UNIVERSIDAD POLITÉCNICA DE MADRID

Departamento de Ingeniería Telemática y Electrónica

Escuela Técnica Superior de Ingeniería y Sistemas de
Telecomunicación

APORTACIONES METODOLÓGICAS PARA EL
DISEÑO DE DESCODIFICADORES DE VÍDEO DE
ÚLTIMA GENERACIÓN SOBRE PLATAFORMAS
MULTI-DSP.

TESIS DOCTORAL

Autor: Miguel Chavarrías Lapastora

Máster en Ingeniería de Sistemas y Servicios para la Sociedad de la
Información

Directores:

Fernando Pescador del Oso

Doctor Ingeniero de Telecomunicación

Matías J. Garrido González

Doctor Ingeniero de Telecomunicación

Junio 2017

TÍTULO: Aportaciones metodológicas para el diseño de descodificadores de vídeo de última generación sobre plataformas Multi-DSP

AUTOR: D. Miguel Chavarrías Lapastora

DIRECTORES: D. Fernando Pescador del Oso

D. Matías Javier Garrido González

El Tribunal nombrado con fecha XX de XX de 2017 por el Mgfco. Excmo. Sr. Rector de la Universidad Politécnica de Madrid, compuesto por los doctores:

Presidente: D.

Vocal: D.

Vocal: D.

Vocal: D.

Secretario: D.

Suplente: D.

Suplente: D.

realizado el acto de lectura y defensa de la tesis en la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la UPM, acuerda otorgar la calificación de

Madrid, a XX de XX de 2017

EL PRESIDENTE

EL SECRETARIO

LOS VOCALES

*A mis padres Mercedes y Pascual,
por todo su apoyo y cariño
y porque gracias a ellos hice realidad mis ilusiones.*

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis directores de tesis Fernando y Matías todo su apoyo, su ayuda y la guía que han sido para mí durante estos últimos años. Estoy seguro de que este trabajo que ahora cierro no habría sido algo de lo que poder sentirme orgulloso sin su confianza.

También quiero agradecer a todos los miembros de mi grupo de investigación, el GDEM, el apoyo recibido al hacerme sentir un miembro más del equipo, y en especial a César, Edu y Rubén por todo lo que de vosotros aprendo dentro y fuera del laboratorio.

A mi departamento, el DTE, donde me han abierto las puertas desde el primer día que empecé a perderme por sus pasillos. Hoy tengo la suerte de formar parte de este departamento ya como profesor. Espero seguir aprendiendo para llegar a ser un buen docente. No faltan, cerca de mí, grandes ejemplos. No quiero olvidarme de agradecer su dedicación a Marisa y a Rosa, cuyo trabajo muchas veces no se ve, pero sin las cuales, estoy seguro, no saldríamos adelante.

Durante el desarrollo de mi tesis he tenido la suerte de poder realizar mi segunda estancia en el grupo IETR del INSA de Rennes. Quiero agradecer especialmente a Maxime, su atención y cercanía, pero sobre todo esas largas charlas sobre *dataflow* con las que tanto he aprendido, y también a todas las personas que allí trabajan o con las que he compartido mi vida esos meses en el extranjero: Alexandre, Khaled, Karol, Clementine y Avae, *merci beaucoup pour me montrer une région exceptionnelle!*

Quiero dar también las gracias a todos los alumnos que tuvieron el valor de hacer su proyecto fin de carrera, grado o máster conmigo: Pablo R., Fernando, Adrián, Julio, Pablo C., Estefanía, Álvaro, Rafa, Luis y Daniel. Gracias por confiar en mí y por poner vuestra parte en este trabajo que ahora termino, una parte de todo esto es vuestra.

Quiero agradecer a mis padres Mercedes y Pascual, y a mi hermano Guillermo, su apoyo incondicional desde que decidí empezar este camino. Siempre me habéis animado a dedicarme a aquello que más me gustaba, al tiempo que me ayudábais con los problemas, y a veces las dudas, que la vida te va poniendo por delante.

No me olvido de mi primo Pablo y de mis amigos María, Miguel, Óscar, Laura, Fer, Mario, Javi, Alex, Elena, y a mis “Erasmus” de Lavapiés... gracias a todos por sacarme de mi rutina y por reservarme todos esos momentos de fiesta que me he perdido durante la tesis.

Hay una persona muy especial con la que comparto mi vida casi desde el mismo día en el que me matriculé de la tesis. Gracias por aguantarme la eterna escritura de la tesis, mis explicaciones incomprensibles sobre “*de qué va mi tesis*”, las frustraciones tras semanas de bloqueo... pero también por los viajes con mochila y sin ruta fija descubriendo sitios increíbles, por hacerme reír tanto los días buenos como los malos y por tu confianza en todo lo que he necesitado. Gracias por hacerme feliz y por compartir tu vida conmigo, te quiero Miguel.

ÍNDICE GENERAL

AGRADECIMIENTOS	V
ÍNDICE GENERAL	VII
RELACIÓN DE TABLAS	XIII
RELACIÓN DE FIGURAS.....	XV
RESUMEN	XIX
ABSTRACT	XXI
PRIMERA PARTE: INTRODUCCIÓN, OBJETIVOS Y METODOLOGÍA	1
1. INTRODUCCIÓN, OBJETIVOS Y METODOLOGÍA	1
1.1. <i>Introducción</i>	1
1.2. <i>Objetivos de la investigación</i>	3
1.3. <i>Metodología de trabajo</i>	4
1.4. <i>Estructura de la memoria</i>	5
SEGUNDA PARTE: ANTECEDENTES	9
2. CODIFICACIÓN DE VÍDEO DIGITAL	9
2.1. <i>Introducción a la codificación de vídeo digital</i>	10
2.1.1. Origen de la codificación de vídeo digital	10
2.1.2. Fundamentos de la codificación de vídeo	11
2.1.3. El esquema de codificación híbrido.....	12
2.1.4. Estándares de codificación de vídeo	16
2.2. <i>High Efficiency Video Coding</i>	17
2.2.1. Introducción al estándar HEVC.....	17
2.2.1.1. Particionado de imágenes	18
2.2.1.2. Sintaxis de alto nivel	20
2.2.1.3. Perfiles y niveles de HEVC	22
2.2.2. Modelo de un decodificador HEVC.....	24
2.3. <i>MPEG Reconfigurable Video Coding (RVC)</i>	25
2.3.1. Introducción al estándar RVC.....	25
2.3.2. Descripción de decodificadores de vídeo con MPEG RVC	26
2.3.3. RVC-CAL Actor Language, conceptos básicos.....	28
2.3.4. Funcionamiento interno de los actores	30
2.3.5. Implementaciones de decodificadores de vídeo basadas en MPEG RVC	31
2.4. <i>Resumen</i>	33
3. TECNOLOGÍAS DE IMPLEMENTACIÓN	35
3.1. <i>Introducción a las tecnologías de implementación</i>	35
3.2. <i>Tecnologías de implementación basadas en GPPs de altas prestaciones</i>	36
3.2.1. GPPs de altas prestaciones	36
3.2.2. Implementaciones desarrolladas sobre plataformas tipo GPP de altas prestaciones.....	38
3.3. <i>Tecnologías de implementación basadas en GPPs para sistemas empujados</i>	39
3.3.1. GPPs para sistemas empujados.....	39
3.3.2. Implementaciones desarrolladas sobre plataformas GPP para sistemas empujados.	40
3.4. <i>Tecnologías de implementación basadas en DSPs</i>	41
3.4.1. Tecnología DSP.....	41
3.4.2. Implementaciones desarrolladas sobre plataformas tipo DSP	43
3.5. <i>Comparativa entre las distintas implementaciones</i>	43
3.6. <i>Resumen</i>	44

4.	METODOLOGÍAS Y HERRAMIENTAS.....	47
4.1.	<i>Introducción</i>	47
4.2.	<i>Proyecto HEAP</i>	48
4.3.	<i>Proyecto ALMA</i>	49
4.4.	<i>PREESM</i>	50
4.5.	<i>Metodología basada en flujos de datos DPN</i>	52
4.5.1.	Modelo de computación.....	52
4.5.2.	Estrategias de planificación.....	54
4.5.2.1.	Planificación a nivel de acciones.....	54
4.5.2.2.	Planificación a nivel de actores.....	55
4.5.2.3.	Planificación a nivel de procesadores.....	56
4.5.3.	Aplicación de estrategias de planificación sobre plataformas multinúcleo.....	57
4.5.4.	Mapeo dinámico de actores en plataformas multinúcleo.....	58
4.5.5.	Limitaciones de la metodología.....	60
4.6.	<i>Resumen</i>	60
	TERCERA PARTE: TECNOLOGÍAS DE REFERENCIA	63
5.	OPENHEVC.....	63
5.1.	<i>El decodificador OpenHEVC</i>	64
5.2.	<i>Funcionamiento de OpenHEVC</i>	65
5.3.	<i>Resumen</i>	66
6.	OPENMP.....	67
6.1.	<i>Origen de OpenMP</i>	67
6.2.	<i>Ejecución fork/join de OpenMP</i>	68
6.3.	<i>Componentes de OpenMP</i>	69
6.4.	<i>Directivas y funciones de la biblioteca OpenMP</i>	70
6.5.	<i>Cláusulas fundamentales</i>	72
6.6.	<i>Ventajas e inconvenientes de la utilización de OpenMP</i>	72
6.7.	<i>Alternativas a OpenMP sobre plataformas multi-núcleo</i>	74
6.7.1.	Desarrollo <i>ad-hoc</i> : gestión de la memoria y tareas.....	74
6.7.2.	OpenCL.....	74
6.8.	<i>Resumen</i>	76
7.	PLATAFORMAS DE DESARROLLO.....	77
7.1.	<i>La plataforma TMDSEVM6678L</i>	77
7.1.1.	Elección del procesador TMS320C6678.....	78
7.1.2.	Principales características del procesador TMS320C6678.....	79
7.1.3.	Consideraciones sobre las memorias en los procesadores de la familia C66 de Texas Instruments	81
7.2.	<i>Entorno de trabajo para plataformas de Texas Instruments</i>	83
7.3.	<i>Otras plataformas utilizadas en el trabajo de investigación</i>	85
7.4.	<i>Tarjetas de desarrollo KeyStone II: plataformas heterogéneas</i>	87
7.5.	<i>Resumen</i>	89
8.	ENTORNO DE DESARROLLO DE ORCC PARA RVC-CAL.....	91
8.1.	<i>Introducción a Open RVC-CAL compiler</i>	91
8.2.	<i>Entorno de trabajo de Orcc</i>	92
8.2.1.	Requisitos del entorno de trabajo de Orcc.....	92
8.2.2.	Código fuente generado por Orcc.....	93
8.2.3.	Flujo de trabajo con Orcc.....	94
8.3.	<i>Infraestructura interna de Orcc</i>	95
8.3.1.	Descripción general.....	95
8.3.2.	Los <i>back-ends</i> de Orcc.....	96
8.4.	<i>Utilización de módulos optimizados desde el código fuente generado por Orcc</i>	97

8.5.	<i>Resumen</i>	99
CUARTA PARTE: DISEÑO DE DESCODIFICADORES DE VÍDEO SOBRE PLATAFORMAS MULTI-DSP		101
9.	METODOLOGÍA DE DISEÑO	101
9.1.	<i>Introducción</i>	102
9.2.	<i>Generación automática de código</i>	104
9.2.1.	Planteamiento del problema	104
9.2.2.	Solución propuesta	104
9.2.2.1.	Análisis del código generado del fichero Top	104
9.2.2.2.	Análisis del código de las bibliotecas generado por Orcc	105
9.2.2.3.	Modificaciones sobre el código generado de los actores	107
9.2.3.	Generalización	107
9.3.	<i>Paralelización del código para arquitecturas multinúcleo</i>	108
9.3.1.	Planteamiento del problema	108
9.3.2.	Solución propuesta	108
9.3.3.	Generalización	110
9.4.	<i>Gestión del flujo de información y coherencia de caché</i>	111
9.4.1.	Planteamiento del problema	111
9.4.2.	Solución propuesta	113
9.4.2.1.	Rediseño de las FIFOs	115
9.4.2.2.	Inicialización de parámetros	115
9.4.2.3.	Sincronización del procedimiento	116
9.4.2.4.	<i>Buffers de prefetch</i> y efectos sobre el vídeo	117
9.4.2.5.	Modificación sobre el código de los actores	118
9.4.3.	Generalización	119
9.5.	<i>Mapeo de algoritmos a núcleos</i>	119
9.5.1.	Planteamiento del problema	119
9.5.2.	Solución propuesta	120
9.5.2.1.	Análisis y modificaciones sobre el código fuente	121
9.5.2.2.	Compilación de la biblioteca METIS	122
9.5.3.	Generalización	123
9.6.	<i>Control de consumo</i>	125
9.6.1.	Planteamiento del problema	125
9.6.2.	Solución propuesta	125
9.6.2.1.	Modo IDLE	126
9.6.2.2.	Cambio de la frecuencia de reloj	128
9.6.2.3.	Modelo de consumo	130
9.6.2.4.	Integración de las técnicas de reducción de consumo en el código fuente	134
9.6.3.	Generalización	136
9.7.	<i>Mejora del rendimiento mediante módulos optimizados</i>	136
9.7.1.	Planteamiento del problema	136
9.7.2.	Solución propuesta	137
9.7.3.	Generalización	140
9.8.	<i>Personalización de las aplicaciones para las plataformas objetivo</i>	140
9.8.1.	Planteamiento del problema	140
9.8.2.	Solución propuesta	140
9.8.2.1.	Configuración de la plataforma multi-DSP	142
9.8.2.2.	Configuración con OpenMP	144
9.8.3.	Generalización	145
9.9.	<i>Automatización del proceso de diseño</i>	145
9.9.1.	Definición del nuevo <i>back-end</i> DSP	146
9.9.2.	Generación de los ficheros de configuración	148
9.9.3.	Modificación de la plantilla <i>Network Printer</i>	150
9.9.4.	Modificación de la plantilla <i>Instance Printer</i>	152

9.10.	<i>Resumen</i>	153
10.	IMPLEMENTACIÓN DE DESCODIFICADORES HEVC	155
10.1.	<i>El decodificador HEVC-RVC</i>	155
10.1.1.	Decodificador HEVC RVC de referencia	156
10.1.2.	Decodificador HEVC RVC YUV	157
10.1.3.	Decodificador HEVC RVC FBn	159
10.2.	<i>Banco de pruebas y secuencias utilizadas</i>	160
10.3.	<i>Análisis del factor de mejora</i>	161
10.3.1.	Cálculo del factor de mejora	162
10.3.2.	Estimación del factor de mejora de las implementaciones HEVC-RVC con el procesador TMS320C6678	165
10.3.2.1.	Reparto de la carga computacional con la versión HEVC-RVC REF	165
10.3.2.2.	Reparto de la carga computacional con la versión HEVC-RVC YUV	166
10.3.2.3.	Factores de mejora calculados con los diferentes estimadores	168
10.4.	<i>Pruebas de verificación de la metodología de diseño</i>	169
10.4.1.	Pruebas realizadas con la plataforma Multi-DSP TMS320C6678	170
10.4.2.	Verificación sobre plataformas GPP	173
10.4.3.	Verificación con la versión HEVC-RVC FB3	179
10.4.3.1.	Implementación y resultados de la versión HEVC RVC FB3 sobre la plataforma Multi-DSP	180
10.4.3.2.	Implementación y resultados de la versión HEVC RVC FB3 sobre la plataforma GPP de altas prestaciones	182
10.4.4.	Reducción del consumo de energía sobre la plataforma TMS320C6678	183
10.5.	<i>Resumen</i>	187
QUINTA PARTE: RESULTADOS, APORTACIONES Y LÍNEAS DE TRABAJO FUTURO		189
11.	RESULTADOS Y APORTACIONES	189
11.1.	<i>Objetivos de la tesis</i>	189
11.2.	<i>Resultados y aportaciones de la tesis doctoral</i>	190
11.2.1.	Metodología de diseño	191
11.2.2.	Resultados de las implementaciones	191
11.2.3.	Utilización de distribuciones dinámicas de actores	193
11.2.4.	Reducción del consumo energético	194
12.	PUBLICACIONES Y MÉRITOS	195
12.1.	<i>Trabajos publicados en relación con la tesis</i>	195
12.2.	<i>Otros resultados relacionados con la tesis</i>	197
12.2.1.	Proyectos de investigación	198
12.2.2.	Colaboración y estancia en el INSA de Rennes	198
12.2.3.	Dirección de Proyectos Fin de Grado	199
12.2.4.	Becas y premios obtenidos	200
12.3.	<i>Resultados susceptibles de publicación en el futuro</i>	200
13.	LÍNEAS DE TRABAJO FUTURO	201
BIBLIOGRAFÍA		203
ACRÓNIMOS		217
ANEXO I – ARCHIVOS DE CONFIGURACIÓN		221
ANEXO II – DIAGRAMAS DE LOS DESCODIFICADORES RVC-HEVC		223
ANEXO III – ANÁLISIS DE LA UTILIZACIÓN DE MÓDULOS OPENHEVC SOBRE LA PLATAFORMA MULTI-DSP		227
III.1.	ANÁLISIS DE LA VARIACIÓN EN EL RENDIMIENTO SOBRE LA PLATAFORMA DSP	227
ANEXO IV – IMPLEMENTACIÓN DE UN DESCODIFICADOR CON EL CÓDIGO DE REFERENCIA DE HEVC		231

ANEXO V – CÓDIGO RELATIVO A LA GESTIÓN DE INTERRUPCIONES CON EL PROCESADOR TMS320C6678	237
ANEXO VI: SUMMARY OF THE DISSERTATION.....	239
OBJECTIVE OF THIS SUMMARY	239
1. INTRODUCTION AND OBJECTIVES.....	239
1.1. INTRODUCTION.....	239
1.2. WORK METHODOLOGY.....	240
1.3. STRUCTURE OF THE THESIS REPORT	241
2. STATE OF THE ART OF THE RESEARCH WORK.....	242
2.1. DIGITAL VIDEO CODING	242
2.1.1. <i>Basis of the digital video coding</i>	242
2.1.2. <i>High-Efficiency Video Coding</i>	242
2.1.3. <i>MPEG Reconfigurable Video Coding</i>	242
2.2. IMPLEMENTATION TECHNOLOGIES.....	243
2.2.1. <i>High-performance GPPs</i>	243
2.2.2. <i>GPPs for embedded systems</i>	243
2.2.3. <i>Multi-DSP platforms</i>	244
2.3. METHODOLOGIES FOR MULTICORE PROGRAMMING	244
3. CONCERNING TECHNOLOGIES	245
3.1. OPENHEVC.....	245
3.2. OPENMP	245
3.3. DEVELOPMENT PLATFORMS	246
3.4. OPEN RVC-CAL COMPILER – WORKING ENVIRONMENT.....	247
4. DESIGN OF DIGITAL VIDEO DECODERS FOR MULTICORE DSPS	248
5. IMPLEMENTATION OF HEVC DECODERS	250
5.1. THE HEVC DECODERS.....	251
5.2. TEST BENCH	252
5.3. RESULTS AND VERIFICATION OF THE DESIGN METHODOLOGY	252
6. CONTRIBUTIONS, PUBLISHED RESULTS AND FUTURE RESEARCH TOPICS	255
6.1. CONTRIBUTIONS	255
6.1.1. <i>Design Methodology</i>	255
6.1.2. <i>Results of the implementations</i>	256
6.1.3. <i>Power consumption reduction</i>	256
6.2. PUBLISHED WORKS AND OTHER MERITS	256
6.2.1. <i>Published results</i>	256
6.2.2. <i>Other merits achieved</i>	258
6.3. FUTURE RESEARCH LINES	258
6.3.1. <i>Heterogeneous platforms</i>	258
6.3.2. <i>Future Video Coding</i>	259
6.3.3. <i>Multi-application extension</i>	259

RELACIÓN DE TABLAS

TABLA 1. DIFERENCIA ENTRE LOS ÓRDENES DE PRESENTACIÓN Y CODIFICACIÓN DE LAS IMÁGENES.	16
TABLA 2. LISTADO DE LOS DISTINTOS TIPOS DE CABECERAS NAL EN EL ESTÁNDAR HEVC.	21
TABLA 3. LISTADO DE PERFILES INCLUIDOS EN EL ESTÁNDAR HEVC.	23
TABLA 4. VALORES LÍMITE PARA LOS PRINCIPALES NIVELES DEL PERFIL <i>MAIN</i>	23
TABLA 5. ELEMENTOS EXISTENTES EN EL DIAGRAMA FNL DE LA FIGURA 12.	28
TABLA 6. COMPARATIVA DE LAS DISTINTAS IMPLEMENTACIONES ANALIZADAS.	44
TABLA 7. PROPIEDADES DE LOS DISTINTOS TIPOS DE <i>MOCS</i> BASADOS EN FLUJOS DE DATOS.	53
TABLA 8. RELACIÓN DE PARÁMETROS DE CONFIGURACIÓN CON MAPEOS DINÁMICOS.	59
TABLA 9. ALGORITMOS DE MAPEO DINÁMICO.	59
TABLA 10. RELACIÓN DE CARACTERÍSTICAS SOPORTADAS POR <i>OPENHEVC</i>	65
TABLA 11. DIRECTIVAS DE <i>OPENMP</i> PARA LA GESTIÓN DE REGIONES PARALELAS.	71
TABLA 12. CLÁUSULAS FUNDAMENTALES DE <i>OPENMP</i>	72
TABLA 13. RELACIÓN DE PROS Y CONTRAS AL UTILIZAR <i>OPENMP</i>	73
TABLA 14. PRINCIPALES CARACTERÍSTICAS DE LA PLATAFORMA <i>TMDSEVM6678L</i>	77
TABLA 15. RESUMEN DE CARACTERÍSTICAS DEL PROCESADOR <i>TMS320C6678</i>	79
TABLA 16. MÓDULOS UTILIZADOS DEL PAQUETE <i>MCSDK</i> DE TEXAS INSTRUMENTS.	85
TABLA 17. RELACIÓN DE PROCESADORES UTILIZADOS Y SUS CARACTERÍSTICAS.	87
TABLA 18. RESUMEN DE CARACTERÍSTICAS DE LOS PROCESADORES DEL <i>SOC 66AK2H14</i>	88
TABLA 19. EJEMPLO DE FUNCIONAMIENTO DEL ESQUEMA DEL <i>WRAPPER</i> ENTRE <i>OPENHEVC</i> Y <i>RVC</i>	99
TABLA 20. RELACIÓN DE PLATAFORMAS Y DESCODIFICADORES IMPLEMENTADOS DURANTE EL DISEÑO DE LA METODOLOGÍA.	102
TABLA 21. RESUMEN DE MODIFICACIONES REALIZADAS EN EL FICHERO <i>TOP</i> GENERADO POR <i>ORCC</i>	105
TABLA 22. CÓDIGO DEL LANZADOR QUE ARRANCA EL PROCESO DE DESCODIFICACIÓN.	110
TABLA 23. FUNCIONES PARA LA GESTIÓN DE LAS <i>FIFOS</i> POR PARTE DE LOS ACTORES.	118
TABLA 24. PSEUDOCÓDIGO DE LAS NUEVAS FUNCIONES DE GESTIÓN DE <i>FIFOS</i> EN LOS ACTORES.	118
TABLA 25. PARÁMETROS REGISTRADOS PARA EL CÁLCULO DE MAPEOS DINÁMICOS.	121
TABLA 26. MODIFICACIÓN DE FUNCIONES EN LAS BIBLIOTECAS DE <i>METIS</i>	123
TABLA 27. INSTRUCCIONES PARA LA MODIFICACIÓN DE LA FRECUENCIA DEL RELOJ.	129
TABLA 28. RELACIÓN DE MACROS DESHABILITADAS EN EL ENTORNO <i>MULTI-DSP</i>	138
TABLA 29. RELACIÓN DE CARPETAS Y FICHEROS EXCLUIDOS DE COMPILACIÓN.	138
TABLA 30. RELACIÓN DE ACTORES Y FUNCIONES OPTIMIZADAS.	139
TABLA 31. MAPA DE MEMORIA DE LA PLATAFORMA <i>TMS320C6678</i>	142
TABLA 32. MODIFICACIÓN DEL MAPA DE MEMORIA DE LA PLATAFORMA <i>TMS320C6678</i>	142
TABLA 33. COMANDOS DE CONFIGURACIÓN DEL FICHERO <i>CFG</i>	143
TABLA 34. MAPA DE MEMORIA CON <i>OPENMP</i>	144
TABLA 35. PARÁMETROS DE CONFIGURACIÓN DE <i>OPENMP</i> EN EL FICHERO <i>CFG</i>	145
TABLA 36. RELACIÓN DE MODIFICACIONES INTRODUCIDAS EN LA PLANTILLA <i>NETWORK PRINTER</i>	150
TABLA 37. SECCIÓN DE CÓDIGO ENCARGADO DE LA DECLARACIÓN Y CONFIGURACIÓN DE LAS <i>FIFOS</i>	151
TABLA 38. COMPARATIVA ENTRE EL CÓDIGO GENERADO RELATIVO A LA DECLARACIÓN DE <i>FIFOS</i> ENTRE LOS <i>BACK-ENDS C</i> Y <i>DSP</i>	151
TABLA 39. RELACIÓN DE FUNCIONALIDADES MODIFICADAS DE LA PLANTILLA <i>INSTANCE PRINTER</i>	152
TABLA 40. RELACIÓN ENTRE PARTE DEL CÓDIGO INSERTADO EN LA PLANTILLA Y EL GENERADO POSTERIORMENTE.	153
TABLA 41. RELACIÓN DE BLOQUES FUNCIONALES DEL DESCODIFICADOR <i>HEVC-RVC</i>	157
TABLA 42. CARACTERÍSTICAS DE LAS <i>FUS</i> PRINCIPALES QUE INTEGRAN EL DESCODIFICADOR <i>HEVC-RVC</i>	157
TABLA 43. CARACTERÍSTICAS DE LAS PRINCIPALES <i>FUS</i> QUE INTEGRAN EL DESCODIFICADOR <i>HEVC-RVC YUV</i>	158
TABLA 44. LISTADO Y CARACTERÍSTICAS DE LAS SECUENCIAS DE PRUEBA UTILIZADAS.	161

TABLA 45. COMPARATIVA DE LOS RESPECTIVOS FACTORES DE MEJORA MÁXIMOS ESPERADOS PARA LAS DISTINTAS IMPLEMENTACIONES.....	169
TABLA 46. RELACIÓN DE PLATAFORMAS Y DESCODIFICADORES IMPLEMENTADOS DURANTE EL DISEÑO DE LA METODOLOGÍA.	170
TABLA 47. COMPARATIVA DEL RENDIMIENTO (EN FPS) DE LAS IMPLEMENTACIONES CON OPENMP Y THREADS DEL DESCODIFICADOR HEVC-RVC SOBRE PLATAFORMAS PC Y ODROID PARA VARIOS NÚCLEOS.	179
TABLA 48. MODIFICACIÓN DEL MAPA DE MEMORIA DE LA PLATAFORMA EVMK2H.....	180
TABLA 49. RENDIMIENTO MEDIO, EN FPS, Y FACTOR DE GANANCIA CON DISTINTAS ESTRATEGIAS DE MAPEO DINÁMICO SOBRE LA PLATAFORMA EVMK2H.	181
TABLA 50. RENDIMIENTO MEDIO, EN FPS, Y FACTOR DE GANANCIA CON DISTINTAS ESTRATEGIAS DE MAPEO DINÁMICO. ...	183
TABLA 51. ESCENARIOS DE CONSUMO SELECCIONADOS EN CADA MOMENTO DURANTE LA DESCODIFICACIÓN DE VARIAS SECUENCIAS CONCATENADAS.....	185
TABLA 52. COMPARATIVA DEL CONSUMO ENERGÉTICO Y TIEMPO DE DESCODIFICACIÓN DE DOS ESCENARIOS.	187
TABLA 53. COMPARATIVA DEL FACTOR DE MEJORA CONSEGUIDO PARA LAS DISTINTAS IMPLEMENTACIONES CON EL PROCESADOR TMS320C6678.	192
TABLA 54. VARIACIÓN DEL RENDIMIENTO POR ACTORES DEL DESCODIFICADOR HEVC-RVC YUV (v1) AL UTILIZAR MÓDULOS DE OPENHEVC (v2) CON UNA SECUENCIA DE QP 32.	229
TABLA 55. VARIACIÓN DEL RENDIMIENTO POR ACTORES DEL DESCODIFICADOR HEVC-RVC YUV (v1) AL UTILIZAR MÓDULOS DE OPENHEVC (v2) CON UNA SECUENCIA DE QP 27.	230
TABLA 56. RENDIMIENTO EN CLKs Y FPS DEL DESCODIFICADOR HM v9.0 SOBRE UN DSP.....	235

RELACIÓN DE FIGURAS

FIGURA 1. ESQUEMAS DE MUESTRO.	12
FIGURA 2. ESTRUCTURA DE UN MACROBLOQUE SEGÚN DISTINTOS ESQUEMAS DE MUESTREO.....	12
FIGURA 3. DIAGRAMA DE BLOQUES DE UN CODIFICADOR HÍBRIDO.	14
FIGURA 4. REPRESENTACIÓN DE UNA SECUENCIA CON IMÁGENES I, P Y B.	16
FIGURA 5. REPRESENTACIÓN CRONOLÓGICA DEL DESARROLLO DE LOS PRINCIPALES ESTÁNDARES DE CODIFICACIÓN DE VÍDEO.	17
FIGURA 6. SUBDIVISIÓN DE UN CTB EN CBS Y TBS. LAS LÍNEAS SÓLIDAS REPRESENTAN SUBDIVISIONES EN CBS Y LAS PUNTEADAS EN TBS.....	19
FIGURA 7. EJEMPLO DE LA DIVISIÓN DE BLOQUES REALIZADA EN UNA IMAGEN POR UN CODIFICADOR H.264 (IZQUIERDA) Y POR UN CODIFICADOR HEVC (DERECHA).	20
FIGURA 8. REPRESENTACIÓN GRÁFICA DE LOS NIVELES JERÁRQUICOS DE UNA TRAMA DE VÍDEO CODIFICADO BAJO EL ESTÁNDAR HEVC.	21
FIGURA 9. SUBDIVISIÓN DE UNA IMAGEN EN <i>SLICES</i> (A) Y EN <i>TILES</i> (B).	22
FIGURA 10. DIAGRAMA DE BLOQUES SIMPLIFICADO DE UN DESCODIFICADOR HEVC.	24
FIGURA 11. MODELO DE ADMs UTILIZADO EN EL ESTÁNDAR RVC.....	27
FIGURA 12. DIAGRAMA BÁSICO DE UN MODELO BASADO EN FNL.	28
FIGURA 13. CABECERA Y CUERPO EN LENGUAJE RVC-CAL DEL ACTOR MERGEYUV DEL PROYECTO <i>RESEARCH</i>	29
FIGURA 14. EJEMPLO DE SECCIÓN DE CÓDIGO NO DETERMINÍSTICA EN RVC-CAL.	30
FIGURA 15. EJEMPLO DE CÓDIGO EN RVC-CAL DE UNA ACCIÓN DEL ACTOR QpGEN CON CONDICIÓN DE GUARDA.	30
FIGURA 16. DIAGRAMA DEL FUNCIONAMIENTO INTERNO DE LOS ACTORES.	31
FIGURA 17. DIAGRAMA DE BLOQUES DE UN PROCESADOR DE LA FAMILIA INTEL I7.....	37
FIGURA 18. DIAGRAMA DE BLOQUES DE UN PROCESADOR DSP MULTINÚCLEO DE ÚLTIMA GENERACIÓN.....	42
FIGURA 19. DIAGRAMA DE BLOQUES DE UNA UNIDAD MAC EN UN PROCESADOR TIPO DSP.	42
FIGURA 20. DIAGRAMA DEL FLUJO DE TRABAJO SEGÚN EL PROYECTO ALMA.....	50
FIGURA 21. FLUJO DE TRABAJO CON PREESM.	51
FIGURA 22. PLANIFICACIÓN A NIVEL DE ACCIONES.....	55
FIGURA 23. EJEMPLO DE DIAGRAMA DPN CON CINCO ACTORES INTERCONECTADOS MEDIANTE FIFOS.....	55
FIGURA 24. ESTRATEGIAS <i>ROUND-ROBIN</i> (IZQUIERDA) Y <i>DATA-DRIVEN/DATA-DEMAND</i> (DERECHA).	56
FIGURA 25. ILUSTRACIÓN DE LA PLANIFICACIÓN A NIVEL DE PROCESADORES Y ACTORES.	57
FIGURA 26. DIAGRAMA DEL PROCESO DE DESCODIFICACIÓN AL UTILIZAR MAPEO DINÁMICO.	60
FIGURA 27. DIAGRAMA DEL FUNCIONAMIENTO DEL DESCODIFICADOR OPENHEVC.	65
FIGURA 28. DIAGRAMA DE LA EJECUCIÓN <i>FORK/JOIN</i> DE OPENMP.	69
FIGURA 29. MODELO DE DIRECTIVA OPENMP.	69
FIGURA 30. EJEMPLO DE DECLARACIÓN Y EJECUCIÓN DE UNA REGIÓN PARALELA CON OPENMP.....	70
FIGURA 31. DIAGRAMA DE LA DISTRIBUCIÓN JERÁRQUICA CON OPENCL.	75
FIGURA 32. FOTOGRAFÍA DE LA PLATAFORMA DE DESARROLLO TMDSEVM6678L, REMARCADO EN ROJO EL EMULADOR XDS560v2.	78
FIGURA 33. DIAGRAMA DE BLOQUES DEL PROCESADOR TMS320C6678 [C6678].	80
FIGURA 34. DIAGRAMA DE VELOCIDADES DE TRANSMISIÓN DE DATOS ENTRE LAS DISTINTAS MEMORIAS DEL PROCESADOR TMS320C6678.....	81
FIGURA 35. POSIBLES DISTRIBUCIONES DE LA MEMORIA L1P COMO MEMORIA CACHÉ O SRAM EN EL PROCESADOR TMS320C6678 [C6678].	82
FIGURA 36. DIAGRAMA DEL FLUJO DE TRABAJO CON CODE COMPOSER STUDIO.	84
FIGURA 37. FOTOGRAFÍA DE LA PLATAFORMA ODROID U3.....	86
FIGURA 38. DIAGRAMA DE BLOQUES DE LA PLATAFORMA ODROID U3.....	86
FIGURA 39. DIAGRAMA DE BLOQUES DEL SoC 66AK2H14 [AK2H].....	88

FIGURA 40. DIAGRAMA DEL FLUJO DE TRABAJO EN EL ENTORNO DE ORCC.....	94
FIGURA 41. DIAGRAMA DE LA INFRAESTRUCTURA DEL PROCESO DE COMPILACIÓN DE ORCC.	96
FIGURA 42. DIAGRAMA DE MÓDULOS INTERNOS DEL <i>BACK-END C</i>	97
FIGURA 43. SECCIÓN DE CÓDIGO QUE PERMITE LA UTILIZACIÓN DE FUNCIONES OPTIMIZADAS DE OPENHEVC SOBRE CÓDIGO GENERADO CON ORCC.	97
FIGURA 44. ESQUEMA DE UTILIZACIÓN DE MÓDULOS OPTIMIZADOS DE OPENHEVC SOBRE RVC.	98
FIGURA 45. INSTRUCCIONES PARA DEFINIR EL TAMAÑO DE LAS CACHÉS L1 Y L2.	105
FIGURA 46. ACTIVACIÓN DE LAS CONSTANTES GLOBALES PARA ENTORNO MULTI-DSP.	107
FIGURA 47. MODIFICACIÓN DE LA INCLUSIÓN DEL FICHERO <i>THREAD.H</i>	109
FIGURA 48. REDEFINICIÓN DE FUNCIONES PARA EL MANEJO DE SEMÁFOROS CON OPENMP.	109
FIGURA 49. ACTIVACIÓN DE OPENMP EN EL FICHERO <i>CMAKELISTS.TXT</i>	110
FIGURA 50. EJEMPLO DEL PROCESO DE LECTURA/ESCRITURA MEDIANTE UNA FIFO Y DOS ACTORES.	111
FIGURA 51. DIAGRAMA DEL ACCESO DE LOS ACTORES A LAS FIFOS.	113
FIGURA 52. COMPARACIÓN ENTRE LA ESTRUCTURA DE LAS FIFOS ORIGINAL (IZQUIERDA) Y LA MODIFICADA (DERECHA). ...	115
FIGURA 53. DIAGRAMA DE TRABAJO DURANTE LA INICIALIZACIÓN DE PARÁMETROS DE LAS FIFOS.	116
FIGURA 54. SECCIÓN DE CÓDIGO QUE IMPLEMENTA EL REGISTRO DEL NÚMERO DE NÚCLEO.	116
FIGURA 55. ARTEFACTOS (EN VERDE) PRODUCIDOS POR LA NO-INVALIDACIÓN DE LOS <i>BUFFERS DE PREFETCH</i>	117
FIGURA 56. INSTRUCCIÓN PARA EFECTUAR UN BORRADO DEL <i>BUFFER</i> DE <i>PREFETCH</i>	117
FIGURA 57. DIAGRAMA DE FLUJO TRABAJANDO CON MAPEOS DINÁMICOS O ESTÁTICOS.	120
FIGURA 58. TOMA DE MEDIDAS DURANTE LA EJECUCIÓN DEL PLANIFICADOR.	122
FIGURA 59. DIAGRAMA DEL FUNCIONAMIENTO GENERAL DEL DESCODIFICADOR CON <i>THREADS</i> Y OPENMP.	124
FIGURA 60. INSTRUCCIONES PARA ACTIVAR EL MODO <i>IDLE</i>	127
FIGURA 61. DIAGRAMA DE FUNCIONAMIENTO CON MODO <i>IDLE</i> DE UN DESCODIFICADOR BASADO EN RVC.....	128
FIGURA 62. CÓDIGO DE LLAMADA A LAS INTERRUPCIONES PARA SACAR DEL MODO <i>IDLE</i> A DOS NÚCLEOS.....	128
FIGURA 63. DIAGRAMA DEL CIRCUITO PARA LA GENERACIÓN DE LAS SEÑALES DE RELOJ DEL PROCESADOR TMS320C6678.	129
FIGURA 64. INSTRUCCIONES PARA EL BLOQUEO/DESBLOQUEO DE LOS REGISTROS KICK.....	130
FIGURA 65. CONSUMO DE CORRIENTE DURANTE EL PROCESO DE ARRANQUE DE LA PLATAFORMA TMS320C6678.	130
FIGURA 66. EVOLUCIÓN DE LA CORRIENTE DEMANDADA POR LA PLATAFORMA TMDSEVM6678 EXPRESADA EN AMPERIOS, PARA DISTINTAS FRECUENCIAS DE RELOJ Y NÚMERO DE NÚCLEOS.	132
FIGURA 67. CORRIENTE CONSUMIDA POR EL PROCESADOR TMS320C6678 PARA DISTINTOS ESCENARIOS DE FRECUENCIA Y NÚMERO DE NÚCLEOS.....	133
FIGURA 68. DECLARACIÓN DE MATRICES CON LOS ESCENARIOS DE CONSUMO.....	134
FIGURA 69. DIAGRAMA DE FUNCIONAMIENTO DEL PROCESO DE DESCODIFICACIÓN CON CONTROL DE CONSUMO ENERGÉTICO.	135
FIGURA 70. SECCIÓN DE CÓDIGO QUE MIDE EL NÚMERO DE FPS DESCODIFICADOS.	139
FIGURA 71. CAPTURA DE LA HERRAMIENTA DE CCS PARA CONFIGURAR NUEVAS PLATAFORMAS.....	141
FIGURA 72. ALOJAMIENTO DE UN ARRAY EN MEMORIA MSM.....	143
FIGURA 73. PROYECTOS DEL COMPILADOR ORCC IMPORTADOS EN ECLIPSE.....	146
FIGURA 74. ASPECTO DE LA VENTANA DE OPCIONES DEL COMPILADOR ORCC SOBRE ECLIPSE.	148
FIGURA 75. SECCIÓN DE CÓDIGO ENCARGADA DE LA GENERACIÓN DE LOS FICHEROS DE CONFIGURACIÓN.	149
FIGURA 76. SECCIÓN DE CÓDIGO ENCARGADA DE LA GESTIÓN DE BIBLIOTECAS Y ARCHIVOS EN EL <i>BACK-END</i>	150
FIGURA 77. DIAGRAMA DEL NIVEL MÁS ALTO DE LA JERARQUÍA DEL DESCODIFICADOR HEVC-RVC DE REFERENCIA.	156
FIGURA 78. DIAGRAMA DEL NIVEL MÁS ALTO DE LA JERARQUÍA DEL DESCODIFICADOR HEVC-RVC YUV.	158
FIGURA 79. DIAGRAMA SIMPLIFICADO DEL DESCODIFICADOR HEVC-RVC YUV.....	159
FIGURA 80. DIAGRAMA SIMPLIFICADO DEL NIVEL MÁS ALTO DE LA JERARQUÍA DEL DESCODIFICADOR HEVC-RVC FB3.....	160
FIGURA 81. VARIACIÓN DE LA CARGA COMPUTACIONAL DE LAS DISTINTAS FUS AL UTILIZAR 1 Y 7 NÚCLEOS CON LA VERSIÓN HEVC-RVC YUV.	163
FIGURA 82. FACTORES DE MEJORA CALCULADOS PARA UN NÚMERO DE NÚCLEOS UTILIZADOS Y DISTINTOS PORCENTAJES DE CARGA PARALELIZABLE.	164

FIGURA 83. REPARTO DE LA CARGA COMPUTACIONAL POR FUS DEL DESCODIFICADOR HEVC-RVC REF SOBRE LA PLATAFORMA TMDSEVM320C6678.	166
FIGURA 84. REPARTO DE LA CARGA COMPUTACIONAL MEDIA POR FUS DEL DESCODIFICADOR HEVC-RVC YUV AL DESCODIFICAR LAS SECUENCIAS <i>LOW-DELAY</i> CON LA PLATAFORMA TMDSEVM320C6678.	167
FIGURA 85. REPARTO DE LA CARGA COMPUTACIONAL POR FUS DEL DESCODIFICADOR HEVC-RVC YUV AL DESCODIFICAR SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA TMDSEVM320C6678.	168
FIGURA 86. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF CON Y SIN MÓDULOS OPTIMIZADOS Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA TMDSEVM6678, ESTRATEGIA DE MAPEO MKEC.	171
FIGURA 87. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA TMDSEVM6678, ESTRATEGIA DE MAPEO MKEC.	171
FIGURA 88. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV CON Y SIN MÓDULOS OPTIMIZADOS Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA TMDSEVM6678, ESTRATEGIA DE MAPEO DINÁMICO MKEC.	172
FIGURA 89. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA TMDSEVM6678, ESTRATEGIA DE MAPEO DINÁMICO MKEC.	173
FIGURA 90. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA PC, ESTRATEGIA DE MAPEO MKEC.	174
FIGURA 91. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA ODROID, ESTRATEGIA DE MAPEO MKEC.	175
FIGURA 92. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA PC, ESTRATEGIA DE MAPEO MKEC.	176
FIGURA 93. FACTOR DE MEJORA Y FPS CON HEVC-RVC REF Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA ODROID, ESTRATEGIA DE MAPEO MKEC.	176
FIGURA 94. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA PC, ESTRATEGIA DE MAPEO MKEC.	177
FIGURA 95. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV Y SECUENCIAS <i>LOW-DELAY</i> SOBRE LA PLATAFORMA ODROID, ESTRATEGIA DE MAPEO MKEC.	177
FIGURA 96. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA PC, ESTRATEGIA DE MAPEO MKEC.	178
FIGURA 97. FACTOR DE MEJORA Y FPS CON HEVC-RVC YUV Y SECUENCIAS <i>ALL-INTRA</i> SOBRE LA PLATAFORMA ODROID, ESTRATEGIA DE MAPEO MKEC.	178
FIGURA 98. EVOLUCIÓN DEL RENDIMIENTO DURANTE LA DESCODIFICACIÓN DE UNA SECUENCIA <i>ALL-INTRA</i> ENTRE 1 Y 4 NÚCLEOS Y FACTOR DE GANANCIA OBTENIDO CADA 50 IMÁGENES SOBRE LA PLATAFORMA EVMK2H, ESTRATEGIA MKEC.	182
FIGURA 99. EVOLUCIÓN DEL RENDIMIENTO DURANTE LA DESCODIFICACIÓN DE UNA SECUENCIA <i>ALL-INTRA</i> EMPLEANDO ENTRE 1 Y 4 NÚCLEOS Y FACTOR DE GANANCIA OBTENIDO CADA 50 IMÁGENES SOBRE UNA PLATAFORMA GPP DE ALTAS PRESTACIONES, ESTRATEGIA MKEC.	183
FIGURA 100. ENERGÍA CONSUMIDA EN EL PROCESO DE DESCODIFICACIÓN POR EL PROCESADOR TMS320C6678 PARA DISTINTOS ESCENARIOS DE CONSUMO CON LA VERSIÓN HEVC-RVC YUV.	184
FIGURA 101. DESCODIFICACIÓN DE VARIAS SECUENCIAS CONCATENADAS Y EVOLUCIÓN DEL RENDIMIENTO.	186
FIGURA 102. DIAGRAMA CON LA EVOLUCIÓN DEL TRABAJO REALIZADO POR EL DOCTORANDO, PUBLICACIONES Y TRABAJOS RELACIONADOS CON LA INVESTIGACIÓN.	190
FIGURA 103. COMPARATIVA GENERAL DE LOS RESULTADOS OBTENIDOS CON TRES PLATAFORMAS MULTINÚCLEO.	193
FIGURA 104. DISTRIBUCIÓN JERÁRQUICA DE LAS CLASES UTILIZADAS POR EL DESCODIFICADOR HM 9.0.	232
FIGURA 105. DIAGRAMA DE FLUJO DEL FUNCIONAMIENTO DEL DESCODIFICADOR HM V9.0.	233
FIGURA 106. BANCO DE PRUEBAS UTILIZADO PARA LA TOMA DE MEDIDAS CON LAS DISTINTAS VERSIONES DE LOS DESCODIFICADORES IMPLEMENTADOS.	234
FIGURA 109. CARGA COMPUTACIONAL POR FUS PARA DISTINTAS CONFIGURACIONES.	235

RESUMEN

Esta tesis doctoral se enmarca dentro de las líneas de investigación centradas en la codificación y decodificación de vídeo digital que durante los últimos años desarrolla en la Universidad Politécnica de Madrid el Grupo de Diseño Electrónico y Microelectrónico. Dentro de estas líneas de investigación se han llevado a cabo varias tesis doctorales, donde se han propuesto distintas metodologías para la implementación de codificadores y decodificadores de vídeo sobre arquitecturas *hardware* específicas, así como para su optimización en tiempo de ejecución. Uno de los objetivos principales de estas metodologías es la reducción de los tiempos de desarrollo. Sin embargo, todas estas propuestas están centradas en arquitecturas integradas por un solo procesador. Tanto el incremento en la complejidad de los algoritmos que integran los nuevos estándares de codificación, como la utilización de procesadores multinúcleo, hacen necesaria la concepción de nuevas metodologías de diseño que aporten soluciones flexibles para la implementación de los nuevos decodificadores sobre este tipo de plataformas.

Durante la última década se ha generalizado el uso de todo tipo de arquitecturas y plataformas multinúcleo en los terminales multimedia. Los Procesadores Digitales de Señal (DSP) multinúcleo (Multi-DSP) están específicamente diseñados para mejorar el procesamiento de algoritmos como los incluidos en los decodificadores de vídeo. Sin embargo, la programación de aplicaciones para procesadores Multi-DSP no es una tarea trivial. Esto se debe principalmente a que, por una parte, el código de referencia de los decodificadores suele estar optimizado para otras arquitecturas y debe migrarse a este entorno, lo que implica largos procesos de desarrollo y, por otra, el soporte software para el desarrollo de las aplicaciones es escaso, ya que este tipo de plataformas no disponen de un Sistema Operativo (SO) que facilite la tarea del programador. Así, resulta necesario buscar soluciones metodológicas que permitan aprovechar las ventajas que las plataformas DSP ofrecen en el procesamiento de algoritmos como los que se integran en las aplicaciones multimedia manteniendo tiempos de desarrollo razonables.

El principal objetivo de esta tesis es el desarrollo de una metodología de diseño que permita implementar decodificadores de vídeo de última generación sobre plataformas Multi-DSP.

Para lograr este objetivo, en primer lugar, se ha realizado un profundo estudio sobre el estado del arte en codificación y decodificación de vídeo. En este sentido, el recientemente estandarizado *High Efficiency Video Coding* (HEVC) es el códec más novedoso de entre los existentes. De forma complementaria a los distintos estándares de codificación, el estándar *Reconfigurable Video Coding* (RVC) permite el desarrollo de los mismos en base a una descripción de alto nivel mediante unidades funcionales y flujos de datos. Dicho estándar dota de un elevado grado de flexibilidad y modularidad al proceso de diseño e implementación de decodificadores, exponiendo un elevado grado de paralelismo que facilita su ejecución en entornos multinúcleo.

Seguidamente, se ha llevado a cabo un análisis de las distintas tecnologías utilizadas en la implementación de decodificadores de vídeo y se han evaluado las distintas metodologías para el diseño de aplicaciones en entornos multinúcleo. La metodología que se propone en esta tesis tiene dos características que no poseen las metodologías que se han evaluado: que está específicamente pensada para facilitar el diseño con tecnologías DSP multinúcleo (aunque es lo suficientemente general como para ser de utilidad con otros tipos de arquitecturas) y que está muy orientada al desarrollo de codecs de vídeo (porque está basada en RVC).

Para el desarrollo de decodificadores sobre plataformas Multi-DSP ha sido necesario evaluar, seleccionar y utilizar distintas herramientas y tecnologías que dan soporte a aspectos clave como la paralelización del código, la posibilidad de optimizar alguno de los algoritmos que integran los decodificadores, o la generación automática del código fuente. Las distintas soluciones adoptadas, así como su justificación, han constituido también una parte importante de este trabajo.

A partir de la experiencia de diseño obtenida en la implementación de diferentes decodificadores en plataformas Multi-DSP se ha sintetizado una metodología de diseño, que además se ha automatizado incorporándola a *Open RVC Cal compiler* (Orcc), una herramienta de código abierto que permite la generación automática de código a partir de modelos RVC-CAL. Es más, la metodología propuesta es lo suficientemente general como para permitir la realización de implementaciones sobre plataformas multinúcleo basadas en Procesadores de Propósito General (GPPs).

ABSTRACT

This Ph.D. work is integrated into the digital video coding and decoding research lines of the Electronic and Microelectronic Research Group (GDEM), Universidad Politécnica de Madrid. Previous dissertations were mainly focused in the implementation of video codecs over specific hardware architectures, specifically in performance optimization and development time reduction. However, these proposals centered their attention on single-core architectures. Nowadays, the increase in the algorithmic complexity of new coding standards, together with the massive adoption of multicore processors, is pushing towards the conception of new design methodologies that improve designer's efficiency. Succeeding in this goal, would help in providing more flexible solutions for the implementation of new video decoders over multicore platforms.

During the last decade, the use of a large range of multicore architectures in multimedia devices has become widespread. The Multicore Digital Signal Processors (Multi-DSP) are specially designed to improve the performance of algorithms like those used on video decoders. However, programming this kind of processors has posed two main difficulties. On one side, the reference code of video codecs is typically optimized for other, more generic platforms, so it needs to be efficiently migrated, which involves long development times. Besides, software support for these platforms is reduced, as proven by the fact, among others, that no operating system is usually provided to both, help the development tasks and support the applications to take advantage of the hardware resources. Therefore, the proposal of new methodological solutions becomes necessary to take advantage of the Multi-DSP benefits while reducing the time to market.

The main objective of this thesis is the development of a design methodology that allows the implementation of state-of-the-art video decoders for Multi-DSP platforms.

First, a detailed study about the state-of-the-art on video codecs has been done. In this regard, the High Efficiency Video Coding (HEVC) standard is the newest one and it has been recently standardized. Complementary to the coding tools, the Reconfigurable Video Coding (RVC) standard allows the development of video codecs from a high-level description based on the paradigm of dataflow models of computation. The use of RVC-based solutions introduces flexibility, modularity and reusability of video codecs implementations. Also, it eases the runtime management and the parallel execution of the different actor, i.e., functional units, of the dataflow graph on the available computational resources.

Secondly, a review of video coding implementation technologies has been done too, mainly focused on DSPs and GPPs platforms. Besides, an analysis of the state-of-

the-art on multicore programming methodologies has been carried out. As a result, it turned that none of them had addressed the main target of this work: to contribute to improve the design phase for Multi-DSP technologies. Although the proposal contained herein is highly focused on the development of video codecs, as it is based on the RVC standard, it is general enough to be useful with other types of architectures and applications.

During the development of this work, it has been necessary to evaluate, select and use different tools and technologies in order to develop video decoders for Multi-DSP platforms under the analyzed conditions. These tools support key aspects such as code parallelization, algorithm optimization features (for the decoders), as well as automatic source code generation.

Finally, the proposed design methodology is based on the experience obtained during the practical implementation of different video decoders over Multi-DSP platforms. The methodology and tools developed in this work have been incorporated into Open RVC CAL Compiler (Orcc), an open source tool that allows the automatic source code generation from RVC-based models. Moreover, the proposed methodology is general enough to allow the implementation of generic multicore applications on GPP-based platforms.

PRIMERA PARTE: Introducción, objetivos y metodología

En esta primera parte de la memoria se realiza la introducción a la temática de la tesis doctoral, se resumen los objetivos de la investigación, la metodología seguida durante el desarrollo del trabajo de investigación y se define la estructura del resto de la memoria.

1. Introducción, objetivos y metodología

1.1. Introducción

La constante evolución de los estándares de codificación de vídeo y las sucesivas modificaciones que se efectúan sobre los mismos, hace que el tiempo disponible para el desarrollo de nuevas versiones de los codificadores y decodificadores sea cada vez más corto.

La codificación y la decodificación de vídeo son aplicaciones muy exigentes que requieren soportes tecnológicos con una capacidad de cálculo elevada. En la actualidad, las diferentes alternativas tecnológicas que se utilizan para implementar este tipo de aplicaciones pueden clasificarse en tres grandes grupos: las basadas en procesadores de propósito general, las basadas en arquitecturas específicas y las basadas en procesadores digitales de señal.

Desde el año 1996, el Grupo de Diseño Electrónico y Microelectrónico (GDEM) de la E.T.S.I.S. de Telecomunicación¹ (ETSIST) perteneciente a la Universidad

¹ Antigua Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones (EUITT), campus sur UPM (Madrid).

Politécnica de Madrid (UPM) tiene una línea de investigación en arquitecturas *hardware* orientadas a la codificación de vídeo digital. En el marco de esta investigación se desarrollaron dos tesis doctorales [San98] [Gar04] en las que se propusieron arquitecturas específicas avanzadas para la realización de codificadores de vídeo.

Sin embargo, a partir de 2004, algunas compañías comenzaron a comercializar Procesadores Digitales de Señal (DSP) de muy altas prestaciones, con una capacidad de cómputo elevada, lo que permite implementar aplicaciones de codificación y descodificación de vídeo sobre ellos. Adicionalmente, estos procesadores incorporan una serie de periféricos (puertos de vídeo de entrada y salida, convertidores digitales-analógicos o acceso a la red local) que facilitan las interfaces propias de este tipo de aplicaciones, permitiendo implementar sistemas completos en un solo chip (System on Chip o SoC).

Las ventajas de estos procesadores respecto a las arquitecturas *hardware* específicas se encuentran en la facilidad para modificar el estándar que implementan, ya que sólo supone una sustitución del *software* que ejecuta el procesador. Esta versatilidad permite disponer de sistemas de codificación/descodificación con tiempos de desarrollo menores que los obtenidos con implementaciones basadas en arquitecturas *hardware*. Como contrapartida, tanto el precio como el consumo de los DSPs son superiores a los de los sistemas diseñados específicamente para implementar un estándar de codificación.

Otro tipo de SoC son aquellos integrados por Procesadores de Propósito General (GPPs). Si bien históricamente los GPPs han estado asociados a dispositivos de altas prestaciones en terminales fijas, los fabricantes de chips ya comercializan dispositivos con menores prestaciones que los GPPs clásicos destinados a integrarse en terminales móviles.

Dentro del proyecto PccMuTe [PCC] y empleando DSPs de última generación como soporte tecnológico, se ha desarrollado dentro del grupo de investigación una tesis doctoral [Pes11] en la que se han propuesto metodologías para la optimización del tiempo de ejecución de descodificadores de vídeo compatibles con diferentes estándares (MPEG-2, MPEG-4, H.264 y H.264/SVC) utilizando tecnología DSP.

Las metodologías propuestas en la tesis mencionada son válidas para arquitecturas con un único DSP. En la actualidad existen dos factores que impiden implementar descodificadores de vídeo funcionando en tiempo real con este tipo de arquitecturas. Por una parte los nuevos estándares de codificación como *High Efficiency Video Coding* (HEVC) [ISO15] [SOHW12] demandan una capacidad de cálculo más elevada que sus antecesores y, por otra, las resoluciones espaciales y temporales de las imágenes cada vez son mayores, lo que implica también un coste computacional mayor.

Para abordar la implementación de estos nuevos estándares y soportar elevadas resoluciones espaciales, es necesario emplear más de un procesador; de esta forma, la

carga computacional del proceso de decodificación puede repartirse entre varios procesadores. En este sentido, a comienzos de 2012 algunos de los fabricantes más representativos han comercializado dispositivos que integran varios DSPs (en adelante, procesadores Multi-DSP). Estos dispositivos poseen una elevada capacidad de cómputo para un consumo reducido lo que les hace, además, especialmente adecuados para aplicaciones que deben trabajar autónomamente empleando baterías.

El particionado de la carga computacional de un algoritmo que se ejecuta en un sistema multinúcleo es un tema que ha sido ampliamente abordado en sistemas complejos que integran Sistemas Operativos (SOs) como pueden ser Linux o Windows. En la actualidad ya existen grupos de investigación trabajando en metodologías para optimizar de forma automática la paralelización de algoritmos entre varios procesadores [YSJ⁺14] [PSB14], si bien hasta la fecha estos trabajos se han focalizado en procesadores de propósito general.

Sin embargo, los Sistemas Operativos en Tiempo Real (RTOS) [SYS] que emplean los procesadores multi-DSP no incorporan las facilidades proporcionadas por los SOs de altas prestaciones, por lo que es tarea del diseñador el realizar la división de forma manual. Esto implica un trabajo laborioso y poco flexible puesto que los cambios en el reparto de la carga computacional entre los procesadores requieren modificaciones significativas y un elevado coste en términos de tiempo de desarrollo.

En los últimos años, el GDEM ha orientado su investigación en la línea de arquitecturas para la codificación de vídeo hacia las metodologías para la implementación de decodificadores de vídeo de última generación con procesadores multinúcleo. En este contexto, el GDEM ha participado en los proyectos H2B2VS [H2B], MR-UHDTV [MRU] e IVME [IVME].

1.2. Objetivos de la investigación

Para flexibilizar el particionado de tareas entre los diferentes procesadores se ha investigado en metodologías que automaticen el proceso de reparto y resuelvan los problemas de sincronización y movimiento de datos entre los núcleos disponibles.

El trabajo realizado en esta tesis se centra en aprovechar las metodologías desarrolladas en otros grupos de investigación [BEP⁺08] [PRP⁺08] para procesadores de propósito general, así como la experiencia acumulada en el GDEM en la optimización de algoritmos de decodificación de vídeo para DSPs con el objetivo de proponer una metodología que reduzca el tiempo de desarrollo de algoritmos de codificación de vídeo con elevados requisitos de cálculo en plataformas Multi-DSP. Además, la metodología propuesta debe ser automatizada y validada de forma práctica utilizando tecnologías *hardware* y algoritmos de codificación que estén en el estado del arte.

De esta forma quedan definidos los tres objetivos principales de esta tesis:

- 1) La formalización de una metodología de trabajo que facilite el diseño de descodificadores de vídeo sobre plataformas multinúcleo basadas en tecnología DSP.
- 2) La creación de una herramienta que permita automatizar las tareas de diseño descritas en la metodología, con el objeto de poder generar diferentes versiones de descodificadores que corran sobre diferentes plataformas (por ejemplo, con un número mayor o menor de núcleos) en un tiempo razonable.
- 3) La validación de la metodología propuesta y de la herramienta generada realizando diferentes versiones de un descodificador² de vídeo conforme al estándar HEVC, utilizando tecnología DSP (Multi-DSP). Esta implementación debe superar con ventaja las realizaciones con un solo DSP, permitiendo incrementar la resolución espacial y/o temporal de las imágenes empleando varios núcleos de procesamiento.

Asociados a los objetivos anteriores hay una serie de objetivos secundarios que han surgido durante la realización del trabajo de investigación:

- 1) La integración de algoritmos de asignación dinámica de carga computacional y de tecnologías multinúcleo basadas en GPPs en la herramienta anteriormente mencionada.
- 2) La realización de un estudio sobre el consumo energético en un descodificador HEVC basado en tecnología Multi-DSP, teniendo en cuenta la relación entre el número de núcleos utilizados, el incremento del rendimiento y el consecuente aumento de energía consumida.

1.3. Metodología de trabajo

La metodología de trabajo que se ha empleado contempla cuatro fases:

- 1) Estudio del estado del arte en la implementación de algoritmos de procesamiento en plataformas multiprocesador. Se ha realizado un estudio detallado de otras contribuciones que se han publicado en la literatura científica sobre aplicaciones y herramientas que facilitan la tarea de distribución de la carga computacional entre los núcleos de procesamiento. Este estudio no se ha limitado a aplicaciones de procesamiento de vídeo, ni a sistemas basados en procesadores Multi-DSP, si no que se ha generalizado a otro tipo de aplicaciones y empleando otras plataformas para tener una visión más general respecto a la implementación de algoritmos basados en sistemas multiprocesador.
- 2) Implementación de un descodificador de vídeo sobre dos plataformas Multi-DSP. Con objeto de realizar los experimentos que han permitido poner de manifiesto los diferentes problemas, ensayar soluciones y, en definitiva, extraer las conclusiones

² La elección de un descodificador, en lugar de un codificador, para validar la metodología propuesta responde a consideraciones prácticas relacionadas tanto con el tiempo disponible como con la trayectoria investigadora previa del GDEM.

que han alimentado el desarrollo de la metodología de trabajo propuesta, se ha implementado un descodificador compatible con el estándar HEVC en una plataforma Multi-DSP. Para ello, se ha seleccionado una plataforma de desarrollo adecuada entre las existentes en el mercado ([TMS0] [TMS1], entre otras). A partir de una implementación de referencia del descodificador se ha realizado el proceso de portado a la tecnología DSP y la división de la carga computacional entre los núcleos de procesamiento, entre otras tareas necesarias.

- 3) Síntesis de la metodología. A partir de la experiencia obtenida durante la implementación del descodificador HEVC, se han extraído conclusiones metodológicas que han permitido definir una metodología de trabajo que permite reducir los tiempos de desarrollo de estas aplicaciones, y que ayuda en la flexibilización de la implementación en sistemas Multi-DSP con un número variable de procesadores.
- 4) Creación de una herramienta de diseño. Los diferentes pasos de la metodología de diseño propuesta se han integrado en una herramienta que permite su aplicación de forma automática. La herramienta ha sido verificada con el diseño de varios descodificadores HEVC sobre varias plataformas basadas en DSPs, pero también en GPPs.

Estas cuatro fases no se han desarrollado de manera completamente secuencial. El esfuerzo principal de la primera fase se realizó al comienzo de la tesis, pero ha sido necesario seguir actualizando la información recopilada con las nuevas contribuciones que se han seguido publicando durante todo el desarrollo de la misma. La segunda y la tercera fase han discurrido en paralelo y se han realimentado mutuamente durante gran parte del desarrollo de la tesis. La cuarta fase se ha llevado a cabo en la parte final de la tesis, una vez quedaron fijados los diferentes pasos de la metodología de diseño.

1.4. Estructura de la memoria

La presente memoria se ha estructurado en 5 partes, compuestas por 13 capítulos, y 6 anexos.

La primera parte se compone del capítulo primero exclusivamente; aquí se realiza la introducción y se establece el marco del trabajo de investigación desarrollado dentro de las tareas que viene realizando el GDEM de la UPM. En esta primera parte también se presentan los objetivos de la tesis doctoral.

En la segunda parte de la memoria se reúne la información relevante sobre el estado del arte en técnicas de codificación de vídeo, tecnologías de implementación y metodologías de diseño. Esta parte contiene tres capítulos:

En el capítulo 2 se introducen las últimas técnicas en codificación de vídeo digital, y se presentan los principios de los estándares HEVC y *Reconfigurable Video Coding* (RVC) [MRJ13], claves en el desarrollo de la tesis.

El capítulo 3 versa sobre las principales tecnologías utilizadas para la implementación de codificadores y descodificadores de vídeo. En este sentido se analizan las principales características de las arquitecturas de aquellas plataformas utilizadas durante la fase de desarrollo de este trabajo. También se analiza el estado del arte de las implementaciones llevadas a cabo, y publicadas en la literatura científica, por parte de otros grupos de investigación.

El capítulo 4 cierra la segunda parte de la memoria y aborda el estado del arte en las metodologías para el diseño de *software* sobre plataformas multinúcleo. En este capítulo se presenta la metodología basada en flujos de datos que ha servido como punto de partida para la consecución de los objetivos planteados en este trabajo.

En la tercera parte de la memoria se presentan las distintas tecnologías que se han utilizado durante el desarrollo de la tesis doctoral, en cuatro capítulos:

El capítulo 5 presenta el *software* del descodificador de vídeo OpenHEVC [HRD14b]. Este *software* está escrito en lenguaje de programación C y ha sido principalmente desarrollado por investigadores del INSA de Rennes [INSA].

En el capítulo 6 se presenta la Interfaz de Programación de Aplicaciones (API) OpenMP. OpenMP es un estándar *de facto* que facilita la programación paralela sobre plataformas con memoria compartida.

A lo largo del capítulo 7 se presentan las características de las plataformas utilizadas para la realización de pruebas y toma de resultados durante la fase de desarrollo.

El capítulo 8 detalla las particularidades del entorno de trabajo del compilador Orcc [ORC]. Ésta es una herramienta fundamental en el desarrollo de la tesis doctoral, ya que sobre parte de ella se implementarán las modificaciones necesarias que den soporte a la nueva metodología propuesta.

En la cuarta parte de la memoria se presenta la metodología propuesta para el diseño de descodificadores de vídeo sobre plataformas Multi-DSP. También se presentan las distintas implementaciones llevadas a cabo para validar la nueva metodología. Esta parte contiene dos capítulos:

En el capítulo 9 se detallan los distintos pasos seguidos para desarrollar la metodología propuesta. Aquí se presentan, pormenorizadamente, todos los aspectos que deben tenerse en cuenta a la hora de abordar el diseño de descodificadores de vídeo sobre plataformas Multi-DSP. También se incluyen las mejoras incluidas en la fase de diseño relativas a las técnicas de reducción de consumo energético y optimización del rendimiento mediante el *software* de OpenHEVC.

El capítulo 10 presenta el banco de pruebas utilizado para validar las distintas técnicas presentadas en el capítulo 9. Además, se detallan las particularidades de las implementaciones desarrolladas y se presentan los resultados obtenidos.

Para finalizar la memoria, en la quinta parte se presentan los resultados y las aportaciones de la tesis doctoral, en tres capítulos:

En el capítulo 11 se hace un repaso de los objetivos de la tesis y se resumen los resultados conseguidos. En el capítulo 12 se detallan las publicaciones científicas llevadas a cabo durante la tesis así como otros méritos relacionados con el desarrollo de la misma. Por último, en el capítulo 13, se presentan las distintas propuestas de trabajos futuros a desarrollar a partir del trabajo realizado en la presente tesis doctoral.

La memoria concluye con el listado de referencias bibliográficas, los acrónimos utilizados y los anexos:

En el Anexo I se facilita la configuración establecida en el fichero *config.cfg* para la plataforma Multi-DSP. El Anexo II incluye los diagramas completos de las tres versiones de decodificadores HEVC-RVC utilizados. En el Anexo III se realiza un análisis de la utilización de módulos optimizados sobre implementaciones HEVC-RVC. El Anexo IV detalla la implementación de un decodificador HEVC en base al código de referencia del estándar (HM) sobre una plataforma DSP. El Anexo V contiene el código relativo a la gestión de interrupciones con el procesador TMS320C6678. Por último, el Anexo VI es un resumen en lengua inglesa de esta memoria.

SEGUNDA PARTE: Antecedentes

En esta parte de la memoria se presentan los antecedentes que dan lugar al trabajo desarrollado. Aquí se agrupan los conceptos tecnológicos que, en su mayoría, dan soporte a la parte teórica de la tesis doctoral.

Así, el capítulo 2 presenta los elementos principales de la codificación de vídeo digital y los últimos estándares de codificación desarrollados. En el capítulo 3 se realiza un estudio sobre el estado del arte en la implementación de descodificadores HEVC. Finalmente en el capítulo 4 se lleva a cabo un estudio sobre las distintas metodologías en el diseño de aplicaciones sobre plataformas multinúcleo.

2. Codificación de Vídeo Digital

En este capítulo se describen los conceptos más relevantes de la codificación de vídeo digital, introduciendo en primer lugar tanto los fundamentos como las técnicas que permiten la compresión de la información. Seguidamente se hace un breve repaso de los principales estándares de codificación, desde los años 80 del siglo pasado hasta la actualidad. Finalmente, se introducen los estándares HEVC y RVC que han sido utilizados en la tesis.

Así, el capítulo queda estructurado de la siguiente manera: en el apartado 2.1 se introducen los conceptos básicos y comunes a todos los estándares de codificación. También se realiza un repaso en orden cronológico de los principales estándares, destacando aquellos hitos de mayor relevancia. Seguidamente, en el apartado 2.2 se introduce con mayor detalle el estándar de codificación HEVC utilizado durante la realización de esta tesis doctoral. Finalmente, en el apartado 2.3 se presenta el estándar RVC, clave en el desarrollo de nuevos descodificadores de vídeo y que ha sido una pieza central en la investigación realizada.

2.1. Introducción a la codificación de vídeo digital

En este apartado se describen brevemente los fundamentos de la codificación de vídeo digital empleados en el desarrollo de los estándares de codificación de vídeo digital durante las últimas décadas. Gran parte de la nomenclatura utilizada en los posteriores capítulos de esta memoria se introduce en este capítulo.

2.1.1. Origen de la codificación de vídeo digital

Desde que a finales del siglo XIX comenzaran las primeras reproducciones de vídeo, hasta nuestros días, las tecnologías multimedia han experimentado grandes avances animadas por el éxito obtenido entre los consumidores de contenidos y una industria próspera y en constante expansión.

El rápido desarrollo experimentado, principalmente en los últimos años, ha permitido la diversificación de la industria multimedia, que hoy en día ofrece servicios como la televisión digital inteligente de alta definición o la reproducción bajo demanda de contenidos en terminales móviles o sistemas de videoconferencia 3D, entre otros. Buena parte del éxito y de la rápida expansión de estas tecnologías durante el último cuarto de siglo se han conseguido gracias a la digitalización de las señales de televisión. La era digital facilitó el almacenamiento y la difusión de contenidos, su edición y montaje, y abrió la puerta a una trepidante fase de desarrollo de nuevas tecnologías multimedia.

El primer formato para la digitalización y codificación de vídeo digital surgió en el año 1984 de la mano de la Unión Internacional de Telecomunicaciones (ITU) [ITU0]. En este primer estándar se define una frecuencia de muestreo fija tanto para sistemas PAL³ como NTSC⁴, al mismo tiempo que en la señal transmitida se incluyen impulsos para la sincronización de la imagen. Bajo este primer estándar el área útil del vídeo es de 625 líneas y 25 imágenes por segundo en el sistema PAL y de 525 líneas y 30 imágenes por segundo en NTSC. Además, se decide establecer una representación del vídeo basada en la utilización de tres componentes por cada *píxel*: una para la luminancia (Y) y dos para las crominancias o diferencias de color (C_R y C_B). Existen otras formas de representación de la señal de vídeo, una de las más destacables es el formato RGB, en el que se recogen tres componentes de color por píxel: rojo, verde y azul (en inglés *Red, Green, Blue*).

Tanto el régimen binario necesario para la transmisión de la señal de vídeo como la cantidad de memoria necesaria para su almacenamiento exceden las capacidades de los sistemas existentes. Por ejemplo, una secuencia de vídeo de alta definición requeriría un régimen binario de 1.24 Gbps⁵. Para almacenar únicamente 5 minutos de vídeo sería

³ *Phase Alternating Line* (línea de fase alternada), este sistema de codificación analógica era el utilizado en la mayoría de países de Europa, Asia, África y Oceanía.

⁴ *National Television System Committee* (Comité Nacional de Sistema de Televisión), sistema de codificación mayoritariamente utilizado en América y en países como Japón o Filipinas.

⁵ Este régimen binario se calcula considerando una resolución de 1920x1080 píxeles, y según la fórmula: (1920x1080) píxeles/imagen * 25 imágenes/segundo * 8 bits/píxel * 3 componentes = 1.24 Gbps.

necesario un dispositivo capaz de albergar 373,2 Gbits, el equivalente a un disco Blu-Ray⁶.

Afortunadamente, la señal de vídeo posee información redundante, y esta redundancia se puede reducir utilizando algoritmos de compresión. Normalmente, estos algoritmos permiten reducir el régimen binario de la señal de vídeo en dos órdenes de magnitud, aunque en el proceso se introducen pérdidas. Con el paso de los años se han estandarizado numerosos algoritmos de codificación, los cuales se valen de distintas técnicas de muestreo de imagen y compresión de información. Los avances en las tecnologías de compresión de vídeo han posibilitado el almacenamiento y transmisión de vídeo con resoluciones espaciales y temporales cada vez más elevadas, con regímenes binarios más reducidos y con una mayor calidad.

2.1.2. Fundamentos de la codificación de vídeo

A la hora de realizar la digitalización de una secuencia de vídeo se pueden utilizar distintos esquemas de muestreo en función de la calidad deseada, de acuerdo con las especificaciones dadas para la posterior difusión del contenido multimedia.

El muestreo de mayor calidad es aquel que en cada muestra registra, por cada píxel de la imagen, una componente de luminancia y dos de crominancias, o muestreo 4:4:4 (Figura 1-a). Sin embargo, existen otros esquemas de muestreo en los que se reduce la proporción de componentes de crominancia aprovechando que el sistema de visión humana es más sensible a los cambios en el brillo que a los matices de color. De esta forma se puede aplicar un muestreo 4:2:2 (Figura 1-b) en el que por cada dos componentes de luminancia se obtienen únicamente dos componentes de crominancia, una C_R y otra C_B . Esta proporción se ve todavía más reducida en los esquemas 4:1:1 (Figura 1-c) y 4:2:0 (Figura 1-d), en los que se registran dos muestras de crominancia C_R C_B por cada cuatro muestras de luminancia. La diferencia entre estos dos últimos esquemas radica en que en el primero de ellos el submuestreo se realiza únicamente en el eje horizontal, mientras que en el segundo se realiza tanto en el horizontal como en el vertical.

⁶ Considerando que la capacidad media de un disco Blu-Ray es de 50 Gbytes.

a) Muestreo 4:4:4 b) Muestreo 4:2:2 c) Muestreo 4:1:1 d) Muestreo 4:2:0

Figura 1. Esquemas de muestreo.

Ya en los primeros estándares MPEG⁷-2 ISO 13818 [ISO94] las imágenes que se codifican se dividen en áreas de píxeles, denominadas *macrobloques*. Estos macrobloques se componen de 16x16 muestras de luminancia y un número variable de muestras de crominancia, que depende del esquema de muestreo utilizado. En el estándar H.264/AVC se modificó la estructura del macrobloque, quedando éste dividido en bloques de 8x8 píxeles. Posteriormente, en el estándar HEVC (ver apartado 2.2), el tamaño del macrobloque se aumentó hasta las 64x64 muestras. La Figura 2 muestra la estructura de un macrobloque para distintos esquemas de muestreo.

Figura 2. Estructura de un macrobloque según distintos esquemas de muestreo.

2.1.3. El esquema de codificación híbrido

Para realizar la compresión de una secuencia de vídeo se aprovechan diferentes propiedades que en mayor o menor medida permiten obtener una buena ratio de compresión con unas pérdidas de calidad aceptables. La mayoría de las bases para la compresión del vídeo se establecieron en el estándar MPEG-2, dichas bases han servido como punto de partida al resto de estándares desarrollados.

Realizando un análisis estadístico se puede observar que se dan correlaciones tanto entre los píxeles próximos entre sí dentro de cada imagen como entre grupos de

⁷ Moving Pictures Experts Group [MPEG].

píxeles pertenecientes a diferentes imágenes. Las primeras son consecuencia de la llamada redundancia espacial de las imágenes, mientras que las segundas son consecuencia de la redundancia temporal entre imágenes.

La redundancia espacial puede reducirse mediante la implementación de un proceso de transformación al dominio de la frecuencia, donde se realiza una compactación de la información en el rango de las bajas frecuencias espaciales; posteriormente se aplican procesos de cuantificación y codificación de entropía. Todo ello permite reducir la cantidad de información de la secuencia a cambio de una reducción en su calidad que depende, básicamente, del grado de cuantificación empleado.

La reducción de la redundancia temporal se consigue mediante la aplicación de técnicas de predicción basadas en la estimación de movimiento. Para cada macrobloque de la imagen que se va a codificar se busca una predicción dentro de imágenes previamente codificadas. Los algoritmos de estimación de movimiento realizan esta búsqueda comparando el macrobloque que se va a codificar con diferentes zonas del mismo tamaño dentro de las imágenes ya codificadas utilizando un criterio de parecido, normalmente la Suma Absoluta de Diferencias. La reducción de la redundancia temporal se obtiene al codificar la diferencia entre cada macrobloque y su predicción, en lugar de codificar el macrobloque.

Adicionalmente se deben tener en cuenta las particularidades del sistema de visión humano, para reducir aquellas partes de información que dicho sistema no percibe o que percibe con poca precisión, como son las componenets espaciales de alta frecuencia. Este hecho se conoce como redundancia de percepción.

La mayoría de los codificadores de vídeo estandarizados durante los últimos 30 años utilizan un esquema de codificación híbrido que permite reducir la redundancia espacial y la redundancia temporal de manera simultánea. La Figura 3 representa el diagrama de bloques básico de un codificador híbrido de MPEG-2. Como se observa en el diagrama, en la parte superior están los bloques o módulos más relacionados con la reducción de la redundancia espacial, y en la parte inferior aquellos más relacionados con la reducción de la redundancia temporal.

Figura 3. Diagrama de bloques de un codificador híbrido.

El codificador híbrido realiza la codificación macrobloque a macrobloque y, dentro de cada macrobloque, bloque a bloque.

En primer lugar, para cada bloque de la imagen, se realiza una transformación al dominio de las frecuencias espaciales. El resultado de esta transformación es un conjunto de coeficientes que representan las componentes de frecuencia de cada bloque. Seguidamente, estos coeficientes son cuantificados con el objetivo de reducir el rango de valores de las muestras, permitiendo así su codificación con un número de bits menor. Dependiendo del nivel de cuantificación utilizado, habitualmente se elimina una parte de las componentes asociadas a las altas frecuencias espaciales, lo que permite una reducción de la cantidad de información a costa de una pérdida moderada de calidad. Finalmente se utiliza una codificación de entropía mediante la que se asignan códigos de menor longitud a las combinaciones más probables, a este procesado se le conoce como *Context-Adaptive Variable-Length Coding* (CAVLC). Dentro de todo este proceso se producen pérdidas, fundamentalmente debidas al proceso de cuantificación; estas pérdidas no podrán ser compensadas en el proceso de descodificación.

Por otro lado, las imágenes así codificadas son sometidas a un proceso de descodificación análogo al que ocurrirá en el descodificador. El resultado de este proceso son las llamadas *imágenes reconstruidas*, que son las mismas que se han codificado, pero incluyendo las pérdidas introducidas durante el proceso de codificación. Estas imágenes son almacenadas en una memoria o *buffer* de imágenes reconstruidas. Para reducir la redundancia temporal, el codificador realiza una predicción del bloque de la imagen que debe codificar en cada momento, basada en las imágenes previamente codificadas y almacenadas en la memoria de reconstrucción. Para ello, buscará entre las imágenes ya codificadas un bloque que sirva de predicción;

una vez obtenido se codificará únicamente la diferencia entre el bloque que se va a codificar y su predicción.

El proceso de búsqueda de las predicciones se realiza mediante el módulo de estimación de movimiento. Aquellos macrobloques necesarios para el procesamiento de las predicciones pasan a la estimación de movimiento junto con las imágenes almacenadas en el *buffer*. Posteriormente, en la fase de decodificación, las imágenes pueden recuperarse sin pérdidas mediante un módulo de compensación de movimiento.

Para cada macrobloque, el codificador puede elegir entre realizar una codificación con o sin predicción. Los macrobloques que se codifican sin predicción son macrobloques intracodificados, o simplemente macrobloques *intra*, y son identificados en la trama de bits como macrobloques tipo I. Los macrobloques que se codifican con predicción son macrobloques intercodificados o macrobloques *inter*. Aquellos macrobloques *inter* para los que la predicción ha sido calculada en base a una sola imagen de entre las previamente codificadas se identifican como macrobloques tipo P (con Predicción). Si la predicción ha sido calculada en base a la información de dos imágenes previamente codificadas, entonces se identifican como macrobloques tipo B (con predicción Bidireccional).

A partir de los distintos tipos de macrobloques que integran una imagen, éstas quedan definidas como imágenes de tipo I, B o P:

- Las imágenes tipo I se componen únicamente de macrobloques tipo I. Dado que se codifican sin predicción, la compresión que puede obtenerse es pequeña. Son, por tanto, las que más tamaño ocupan y se utilizan cada cierto tiempo para evitar que los errores derivados de las pérdidas introducidas durante el proceso de codificación se vayan acumulando, con la consecuente pérdida de calidad. Adicionalmente, facilitan el acceso aleatorio a la trama de bits codificada, la recuperación frente a errores en la transmisión y la realización de búsquedas rápidas hacia adelante o hacia atrás (*fast forward* o *rewind*).
- Las imágenes tipo P deben contener al menos un macrobloque tipo P, el resto deben ser tipo I. Permiten una compresión moderada de la información a cambio de un coste de procesamiento también moderado.
- Finalmente, las imágenes tipo B deben contener al menos un macrobloque tipo B. El resto, en su caso, pueden ser de tipo P o I. Con este tipo de imágenes se consiguen las mejores ratios de compresión con un mayor coste de procesamiento.

La Figura 4 muestra una secuencia típica del ordenamiento de imágenes con los tres tipos descritos, mediante las flechas se indican las dependencias entre las imágenes. En esta secuencia la cadencia entre los diferentes tipos de imágenes es periódica, cada imagen tipo P se predice en base a la imagen P o I inmediatamente anterior, y cada imagen tipo B se predice en base a la imagen I o P inmediatamente anterior y a la

imagen I o P inmediatamente posterior. Para que un descodificador sea capaz de descodificar tal secuencia, es necesario que el codificador codifique las imágenes en un orden diferente al de presentación (visualización). En la Tabla 1 se muestra la diferencia entre el orden de presentación y el orden de codificación de las imágenes de la secuencia de la figura. Por ejemplo, para descodificar las imágenes 2 y 3, el descodificador deberá haber descodificado previamente las imágenes 1 y 4.

Figura 4. Representación de una secuencia con imágenes I, P y B.

	1	2	3	4	5	6	7	8	9	10
Orden de presentación	I	B	B	P	B	B	P	B	B	I
Orden de codificación	1	4	2	3	7	5	6	10	8	9
	I	P	B	B	P	B	B	I	B	B

Tabla 1. Diferencia entre los órdenes de presentación y codificación de las imágenes.

La secuencia de la Tabla 1 sigue un orden típico de codificadores de baja complejidad como MPEG-2 [ISO94]. Los codificadores más recientes como MPEG-4 Parte 10 [ISO03] o HEVC (ISO 23008-2) [ISO15] [SOHW12] generan típicamente secuencias más complejas, en las cuales, por ejemplo, la predicción de un macrobloque puede involucrar información contenida en varias imágenes codificadas además en cualquier instante previo.

2.1.4. Estándares de codificación de vídeo

Desde el desarrollo del primer estándar de codificación a principios de los años ochenta, y hasta el día de hoy, dos organismos internacionales han liderado los procesos de desarrollo y estandarización de nuevos sistemas de codificación de vídeo. Por una parte la ITU (*International Telecommunication Union*) desarrolló los estándares H.261 y H.263 entre 1984 y 1998. Mientras que la ISO (*International Organization for Standardization*) desarrolló los estándares MPEG-1, MPEG-2 y MPEG-4 en los años 1990, 1995 y 1999 respectivamente.

Sin embargo, hacia el año 2003, ambas organizaciones, la ITU y la ISO aunaron esfuerzos y desarrollaron conjuntamente el estándar H.264 en nomenclatura ITU o MPEG-4 parte 10 en nomenclatura ISO. Este es uno de los estándares más extendidos y

es ampliamente utilizado en sistemas de difusión de televisión digital o de vídeo en *streaming*, entre muchos otros.

Finalmente hacia el año 2010 se comenzó a trabajar, por parte de ambas organizaciones, en un nuevo estándar denominado H.265 (según la ITU) o HEVC (*High Efficiency Video Coding* – según la ISO). El proceso de estandarización de HEVC culminó en 2012; desde entonces HEVC se considera el sucesor natural de H.264.

La Figura 5 resume gráficamente el desarrollo de los distintos estándares de codificación durante los últimos 30 años por parte de la ITU y la ISO.

Figura 5. Representación cronológica del desarrollo de los principales estándares de codificación de vídeo.

El estándar HEVC se ha utilizado como referencia durante la realización de esta tesis doctoral, por esta razón, en el siguiente apartado se analizan sus particularidades, con especial énfasis en aquellos detalles más relevantes para este trabajo. La información relativa a los estándares de MPEG puede consultarse en [MPEGS], y en el sitio web de la ITU-T se encuentran disponibles los documentos de las distintas normas que afectan a sus estándares [ITU].

2.2. High Efficiency Video Coding

En este apartado se presenta un breve resumen del estándar de codificación de vídeo HEVC. Para ampliar esta información, en la literatura científica pueden consultarse numerosas referencias entre las que cabe destacar el documento de la ITU-T relativo al estándar [ISO15], el resumen de sus principales características publicado en [SOHW12] o el análisis de una primera implementación sobre dos plataformas basadas en GPPs [BBSF12].

2.2.1. Introducción al estándar HEVC

El estándar para la codificación de vídeo digital HEVC ha sido desarrollado conjuntamente por las organizaciones ISO/IEC [ISO] y MPEG [MPEG] mediante el consorcio *Joint Collaborative Team on Video Coding* (JCT-VC) [JCT]. Dicho consorcio surge a finales de los años 90 con el ánimo de desarrollar nuevos estándares de codificación de vídeo. Frutos de esta colaboración fueron los estándares H.262/MPEG-2 y H.264/MPEG-4 parte 10, detallados en el apartado 2.1.4. La primera versión oficial del estándar HEVC fue estandarizada en 2012 bajo la norma ISO/IEC 23008-2. HEVC

surge con el doble objetivo de mejorar la capacidad de descodificar secuencias de vídeo con mayores resoluciones espaciales, incluyendo ultra alta definición (*Ultra High Definition* – UHD⁸), al mismo tiempo que se consigue reducir el régimen binario de una secuencia de vídeo codificada con su predecesor H.264/AVC perfil alto, sin reducir con ello la calidad del vídeo. Todo esto se consigue incrementando considerablemente la complejidad de los códecs, aumentando con ello también el coste computacional de sus implementaciones.

Debido al incremento en la complejidad de los algoritmos que integran las implementaciones HEVC se hace necesaria la inclusión de herramientas que faciliten su paralelización. En este sentido, se incluyen en el estándar técnicas que permiten paralelizar el procesamiento de las imágenes a nivel de *slices*, de imágenes y por filas de CTUs (llamado *Wavefront Parallel Processing* – WPP).

2.2.1.1. Particionado de imágenes

El estándar HEVC no presenta una estructura global disruptiva con respecto a su predecesor, sino que se apoya en numerosas pequeñas mejoras efectuadas sobre los distintos bloques funcionales del codificador. Estas pequeñas mejoras consiguen una mejora global de hasta el 50% en el régimen binario necesario para transmitir una secuencia de vídeo codificado con la misma calidad respecto a su predecesor H.264.

Sin embargo, HEVC sí introduce algunos elementos novedosos con respecto a sus predecesores. Algunas de estas novedades se centran en el modelo de elementos que se utilizan a la hora de realizar el particionado de las imágenes. Antes de la llegada de HEVC el elemento utilizado para realizar el particionado durante el proceso de codificación era el macrobloque. Cada macrobloque se compone, típicamente utilizando un muestreo 4:2:0, de un total de 16x16 muestras de luminancia y 8x8 muestras de crominancia. De distinta forma, el estándar HEVC se apoya sobre un modelo más flexible basado en una serie de estructuras más complejas:

- Unidades en estructura de árbol (*Coding Tree Units* – CTU) y Bloques en estructura de árbol (*Coding Tree Blocks* – CTB). Al contrario que sus predecesores, HEVC utiliza como elemento principal de particionado el CTU, cuyo tamaño se especifica en la codificación y puede ser mayor que el del macrobloque. Cada CTU se compone un número de CTBs correspondientes a luminancias y crominancias. Además, el tamaño del CTB es variable, pudiendo ser de 16x16, 32x32 ó 64x64 muestras. HEVC también contempla un particionado menor de los CTBs mediante el modelo *quad-tree* [Sam84]. La Figura 6 representa gráficamente esta subdivisión. En la Figura 6 (a) se observa el particionado realizado sobre un CTB, y en la Figura 6 (b) la correspondiente estructura según el modelo *quad-tree*.
- Unidades de codificación (*Coding Units* – CUs) y bloques de codificación (*Coding Blocks* CBs). Cada CTB contiene una CU, aunque ésta puede a su vez dividirse para

⁸ UHD – 4K tiene una resolución de 3840x2160 *píxeles* por imagen.

formar múltiples CUs. A su vez, una CU está formada generalmente por un CB de luminancia y dos CBs de crominancia. La relación entre una CTU y el tamaño y posición de las CBs viene dado por el índice del modelo quad-tree (ver Figura 6). Finalmente cada CU tiene asociado un sub-particionado en otros dos elementos menores: Unidades de Predicción (Prediction Units – PUs) y Unidades de Transformación (*Transform Units* – TUs).

Las decisiones a la hora de realizar predicciones tanto inter como intra se toman a nivel de CU. De esta forma, el particionado en estructuras PU viene también dado a partir del nivel de particionado de CUs. En función de las decisiones aplicadas en las predicciones, las componentes de luminancia y crominancia de los CBs pueden separarse a su vez en Bloques de Predicción (*Prediction Blocks* – PBs). Bajo el estándar HEVC el tamaño de los PBs es variable y puede tomar valores desde las 4x4 muestras hasta 64x64 muestras.

Tanto los TBs, como las TUs se integran en la estructura de árbol mostrada en la Figura 6 y tienen como referencia las CUs. Un CB de luminancia residual puede ser igual a su correspondiente TB de luminancia o bien puede dividirse en TBs de luminancia de menor tamaño. El mismo procedimiento puede aplicarse con los CBs de crominancia. Los TBs pueden tener un tamaño de 4x4, 8x8, 16x16 ó 32x32.

Figura 6. Subdivisión de un CTB en CBs y TBs. Las líneas sólidas representan subdivisiones en CBs y las punteadas en TBs.

Esencialmente, los nuevos elementos introducidos en HEVC permiten la codificación de regiones homogéneas de mayor tamaño que el alcanzable con sus estándares predecesores. La Figura 7 muestra un ejemplo de la división en bloques realizada en una imagen por un codificador H.264 (imagen izquierda) y otro HEVC. Esta característica permite una reducción importante del *bitrate* necesario para codificar el vídeo [OSS⁺12]. Así, el conjunto de mejoras introducidas en HEVC permiten reducir hasta en un 50% el *bitrate* necesario para codificar la misma secuencia (misma resolución y factor QP) respecto del necesario con H.264.

Figura 7. Ejemplo de la división de bloques realizada en una imagen por un codificador H.264 (izquierda) y por un codificador HEVC (derecha).

2.2.1.2. Sintaxis de alto nivel

Además de las modificaciones introducidas en el modelo utilizado para realizar el particionado de las imágenes, también se han llevado a cabo una serie de cambios en la sintaxis de alto nivel mediante la que se gestiona la transmisión del vídeo codificado. Los cambios introducidos tienen como objetivo mejorar la adaptación del estándar a distintos entornos de transmisión de datos para ampliar su aplicabilidad.

Los datos listos para transmitirse a la salida del codificador se organizan jerárquicamente formando una trama cuyo protocolo está bien definido en el estándar. El codificador incluye en la trama la información necesaria para la sincronización y la descodificación de las imágenes codificadas encapsuladas en la trama. Cada paquete de la trama viene precedido por una cabecera en la que se incluye la información necesaria para una correcta interpretación del subsiguiente paquete de datos. En el lado del descodificador se procesa en primer lugar cada cabecera antes de comenzar la descodificación de cada imagen. La Figura 8 representa gráficamente una trama a la salida del codificador; como se puede ver, en primer lugar se transmiten varios paquetes relativos a la información necesaria para proceder a la descodificación, seguidamente se envían secuencialmente las imágenes codificadas. A continuación se describen las cabeceras y tipos de paquetes más relevantes que se pueden encontrar en una trama codificada con el estándar HEVC.

Figura 8. Representación gráfica de los niveles jerárquicos de una trama de vídeo codificado bajo el estándar HEVC.

- **Cabecera NAL** (*Network Abstraction Layer* – Capa de abstracción de red): todo paquete transmitido viene precedido de la cabecera NAL de dos *bytes*, mediante la que se informa del propósito del paquete de datos transmitido a continuación. Se han definido distintos tipos de cabeceras NAL en función de la información encapsulada, la Tabla 2 lista todos ellos. Como se puede observar, existen numerosos tipos de cabeceras que permiten una rápida identificación de la información que se transmite por parte del codificador. El detalle de cada uno de los tipos de cabecera NAL se puede consultar en [ITU15].

Tipo	Significado	Clase	Tipo	Significado	Clase
0, 1	<i>Slice</i> ordinario de imagen	VCL	32	Paquete VPS	No VCL
2, 3	<i>Slice</i> de imagen TSA	VCL	33	Paquete SPS	No VCL
4, 5	<i>Slice</i> de imagen STSA	VCL	34	Paquete PPS	No VCL
6, 7	<i>Slice</i> de imagen RADL	VCL	35	Delimitador de acceso a unidad	No VCL
8, 9	<i>Slice</i> de imagen RASL	VCL	36	Fin de secuencia	No VCL
10~15	Reservado para futuras ampliaciones	VCL	37	Fin del <i>bitstream</i>	No VCL
16~18	<i>Slice</i> de imagen BLA	VCL	38	Datos de relleno	No VCL
19, 20	<i>Slice</i> de imagen IDR	VCL	39, 40	Mensaje SEI	No VCL
21	<i>Slice</i> de imagen CRA	VCL	41~47	Reservado para futuras ampliaciones	No VCL
22~31	Reservado para futuras ampliaciones	VCL	48~63	No especificado (disponible para uso del sistema)	No VCL

Tabla 2. Listado de los distintos tipos de cabeceras NAL en el estándar HEVC.

- El primer paquete que se transmite contiene información común a toda la trama de vídeo codificado, este paquete es una extensión del mismo utilizado en el estándar H.264 y se denomina VPS (*Video Parameter Set* – conjunto de parámetros del vídeo).

- Tras el VPS se transmite un paquete SPS (*Sequence Parameter Set* – conjunto de parámetros de secuencia), donde se incluye información común a toda la secuencia de vídeo, como el perfil y el nivel utilizados en la codificación.
- Finalmente, y antes de comenzar el envío de las imágenes codificadas, se adjunta un paquete PPS (*Picture Parameter Set* – conjunto de parámetros de imagen). Este paquete se utiliza para transmitir información relativa a la codificación, similar al paquete SPS, pero de una o varias imágenes individuales.
- *Slice* (rebanada o porción): una *slice* es el conjunto de datos codificados correspondientes a una parte, o a la totalidad, de una imagen. Cada *slice* puede ser descodificada de forma independiente de otra en términos de codificación de entropía, predicción o reconstrucción de la señal. De esta forma los errores producidos en descodificación en caso de pérdidas en la transmisión se ven reducidos. Cada *slice* está compuesta de un número variable de CTUs. La Figura 9 (a) muestra un ejemplo de cómo una imagen puede subdividirse en *slices*.
- *Tiles* (mosaicos): bajo el estándar HEVC se ha definido un subnivel adicional cuyo propósito es facilitar el procesamiento en paralelo de distintas regiones de imágenes. Los *tiles* se corresponden a regiones rectangulares de una imagen y pueden ser descodificados de forma independiente, además pueden compartir la misma información de una cabecera del *slice* en el que se encuentren. La Figura 9 (b) muestra un ejemplo de subdivisión en *tiles*.

Figura 9. Subdivisión de una imagen en *slices* (a) y en *tiles* (b).

2.2.1.3. Perfiles y niveles de HEVC

Como es habitual en los estándares de codificación de vídeo MPEG, en HEVC se han definido una serie de perfiles y niveles que ofrecen distintos grados de funcionalidad y calidad. Éstos permiten establecer diversos grados de operatividad de cara a la implementación del estándar en distintos campos de aplicación. Los perfiles representan subconjuntos de la sintaxis de la trama de bits de salida del codificador, mientras que los niveles especifican, dentro de cada perfil, un conjunto de restricciones a aplicar sobre los distintos parámetros de esa trama.

A continuación se detallan todos aquellos perfiles que han sido incluidos de forma oficial en el estándar; aunque para la realización de este trabajo se han utilizado distintas versiones de decodificadores HEVC (capítulo 10), el perfil seleccionado ha sido en todos los casos *Main*, ya que su uso es el más extendido y maximiza las oportunidades de comparación de los resultados de esta investigación con los de otros trabajos. La Tabla 3 muestra las principales características de los distintos perfiles existentes para el estándar HEVC.

Perfiles	Main	Main 10	Main 12	Main 4:2:2 10	Main 4:2:2 12	Main 4:4:4	Main 4:4:4 10	Main 4:4:4 12	Main 4:4:4 16 Intra
Características									
Profundidad de color	8	8~10	8~12	8~10	8~12	8	8~10	8~12	8~16
Formatos de muestreo	4:2:0	4:2:0	4:2:0	4:2:0/ 4:2:2	4:2:0/ 4:2:2	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4
4:0:0 (Monocromo)	No	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Predicción de alta precisión ponderada	No	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Predicción de componentes cruzados	No	No	No	No	No	Sí	Sí	Sí	Sí
Desactivación de suavizado intra	No	No	No	No	No	Sí	Sí	Sí	Sí
Adaptación Rice persistente	No	No	No	No	No	Sí	Sí	Sí	Sí
RPDCM implícita / explícita	No	No	No	No	No	Sí	Sí	Sí	Sí
Transformación de tamaños de bloque mayores de 4x4	No	No	No	No	No	Sí	Sí	Sí	Sí
Transformar salto de contexto / rotación	No	No	No	No	No	Sí	Sí	Sí	Sí
Procesamiento de precisión extendida	No	No	No	No	No	No	No	No	Sí

Tabla 3. Listado de perfiles incluidos en el estándar HEVC.

La Tabla 4 muestra los valores límite para determinados parámetros que se han definido en los diferentes niveles dentro del perfil *Main*. Así mismo el perfil *Main* incluye dos clasificaciones, *Main Tier* (clasificación principal) y *High Tier* (clasificación alta), éstas se diferencian únicamente en el régimen binario y la capacidad del *buffer* de imágenes en el decodificador.

Nivel	Número máximo de muestras Y	Ratio máximo de muestras Y (muestras/segundo)	Bitrate máximo para el nivel <i>Main</i>	Bitrate máximo para el nivel <i>High</i>	Ratio de compresión mínimo
1	36.864	552.960	128	-	2
2	122.880	3.686.400	1.500	-	2
3	552.960	16.588.800	3.000	-	2
4	2.228.224	66.846.720	12.000	30.000	4
5	8.912.896	267.386.880	25.000	100.000	6
6	35.651.584	1.069.547.520	60.000	240.000	8

Tabla 4. Valores límite para los principales niveles del perfil *Main*.

2.2.2. Modelo de un descodificador HEVC

En este apartado se describen, una vez introducidas las principales características del estándar de codificación HEVC, los módulos que integran un descodificador de vídeo HEVC genérico. La información relativa a la funcionalidad de cada etapa de procesado, así como la implementación de cada una de ellas, es crucial a la hora de establecer técnicas que permitan explotar el procesamiento paralelo de las mismas y tomar decisiones sobre la optimización del código relativo a cada sección. El diagrama de la Figura 10 muestra los distintos bloques que deben integrar un descodificador HEVC. Seguidamente se resume la funcionalidad de cada uno de los bloques.

Figura 10. Diagrama de bloques simplificado de un descodificador HEVC.

- **Decodificación de entropía:** En el estándar HEVC el método utilizado para la codificación/descodificación de entropía es el llamado *Context-based Adaptive Binary Arithmetic Coding* (CABAC) [MSW03]. A partir de la trama de bits codificada, el descodificador de entropía extrae la información sobre el tipo de codificación empleada (inter/intra), los coeficientes de cada CU (transformados y cuantificados) y los vectores de movimiento (en su caso).
- **Cuantificación y transformada inversa:** una vez se han extraído los datos del vídeo codificado mediante la descodificación de entropía se realiza la decuantificación de los coeficientes transmitidos, dichos coeficientes deben ser entonces transformados al dominio temporal mediante la Transformada Discreta del Coseno Inversa (IDCT), obteniendo así los bloques de píxeles que conforman la imagen.
- **Reconstrucción de imágenes:** las CTUs se reconstruyen mediante la suma de las predicciones inter/intra y las CTUs salientes del proceso de cuantificación y transformación.
- **Predicción Intra:** en el caso de que una CU se haya codificado con predicción intra-*frame* se procede a calcular su predicción a partir de la información de los PBs vecinos. Además, en este caso los bloques intra descodificados pasarán directamente a la memoria de imágenes.

- **Predicción Inter:** en este caso el bloque de compensación de movimiento obtiene la predicción a partir de los vectores de movimiento y de las imágenes previamente almacenadas en la memoria de imágenes. Este procesado se realiza a nivel de PUs y puede utilizarse una compensación de movimiento con precisión de hasta un cuarto de píxel.
- **Filtrado antibloques y filtrado SAO:** tras la obtención de la imagen se aplica un primer filtrado antibloques (DBF) [SOHW12] consistente en suavizar los flancos existentes entre los distintas PUs y TUs. Este filtrado se realiza a nivel de CU y se aplica en bloques de 8x8 píxeles. Seguidamente, se aplica un segundo filtrado (SAO, *Sample Adaptive Offset*) [SOHW12]. Este filtrado no está acotado a los bordes de los bloques que integran las imágenes, y consiste en añadir un valor de base (*offset*) a algunos píxeles ya reconstruidos. Los valores a añadir se eligen en función de una tabla de valores transmita y generada por el codificador. Ambos filtros se aplican en el bucle correspondiente a la predicción de imágenes inter, y sus efectos de mejora sobre la imagen descodificada se incrementan al utilizarse de forma conjunta.
- **Memoria de imágenes descodificadas:** las imágenes descodificadas se almacenan en memoria con un doble objetivo: disponer de las imágenes utilizadas durante la predicción inter ya que son necesarias para su correcta descodificación, así como para poder reordenar correctamente las imágenes en su orden natural ya descodificadas como se explicó en el apartado 2.1.3.

2.3. MPEG Reconfigurable Video Coding (RVC)

En este apartado se presentan las principales características del estándar de Codificación de Vídeo Reconfigurable (RVC, de sus siglas en inglés *Reconfigurable Video Coding*). El estándar ha sido desarrollado por el grupo MPEG y actualmente se encuentra estandarizado por la ISO. El estándar RVC ofrece una solución al desarrollo clásico y monolítico de descodificadores de vídeo. Esto es posible mediante la combinación de un nuevo lenguaje para desarrollar aplicaciones y una serie de bibliotecas que incluyen los algoritmos de codificación utilizados en los códecs de MPEG.

2.3.1. Introducción al estándar RVC

El desarrollo del estándar RVC comenzó en el año 2005 con el ánimo de ofrecer nuevas soluciones en la implementación de códecs de vídeo digital. Dichos códecs se sirven de algoritmos cuya complejidad ha ido creciendo a la vez que se han ido consiguiendo mayores tasas de compresión. Además, el desarrollo de nuevos dispositivos multimedia, como *tablets* o *smartphones*, y la implantación de la televisión digital, requieren de nuevas implementaciones más versátiles. La necesidad de poder reutilizar elementos ya desarrollados, y de integrar en un solo descodificador diversos códecs, permitiendo que el mismo se adapte dinámicamente al flujo de datos entrante, hace de RVC el estándar idóneo para afrontar los retos del momento.

Con el desarrollo de RVC, MPEG realizó un trabajo de reorganización de los estándares ya existentes, de esta forma MPEG-1, 2 y 4 se integraron en MPEG-A (formatos de aplicaciones multimedia), MPEG-B (Sistemas tecnológicos), MPEG-C (tecnologías de vídeo) y MPEG-H (para la difusión en medios heterogéneos).

El nuevo *framework* MPEG RVC fue estandarizado como MPEG-B parte 4 y MPEG-C parte 4 bajo las normas ISO/IEC 23001-4 [ISO09] e ISO/IEC 23002-4 [ISO10] respectivamente, con el objetivo de reemplazar las referencias *software* de los descodificadores, escritas en lenguaje C o HDL, por especificaciones de la descodificación de vídeo a nivel de bibliotecas de componentes. La primera de estas normas ofrece un nuevo lenguaje para la descripción de códecs de vídeo; la segunda, una biblioteca de herramientas utilizada por los algoritmos de descodificación de vídeo en los estándares de MPEG. De esta forma, y bajo el estándar RVC, las antiguas referencias *software* son sustituidas por representaciones Abstractas de Modelos de Descodificadores (*Abstract Decoder Models* – ADMs). Los detalles que ilustran este modelo se describen en siguiente apartado y en la Figura 11.

Uno de los principales objetivos de MPEG RVC es el desarrollo de descodificadores de vídeo digital a partir de una serie de especificaciones de alto nivel basadas en modelos de flujos de datos (*dataflow*). De esta forma, un ADM representa el modelo de un descodificador como un diagrama de flujos de datos. Este tipo de representaciones ofrecen una serie de ventajas claves en el estándar RVC, las cuales se enumeran a continuación:

- Un lenguaje de programación abstracto, que permite a los desarrolladores abstraerse de los detalles de bajo nivel de la arquitectura y poner el énfasis en la mejora y optimización de los algoritmos de compresión.
- Un paralelismo escalable, gracias a que los nuevos métodos de programación se basan en modelos de flujos de datos, lo cual facilita un mayor paralelismo de los programas desarrollados.
- La reutilización de módulos y algoritmos es más fácil, ya que existen numerosas similitudes entre los módulos utilizados en distintas implementaciones. Además, el *framework* MPEG RVC facilita el desarrollo de algoritmos que puedan reutilizarse en futuros estándares.

2.3.2. Descripción de descodificadores de vídeo con MPEG RVC

En esta subsección se introducen los principales conceptos necesarios para entender el funcionamiento y la implementación de los módulos que integran una aplicación basada en MPEG RVC. La mayoría de estos conceptos son clave para entender el desarrollo de esta tesis doctoral.

Hoy en día los estándares MPEG-B y MPEG-C, sobre los que se apoya MPEG RVC, siguen evolucionando con el objetivo de incluir nuevos algoritmos sobre los que

se puedan desarrollar futuros estándares de descodificación de vídeo. Como se introducía en la subsección anterior, los ADMs pueden representar las especificaciones de un decodificador de vídeo basado en alguno de los estándares existentes de MPEG.

Desde un punto de vista técnico, MPEG RVC queda definido mediante una biblioteca de herramientas de codificación, una descripción sintáctica del descodificador y una descripción del flujo de datos entre las unidades funcionales que integran las aplicaciones. Cada uno de estos elementos queda integrado en el ADM como una Biblioteca de Herramientas de Vídeo (*Video Tool Library* – VTL), un Lenguaje Funcional de unidad de Red (*Functional unit Network Language* – FNL) [ISO11] y un Lenguaje para la Descripción Sintáctica del *Bitstream* (*Bitstream Syntax Description Language* – BSDL) respectivamente. La Figura 11 ilustra gráficamente el modelo basado en ADMs.

Figura 11. Modelo de ADMs utilizado en el estándar RVC.

Los algoritmos que implementan la funcionalidad se encapsulan en entidades independientes llamadas Unidades Funcionales (*Functional Units* – FUs), estas entidades conforman una serie de herramientas de codificación proporcionadas por la VTL. La funcionalidad de cada FU es implementada a su vez por otra entidad denominada *actor*. De esta forma una FU puede contener a su vez otras FUs, sin embargo, en el nivel más bajo de la jerarquía cada FU debe contener, al menos, un actor. La programación de los actores se realiza en el lenguaje de referencia RVC-CAL Actor Language (RVC-CAL) [CAL] [BEJ⁺09]. Los conceptos básicos, tanto de la programación interna, como del funcionamiento de los actores, se describen someramente en la subsección siguiente.

El FNL permite realizar la interconexión entre las FUs. En el estándar RVC, una aplicación queda definida mediante una serie de instancias de FUs o actores, representadas gráficamente mediante cajas, y la interconexión de las mismas mediante flechas. Las flechas representan el flujo de datos entre los actores y se conectan a éstos mediante puertos de entrada/salida, la implementación de los flujos de datos se realiza

mediante colas de datos tipo FIFO (*First In First Out*, primero en entrar, primero en salir). Existen tres tipos de conexiones definidas en FNL:

- *Input* (entrada): entre un puerto de entrada de la red y una instancia.
- *Output* (salida): entre la salida de una instancia y un puerto de salida de la red.
- *Connection* (conexión): entre la salida de una instancia y la entrada de otra instancia.

La Tabla 5 ilustra los distintos tipos de elementos que pueden darse en este tipo de modelos utilizando FNL a partir del diagrama de la Figura 12.

Figura 12. Diagrama básico de un modelo basado en FNL.

Instancias	<pre> <Instance id=" FU_1 "> <Class name =" Algo_Ejemplo_1 "/> </ Instance > <Instance id=" FU_2 "> <Class name =" Algo_Ejemplo_2 "/> </ Instance > </pre>
Connections	<pre> <Connection src= » FU_1 « src -port = »B » dst= » FU_2 « dst -port = »D »/> <Connection src= » FU_1 « src -port = »C » dst= » FU_2 « dst -port = »E »/> </pre>
Inputs	<pre> <Input src=" FU_1 " src -port ="A"/> </pre>
Outputs	<pre> <Output src= » FU_2 « src -port = »F »/> </pre>

Tabla 5. Elementos existentes en el diagrama FNL de la Figura 12.

Finalmente se incluyen en el *framework* las herramientas necesarias para realizar el proceso de análisis y traducción sintáctica de *bitstreams* mediante RVC-BSDL. El BSDL es capaz de realizar la traducción sintáctica de cualquier *bitstream* compatible con el estándar y está definido en la norma ISO/IEC 23001-5 [ISO08].

2.3.3. RVC-CAL Actor Language, conceptos básicos

Con el objetivo de hacer esta memoria lo más autocontenida posible, este apartado presenta únicamente aquellos conceptos principales sobre el lenguaje de programación RVC-CAL Actor Language. Para tener un conocimiento más detallado de la especificación CAL Actor Language se recomienda acudir al informe realizado por la Universidad de California en Berkeley donde fue desarrollado [CAL].

RVC-CAL Actor Language es un lenguaje de programación de alto nivel utilizado en la descripción del comportamiento de los actores que integran las aplicaciones basadas en el estándar MPEG RVC. Los actores se componen de interfaces, puertos de entrada/salida, variables de estado y otros parámetros. A lo largo de esta subsección se van a utilizar pequeños extractos de código para que sirvan de ejemplo.

El primer elemento de un actor es su cabecera, en la cual se deben declarar los puertos de entrada/salida necesarios. La Figura 13 muestra la parte de código relativa a la cabecera, los puertos de entrada (Y, U y V) y de salida (YUV), las acciones y la FSM del actor MergeYUV integrado en un decodificador HEVC-RVC.

```

Package org.ietr.mpeghe.part2.common;

actor MergeYUV () //Entradas al actor
    int(size=8) Y,
    int(size=8) U,
    int(size=8) V

    ==> //Salidas del actor
    int(size=8) YUV :

    // Action, select the Y blocks
    Read_Y: action Y:[ y ] repeat 4096 ==> YUV:[ y ] repeat 4096
end
(...) //Resto de acciones (Read_U y Read_V)

fsm Y :
    Y ( Read_Y ) ==> U;
    U ( Read_U ) ==> V;
    V ( Read_V ) ==> Y;
end
end

```

Figura 13. Cabecera y cuerpo en lenguaje RVC-CAL del actor MergeYUV del proyecto *Research*.

El procesamiento de un actor se describe mediante una secuencia de pasos que se ejecutan de forma atómica, de manera similar a como lo hacen las funciones escritas en otros lenguajes de programación, como el lenguaje C. Estas secuencias de pasos se denominan acciones. La funcionalidad de cada actor debe describirse como parte del código de cada acción. A la ejecución de una acción se le denomina disparo (*firing*); en cada ejecución el actor consume elementos (*tokens*) disponibles en sus entradas, produce *tokens* en los puertos de salida y modifica las variables de estado. Este proceso ocurre para cualquier actor y es el mismo en cualquier proceso de disparo. En el ejemplo de la Figura 13 se muestra la acción *Read_Y* junto con el código de la misma y sus acciones sobre los puertos de E/S. La descripción del actor concluye con la especificación de la FSM, en este caso tras ejecutar la acción *Read_Y* se continuará con *Read_U*, y seguidamente *Read_V*, para volver entonces al punto de partida.

Un actor puede estar compuesto por una o varias acciones. Pueden existir situaciones no determinísticas si las condiciones de ejecución de cada una de las acciones no han sido especificadas para cualquier situación posible. En la Figura 14 se ejemplifica un caso no determinístico básico, ya que ambas acciones pueden ejecutarse al no haberse descrito condiciones de ejecución; en este caso la acción que será finalmente ejecutada es indeterminada.

```

action Input1: [x] → [x] end
action Input2: [x] → [x] end

```

Figura 14. Ejemplo de sección de código no determinística en RVC-CAL.

La ejecución de una acción se produce cuando hay datos disponibles en las entradas, espacio suficiente disponible en los puertos de salida afectados y las condiciones de *guarda* se cumplen, si existen. Las condiciones de guarda pueden referirse al valor de los datos de entrada, al estado interno del actor o a ambos. La Figura 15 ilustra un ejemplo: mientras el valor de la entrada ‘sel’ sea TRUE el dato en ‘A’ pasa al puerto de salida, en caso contrario se selecciona el dato presente en el puerto ‘B’. Dependiendo del propio estado del actor, las condiciones de guarda pueden utilizarse para decidir las próximas acciones a ejecutar. Toda la gestión de la ejecución de las acciones se realiza mediante una Máquina de Estados Finita (*Finite State Machine* – FSM), donde se pueden tener en cuenta los valores de los *tokens* de entrada, las condiciones de guarda y las prioridades.

```
Read_CodingUnit_end_sendQp.blk4x4:action QpData:[length, qp_y]→Qp:[qp_y]
guard //Condición de guarda para que se pueda ejecutar la acción precedente
length = 0
end
```

Figura 15. Ejemplo de código en RVC-CAL de una acción del actor QpGen con condición de guarda.

2.3.4. Funcionamiento interno de los actores

El código fuente que implementa cada uno de los actores es el resultado de las transformaciones realizadas sobre el código CAL de los mismos. Además, se genera el código relativo a la gestión de entrada/salida de datos mediante las FIFOs y la FSM que controla el comportamiento interno de cada actor.

Todos los actores disponen además de dos funciones: la función de inicialización, que establece el estado inicial de la FSM y realiza una primera lectura de *tokens* en las FIFOs de entrada del actor, y la función del planificador, la cual es llamada secuencialmente por el hilo de ejecución y que controla la ejecución de las funciones internas del actor mediante la FSM. La Figura 16 esquematiza el funcionamiento de los actores, cuando un actor finaliza su ejecución (no hay más *tokens* en sus FIFOs de entrada) el planificador de acciones llamará a ejecución al siguiente actor en ese hilo de ejecución (estos conceptos se explican en detalle en el apartado 4.5.2).

Figura 16. Diagrama del funcionamiento interno de los actores.

2.3.5. Implementaciones de descodificadores de vídeo basadas en MPEG RVC

Desde que en el año 2002 comenzara el desarrollo de MPEG RVC, diferentes grupos de investigación han llevado a cabo numerosas aplicaciones, tanto para la descodificación de vídeo digital, como dentro de otros campos de estudio. Algunos de los grupos más destacados son: IETR (*Institut d'Électronique et de Télécommunications de Rennes*) [IETR] e IRISA (*Institut de Recherche en Informatique et Systèmes Aléatoires*) [IRISA], ambos dentro de la red de centros de investigación del INSA (*Institut National des Sciences Appliquées*) de Rennes (Francia), y l'EPFL (*École Polytechnique Fédérale de Lausanne*) [EPFL] en Suiza. Cabe destacar que el GDEM, donde se ha desarrollado esta tesis doctoral, ha realizado varias publicaciones relacionadas con MPEG RVC [JRR⁺13] [RCS⁺13] [JRW⁺13]. Existe una estrecha relación entre los cuatro grupos de investigación mencionados, y en varias ocasiones se han realizado intercambios de personal docente e investigador con el fin de fomentar la colaboración internacional y la superación conjunta de algunos de los retos planteados por MPEG RVC.

Tanto para el desarrollo de esta tesis doctoral, como en la mayoría de proyectos impulsados por los distintos grupos de investigación anteriormente mencionados, se ha utilizado un repositorio web [ORC] donde se encuentran disponibles numerosas aplicaciones basadas en RVC. En el repositorio se pueden encontrar los siguientes descodificadores de vídeo digital basados en MPEG RVC:

- Descodificador MPEG-4 parte 2 SP.
- Descodificador MPEG-4 parte 10 (AVC o H.264), perfiles PHP y CBP.
- Descodificador MPEG-H parte 2 (HEVC o H.265).

Durante la elaboración del Trabajo Fin de Máster del doctorando en 2012 se realizó un estudio detallado de los descodificadores MPEG-4 parte 2 y MPEG-4 parte 10. En este trabajo se analizó el rendimiento de estas implementaciones, tanto sobre plataformas mono-núcleo como multinúcleo en arquitecturas tipo PC y ARM. La memoria está disponible en [Cha12].

El descodificador HEVC ha sido el utilizado como aplicación de referencia para el desarrollo de esta tesis doctoral. Cabe destacar que hay desarrolladas distintas versiones de este descodificador. En el apartado 10.1 se describen pormenorizadamente cada una de estas versiones y su importancia en el desarrollo de este trabajo.

- Descodificador HEVC-RVC de referencia (REF): primera versión estable que se desarrolló. En esta primera versión el grado de paralelismo alcanzable está muy limitado, sin embargo se incluyen ya las etapas de filtrado SAO y antibloques, y está preparada para resoluciones de hasta 4K.
- Descodificador HEVC-RVC YUV: versión con el procesado dividido en FUs independientes de las componentes Y, U y V. Esta versión ofrece un mayor grado de paralelismo que el alcanzable con la versión de referencia. Las versiones de descodificadores MPEG-4 parte 10 con RVC ya incluían implementaciones basadas en el procesado en paralelo de luminancias y crominancias.
- Descodificador HEVC-RVC FBn: ésta es una de las últimas versiones desarrolladas, y en este caso se explota el paralelismo a nivel de imagen (*frame-based*). El procedimiento consiste en replicar tantos descodificadores como imágenes en paralelo se deseen procesar al mismo tiempo. Una de las principales ventajas de esta versión radica en que de esta manera se puede paralelizar el módulo *Parser*, el cual no tiene paralelismo con el resto de unidades funcionales.

En el repositorio también se encuentran proyectos de aplicaciones no relacionados con el procesado de vídeo digital basados en RVC. A continuación se enumeran algunos de los más relevantes:

- *Crypto*: descripción de algoritmos para tareas de encriptación.
- *DigitalFiltering*: implementaciones de filtros FIR e IIR.
- *JPEG y Jpeg2000*: codificador/descodificador para JPEG.
- *Stereo*: Implementación de algoritmos de *stereo matching*.

2.4. Resumen

A lo largo de este capítulo se han presentado los principios de la codificación de vídeo digital, incluyendo un breve repaso a la evolución de los estándares de codificación en los últimos años.

Seguidamente se ha introducido el estándar H.265/HEVC. La característica más destacable de dicho estándar es la mejora de hasta un 50% en la tasa binaria necesaria para la transmisión de una secuencia de vídeo respecto de su antecesor H.264/AVC para un mismo nivel de calidad. Se han presentado las principales características del estándar, perfiles y niveles, así como los principales bloques funcionales que integran los códecs.

Por último, se ha presentado el estándar de vídeo reconfigurable RVC. RVC permite realizar descripciones de alto nivel de descodificadores de vídeo e integra un conjunto de bibliotecas de MPEG con diversas funcionalidades necesarias en la implementación de este tipo de aplicaciones. Este estándar supone una revolución en el desarrollo de descodificadores de vídeo, ya que posibilita llevar a cabo diseños con una alta modularidad al tiempo que permite reutilizar elementos previamente desarrollados, reduciendo así los tiempos de desarrollo de las implementaciones. Además, permite llevar a cabo particionados funcionales de las FUs que integran los descodificadores en entornos multinúcleo. Esto se hace asociando la ejecución de cada una de los actores a cualquiera de los núcleos disponibles en el procesador.

3. Tecnologías de implementación

En este capítulo se realiza una revisión del estado del arte en las tecnologías de implementación de códecs de vídeo con procesadores multinúcleo. Se han considerado tres tipos de tecnologías: GPPs de altas prestaciones, GPPs orientados a sistemas empuotrados y DSPs. Las tecnologías basadas en arquitecturas *hardware* específicas no se han tenido en cuenta porque las metodologías que se utilizan en su diseño están fuera del marco de esta tesis. A lo largo del capítulo se describen las características más relevantes de las tecnologías de implementación mencionadas y, para cada una de ellas, se recopilan las implementaciones más destacadas, poniendo el foco en los descodificadores de vídeo basados en el estándar HEVC.

En el apartado 3.1 se enumeran e introducen brevemente las tecnologías de implementación relevantes para el desarrollo de esta tesis. En el apartado 3.2 se muestra la arquitectura típica de los chips multinúcleo basados en GPPs de altas prestaciones, se analizan sus ventajas e inconvenientes y se recopilan las implementaciones más destacadas de códecs HEVC realizadas en base a ese tipo de tecnología, haciendo hincapié en las metodologías y herramientas utilizadas. En los apartados 3.3 y 3.4 se hace lo mismo para los chips multinúcleo basados en GPPs para sistemas empuotrados y para los basados en DSPs, respectivamente.

3.1. Introducción a las tecnologías de implementación

Durante la última década los fabricantes de procesadores de todo tipo han venido desarrollado sistemas multinúcleo como alternativa a seguir aumentando la frecuencia de reloj de los chips para incrementar sus prestaciones.

A lo largo de los siguientes apartados se describen las principales características y ventajas ofrecidas por tres tipos de sistemas utilizados en el desarrollo de descodificadores de vídeo. En primer lugar, en el apartado 3.2, se estudian las particularidades de los sistemas integrados por procesadores tipo GPP de altas prestaciones. Estos sistemas, típicamente utilizados en los ordenadores personales (PCs), conforman dispositivos multipropósito típicamente utilizados en las fases de desarrollo y prueba de aplicaciones. Son los que ofrecen mayores prestaciones, pero su coste es elevado y su consumo energético es alto, requiriendo habitualmente el uso de disipadores activos y sistemas de ventilación.

Sin embargo, también existen dispositivos integrados por procesadores tipo GPP con menores prestaciones que son adecuados para ser utilizados en terminales móviles. Dichos sistemas conforman típicamente un sistema empotrado que integra, además de los GPPs, aquellos periféricos, memorias o dispositivos complementarios necesarios para el funcionamiento de una amplia variedad de diseños. Estos dispositivos se revisan en el apartado 3.3.

Por último, en el apartado 3.4 se detallan las particularidades e implementaciones con procesadores tipo Multi-DSP, que como se ha dicho a lo largo de esta memoria son el principal objetivo de esta tesis doctoral.

Cada tipo de arquitectura presenta distintas ventajas que marcan la idoneidad de ésta para su utilización en entornos diversos, tales como el consumo energético o las facilidades disponibles a la hora de realizar la programación sobre un sistema multinúcleo⁹. Sin embargo la utilización de técnicas como *Single Instruction, Multiple Data*¹⁰ (Una Instrucción, Múltiples Datos – SIMD) o la sustitución de fragmentos de código por transcripciones directas de éstos en lenguaje ensamblador¹¹ pueden reportar importantes mejoras en el rendimiento, si bien su integración en el *software* puede ser costosa. La complejidad en la utilización de estas técnicas sobre las distintas plataformas no es idéntica entre ellas, así como la mejora derivada de este proceso.

3.2. Tecnologías de implementación basadas en GPPs de altas prestaciones

3.2.1. GPPs de altas prestaciones

Los procesadores más potentes están formados por varios núcleos GPP de altas prestaciones como los fabricados por Intel [INT] o AMD [AMD]. Estos procesadores soportan SOs como Windows o Linux, su programación está muy extendida y pueden desarrollar capacidades de cómputo muy elevadas. Además, las plataformas basadas en este tipo de procesadores (ordenadores personales, portátiles,...) tienen relativamente bajo coste y son muy accesibles. Por todas estas razones, las primeras implementaciones de un estándar de codificación que se desarrollan en el entorno científico suelen utilizar este soporte tecnológico. Es sobre este tipo de plataformas donde se encuentran las implementaciones capaces de descodificar en tiempo real secuencias de mayor calidad y resolución. Sin embargo, este tipo de sistemas tienen un elevado consumo de recursos, energía y espacio, por lo que no resultan ser eficaces para el desarrollo de sistemas

⁹ Estas características dependen de diversos factores, como la compatibilidad de la arquitectura con APIs de gestión de multitareas, como OpenMP (capítulo 6).

¹⁰ Las técnicas SIMD dependen de la arquitectura, y permiten conseguir un mayor paralelismo en el procesamiento de datos. Las instrucciones SIMD se basan en utilizar una sola operación sobre un conjunto amplio de datos siendo éstos procesados de forma simultánea por distintas unidades de procesamiento. Para un conocimiento más exhaustivo se recomienda consultar [DJL⁺06].

¹¹ El nivel de optimización del código máquina generado por un compilador no siempre es lo suficientemente elevado, por lo que éste puede ser reescrito por el desarrollador pudiendo obtener así importantes mejoras en el rendimiento de la aplicación. Este tipo de tareas son costosas y requieren de largos periodos de desarrollo.

empotrados. Tampoco su coste es lo suficientemente bajo para que resulten competitivos en este tipo de aplicaciones.

Otra ventaja de este tipo de plataformas, y de los SOs capaces de funcionar sobre ellas, es la disponibilidad del conjunto de herramientas *software* necesarias para los grupos de trabajo de desarrollo de aplicaciones. El hecho de que todas las fases de diseño, prueba y depuración puedan llevarse a cabo sobre la misma plataforma facilita y acorta los tiempos de diseño del *software*.

Además, la mayoría de GPPs de altas prestaciones, como los de las familias más recientes de los fabricantes Intel y AMD, disponen de los repertorios de instrucciones SIMD como SSE (SSE) [SSE] y *3Dnow!* (3DN) [3DN] respectivamente. Dichos repertorios de instrucciones pueden ser utilizados para optimizar el código de una aplicación, consiguiendo importantes mejoras en el rendimiento. Un ejemplo de su uso sobre decodificadores de vídeo HEVC se puede consultar en los trabajos publicados por Khaled et al. [JRS⁺14] y [JYS⁺16].

La Figura 17 muestra el diagrama de bloques genérico de un procesador de la familia i7 de última generación del fabricante Intel. Como se puede observar este tipo de procesadores integran, desde hace varios años, varios núcleos en su arquitectura. Además disponen de un nivel típicamente adicional de memoria caché (L3) con una capacidad elevada. También incorporan el acceso a aquellos periféricos y subsistemas externos de comunicación, visualización y gestión de memoria externa de alta capacidad. Más recientemente, este tipo de procesadores incorporan facilidades *hardware*, como el *HyperThreading* del fabricante Intel [INT], que permiten paralelizar la ejecución de los procesos emulando dos procesadores sobre un único núcleo físico, obteniendo rendimientos más elevados de las aplicaciones desarrolladas.

Figura 17. Diagrama de bloques de un procesador de la familia Intel i7.

A continuación se presenta una relación de algunas implementaciones desarrolladas sobre este tipo de arquitecturas.

3.2.2. Implementaciones desarrolladas sobre plataformas tipo GPP de altas prestaciones

En [YDSG12] investigadores del Instituto de Informática y Tecnología de la Universidad de Pekín presentan un descodificador HEVC implementado sobre un Intel i5 con optimizaciones SIMD. Tras identificar los módulos del descodificador de mayor carga computacional, en su caso la compensación de movimiento, el filtrado antibloques y la transformada inversa, proceden a optimizar el código de referencia del descodificador (versión HM 4.0) de dichos módulos mediante el uso de instrucciones SIMD para arquitecturas x86¹². Con estas mejoras, el tiempo de ejecución de los módulos optimizados se ve reducido hasta en un 86% para la compensación de movimiento y el filtro antibloques, y hasta un 70% para la transformada. El número de imágenes por segundo (FPS) decodificadas se ve globalmente incrementado en 3,14 veces, respecto al rendimiento conseguido con el código del estándar, en el peor de los casos y hasta por 5,23 veces para la secuencia que presenta los mejores resultados. Sin embargo, no se hace ninguna mención a la utilización de varios núcleos de los dos disponibles en un Intel i5. En este trabajo se utilizan secuencias con resoluciones desde los 416x240 píxeles hasta 1920x1080 píxeles.

Posteriormente, los mismos investigadores publicaron un trabajo más extenso [DSY⁺14] en el que se realizan dos implementaciones de descodificadores HEVC, una sobre un Intel i7 y otra sobre un ARM Cortex-A9 (ver apartado 3.3). En este caso sí se utilizan varios núcleos de los disponibles en el procesador y los resultados se comparan con una de las implementaciones del descodificador OpenHEVC (ver capítulo 5). El uso combinado de instrucciones SIMD en ciertos módulos del descodificador, y la ejecución paralelizada del procesamiento a nivel de *frame* (*Frame Based* ver apartado 10.1.3) permiten obtener un factor de mejora medio en la tasa de imágenes por segundo de 6,32 con un procesador, respecto a los resultados con el descodificador de referencia (HM 10.0), y de 13,24 al utilizar simultáneamente los cuatro procesadores del Intel i7.

Por otra parte, uno de los grupos que lideran el desarrollo de implementaciones para aplicaciones de decodificación de vídeo es el ya mencionado IETR del INSA de Rennes (ver apartados 2.3.5 y 5.1). En algunos de sus trabajos más recientes [HRD14] y [HRD16], se presentan implementaciones del descodificador OpenHEVC (ver capítulo 5) incluyendo algunas de las extensiones incluidas en el estándar HEVC [SBCO⁺13], como *Scalable HEVC*¹³ (SHVC) [ISO12]. En el primero de los trabajos se realiza una implementación del descodificador SHVC sobre un Intel Xeon con seis procesadores. Aplicando distintas técnicas de paralelismo contempladas en el estándar HEVC, como *Wavefront Parallel Processing* (WPP) [CHP11], se consigue la decodificación en tiempo real de secuencias de hasta 2560x1600 píxeles. En el segundo de los trabajos

¹² x86 Familia de procesadores cuyo repertorio de instrucciones se basa en la CPU del Intel 8086.

¹³ La extensión SHVC del estándar HEVC contempla la implementación escalable de las características de resolución espacial, temporal y de calidad (o *Signal Noise Ratio* – SNR).

publicados se incluye la extensión *Multiview HEVC*¹⁴ (MV-HEVC) [TWCH⁺14] en el código del decodificador OpenHEVC. En este caso se consigue decodificar en tiempo real una secuencia de resolución 4K a 60 FPS sobre un procesador Intel i7 de seis núcleos.

En otro trabajo [JRS⁺14] realizado por investigadores del IETR, se realiza una implementación de un decodificador HEVC-RVC sobre un Intel Xeon de seis núcleos. En esta ocasión se realiza una optimización del código generado por Orcc (ver capítulo 8) y se emplea la versión YUV del decodificador HEVC con el estándar RVC. Además de implementar el reparto del procesamiento de los actores en componentes YUV, lo que permite elevar el paralelismo de la solución, se integran en el código fuente llamadas a funciones optimizadas en lenguaje C para arquitecturas x86 incluidas en el código de OpenHEVC (ver capítulo 5). También se realiza una mejora en la implementación de las FIFOs que transmiten la información entre los actores. Todas estas mejoras permiten obtener, en el mejor caso, una ganancia de 7,6 con respecto a la implementación de referencia sin optimizaciones. Esta situación se da al utilizar cinco núcleos de los seis disponibles, pasando de una velocidad de decodificación de 6,1 FPS a 46,6 FPS. Cabe destacar que tanto este trabajo, como otras publicaciones que lo complementan [JYS⁺16], tienen una especial relevancia para esta tesis doctoral, como se verá en los apartados 9.4.2.1 y 9.7 de esta memoria.

3.3. Tecnologías de implementación basadas en GPPs para sistemas empujados

3.3.1. GPPs para sistemas empujados

Para satisfacer las necesidades de los sistemas empujados, diferentes fabricantes han desarrollado procesadores basados en núcleos GPP de menores prestaciones que los anteriores. Estos procesadores son típicamente utilizados en terminales portátiles, tanto de telefonía móvil como de otros dispositivos multimedia cuya fuente energética es una batería. La utilización de esta alternativa tecnológica ha experimentado un incremento constante durante la última década, debido principalmente a su reducido coste, su bajo consumo energético y a una potencia de procesamiento siempre creciente. En la actualidad el fabricante ARM Holdings [ARM] es líder en este segmento de mercado, y sus procesadores Advanced RISC Machine (ARM) se utilizan por parte de fabricantes de dispositivos comerciales tan destacados como Samsung, Sony Ericsson, Apple o Nokia, entre muchos otros.

Este tipo de plataformas han experimentado un gran éxito comercial en el mercado de las telecomunicaciones durante la última década. A continuación se describen sus principales características:

- *Reduced Instruction Set Computer* (RISC – Computadora con un Conjunto de Instrucciones Reducido). Al contrario que las CPU basadas en la filosofía de diseño

¹⁴ La extensión MV-HEVC del estándar HEVC permite la codificación de varias vistas en un solo *bitstream*.

CISC (*Complex Instruction Set Computing*), la arquitectura RISC busca un diseño de CPU que ejecute una serie de instrucciones reducidas, de tamaño fijo y reduciendo el número de accesos a memoria. Este tipo de arquitectura permite simplificar el diseño interno del procesador, reduciendo el número de transistores que lo integran. De esta manera se consigue reducir el consumo energético y la disipación.

- Consumo y disipación de calor. Debido al menor consumo energético de este tipo de plataformas la mayoría de ellas no requieren de sistemas activos para la disipación de calor, siendo suficiente una disipación pasiva mediante elementos metálicos conductores. Esto redundará en una reducción del consumo total de energía, además de reducir el espacio necesario dentro del dispositivo, así como el ruido producido por los ventiladores. En el apartado 7.3 se presenta la plataforma Odroid U3, utilizada durante la fase de desarrollo de esta tesis doctoral, la cual integra un procesador tipo ARM multinúcleo.

3.3.2. Implementaciones desarrolladas sobre plataformas GPP para sistemas empujados.

En [NBPM⁺14] investigadores del IETR del INSA de Rennes presentan una implementación del descodificador HEVC sobre la plataforma Odroid, con el SoC Exynos de Samsung (ver apartado 7.3), el cual dispone de ocho núcleos en su arquitectura, si bien sólo se utilizarán cuatro para la toma de resultados. El código escogido para la realización de pruebas es una versión del descodificador OpenHEVC con optimizaciones SIMD para esta arquitectura. El objetivo de este trabajo se centra en el ahorro de consumo energético orientado a terminales móviles mediante el uso de la técnica *Dynamic Voltage Frequency Scaling*¹⁵ (DVFS). En este contexto presentan resultados en los que se consigue descodificar una secuencia de 1280x720 píxeles a 60 FPS en tiempo real, con un consumo aproximado de 1,1 W. Sin embargo, en los resultados publicados no se incluye el costo de la gestión y representación de las imágenes por pantalla.

Como se explicó en el apartado anterior, en [DSY⁺14] investigadores del Instituto de Informática y Tecnología de la Universidad de Pekín presentan dos implementaciones de descodificadores HEVC, una sobre un Intel i7 y otra sobre un ARM Cortex-A9 con dos núcleos en su arquitectura. La implementación sobre la plataforma ARM consigue una velocidad de descodificación de hasta 55 FPS para secuencias de 1280x720 píxeles, utilizando simultáneamente los dos núcleos disponibles en la plataforma. El código fuente utilizado como punto de partida es la versión HM 10.0. Por último, los autores no facilitan información sobre el consumo energético de la plataforma en este estudio.

¹⁵ La técnica DVFS permite variar dinámicamente tanto la alimentación (tensión), como la frecuencia de funcionamiento, de un procesador con el fin de reducir su consumo energético.

En [BLRP14] investigadores de la Universidad de Genoa realizan una implementación del decodificador HEVC con optimizaciones SIMD sobre un ARM Cortex A9 con dos núcleos. En los resultados presentados se consiguen, en el mejor de los casos, tasas de decodificación de hasta 73 FPS para una secuencia con una resolución de 1280x720 píxeles. En este caso los autores tampoco facilitan información sobre el consumo energético de la plataforma.

Finalmente, en [RNH⁺16] investigadores del INSA de Rennes realizan una implementación del decodificador OpenHEVC (ver capítulo 5) sobre un procesador Exynos 5410 con cuatro núcleos ARM Cortex-A15 y cuatro núcleos ARM Cortex-A7. En esta implementación se han llevado a cabo optimizaciones de bajo nivel utilizando instrucciones SIMD para el procesador utilizado en aquellas secciones de código que más carga computacional suponen. También se incluyen medidas para paralelizar la ejecución del decodificador a nivel de imágenes (*frame-based*), y se implementan técnicas DVFS para la reducción del consumo energético. Los resultados muestran un factor de mejora de 3 en el mejor de los casos y al utilizar 64 hilos de ejecución, con respecto a la tasa de FPS con un núcleo. Finalmente, en el trabajo mencionado se muestra cómo la energía consumida al ejecutar una aplicación utilizando 4 hilos de ejecución se reduce en un 46% con relación a la que se consume utilizando un único hilo.

3.4. Tecnologías de implementación basadas en DSPs

3.4.1. Tecnología DSP

Finalmente, la tecnología DSP ha estado históricamente ligada al procesamiento de audio y vídeo. Debido a las limitaciones que presentan las plataformas basadas en DSP relacionadas con el soporte de SOs y de otros recursos (ver apartado 7.1) en muchas ocasiones, éstas se utilizan como aceleradores dentro de arquitecturas en las cuales un GPP realiza la gestión principal de la ejecución del *software* y despacha al DSP la ejecución de aquellos algoritmos susceptibles de ser optimizados sobre este tipo de arquitectura. Es por esto que en la literatura científica no suelen encontrarse implementaciones completas de decodificadores de vídeo sobre DSPs, sino que se suelen presentar optimizaciones de ciertos módulos que integran el decodificador.

En el caso de los DSPs, una de sus principales ventajas reside en el diseño de su arquitectura y de instrucciones específicas que permiten obtener importantes mejoras en el rendimiento, como se expuso en la tesis doctoral de Fernando Pescador [Pes11]. La arquitectura de los DSP está basada en el modelo *Super Harvard*, en dicho modelo existen memorias diferenciadas para datos e instrucciones. Estando ambas conectadas a la CPU mediante sus correspondientes *buses* de datos y direcciones. Cabe destacar que la CPU dispondrá de al menos un nivel de memoria caché, donde se copiarán de las memorias aquellos datos a procesar. Para el acceso a datos almacenados en memoria externa se dispone de un controlador de entrada/salida que gestiona la transferencia de los mismos a la memoria de datos. Por último, otra ventaja de los dispositivos DSP es

su reducido consumo energético, lo que los hace idóneos para su integración en terminales cuya fuente de alimentación sea una batería.

La Figura 18 presenta, a modo de ejemplo, un diagrama con los bloques que integran un procesador DSP multinúcleo de última generación del fabricante Texas Instruments [TI].

Figura 18. Diagrama de bloques de un procesador DSP multinúcleo de última generación.

El diseño del núcleo DSP se diferencia además por disponer de una unidad *Multiply-Accumulate* (MAC), capaz de realizar operaciones de multiplicación acumulación en un solo ciclo de reloj. En la Figura 19 se muestra un diagrama de una unidad MAC como las que se integran en este tipo de procesadores.

Figura 19. Diagrama de bloques de una unidad MAC en un procesador tipo DSP.

Por último, los procesadores DSP también soportan la ejecución de instrucciones SIMD sobre las unidades MAC, lo que permite a este tipo de plataformas superar en rendimiento a las plataformas GPP cuando se trata de realizar tareas de procesamiento numérico repetitivo.

3.4.2. Implementaciones desarrolladas sobre plataformas tipo DSP

Las plataformas DSP se emplean típicamente como aceleradores asociados a otros procesadores los cuales delegan el procesamiento de ciertos algoritmos a los núcleos DSP. Así por ejemplo, en [LSL⁺15] investigadores del centro de I+D DMC de Samsung en Suwon (Corea del Sur), realizan una implementación del decodificador HEVC en un SoC específico para televisión digital que trabaja a una frecuencia de reloj inferior a los 400 MHz. El SoC está dividido en dos unidades de procesamiento independientes que pueden funcionar de forma simultánea. Los resultados presentan que el sistema permite decodificar una secuencia de 1920x1080 píxeles en tiempo real (30 FPS).

En [PGJS13] se presenta una implementación del código de referencia (HM 5.0) del decodificador HEVC sobre el DSP de Texas Instruments TMS320DM6437, el cual dispone de un solo núcleo en su arquitectura. Los autores presentan una comparativa entre el nuevo estándar y su predecesor H.264. Se utiliza una secuencia de resolución 352x288 píxeles y QP 27 para la cual se consigue una tasa de decodificación de hasta 14 FPS. Se debe tener en cuenta que en esta implementación no se han llevado a cabo optimizaciones de bajo nivel.

Finalmente, en [BBA⁺15] se realiza la implementación de un codificador H.264/AVC sobre la plataforma TMS320C6678. En este trabajo, investigadores de la Universidad de Sfax presentan un codificador HD capaz de funcionar en tiempo real (26 FPS) y una resolución de 1280x720 píxeles. Este resultado se consigue aprovechando los ocho núcleos de la plataforma y realizando la codificación a nivel de imagen de forma simultánea.

3.5. Comparativa entre las distintas implementaciones

Con el fin de tener una visión más global de las implementaciones vistas en este capítulo, se ha establecido un índice de calidad que permite comparar diferentes realizaciones. Mediante este índice, cuya expresión se muestra en la Fórmula 1, se normaliza la tasa de imágenes por segundo decodificadas, FPS, ajustando el número de núcleos a uno, la frecuencia de reloj del sistema a 1GHz y una resolución estándar de 1280x720 píxeles.

$$Indice = FPS * \frac{Resolución/(1280 * 720)}{N^{\circ}núcleos * Frecuencia\ de\ reloj}$$

Fórmula 1. Fórmula para calcular el índice de calidad de las implementaciones propuesto.

Comparar implementaciones llevadas a cabo sobre distintas plataformas con distintas versiones del *software* y secuencias de vídeo de diferente resolución y calidad siempre es complejo. Características relevantes de las plataformas, tales como las velocidades de las memorias, el tamaño de éstas, o el tiempo de desarrollo invertido, son importantes para una comparación detallada. Sin embargo, muchas veces los autores no facilitan toda la información en sus trabajos. Los datos que se resumen en la Tabla 6 representan una buena solución de compromiso entre los datos disponibles y la

calidad de la comparativa. Como se puede observar, las mejores puntuaciones se dan para aquellas implementaciones basadas en GPPs de altas prestaciones. Como se ha explicado anteriormente estas plataformas disponen de unas prestaciones muy elevadas, pero no resultan útiles en sistemas multimedia portátiles o en aquellos cuya fuente de energía sea una batería.

Referencia	Procesador	Núcleos	Frec. (GHz)	FPS	Resolución	Consumo (W/h)	SE	Índice
[YDSG12]	GPP Intel i5	2	3,4	27*	1080p	--	No	8,9
[DSY ⁺ 14]	GPP Intel i7	4	3,4	122*	1080p	--	No	20,1
[HRD14]	GPP Intel Xeon	6	3,2	84**	1080p	--	No	9,8
[HRD16]	GPP Intel i7	6	3,4	60**	4K	--	No	26,4
[JRS ⁺ 14]	GPP Intel Xeon	5/6	3,2	46	1080p	--	No	6,7
[NBP ⁺ 14]	GPP ARM Exynos	4/8	1,6	60	720p	1,1	Sí	9,3
[DSY ⁺ 14]	GPP ARM Cortex A9	2	1,2	35***	720p	0,5****	Sí	14,5
[BLRP14]	GPP ARM Cortex A9	2	1,2	33***	720p	0,5****	Sí	13,7
[PGJS13]	DSP 320DM6437	1	0,6	14	288p	--	Sí	2,5
[LSL ⁺ 15]	SoC específico	2	0,35	30	1080p	--	--	96,4

* *Secuencia Clase B BasketballDrive QP 27.*

** *Se incluye extensión SHVC y/o MV-HEVC.*

*** *Secuencia Clase C KristenAndSara QP 27.*

**** *Consumo medio del SoC.*

Tabla 6. Comparativa de las distintas implementaciones analizadas.

De entre todas las implementaciones destaca el elevado índice obtenido por la presentada en [LSL⁺15]. Esto se debe a que es una implementación diseñada específicamente para un SoC industrial (ASIC), donde se realiza un aprovechamiento elevado de los recursos de la plataforma mediante un desarrollo muy optimizado. Debe tenerse en cuenta que la metodología seguida en esta implementación es radicalmente diferente a la seguida en el resto de trabajos evaluados.

3.6. Resumen

A lo largo de este capítulo se han presentado los soportes tecnológicos programables sobre los que se vienen desarrollando descodificadores de vídeo. También se han presentado las particularidades de cada arquitectura desde un punto de vista general, contraponiendo sus respectivas ventajas y desventajas. Actualmente las implementaciones sobre plataformas tipo GPP de altas prestaciones tienen un uso mayoritariamente experimental y como prueba funcional, mientras que los GPPs tipo ARM han ido tomando una relevancia cada vez mayor acorde con el desarrollo de sistemas cada vez más potentes al tiempo que mantienen un consumo energético reducido. En ambos casos la utilización de procesadores multinúcleo está plenamente instaurada y su programación eficiente sigue siendo uno de los retos en desarrollo.

Por otro lado las plataformas tipo DSP o Multi-DSP siguen desempeñando una parte importante del procesamiento de vídeo, ya sea como co-procesadores aceleradores dentro de un sistema que tenga como centro un GPP, o ya sea formando parte de un SoC específico. En este sentido su programación supone un reto mayor que en las plataformas tipo GPP, no habiéndose encontrado en la literatura científica ninguna referencia a una metodología de diseño que sirva como guía. Este hecho refuerza las motivaciones planteadas al comienzo de este trabajo de investigación.

4. Metodologías y herramientas

Durante la última década, la comunidad científica ha dedicado un considerable esfuerzo a la investigación en metodologías y herramientas para la programación de sistemas multinúcleo. En este capítulo se hará un repaso de algunos de los trabajos de investigación más relevantes que se han desarrollado en los últimos años o que se están desarrollando en la actualidad.

Así, tras una breve introducción (apartado 4.1), en los apartados 4.2, 4.3 y 4.4 se describen los trabajos en los que se proponen metodologías y herramientas que permiten la automatización de los flujos de diseño de aplicaciones para chips multinúcleo. Finalmente, en el apartado 4.5 se describe la metodología y las herramientas utilizadas como punto de partida para el desarrollo de esta tesis.

4.1. Introducción

El diseño de aplicaciones destinadas a funcionar sobre plataformas multinúcleo plantea una serie de retos de diversa índole. En muchos casos la complejidad de los hitos a superar depende tanto de las dificultades planteadas por el, o los, sistemas sobre los que se quiere trabajar, como de las aplicaciones a las que se quiere dar soporte.

A lo largo de los últimos años se han ido desarrollando toda una amplia gama de sistemas multiprocesador para sistemas empuetrados con mucho éxito en el mercado debido a su relativa alta capacidad computacional, su bajo coste y un consumo energético reducido. Este tipo de sistemas se suman a aquellos de mayor recorrido histórico, como los que integran los sistemas de los ordenadores personales o PCs, y que ofrecen un rango más elevado de prestaciones, pero con un coste mayor y consumos energéticos elevados para ser utilizados en terminales portátiles.

Por otro lado, la tendencia comercial dominante durante la última década destaca por la continua salida al mercado de nuevos terminales con soporte multimedia, como *smartphones* o tabletas, integrados por sistemas empuetrados. Además, el diseño de las aplicaciones e interfaces de usuario para este tipo de plataformas ha visto acortados los tiempos de desarrollo debido a los exigentes plazos comerciales.

En este contexto, la búsqueda de metodologías que den soluciones al desarrollo de *software* compatible con los sistemas multiprocesador en tiempos cada vez más reducidos se ha convertido en el objetivo a alcanzar tanto por parte de la industria como por la academia.

4.2. Proyecto HEAP

El proyecto *Highly Efficient Adaptive multi-Processor framework* (HEAP) ha sido financiado por la Unión Europea (UE) y desarrollado entre los años 2010 y 2013 bajo la referencia FP7-ICT 247615 [HEAP] por parte de distintas empresas y universidades de la UE, entre las que destacan STMicroelectronics SRL (Italia), Thales (Francia), la Universidad Politécnica de Turín (Italia), o la Universidad de Genoa (Italia). Este proyecto surge con dos objetivos principales:

- 1) Diseñar un conjunto de herramientas que permitan a los programadores analizar y paralelizar secciones de código serie¹⁶ existente.
- 2) A partir de los datos obtenidos de distintos análisis, mejorar el aprovechamiento de las memorias del sistema, con especial énfasis en las memorias caché, permitiendo adaptar el uso que se hace de éstas en cada aplicación a desarrollar.

Al comienzo del desarrollo de este proyecto se fijaron unos objetivos concretos de mejora: (1) reducir un 20% el tiempo de desarrollo necesario en paralelizar una aplicación ya existente, (2) reducir en otro 20% el consumo de energía derivado de las operaciones del mantenimiento de la coherencia de memoria, y (3) mejorar el rendimiento de las memorias, asegurando la coherencia de los datos almacenados, en un 20%.

HEAP utiliza el compilador Compaan [COM] [KRD00] como herramienta central. Este compilador permite generar modelos de flujos de datos en base a Redes de Procesos Kahn (*Kahn Process Networks* – KPN) [Kah74] (ver apartado 4.5.1) a partir de especificaciones en C, además se reduce la complejidad del código mediante la descripción de *Static Affine Nested Loop Programs* (SANLP). Mediante el uso de SANLPs se pueden tomar decisiones sobre el control del flujo de datos y el acceso a variables en tiempo de compilación permitiendo optimizar la solución y obtener un mayor grado de paralelismo. A partir de un diagrama KPN el compilador Compaan asignará a cada vértice del grafo un hilo de ejecución y resolverá las dependencias de datos entre hilos mediante el uso de FIFOs.

Adicionalmente a lo anteriormente descrito, se ha desarrollado un conjunto de herramientas llamado *HEAP optimistic software parallelization toolset* que da soporte a la paralelización, dentro de este entorno, a aquellas aplicaciones que requieran de modificaciones dinámicas en el uso y localización de memorias, o donde el comportamiento de sus elementos funcionales no pueda ser descrito de forma estática en

¹⁶ El código serie se refiere a aquel cuya ejecución se va a realizar de forma secuencial por un único núcleo.

tiempo de compilación. Si bien esta herramienta accesoria complementa la funcionalidad de HEAP, debe ser el desarrollador de forma manual y en base a su experiencia, el que en estos casos extraiga el paralelismo de la aplicación.

A lo largo del proyecto se han realizado numerosas publicaciones en la literatura científica que pueden consultarse para tener un mejor conocimiento del mismo, todas ellas están disponibles en el sitio web del proyecto [HEAP], algunas de las más destacadas son [LLP⁺13], donde se presentan el conjunto de herramientas desarrolladas dentro del proyecto, y [RK12], donde se presenta un protocolo para el mantenimiento de la información almacenada en las memorias de los sistemas multinúcleo.

4.3. Proyecto ALMA

El proyecto ALMA (*Architecture oriented paraLlelization for high performance embedded Multicore systems using scilAb*) [GCV⁺13] ha sido desarrollado entre los años 2011 y 2015 por parte de varias entidades y centros de investigación de la UE, entre los que destacan Fraunhofer (Alemania), Universidad de Rennes 1 (Francia) o la Universidad del Peloponeso (Grecia) entre otros, y ha sido financiado por la UE según la referencia FP7-ICT 287733 [ALM].

El proyecto ALMA ofrece una solución para el desarrollo de aplicaciones sobre plataformas multinúcleo, abstrayéndose de la complejidad de las capas que representan las aplicaciones y las plataformas; esto se consigue mediante un conjunto de herramientas en torno al *software* Scilab¹⁷ [SCI]. Tanto los conceptos teóricos, como las pruebas realizadas a lo largo del proyecto han sido publicados en distintos medios de difusión científica, entre las que destacan [Str13] y [BOR⁺14]. En estos trabajos se profundiza en los detalles del proyecto, se introducen mejoras y se difunden los resultados.

El flujo de trabajo con ALMA se presenta de forma resumida en la Figura 20. Como se puede observar, a partir de una descripción de alto nivel de la aplicación en Scilab se genera una Representación Intermedia¹⁸ (*Intermediate Representation – IR*) de la misma, se llevan a cabo distintas optimizaciones y se trata de explotar el paralelismo de la arquitectura. Se han elegido dos plataformas como banco de pruebas de la metodología, por un lado KIT's KAHRISMA [KBS⁺10] con fines educativos o de investigación, y por el otro se ha elegido un SoC de *Recore Systems* [REC] para fines industriales y/o comerciales.

¹⁷ Scilab (*Scientific Laboratory*) es un software similar a MATLAB, de uso libre para el procesamiento matemático y que dispone de un lenguaje de programación de alto nivel.

¹⁸ Algunos de los conceptos utilizados en este apartado son comunes a las distintas metodologías estudiadas, aquellos compartidos con la metodología utilizada durante la realización de la tesis doctoral se explican con mayor detalle en apartados posteriores. En concreto los conceptos de Representación Intermedia (IR), *front-end*, *middle-end* y *back-end* se explican en el capítulo 8.

Figura 20. Diagrama del flujo de trabajo según el proyecto ALMA.

En el *front-end* de ALMA se ha incluido un perfilador (*profiler*) que realiza un primer análisis del *software* a nivel de bloques funcionales. Mediante este proceso, se realiza un primer particionado funcional que permite obtener una optimización de alto nivel. Este *profiler* puede ser utilizado en una segunda iteración para localizar puntos calientes o *hot-spots* donde deba llevarse a cabo una optimización más pormenorizada.

En cuanto a la generación de código fuente, ALMA contempla optimizaciones de alto nivel, al mismo tiempo que permite la utilización de instrucciones SIMD. Durante todo el proceso se dispone además de una descripción de alto nivel de la arquitectura, utilizada principalmente en la optimización del código a paralelizar, según la especificación *Architecture Description Language* (ADL) [SKB11]. Dicha descripción sigue un modelo jerárquico en el que se incluye el comportamiento de cada módulo disponible en la arquitectura.

4.4. PREESM

El *framework Parallel and Real-time Embedded Executives Scheduling Method* (PREESM) [PRE] es una herramienta de código abierto desarrollada por investigadores

del grupo IETR del INSA de Rennes (Francia) que facilita la simulación y generación de código fuente para sistemas empotrados y/o plataformas multinúcleo heterogéneas [PDH⁺14]. Mediante el uso de un lenguaje basado en flujos de datos permite extraer fácilmente el paralelismo de aplicaciones basadas en descripciones PiSDF (*Parametrized & Interfaced Synchronous Data-Flow*) [DPN⁺13]. PREESM está integrado por una serie de *plugins* para el IDE de Eclipse [IET14].

PREESM ha sido desarrollado con fines educativos y de investigación, y utilizado con éxito para la implementación de sistemas de telecomunicaciones, sistemas multimedia y aplicaciones de visión artificial sobre distintas plataformas heterogéneas [PAPN12] [HDN⁺12], entre otros.

Las aplicaciones basadas en Modelos de Computación (MoCs – ver apartado 4.5.1) y cuyos flujos de datos son de tipo PiSDF (ver apartado 4.5.1) permiten la división de una aplicación en actores, los cuales se comunican mediante FIFOs. Sin embargo, el comportamiento en la producción y consumo de *tokens* por parte de los actores puede modificarse mediante una serie de parámetros estáticos. De esta forma la descripción de los algoritmos que integran la aplicación sigue un modelo perfectamente predecible.

La Figura 21 muestra el flujo de trabajo con PREESM. Existen tres elementos sobre los que se basa este *framework*: una descripción gráfica de la aplicación (diagrama PiSDF), una descripción gráfica de la arquitectura sobre la que se desea trabajar según un modelo conocido como *System-Level Architecture Model*¹⁹ (S-LAM), y una descripción del comportamiento global en formato *xml*²⁰ llamada *Scenario* (Escenario). Este escenario contiene además la información que permite enlazar los algoritmos de la aplicación con la arquitectura, pudiendo determinar los tiempos de ejecución de cada uno de los actores en cada uno de los núcleos disponibles en la arquitectura.

Figura 21. Flujo de trabajo con PREESM.

¹⁹ S-LAM es un modelo de una arquitectura de alto nivel que permite estudiar los posibles cuellos de botella durante el procesamiento de algoritmos.

²⁰ xml de sus siglas en inglés *Extensible Markup Language* es un formato de texto flexible derivado de la norma ISO 8879.

4.5. Metodología basada en flujos de datos DPN

Esta es la metodología utilizada como guía para la realización de esta tesis doctoral. Las principales razones que motivan su elección en este trabajo son (1) que el estándar RVC tiene su foco de utilización sobre decodificadores de vídeo digital, y (2) que como se mencionaba en el apartado 2.3.5, varios grupos de investigación que colaboran con el GDEM vienen trabajando con este entorno desde hace tiempo, al igual que el propio doctorando, pudiendo aprovechar de esta manera la amplia experiencia ya adquirida.

Cabe destacar que debido a la relación de colaboración existente entre los investigadores del IETR que han desarrollado PREESM e investigadores del GDEM se valoró, durante la fase de desarrollo de esta tesis doctoral, la utilización de PREESM para el desarrollo de decodificadores de vídeo basados en flujos de datos y especificaciones RVC sobre plataformas Multi-DSP. Sin embargo, debido a que los flujos de datos en base PiSDF limitan la expresividad de los actores que integran la aplicación objetivo, numerosos actores que implementan la funcionalidad de los decodificadores en base RVC quedaban excluidos, principalmente aquellos encargados de ejecutar los algoritmos de predicción inter, haciendo imposible la compatibilidad directa entre este tipo de aplicaciones y PREESM.

Este hecho no anula la posibilidad de utilizar PREESM para explotar, de forma independiente, el paralelismo de ciertos actores, como los que integran la transformada inversa y cuyo comportamiento es determinístico o cuasi-estático.

4.5.1. Modelo de computación

Un Modelo de Computación (del inglés *Model of Computation* – MoC), es un conjunto de elementos que pueden utilizarse para describir el comportamiento de una aplicación. Éste debe servir como interfaz entre el campo matemático o algorítmico y la aplicación informática. No se debe confundir un MoC con un lenguaje de programación, ya que lo que hace es dar una especificación independientemente de la sintaxis del lenguaje. En realidad un lenguaje de programación puede implementar uno o varios MoCs.

Los MoCs permiten al desarrollador abstraerse de las capas más bajas de la implementación y centrarse en el desarrollo de aplicaciones de alto nivel, pudiendo así construir modelos no dependientes de la aplicación y para distintas plataformas.

Existen distintos tipos de MoCs, ya que pueden basarse en modelos tipo FSM [SGT10], mediante sucesión de eventos discretos o redes Petri [MMS94], entre otros. Sin embargo, a la hora de abordar la paralelización de aplicaciones sobre distintas arquitecturas, los MoCs más versátiles son aquellos que se basan en flujos de datos. En este sentido el primer MoC basado en un flujo de datos fue introducido en los años setenta [Kah74]. Desde entonces han surgido un gran número de modelos, que pueden clasificarse en función de dos factores principales: expresividad y grado de predicción. La expresividad de un modelo es inversamente proporcional a su grado de predicción,

es decir, a mayor diversidad de los algoritmos implementados e integrados en el flujo de datos menor será la posibilidad de poder predecir el comportamiento del modelo antes de su ejecución. La Tabla 7 resume las propiedades de los distintos modelos existentes.

	SDF	ADF	IBSDF	DSSF	PSDF	PiSDF	SADF	SPDF	DPN	KPN
Grado de expresividad	Bajo				Medio		Alto (<i>Máquina de Turing</i>)			
Jerárquico			✓	✓	✓	✓				
Compositivo			✓	✓		✓				
Reconfigurable					✓	✓	✓	✓	✓	✓
Planificación estática	✓	✓	✓	✓						
Determinable	✓	✓	✓	✓	*	*	✓	*		
Tasa variable		✓			✓	✓	✓	✓	✓	✓
No determinístico							✓	✓	✓	

SDF: Dataflow sincrónico

ADF: Dataflow afín

IBSDF: Dataflow basado en interfaz

DSSF: SDF determinístico con FIFOs compartidas

PSDF: SDF parametrizado

PiSDF: SDF parametrizado y con interfaz

SADF: Dataflow con escenario de aviso

SPDF: Dataflow paramétrico planificable

DPN: Dataflow de red de procesos

KPN: Red de procesos Kahn

Tabla 7. Propiedades de los distintos tipos de MoCs basados en flujos de datos.

En función del tipo de flujo de datos elegido la implementación de los actores o entidades funcionales que integran una aplicación tendrá distintas características. Para que un diagrama tenga un alto grado de predicción el sistema debe estar compuesto por actores cuyo funcionamiento sea determinístico, pudiendo conocer el comportamiento de todo el sistema antes de su ejecución, un ejemplo de este caso es el modelo SDF [LM87]; en este caso los actores producirán un número de *tokens* predecibles para un número de *tokens* consumidos. Por el contrario en los modelos como DPN [LP95] pueden darse situaciones no determinísticas, resultando un comportamiento no predecible. En estos casos el comportamiento global del sistema sólo puede observarse en tiempo de ejecución, incluso el comportamiento en sucesivas ejecuciones puede variar aun estimulando el sistema con las mismas variables.

Existen modelos intermedios, como PSDF o PiSDF que se sirven de modificaciones introducidas sobre las unidades funcionales que integran el modelo y que aumentan la expresividad del mismo. El modelo PiSDF es la base del conjunto de herramientas que conforman PREESM (ver apartado 4.4), y cuyas limitaciones ya fueron expuestas.

En este punto podría concluirse que los modelos estáticos, como SDF o IBSDF, ofrecen una ventaja considerable al poder predecir el comportamiento detallado de las entidades que lo integran, sin embargo, para que un diagrama cumpla los requisitos de este tipo de modelos todos los actores deben tener un comportamiento estático, reduciendo de esta manera el grado de funcionalidad de los mismos. Así, bajo estos modelos, la implementación de ciertas unidades funcionales, como por ejemplo los bloques de predicción en un descodificador de vídeo digital no resulta factible, ya que los algoritmos que deben implementarse pueden tomar diferentes decisiones para un mismo parámetro de entrada.

Así pues la elección del MoC dependerá en gran medida de la funcionalidad que se desee expresar mediante el mismo. También deberán tenerse en cuenta las herramientas que den soporte a cada modelo, la facilidad de acceso a las mismas y su grado de desarrollo. El uso de una herramienta específica para cada MoC es clave para poder extraer el mayor grado de optimización posible de cada uno de ellos, si bien herramientas como Orcc, diseñada para modelos DPN, podrían dar soporte a modelos con un menor grado de expresividad.

El estudio e implementación de aplicaciones con los distintos tipos de flujos de datos es un campo de análisis amplio y excede los objetivos de esta memoria, para un mejor entendimiento de la temática se recomienda acudir a la literatura científica donde puede ampliarse esta información. En [CHL97] se detalla la utilización de distintos MoCs en la herramienta Ptolemy, y en [Kah74] se introducen los modelos KPN para MoCs.

4.5.2. Estrategias de planificación

Una vez se elige el modelo de computación (DPN en este caso) y se implementan las distintas unidades funcionales que integran la aplicación se debe gestionar la ejecución de las mismas. En el caso de los descodificadores de vídeo desarrollados con RVC se especifican tres niveles de planificación (o *scheduling*²¹) que permiten gestionar todo el funcionamiento de un descodificador.

4.5.2.1. Planificación a nivel de acciones

En primer lugar se encuentra la planificación interna de los actores. Este concepto se introdujo en el apartado 2.3.3, donde se presentaba el funcionamiento interno de un actor, gobernado por una FSM que planifica la ejecución de las distintas acciones que lo componen en función de una serie de condiciones de guarda, prioridades y variables de estado interno. A este nivel de planificación se le denomina planificación de acciones o *action scheduler*. La Figura 22 ilustra este mecanismo, donde en función de los datos de entrada (lectura) y del estado interno del actor, el planificador de acciones ejecutará aquellas acciones que consuman los datos de la FIFO de entrada, produciendo *tokens* en la FIFO de salida.

²¹ A lo largo de este apartado se facilita la nomenclatura en lengua inglesa de algunos conceptos claves para la comprensión de las implementaciones realizadas y que se utilizan en capítulos posteriores.

Figura 22. Planificación a nivel de acciones.

4.5.2.2. Planificación a nivel de actores

El siguiente nivel de planificación es el que se refiere a la gestión de la ejecución de los actores que integran una aplicación. Normalmente serán varias decenas los actores que conformen una aplicación, por lo que es preciso prestar atención al orden en que los actores pasan a ejecución. Un actor ejecutará alguna de sus acciones si en sus FIFOs de entrada hay *tokens* disponibles y en sus FIFOs de salida hay espacio libre. De esta forma la ejecución de los actores dependerá en gran medida de la ejecución y producción de *tokens* de sus actores precedentes. Sin embargo la ventaja de este modelo de gestión reside en que en todo momento habrá un actor en ejecución, eliminando así innecesarios estados de espera o tiempos muertos durante el procesado. Las FIFOs deben conectarse a los actores respetando que sólo un actor escriba en cada FIFO, mientras que varios pueden leer de la misma FIFO. La Figura 23 muestra un ejemplo de un diagrama DPN con varios actores interconectados mediante FIFOs.

Figura 23. Ejemplo de diagrama DPN con cinco actores interconectados mediante FIFOs.

Existen dos estrategias de planificación de la ejecución de los actores: *Round Robin* (RR) y *Data-driven/Data-demand* (DD/DD). La estrategia RR consiste en la ejecución secuencial de los actores, en este caso el disparo de las acciones dependerá únicamente del cumplimiento de las condiciones del *planificador de acciones* y de los

datos en las FIFOs. En el caso DD/DD la planificación de los actores a ejecutar se realiza basándose en la demanda de datos en las entradas y salidas de los mismos. De esta forma, la planificación se lleva a cabo ejecutando los actores predecesores productores de *tokens* y requeridos por los sucesores. De la misma forma si la salida de un actor está llena se llamará al actor sucesor para que consuma los *tokens* que están disponibles. La Figura 24 ilustra ambas estrategias de planificación.

Figura 24. Estrategias *Round-Robin* (izquierda) y *Data-driven/Data-demand* (derecha).

En ambas estrategias de planificación no será necesario almacenar información relativa a la ejecución de los actores, ya que toda la planificación depende de los datos transmitidos mediante las FIFOs. Esto proporciona una ventaja fundamental en arquitecturas multinúcleo y permite reducir el coste computacional de las implementaciones. Para el desarrollo de esta tesis se ha utilizado en todo momento la estrategia *Round Robin*. La utilización de la estrategia RR viene motivada por los siguientes factores: ésta es la estrategia utilizada por defecto en las implementaciones que utilizan el estándar RVC, lo que facilita una comparativa posterior de los resultados con otras implementaciones. Por otro lado, la estrategia DD/DD conlleva un aumento en la complejidad del planificador de actores, así como la necesidad de asegurar la coherencia de la información compartida entre los planificadores.

4.5.2.3. Planificación a nivel de procesadores

Con el fin de evitar posibles situaciones de bloqueo de este tipo de topologías sobre entornos multinúcleo hay un tercer nivel de planificación. En este caso la información a transmitir será un listado de los propios actores que deben pasar a ejecución por parte de algún procesador, bien porque sea preciso que consuma *tokens* disponibles en sus entradas o bien porque sus actores sucesores están esperando información en sus respectivas entradas. De darse una situación como ésta en un entorno multinúcleo los planificadores encargados de controlar la ejecución de los actores en cada núcleo deberán poder compartir información relativa a la ejecución de los mismos. Así, se diseña un nuevo tipo de canal FIFO llamado *scheduling-fifo* (FIFOs de planificación), el cual se utilizará únicamente para transmitir este tipo de información. La Figura 25 ilustra este tercer nivel de planificación mediante un diagrama con dos núcleos y sus respectivos planificadores.

Figura 25. Ilustración de la planificación a nivel de procesadores y actores.

4.5.3. Aplicación de estrategias de planificación sobre plataformas multinúcleo

Como ya se ha explicado antes (ver apartados 2.3 y 4.5.1), una de las principales ventajas de los modelos basados en flujos de datos es la facilidad de implementar éstos sobre plataformas multinúcleo. En este sentido resulta clave el hecho de que las variables propias de cada actor no requieren su utilización por parte de los restantes actores, y que la única información compartida se transmite y aloja en FIFOs. De esta manera se puede ejecutar una aplicación integrada por un número determinado de actores organizados y planificados en listas. En entornos multiprocesador se asignará una lista de actores a cada uno de los núcleos disponibles en la plataforma, quedando así cada una de las listas asociadas a un hilo de ejecución (*thread*). Al proceso de repartir la carga de los actores entre los procesadores de una plataforma se le llama mapeo (*mapping*).

Para llevar a cabo la ejecución en paralelo de actores sobre plataformas multinúcleo se deben cumplir dos condiciones fundamentales: que las FIFOs estén alojadas en memoria compartida accesible por parte de aquellos actores que leen o escriben en ellas, y que cada actor quede mapeado en un núcleo, según la expresión de la Fórmula 2.

$$(Actores \text{ en núcleo } 0) \cup (Actores \text{ en núcleo } 1) \cup (Actores \text{ en núcleo } N) = \{\emptyset\}$$

Fórmula 2. Fórmula a cumplir al realizar un mapeo de actores.

Numerosos factores deben tenerse en cuenta a la hora de realizar un mapeo de actores entre los núcleos de un sistema multinúcleo; cuanto mejor sea el mapeo de las unidades funcionales mayor ganancia se obtendrá con respecto a un sistema mono-núcleo.

- *Peso del actor* o carga computacional derivada de la ejecución del mismo. Idealmente un mapeo debe resultar en que la suma de las cargas computacionales de los actores de la lista para cada procesador quede balanceada entre todos los núcleos. Por ejemplo en una distribución para cuatro núcleos el sumatorio del peso de los actores mapeados en cada núcleo debería suponer el 25% de la carga computacional total de la aplicación. Sin embargo, en un MoC dinámico, como es el caso de DPN, esta información sólo se puede obtener en tiempo de ejecución.
- *Paralelismo de los actores*, la funcionalidad que se integra dentro de cada actor tendrá un mayor o menor grado de paralelismo con las acciones del resto de actores. De esta manera se debe buscar la distribución entre los procesadores de aquellos actores cuya funcionalidad se puede realizar de forma paralela a otras tareas, evitando al máximo ejecuciones secuenciales de actores.
- *Reducción del número de comunicaciones entre núcleos*. El peso de las comunicaciones entre actores mapeados en distintos núcleos debe reducirse. Dado que habitualmente la información compartida entre actores ubicados en diferentes núcleos debe alojarse en memorias compartidas, de menor velocidad que las memorias cachés de los procesadores, deberán reducirse, en la medida de lo posible, este tipo de comunicaciones.
- *Reducción del volumen de las comunicaciones*. En relación con la anterior consideración también resulta importante la carga relativa de cada una de las comunicaciones. Se debe tener en cuenta que no todas las FIFOs tienen la misma actividad, aquellas que se utilicen para transmitir un mayor número de datos deben considerarse las primeras candidatas a ubicarse en las memorias más rápidas.

Debido a estas consideraciones la realización de mapeos óptimos en sistemas multinúcleo no resulta una tarea trivial, aumentando su complejidad al aumentar el número de núcleos para los que se desea paralelizar una aplicación. En este sentido la experiencia y los conocimientos del programador son claves para la realización de mapeos que permitan explotar al máximo el paralelismo de un sistema. A este tipo de mapeos se les denomina mapeos estáticos.

4.5.4. Mapeo dinámico de actores en plataformas multinúcleo

Recientemente se han desarrollado sistemas de mapeo dinámico, los cuales se sirven de distintos algoritmos que realizan distribuciones de actores a partir de medidas que se van almacenando en tiempo de ejecución. La base del funcionamiento es la toma de medidas de los parámetros que se identifican con los cuatro factores descritos en el apartado anterior. Así, cada cierto número de imágenes descodificadas un algoritmo propone un mapeo a partir de los datos registrados. Dado que el coste computacional de la ejecución del algoritmo no debe suponer una sobrecarga excesiva sobre el proceso de descodificación del vídeo, en la práctica, los algoritmos están diseñados para optimizar solamente alguno de los cuatro factores descritos.

El diseñador de la aplicación es el encargado de seleccionar los parámetros de configuración necesarios así como de elegir entre los algoritmos de mapeo dinámico disponibles en función del más ventajoso para la arquitectura objetivo. En el caso de los decodificadores basados en RVC y cuyo código fuente vaya a generarse con Orcc (ver capítulo 8) están disponibles distintos algoritmos de mapeo dinámico con distintos parámetros configurables [YCR⁺14]. La Tabla 8 muestra los distintos parámetros que se deben configurar antes de iniciar el proceso de decodificación para estas soluciones, y la Tabla 9 presenta los distintos algoritmos disponibles y el objetivo de su optimización.

Parámetro	Utilidad	Parámetro	Utilidad
-c	Número de procesadores disponibles	-s	Algoritmo a aplicar en mapeos dinámicos
-r	Número de imágenes a decodificar antes de realizar un mapeo	-a	Realizar mapeos dinámicos de forma indefinida cada r imágenes
-q	Guardar en un fichero el mapeo dinámico generado por el algoritmo	-p	Guardar en un fichero los datos de <i>profile</i> de una ejecución

Tabla 8. Relación de parámetros de configuración con mapeos dinámicos.

Id.	Significado	Objetivo del algoritmo	Coste computacional
RR	<i>Round Robin</i>	Mapeo simple RR	Bajo
WLB	<i>Weighted Load Balancing</i>	Optimizar la distribución de la carga de los actores	Medio
QM	<i>Quick Mapping</i>	Reducción de las comunicaciones manteniendo un coste computacional reducido	Muy bajo
MR	<i>METIS [MET] Recursive graph partition</i>	Realiza varias subdivisiones del grafo repartiendo la carga en cada una	Medio
MKCV	<i>METIS K-Way graph partition (Communication Volume)</i>	Optimizar el número de comunicaciones	Alto
MKEC	<i>METIS K-Way graph partition (Edge-Cut)</i>	Optimizar el peso de las comunicaciones	Alto

Tabla 9. Algoritmos de mapeo dinámico.

La Figura 26 resume el proceso de decodificación de vídeo en un entorno multinúcleo haciendo uso de las técnicas de mapeo dinámico descritas. En este caso el proceso de decodificación comienza leyendo los parámetros de configuración (c , r , s y a) y arrancando la decodificación con un único núcleo. Durante todo el proceso de decodificación se van almacenando medidas relativas a la carga computacional de los actores o la utilización de las FIFOs. Transcurridas un número r de imágenes decodificadas se calcula el mapeo en base al número de núcleos c y al algoritmo s , indicados al inicio y se reanuda la decodificación. Si se ha especificado el parámetro a al inicio se aplicarán sucesivos remapeos cada r imágenes decodificadas, en caso contrario la decodificación continuará de forma indefinida con el primer mapeo.

Figura 26. Diagrama del proceso de decodificación al utilizar mapeo dinámico.

Una desventaja adicional a la sobrecarga introducida al utilizar sistemas de mapeo dinámico es la necesidad de utilizar un hilo de ejecución extra encargado de realizar el procesamiento del algoritmo seleccionado; la tarea encargada de realizar dicho procesamiento es *agent_routine*. En sistemas de altas prestaciones esto no supone un problema ya que mientras la decodificación del vídeo está detenida²² el SO permite la ejecución del hilo asociado al algoritmo en un mismo procesador. Sin embargo sobre plataformas Multi-DSP esto no es posible ya que el RTOS (ver capítulo 7) no es capaz de implementar esta funcionalidad. En el apartado 9.5 se detalla la integración de este tipo de sistemas en el trabajo de investigación realizado.

Cabe destacar que el hecho de indicar un número determinado de núcleos disponibles para el mapeo dinámico no implica que todos ellos vayan a ser utilizados en todo momento. Los algoritmos de distribución automática de la carga computacional pueden dejar temporalmente libre uno o varios de los núcleos indicados.

4.5.5. Limitaciones de la metodología

La principal limitación de los modelos DPN reside el comportamiento no determinístico de los actores que implementan la funcionalidad de las aplicaciones. Esto impide conocer por adelantado el comportamiento de la aplicación ya que éste depende de los datos que tenga que procesar. En sistemas estáticos, como SDF, es posible realizar distribuciones óptimas de las unidades funcionales antes incluso de compilar el código, puesto que su comportamiento es completamente determinista (un ejemplo de modelo SDF es el que permite desarrollar la herramienta PREESM). En modelos DPN es necesario tomar decisiones en tiempo real, como las presentadas para la implementación de mapeos dinámicos (ver apartado 4.5.4).

4.6. Resumen

A lo largo de este capítulo se han presentado, en primer lugar, algunas de las metodologías desarrolladas recientemente en el campo de la programación de

²² Para parar y sincronizar correctamente los procesos de decodificación y cálculo de mapeo se utilizan semáforos.

plataformas multinúcleo. En estas metodologías siempre se busca la abstracción de los detalles de bajo nivel de las arquitecturas objetivo, con el fin de proporcionar las herramientas que permitan al desarrollador explotar el paralelismo de las aplicaciones en entornos multinúcleo.

La metodología que se propone en esta tesis doctoral se apoya en el compilador Orcc (ver capítulo 8), una herramienta abierta de programación paralela basada en flujo de datos con un modelo de computación DPN. En el capítulo se discuten las ventajas de esta opción frente a otras en el contexto de la tesis. También se hace una descripción más detallada de las metodologías basadas en flujo de datos y particularmente del modelo de computación DPN.

TERCERA PARTE: Tecnologías de referencia

En esta tercera parte de la memoria se resumen las distintas tecnologías utilizadas en la fase de desarrollo de la tesis doctoral. Se introducen los elementos que las integran y los conceptos necesarios para la comprensión de los siguientes capítulos de la memoria.

El capítulo 5 presenta el *software* de decodificación de vídeo digital para el estándar HEVC denominado OpenHEVC. El capítulo 6 introduce los conceptos principales de la API de OpenMP que da soporte al desarrollo de aplicaciones en entornos multinúcleo. En el capítulo 7 se detallan las plataformas multinúcleo utilizadas durante el desarrollo de la tesis. Por último, el capítulo 8 presenta la infraestructura del compilador Orcc que materializa la metodología expuesta en la sección 4.5 y da soporte a las implementaciones basadas en el estándar RVC introducido en el apartado 2.3.

5. OpenHEVC

En este capítulo se introduce el *software* de decodificación OpenHEVC [OHE]. OpenHEVC es un *software* de código abierto conforme a la norma HEVC que está siendo desarrollado por distintos grupos de investigación y se encuentra disponible para la comunidad científica.

En el apartado 5.1 se introduce OpenHEVC, explicando brevemente sus orígenes y el entorno necesario para ponerlo en marcha. Seguidamente, en el apartado 5.2 se explica brevemente el funcionamiento del decodificador OpenHEVC. En el apartado 8.4 del capítulo 8 se explica cómo puede utilizarse el código de OpenHEVC en combinación con descripciones basadas en RVC-CAL para mejorar rendimiento final de los decodificadores cuyo código ha sido generado de forma automática.

5.1. El descodificador OpenHEVC

OpenHEVC es una implementación de un descodificador HEVC escrita en lenguaje C y su desarrollo forma parte del proyecto libre y con fines de investigación Libav [LAV]. El descodificador es compatible con 16 de los 21 perfiles de HEVC (hasta un muestreo 4:4:4 con 12 bits) e incluye en sus últimas versiones las extensiones SHEVC [ISO12] y MV-HEVC [TWCH⁺14]. El código que integra el descodificador está optimizado para GPPs de Intel [INT] o ARM [ARM]. El código de OpenHEVC se encuentra disponible en la plataforma de desarrollo colaborativo *GitHub* [OHEVC]. Además, OpenHEVC se utiliza en proyectos como 4EVER [4EV], H2B2VS [H2B] o VLC [VLC].

Como se ha mostrado en los apartados 3.2 y 3.3 existen diversas implementaciones basadas en este descodificador, las cuales han ofrecido buenos resultados en cuanto al rendimiento proporcionado sobre GPPs tipo Intel y ARM. El elevado rendimiento conseguido se debe principalmente al alto grado de optimización del código fuente del descodificador y a la utilización de instrucciones SIMD del repertorio SSE para la familia de procesadores x86 de Intel. Todos los ARMs con tecnología Neon (a partir de ARM Cortex 7) disponen de instrucciones SIMD, pero no el mismo repertorio SSE que x86, por lo que estas optimizaciones deben adaptarse al repertorio correspondiente del procesador.

El entorno de trabajo de OpenHEVC está preparado para que, tras la descarga del código del descodificador, el diseñador pueda hacer uso del mismo en un corto espacio de tiempo. Además de un compilador nativo a la plataforma donde se vaya a hacer funcionar el descodificador, serán necesarias las bibliotecas SDL²³ o SDL2 y la herramienta *yasm*²⁴. El conjunto de archivos que componen el *software* de OpenHEVC incluye un fichero de configuración del entorno, el cual debe ser revisado, y en su caso modificado, antes de compilar el código. Mediante este fichero se puede realizar una configuración de prácticamente todos los aspectos que afectan tanto al descodificador como a la plataforma donde se vaya a probar el *software*.

Una vez las herramientas y extensiones están disponibles en la plataforma, y el código ha sido compilado se puede utilizar el descodificador HEVC. En la fecha de redacción de esta memoria el descodificador OpenHEVC soportaba las características descritas en la Tabla 10.

²³ SDL (*Simple DirectMedia Layer*): Biblioteca que permite el acceso del *software* a recursos de bajo nivel como el audio y el *hardware* de gráficos para la reproducción del vídeo.

²⁴ La herramienta Yasm es un ensamblador para la arquitectura x86 de Intel.

Característica	Soporte en OpenHEVC
Perfiles	Desde <i>Main</i> hasta <i>Main 4:4:4 12</i> , y <i>Scalable Main</i> .
Configuración de la codificación (secuencias)	<i>All-Intra</i> , <i>Low-Delay</i> y <i>RandomAccess</i>
Procesamiento paralelo	<i>Tiles</i> , <i>Slices</i> y <i>WPP</i> .
Tamaño de Cus	64, 32, 16 y 8
Tamaño de Tus y Pus	32, 16, 8 y 4
Tamaño de Pus Inter	64, 32, 16 y 8

Tabla 10. Relación de características soportadas por OpenHEVC.

5.2. Funcionamiento de OpenHEVC

A continuación se realiza un breve resumen del funcionamiento del decodificador OpenHEVC. Dado el elevado número de archivos y funciones que conforman el decodificador aquí se presentará únicamente su estructura de alto nivel.

La decodificación de una secuencia de vídeo codificada conforme al estándar HEVC con el decodificador OpenHEVC comienza con la extracción de las cabeceras NAL de la trama. Una vez éstas han sido identificadas se inicia la decodificación de cada una de las imágenes codificadas en la secuencia. Cada vez que una imagen es decodificada se reinicia el proceso. La Figura 27 muestra el diagrama de flujo seguido durante la decodificación de las *slices* que componen las imágenes codificadas con el decodificador OpenHEVC, esta información ha sido extraída de [RNH⁺16], donde puede consultarse con mayor detalle.

Figura 27. Diagrama del funcionamiento del decodificador OpenHEVC.

Para la decodificación de cada *slice* se llama de forma recursiva a la función *hls_decode_row*, que inicia la decodificación a nivel de CTU. Seguidamente se extraen las CUs de cada CTU (*hls_coding_tree*), y se lleva a cabo la transformada inversa (*hls_transform*) y se realizan las predicciones correspondientes (intra/inter). Cuando todas las CUs de la CTU han sido decodificadas se aplican los filtrados antibloques

(*hls_deblocking_filter_ctb*) y SAO (*hls_sao_filter_ctb*), en caso de haberse utilizado en la codificación.

5.3. Resumen

En este capítulo se han resumido las principales características del *software* OpenHEVC, una implementación de código abierto en lenguaje C de un descodificador HEVC. El interés por este *software* reside en la posibilidad de incorporar secciones de código del mismo a las descripciones RVC-CAL de los descodificadores para generar código más eficiente.

6. OpenMP

En este capítulo se presenta la Interfaz de Programación de Aplicaciones (*Application Programming Interface* – API) de *Open Multi-Processing* (en adelante OpenMP). Esta ha sido una de las herramientas centrales en el desarrollo de la tesis doctoral, ya que mediante su uso se han conseguido implementar las distintas versiones de un decodificador HEVC sobre distintos tipos de plataformas, incluyendo una plataforma Multi-DSP. OpenMP permite la paralelización de aplicaciones escritas en C, C++ o Fortran en sistemas que disponen en su arquitectura de memoria compartida. Los conceptos de OpenMP que se van a describir son los más relevantes para esta tesis doctoral, para una consulta detallada sobre la API de OpenMP se recomienda acudir a la referencia [Ope03].

En el apartado 6.1 se introduce brevemente el origen histórico de OpenMP. En los apartados del 6.2 al 6.5 se presentan los fundamentos y elementos principales necesarios para trabajar con OpenMP. En el apartado 6.6 se realiza una breve discusión acerca de las ventajas e inconvenientes de utilizar esta herramienta. Finalmente, en el apartado 6.7, se discuten dos alternativas a la utilización de OpenMP sobre plataformas Multi-DSP.

6.1. Origen de OpenMP

La API de OpenMP [OMP] se crea en el año 1997 por parte de la OpenMP ARB (*Architecture Review Board*), la cual está formada por miembros de diversas empresas del sector de las telecomunicaciones y la electrónica como ARM, Fujitsu, IBM, Intel o Texas Instruments entre otros. La OpenMP ARB es una corporación sin ánimo de lucro que se encarga del mantenimiento y publicación de nuevas versiones de OpenMP, además de organizar las conferencias y eventos relacionados con el consorcio.

A finales de los años 80 surgen los sistemas SMP, (*Symmetric Multiprocessor System* – Sistemas Multiprocesador Simétricos) cuya arquitectura se basa en una memoria compartida por varios núcleos mediante un bus de datos común, con ello surgen los primeros problemas a la hora de gestionar un sistema multiprocesador. El programador de una aplicación que se ejecute en paralelo debe asegurarse de que el acceso a las variables compartidas por varios procesos en ejecución simultánea se realiza correctamente. Además, se deben tener en cuenta posibles diferencias en la sincronización del reloj del sistema de cada procesador, lo que provocaría diferencias en

la velocidad de ejecución por parte de cada núcleo (en la mayoría de sistemas multiprocesador actuales el reloj del sistema es común a todos los procesadores al encontrarse encapsulados en el mismo chip, evitando así este problema).

Las soluciones a estos problemas fueron una serie de instrucciones especiales, o directivas, que se podían añadir al código de los programas y que permitían al compilador generar el código de forma correcta para cada procesador. El problema radicaba en que cada fabricante desarrolló sus propias herramientas para dar solución a sus problemas particulares, no habiendo ningún tipo de estandarización. Con el ánimo de dar solución a esta situación surge el OpenMP ARB a comienzos de los años 90.

OpenMP es una herramienta que permite al programador abstraerse de la gestión de la sincronización y la concurrencia en un sistema multiprocesador mediante el uso de directivas o sencillas sentencias multiplataforma que describen el comportamiento del código paralelizado. Cabe destacar que OpenMP permite la ejecución simultánea de código ya existente mediante un proceso de paralelización incremental, de esta forma puede trabajarse de forma independiente sobre distintas secciones de código sin necesidad de abordar de una sola vez una modificación global.

OpenMP viene utilizándose en grandes proyectos como Altair RADIOSS, MATLAB (NaN y TSA *toolboxes*) o MACROS, todos ellos disponibles en el sitio web de OpenMP [OMP], entre otros. Además, distintos trabajos publicados en la literatura científica complementan el uso de esta API. Así, en [LKR10] investigadores de la Universidad de Pittsburgh presentan un estudio sobre la utilización de forma periódica de la técnica *fork/join*, y en [NMQC14] investigadores de la Universidad de Houston evalúan el impacto de la utilización de la API de OpenMP sobre el consumo energético sobre un procesador Intel.

A lo largo de los siguientes apartados se presentan los distintos elementos que se integran en la API de OpenMP, mediante los cuales se implementa la paralelización de secciones de código de una aplicación.

6.2. Ejecución *fork/join* de OpenMP

OpenMP está basado en el modelo de programación llamado *fork/join* (bifurcación/unión), como se ilustra en la Figura 28. Según este modelo la aplicación comienza con la ejecución de código secuencial por parte de un único hilo (hilo maestro). En cualquier momento en el que un hilo llega a una región de código a ejecutar de forma paralela se produce un *fork*, o bifurcación, inmediatamente después varios hilos (equipo de hilos – *team of threads*) ejecutan simultáneamente la región de código especificada en la región paralela, hasta que ésta termina y se produce el *join*, o unión.

Figura 28. Diagrama de la ejecución *fork/join* de OpenMP.

Las regiones de código paralelizado deben identificarse y parametrizarse mediante el uso de las distintas directivas incluidas en la API. A continuación se describen los distintos componentes que forman parte de OpenMP.

6.3. Componentes de OpenMP

Existen tres componentes integrados en OpenMP que permiten al desarrollador realizar las descripciones de alto nivel deseadas. A continuación se enumeran todos ellos:

- **Directivas:** las directivas empleadas en OpenMP permiten básicamente definir el paralelismo de una aplicación. Las directivas deben incluirse en el código de una aplicación para que el compilador proceda a interpretarlas y generar el código para tantos hilos como se hayan definido previamente. En el caso de código escrito en C o C++, todas las directivas se expresan mediante *pragmas*, sentencias especiales que permiten controlar el comportamiento del compilador; naturalmente, es necesario que el compilador utilizado soporte OpenMP. Todas las directivas de OpenMP deben llevar el identificador *omp* tras el *pragma*, así como el nombre de la directiva y los parámetros asociados a ésta si los hubiese. Mediante las distintas directivas de OpenMP se pueden indicar las regiones de código a ejecutar en paralelo así como los parámetros necesarios que controlen su ejecución. La Figura 29 muestra el modelo de una directiva OpenMP en lenguaje C y en la siguiente subsección se listan las directivas de OpenMP. Además, el comportamiento de las directivas se puede complementar mediante el uso de cláusulas (ver apartado 6.5).

```
#pragma omp directiva [cláusula, cláusula [] ... ]
```

Figura 29. Modelo de directiva OpenMP.

- **Funciones de biblioteca:** las funciones de biblioteca son otro elemento a utilizar en una aplicación con código paralelizado mediante OpenMP. Mediante estas funciones se controlan parámetros tales como el número de hilos que OpenMP

puede gestionar o el control de ciertos elementos de sincronización como *mutex* (cerrojos).

- Variables de entorno: mediante las variables de entorno de OpenMP se puede controlar el comportamiento de las regiones paralelas en tiempo de ejecución, el tipo de planificación (*scheduler*), o ajustar dinámicamente el número de hilos al llegar a un *fork*. Cabe destacar que algunos de los parámetros introducidos como variables de entorno pueden ser modificados dinámicamente por ciertas funciones de la biblioteca.

6.4. Directivas y funciones de la biblioteca OpenMP

A continuación se presenta una relación de las principales directivas de OpenMP disponibles para la programación de aplicaciones en lenguaje C/C++.

A la hora de abordar la paralelización del código de una aplicación el primer paso es identificar aquellas secciones de código que van a ejecutarse en paralelo por parte de varios hilos, y qué secciones se ejecutarán en serie por parte de un único hilo. A aquellas secciones de código paralelizables se las denomina regiones paralelas y deben estar definidas por la directiva *parallel* de OpenMP. La Figura 30 muestra la declaración y ejecución de una región paralela mediante las correspondientes directivas de OpenMP en lenguaje C. En este ejemplo, tres núcleos ejecutarán, de forma simultánea, la sentencia *printf*, imprimiendo por pantalla el número de hilo en ejecución. Como se observa en el ejemplo, se deben indicar el número de hilos que van a realizar la ejecución de la región paralela. En el caso de la Figura 30, se hace uso de la función de biblioteca *omp_set_num_threads(n)* mediante la que se puede fijar este parámetro en tiempo de ejecución. Otra opción que ofrece OpenMP es crear una variable de entorno en el sistema denominada *OMP_NUM_THREADS* que contenga el valor de núcleos que se deseen utilizar en una región paralela.

```
Int main(void){
    omp_set_num_threads(3);           //Se define el número de hilos
    #pragma omp parallel               //Inicio de la región paralela
    {
        int n;                        //Variable privada
        n = omp_get_thread_num();
        printf("Hola mundo soy el hilo %d.\n",n);
        return 0;
    }                                 //Fin de la región paralela
```

Salida por la terminal:

```
Hola mundo soy el hilo 0.
Hola mundo soy el hilo 1.
Hola mundo soy el hilo 2.
```

Figura 30. Ejemplo de declaración y ejecución de una región paralela con OpenMP.

Toda región paralela queda delimitada por llaves que deben abrirse inmediatamente después de la directiva *parallel* y, tras el cierre de las mismas, la ejecución continuará de forma secuencial en un solo hilo. Como se ha observado en el

ejemplo de la Figura 30, el código ejecutado por los tres hilos es el mismo, OpenMP permite identificar cada uno de los hilos que están ejecutándose en una región paralela mediante la función de biblioteca `omp_get_thread_num()`; la cual devuelve el identificador en una variable de tipo entero.

OpenMP ofrece tres directivas principales para gestionar la ejecución de una región paralela, éstas son *for*, *section* y *single*. La Tabla 11 muestra sus particularidades y un pequeño ejemplo que ilustra cómo se utilizan.

	<i>For</i>	<i>Section</i>	<i>Single</i>
Descripción	Permite paralelizar la ejecución de un bucle de tipo <i>for</i> .	OpenMP permite la ejecución simultánea de distintas secciones de código mediante esta construcción.	Dentro de una región paralela se puede indicar, mediante la directiva <i>single</i> , que cierta sección de código se ejecute por un único hilo
Limitaciones o consideraciones	Deben conocerse el número de iteraciones del bucle. No se permiten sentencias del tipo <i>break</i> , <i>continue</i> o <i>goto</i> .	Todos los hilos esperan al resto al terminar la ejecución de su sección. Si el número de secciones es mayor que el de núcleos, se fuerza la ejecución de varias secciones por un hilo.	OpenMP introduce una barrera de sincronización al final de la sección <i>single</i> . Existe una versión <i>master</i> de la directiva en la cual la sección <i>single</i> es ejecutada por el hilo maestro.
Ejemplo	<pre>omp_set_num_threads(n); [...] #pragma omp parallel for for(i=0;i<Index; i++) var[i] = var[i] + 1; [...]</pre>	<pre>#pragma omp parallel { #pragma omp sections { #pragma omp section [...)//Código hilo 1 #pragma omp section [...)//Código hilo 2 #pragma omp section [...)//Código hilo n } } //Fin de las secciones //Fin de región paralela</pre>	<pre>#pragma omp parallel { #pragma omp single {...)//Código ejecutado de forma serializada } [...)//Código ejecutado por varios hilos } //Fin de región paralela</pre>

Tabla 11. Directivas de OpenMP para la gestión de regiones paralelas.

Sin embargo, las posibles dependencias de datos (por ejemplo en la ejecución de un bucle *for*) en el acceso a variables compartidas dentro de una región paralela será responsabilidad del programador.

Finalmente, cabe destacar que por defecto todas las variables declaradas antes del comienzo de una región paralela serán tratadas como variables compartidas por parte de todos los hilos, y serán alojadas en memoria compartida. Así mismo, todas las variables declaradas dentro de una región paralela quedarán definidas por defecto como privadas, en este caso cada hilo tendrá una copia local de la variable no accesible por parte del resto de hilos, generalmente estas variables quedarán alojadas en la memoria local de cada núcleo. Este comportamiento por defecto puede verse modificado mediante el uso de cláusulas en las directivas.

6.5. Cláusulas fundamentales

Como ya se introdujo en el apartado 6.3, las directivas de OpenMP pueden acompañarse de una serie de cláusulas que complementan su funcionalidad. En este sentido se han definido tres grupos de cláusulas: aquellas que afectan a las variables, ya sean compartidas o privadas, las que se ocupan de la sincronización y por último las que complementan la planificación de las regiones paralelas. La Tabla 12 ordena y resume la utilidad de las cláusulas fundamentales disponibles en OpenMP. En [OMPb] se pueden encontrar varios ejemplos de utilización de cláusulas fundamentales, así como una explicación más pormenorizada.

Cláusulas que afectan al uso de variables	
<i>Shared</i>	Los datos declarados en la región paralela estarán considerados como compartidos y serán accesibles por parte del resto de hilos.
<i>Private</i>	Las variables declaradas en la región paralela se consideran privadas, no son inicializadas y el valor de las mismas se pierde al salir de la región paralela.
<i>Default</i>	Permite cambiar la configuración por defecto en la región paralela.
<i>Firstprivate</i>	La variable privada toma el valor que tuviera ésta antes de la región paralela.
<i>Lastprivate</i>	El valor de una variable privada se conserva al terminar la región paralela.
Cláusulas de sincronización	
<i>Critical</i>	Garantiza que varios hilos no accedan de forma simultánea a una variable.
<i>Atomic</i>	La modificación de variables compartidas se realiza de forma atómica por parte de varios hilos.
<i>Ordered</i>	La ejecución de la sección se hará en orden, resulta útil al imprimir datos por pantalla.
<i>Barrier</i>	Los hilos del equipo esperan en la barrera (<i>barrier</i>) a que todos los hilos lleguen a ese punto ante de continuar su ejecución.
<i>Nowait</i>	Al final de la sección paralela los hilos no se esperan a que el resto finalicen.
Cláusulas de planificación con bucles (<i>scheduling</i>)	
<i>Static</i>	Las iteraciones se dividen en secciones que se asignan estáticamente a hilos de ejecución.
<i>Dynamic</i>	Las iteraciones son ejecutadas dinámicamente por los hilos disponibles.
<i>Guided</i>	Similar a <i>dynamic</i> , el número de iteraciones pendientes va disminuyendo dinámicamente.

Tabla 12. Cláusulas fundamentales de OpenMP.

6.6. Ventajas e inconvenientes de la utilización de OpenMP

La principal ventaja de OpenMP es la facilidad con la que pueden paralelizarse con ella secciones de código ya escrito; esta paralelización puede llevarse a cabo, además, de forma incremental. Estas características permiten flexibilizar la migración de código serie a entornos multinúcleo. Además, la estructuración en una serie de directivas para la gestión de las regiones paralelas permite a los desarrolladores sin experiencia llevar a cabo un rápido aprendizaje que les posibilita centrar los esfuerzos de desarrollo en la mejora del paralelismo de aquellas secciones de código especialmente sensibles. Por último, OpenMP permite al programador abstraerse, en la mayoría de los casos, de los problemas de gestión de la sincronización y acceso a memoria compartida, facilitando en caso de ser necesario las instrucciones necesarias para solventar los inconvenientes que puedan surgir.

Por el contrario no todas las distribuciones de la API de OpenMP son idénticas entre sí, dependiendo en gran medida de las implementaciones que los fabricantes de chips llevan a cabo para sus plataformas. Si bien todas las distribuciones están sujetas a las normas especificadas para cada versión de la API, el rendimiento de las aplicaciones puede verse afectado por posibles sobrecargas al utilizar OpenMP²⁵. Otra desventaja de OpenMP es que no se encuentra disponible para todos los tipos de procesadores, como por ejemplo sucede con aquellos sin memoria compartida en su arquitectura. Sin embargo su uso está muy extendido y se encuentra disponible para casi todos los procesadores multinúcleo con memoria compartida.

Resulta de especial importancia prestar atención a las distintas versiones de OpenMP, ya que no todas están disponibles para cada plataforma o fabricante; además las posibilidades ofrecidas en cada versión pueden variar de forma significativa²⁶. La Tabla 13 resume las ventajas y desventajas de utilizar la API de OpenMP.

Ventajas	Desventajas
Código paralelizado fácilmente <i>migrable</i>	Riesgo de introducir situaciones de difícil depuración
Paralelismo incremental	El compilador debe soportar OpenMP
Permite la paralelización de alto y bajo nivel de granularidad	Mayor dificultad al definir variables según sean privadas o compartidas
Compatible con diversos aceleradores	Se requiere una plataforma con memoria compartida
Código unificado tanto para regiones paralelas como de ejecución serie.	Riesgo al gestionar regiones de código compartido o privado
El código de ejecución serie no requiere modificarse	Compleja asignación de hilos a procesadores en secciones con alto grado de granularidad

Tabla 13. Relación de pros y contras al utilizar OpenMP.

Teniendo en cuenta la discusión anterior, la utilización de OpenMP para la paralelización del código de un decodificador de vídeo basado en RVC resulta acertada, siempre y cuando OpenMP esté disponible para la arquitectura objetivo, dado que: (1) se pueden realizar modificaciones incrementales sobre el código, considerablemente extenso y complejo, que integra el decodificador; (2) se puede realizar una paralelización de alto nivel, en el caso de RVC a nivel de actores o unidades funcionales; (3) no es necesario modificar el código interno de una región paralela o que se vaya a ejecutar en serie, (4) el mismo código con pequeñas modificaciones puede utilizarse sobre distintas plataformas multinúcleo, y (5) se puede asignar dinámicamente el número de núcleos a utilizar.

²⁵ La primera versión de OpenMP para plataformas Multi-DSP de Texas Instruments introducía una sobrecarga cercana al 90%. Sin embargo en posteriores versiones dicha sobrecarga se ha visto reducida hasta aproximadamente un 10%.

²⁶ La versión v4.0 de OpenMP incluye facilidades para la paralelización de código sobre ciertas plataformas heterogéneas.

6.7. Alternativas a OpenMP sobre plataformas multi-núcleo

Existen diversas herramientas alternativas a OpenMP que facilitan la programación de aplicaciones cuyo código es susceptible de paralelizarse. Sin embargo, la mayoría son específicas de cada fabricante, su funcionalidad está limitada a unas pocas arquitecturas y/o su desarrollo no está soportado por un consorcio inter-industrial abierto, haciendo difícil el acceso al código fuente de las APIs. A continuación se presentan dos alternativas válidas para plataformas Multi-DSP.

6.7.1. Desarrollo *ad-hoc*: gestión de la memoria y tareas

De forma paralela a APIs como OpenMP, el fabricante Texas Instruments permite la creación, gestión y sincronización de tareas y recursos sobre el SYS/BIOS (o kernel RTOS, ver apartado 7.2), y más concretamente sobre las últimas plataformas Multi-DSP homogéneas de la familia C66 [C66]. El fabricante facilita mediante diversos documentos técnicos las explicaciones y ejemplos necesarios para abordar el diseño de aplicaciones sobre plataformas multinúcleo. Así, en [SPR3P] se facilita la guía de usuario de SYS/BIOS, la cual describe todos los detalles sobre la gestión de recursos del RTOS y de la plataforma, en [TIWP] se perfilan técnicas para reducir el tiempo de desarrollo de aplicaciones multinúcleo con dispositivos KeyStone de Texas Instruments (ver apartado 7.1.2). Por último, en [SPR0C] se detallan las características de los distintos niveles de memoria y de los diferentes periféricos que soporta el chip.

En el caso de elegir esta vía de desarrollo el programador debe diseñar de forma manual e individualizada a cada caso, toda la gestión necesaria de todos los recursos de la plataforma. Es por esto que esta alternativa requiere de un elevado conocimiento de las particularidades de las herramientas propias del fabricante; cabe destacar que, aún en este caso, el tiempo de desarrollo puede ser elevado, y de forma general no se recomienda el uso de esta metodología salvo en aquellas aplicaciones de menor tamaño o complejidad.

Esta alternativa no permite por tanto soportar una solución metodológica en sí. Sin embargo, puede resultar de interés para llevar a cabo pruebas iniciales de estudios en desarrollo, de verificación de ciertos principios de aplicaciones de pequeña complejidad, o en aquellas en las que desee tener un control exhaustivo de todos los aspectos que controlan el comportamiento de bajo nivel del RTOS.

6.7.2. OpenCL

Open Computing Language (OpenCL) [OCL] es una herramienta integrada por una API y un lenguaje de programación específico llamado OpenCL C, la cual fue originalmente diseñada para facilitar la paralelización de aplicaciones sobre Unidades de Procesamiento Gráfico (GPUs) y Unidades Centrales de Procesamiento (CPUs). OpenCL fue estandarizado en 2013 por el consorcio internacional Khronos²⁷ [KHR], si bien fue inicialmente impulsado por Apple y desarrollado de forma conjunta por

²⁷ El grupo Khronos dispone de distintos grupos de trabajo para la creación de APIs, entre los cuales está OpenCL.

grandes grupos de la industria como IBM, Intel o NVIDIA. Cabe destacar que el modelo de programación de OpenCL está basado en el *framework* CUDA [CUDA] de NVIDIA.

El funcionamiento de OpenCL se basa en un *host* (anfitrión) que despacha tareas a otras unidades de procesamiento, bien porque éstas pueden ejecutar los algoritmos de forma más eficiente que el propio *host* o bien porque se desea paralelizar una serie de instrucciones. A estas unidades de procesamiento se las denomina aceleradores.

Como se puede observar en la Figura 31, los aceleradores utilizados bajo OpenCL están ubicados en una estructura jerárquica controlada por el *host*, el cual, utilizando instrucciones propias de la API de OpenCL, despacha tareas a cada uno de ellos (*Compute Device*, según la nomenclatura OpenCL). Éstos a su vez están compuestos de uno o varios clústers (*Compute Unit* - CU), los cuales ejecutarán las tareas paralelizadas en los elementos de procesamiento (*Processing Element*) de que dispongan. Uno o varios dispositivos aceleradores pueden agruparse en lo que, en OpenCL, se denomina un *contexto*. En función del entorno de desarrollo habrá un determinado número de contextos, pudiendo estar todos ellos controlados por un único *host*. Esto permite utilizar distintas arquitecturas dentro de un mismo entorno, pudiendo aprovechar así las ventajas que ofrecen cada una de ellas.

Figura 31. Diagrama de la distribución jerárquica con OpenCL.

De esta forma OpenCL permite gestionar sistemas compuestos de distintos tipos de procesadores, dentro de plataformas heterogéneas (ver apartado 7.4). En este tipo de sistemas las características de cada procesador pueden ser muy diferentes, por lo que OpenCL permite realizar una gestión individualizada de cada dispositivo. Esto se logra utilizando distintas versiones del *framework* OpenCL sobre los diferentes dispositivos, atendiendo así a las funcionalidades específicas de cada uno de ellos.

La herramienta OpenCL es relativamente novedosa, habiéndose iniciado su utilización en entornos Multi-DSP durante el desarrollo de esta tesis doctoral. A día de hoy, su integración con la metodología de diseño propuesta en esta tesis está dificultada por su modelo de reparto de tareas: OpenCL está diseñado para despachar *paquetes de*

trabajo o secciones de código bien definidas a los dispositivos aceleradores para que éstos los procesen, especificando para ello la cantidad de datos transferidos. Por lo tanto la posibilidad de gestionar la paralelización de la ejecución de los actores RVC, cuya ejecución es cíclica en base a su gestión mediante planificadores bien especificados, queda descartada por el momento.

6.8. Resumen

En este capítulo se ha presentado la API de OpenMP, mediante la cual es posible paralelizar código en lenguaje C para procesadores multinúcleo con arquitectura de memoria compartida.

La principal ventaja de OpenMP frente a otras alternativas es su fácil integración en un código ya escrito y aún no paralelizado. Se ha detallado cómo OpenMP permite llevar a cabo una paralelización del código de forma estructurada y modular.

Además, se han presentado las principales características y sentencias de programación que integran la API de OpenMP y que se han utilizado durante la fase de desarrollo de esta tesis doctoral.

Por último se han presentado dos alternativas a uso de OpenMP para la paralelización del código en arquitecturas multinúcleo.

7. Plataformas de desarrollo

En este capítulo se presentan las distintas plataformas sobre las que se han realizado las implementaciones y pruebas del trabajo de investigación llevado a cabo durante la tesis doctoral.

El capítulo se estructura de la siguiente manera: en el apartado 7.1 se explican las principales características de la plataforma de desarrollo TMDSEVM6678L y del procesador TMS320C6678, y se justifica su elección para el trabajo de investigación. En el apartado 7.2 se presenta el entorno de trabajo con plataformas Multi-DSP de Texas Instruments así como los elementos que dan soporte a su programación, depuración y test. En el apartado 7.3 se presentan otras plataformas utilizadas para la realización de pruebas y extracción de resultados. Finalmente en el apartado 7.4 se realiza un breve acercamiento a las plataformas heterogéneas.

7.1. La plataforma TMDSEVM6678L

La plataforma TMDSEVM6678L ha sido utilizada como principal banco de pruebas para llevar a cabo la mayor parte de las tareas relacionadas con la parte experimental del desarrollo de la metodología de diseño propuesta.

La Tabla 14 resume las características de esta plataforma, que puede verse en la fotografía de la Figura 32. La plataforma dispone de un procesador TMS320C6678, que se ubica bajo el disipador con ventilador que se aprecia en el centro de la figura. Para facilitar las tareas de volcado y depuración durante el desarrollo de *software* se conecta a la tarjeta un emulador XDS560v2 [XDS] (enmarcado en rojo en la fotografía de la Figura 32).

Plataforma TMDSEVM6678L	
Procesador TMS320C6678	UART RS232
512 MB de memoria DDR3	Conexión JTAG
64 MB de memoria flash NAND	Puerto Ethernet 10/100/1000
1 MB memoria EEPROM para arranque	LEDs e interruptores

Tabla 14. Principales características de la plataforma TMDSEVM6678L.

Figura 32. Fotografía de la plataforma de desarrollo TMDSEVM6678L, remarcado en rojo el emulador XDS560v2.

7.1.1. Elección del procesador TMS320C6678

Existen numerosas compañías que producen DSPs a día de hoy, como Analog Devices [ANA], Texas Instruments [TI], Motorola (Freescale) [MOT] o Hewlett-Packard [HP] entre otros, si bien el mercado está liderado por las dos primeras. A la hora de tomar la decisión sobre un producto concreto de un fabricante no sólo se deben tener en cuenta las características del DSP, como su potencia de cálculo o consumo energético, sino que la facilidad de acceder y tener soporte a las distintas herramientas *software* (como OpenMP u OpenCL), la posibilidad de solucionar problemas mediante un eficiente servicio técnico, o la posible experiencia ya adquirida deben ser factores muy a tener en cuenta.

Al inicio del desarrollo de esta tesis se evaluaron distintas posibilidades sobre la elección de la plataforma Multi-DSP idónea para la realización de pruebas. Como se ha explicado en el apartado 1.1, el GDEM tiene una larga experiencia trabajando con plataformas DSP, y en concreto con las del fabricante Texas Instruments. Así, el entorno de trabajo y las herramientas específicas eran bien conocidos por distintos miembros del grupo. Algunos de los trabajos anteriores a la realización de esta tesis emplearon plataformas DSP de Texas Instruments con buenos resultados [PGS⁺06], [PSJ⁺12] o [PGJS13].

De esta forma se decidió utilizar el procesador Multi-DSP TMS320C6678 de Texas Instruments, que al comienzo de la tesis era la plataforma Multi-DSP que más prestaciones ofrecía de este fabricante. Además, esta plataforma también había sido elegida por otros grupos de investigación colaboradores, por lo que se facilitaba una posible colaboración entre grupos, al tiempo que permitía una comparativa rápida de los resultados obtenidos. Esta plataforma ha sido utilizada también en distintas pruebas relacionadas con la herramienta PREESM (apartado 4.4). Finalmente, se ha utilizado en trabajos de investigación relacionados con otras disciplinas, como puede verse en [BBEG14], donde investigadores de la universidad de Ayyad (Marruecos) utilizan este Multi-DSP para realizar el procesamiento de un sistema de radar de haces múltiples, o en

[ASIG12], donde investigadores de la Universidad Complutense de Madrid (UCM) y la Universidad de Texas lo utilizan para implementar rutinas algebraicas complejas con vectores y matrices.

El procesador TMS320C6678 es una evolución de otros chips desarrollados por Texas Instruments y bien conocidos por el grupo GDEM; entre los que destacan el procesador TMS320C6472 [C6472], que dispone de seis núcleos DSP pero no resulta compatible con algunos de los módulos *software* de Texas Instruments más recientes, como el de OpenMP (ver capítulo 6), o el procesador TMS320DM6437 [DM6437], el cual dispone de un único núcleo DSP, por lo que no resulta adecuado para la consecución de los objetivos de esta tesis doctoral.

7.1.2. Principales características del procesador TMS320C6678

El procesador TMS320C6678 [C6678] forma parte de la familia C66x de procesadores DSP de Texas Instruments (dispositivos Keystone I), dentro de la que se engloban todos los procesadores DSP multinúcleo, y a la subfamilia C667x, dentro de la cual es el de mayores prestaciones. Sus principales características se resumen en la Tabla 15. La arquitectura dentro de una misma familia de procesadores es muy similar, permitiendo así aprovechar los conocimientos ya adquiridos en caso de cambiar la plataforma de trabajo.

Procesador TMS320C6678			
Número de procesadores	8	Memoria caché L1 datos	32 kB
Velocidad	0.80-1.20 GHz	Memoria L2 (SRAM/cache)	512 kB
Memoria caché L1 programa	32 kB	Memoria compartida (MSMC)	4 MB
Periféricos del procesador TMS320C6678			
SRIO 2.1	PCIe	Gigabit Ethernet	HyperLink
Interfaz 64 bits DDR3	TSIP	EMIF 16 bits	Interfaz UART
16 pines GPIO	I2C	Interfaz SPI	16 <i>timers</i> de 64 bits
512 MB de memoria externa DDR3 @ 667 MHz			

Tabla 15. Resumen de características del procesador TMS320C6678.

En la Figura 33 se muestra el diagrama de bloques de la arquitectura del procesador TMS320C6678. En dicha figura se pueden observar los 8 núcleos con sus memorias locales (memorias caché de niveles 1 y 2) interconectados al conjunto de periféricos y memorias compartidas mediante los controladores pertinentes (*Multicore Shared Memory Controller* – MSMC para la memoria compartida interna y *External Memory InterFace* – EMIF para la externa). Dicha interconexión se realiza mediante un complejo sistema de transmisión de datos llamado *TeraNet*, el cual dispone de un ancho de banda total de 2 Tb/s. A su vez la transmisión de datos está gestionada por el *Multicore Navigator*; este dispositivo controla la transmisión de paquetes entre todos los elementos que integran el SoC mediante una serie de colas de paquetes que transmiten la información a los dispositivos correspondientes.

Figura 33. Diagrama de bloques del procesador TMS320C6678 [C6678].

Además de las características descritas anteriormente, otro factor determinante en el rendimiento de un procesador es la velocidad de transmisión de datos entre los distintos niveles de memoria. La Figura 34 esquematiza las interconexiones entre cada uno de los 8 núcleos DSP del procesador TMS320C6678 con sus respectivas memorias caché, la memoria interna compartida entre todos los núcleos (*Multicore Shared Memory* o MSM) y el acceso a la memoria externa DDR3. Dado que el acceso a la memoria externa DDR3 es considerablemente más lento que el acceso a la memoria interna del procesador, a la hora de abordar el diseño y optimización de una aplicación siempre será recomendable ubicar la mayor cantidad de variables globales y *buffers* posible en memoria interna local o compartida. En el apartado 9.8 se detalla la utilización de los distintos niveles de memoria en el diseño del descodificador HEVC-RVC sobre la plataforma Multi-DSP.

Figura 34. Diagrama de velocidades de transmisión de datos entre las distintas memorias del procesador TMS320C6678.

7.1.3. Consideraciones sobre las memorias en los procesadores de la familia C66 de Texas Instruments

La gestión de los distintos niveles de memoria en las plataformas DSP no es una tarea trivial. El uso de herramientas *software* sobre plataformas multinúcleo, como OpenMP, pueden además requerir de configuraciones especiales que limiten el acceso a algunas regiones de memoria.

En arquitecturas multinúcleo sin sistemas operativos que gestionen el acceso a memoria por parte de los distintos procesadores será responsabilidad del programador asegurar que no se den fallos en dichos accesos. Un caso tipo se da al tener una variable compartida 'a' por varios núcleos y por tanto alojada en memoria compartida (MSM o externa) con un valor inicial de 'a=1'. En un momento dado, el núcleo 1 accede a esta variable, para lo cual la copia en su memoria local (L1/L2), y modifica su valor a 'a=2'. Si no se realiza una operación de *writeback*²⁸ (WB) el nuevo valor de 'a' no se actualiza de forma automática en la memoria compartida; en esta situación si otro núcleo (2) accede a la variable 'a', tomará el valor desactualizado de 'a=1'. Además, aun habiendo realizado el núcleo 1 un *writeback* del nuevo valor de 'a', el núcleo 2 deberá realizar una operación de *invalidate*²⁹ (INV), mediante la cual elimine la variable 'a' de su memoria local y realice una lectura del valor correcto almacenado en memoria compartida. Este es un sencillo ejemplo que pretende ilustrar un problema cuya solución puede resultar ardua para el programador.

²⁸ Una operación *writeback* consiste en indicar de forma manual (vía código) o automática (vía *hardware*) la actualización de una variable o región de memoria local a memoria compartida.

²⁹ Una operación *invalidate* consiste en indicar de forma manual (por programa) o automática (vía *hardware*) el borrado de una variable o región de memoria local, por lo que la próxima vez que se vuelva a utilizar dicha variable o región se accederá a memoria compartida y se copiará de nuevo en memoria local.

En el caso del procesador TMS320C6678 se dan dos niveles de memoria local o privada a cada procesador L1 y L2, entre las que la coherencia de memoria está asegurada vía *hardware*. Ambos niveles pueden configurarse para ser utilizados como memoria caché o como memoria SRAM de propósito general, como ilustra la Figura 35.

Figura 35. Posibles distribuciones de la memoria L1P como memoria caché o SRAM en el procesador TMS320C6678 [C6678].

Las distintas distribuciones configurables para la memoria L1P presentadas en la Figura 35 son idénticas para las memorias L1D y L2 (de forma proporcional al tamaño de la memoria).

Toda la memoria MSM del procesador TMS320C6678 es SRAM y *cacheable*³⁰, no pudiendo modificarse esta configuración. La memoria externa DDR3 también se considera SRAM y por defecto *cacheable*, sin embargo puede configurarse como no *cacheable* mediante los *Memory Address Registers*³¹ (MARs). El uso de regiones de memoria no *cacheables* debe reducirse al mínimo ya que influirá muy negativamente en el rendimiento de la ejecución del código afectado.

De forma adicional, el procesador TMS320C6678 dispone de los llamados *prefetch buffers*, éstos son registros locales donde un procesador puede almacenar código y variables de forma que estén disponibles antes de su ejecución. Sin embargo, éstos pueden ocasionar problemas de coherencia de memoria similares a los descritos anteriormente entre la memoria compartida y la memoria local. Sobre estos *buffers* no se pueden realizar operaciones de *writeback*, la única opción para asegurar la coherencia de memoria es realizar un borrado de los mismos por *software* [SPR0C].

Como se puede observar, se deben tener en cuenta numerosos detalles a la hora de gestionar la memoria de una forma correcta y eficiente. En este sentido también se

³⁰ Se denomina memoria *cacheable* a aquellas memorias o regiones de éstas que pueden copiarse en memoria local caché para optimizar la ejecución del código de un programa.

³¹ Los registros MAR contienen las direcciones de memoria donde se alojan los datos a ser procesados o almacenados por la CPU.

puede consultar la guía de usuario para la gestión de la memoria interna compartida (MSM) disponible en [SPR7A].

7.2. Entorno de trabajo para plataformas de Texas Instruments

El entorno de trabajo para plataformas de Texas Instruments tiene como elemento central el IDE llamado *Code Composer Studio* (CCS), disponible en [CCS]. Es una herramienta gratuita proporcionada por el fabricante, cuya una interfaz de usuario es muy similar a la del IDE de Eclipse al estar basado en éste último. CCS permite la escritura del código de los programas, la configuración de los recursos *hardware* de la plataforma y su depuración mediante el uso de un emulador. Durante el desarrollo de esta tesis se han utilizado las versiones CCSv5.5, CCSv6 y CCSv6.1 con el fin de disponer de las distintas mejoras introducidas en el *software* del IDE, fundamentalmente nuevas versiones del compilador que incluyen más optimizaciones del código.

De forma similar a Eclipse, el entorno de trabajo de CCS se basa en *workspaces* (espacios de trabajo). Cada *workspace* queda ubicado en un directorio de trabajo independiente en el sistema, y contiene uno o varios proyectos. Un proyecto puede estar destinado a generar un archivo ejecutable o un archivo de biblioteca. La configuración y compilación de cada proyecto es independiente pero pueden crearse dependencias entre éstos siempre que estén dentro del mismo *workspace*.

CCS está diseñado para trabajar con una amplia variedad de sistemas empujados, algunos de los más destacados se basan en procesadores como los de la familia OMAP [OMAP], además de las plataformas DSP mencionadas en el apartado 7.1.1. El IDE de CCS se basa en una serie de componentes que permiten al programador acceder y configurar los distintos recursos y funcionalidades de cada sistema.

En primer lugar es necesario disponer de un compilador que genere código máquina para el tipo de procesador para el cual se está diseñando una aplicación. CCS permite instalar compiladores para todas las plataformas a las que da soporte mediante los menús de ayuda del IDE. Pueden tenerse disponibles compiladores para distintas arquitecturas y de versiones diferentes sin que éstas entren en conflicto, únicamente será necesario seleccionar el compilador deseado para cada proyecto en las opciones del menú de propiedades.

Considerando que CCS está diseñado para trabajar con una gran diversidad de arquitecturas el IDE básico contiene únicamente un repertorio de funcionalidades limitadas. En función del proyecto a desarrollar será necesario facilitar a CCS una serie de paquetes adicionales. En este sentido, y a la hora de trabajar con plataformas tipo DSP, CCS se sirve de SYS/BIOS para dar soporte al RTOS de los DSPs.

SYS/BIOS es un conjunto de paquetes, los cuales se componen de módulos, que permiten acceder, configurar y ejecutar los distintos recursos que integran el RTOS,

tales como el reloj del sistema, el uso de elementos de sincronización (semáforos), *timers*, registros, tareas, etc. A este conjunto de paquetes se les denomina *Real Time Software Components* (RTSC). El RTSC se complementa con otro elemento, XDCtools (herramientas XDC) [XDC], cuya función es dar al usuario el soporte para la utilización de los componentes de SYS/BIOS y poder configurar los proyectos de manera acorde con la plataforma. La Figura 36 esquematiza el flujo de trabajo con CCS utilizando el conjunto de componentes que lo integran. Hay una amplia diversidad de paquetes RTSC que dan soporte a distintos recursos del sistema, habiendo a su vez numerosas versiones de cada paquete. Dichos paquetes afectan a distintas funcionalidades de la arquitectura y en ocasiones algunas versiones de éstos no resultan compatibles con otros paquetes. Por ello, Texas Instruments ha agrupado algunos de ellos con el fin de dar un mejor soporte. En el caso de la programación de procesadores Multi-DSP, está disponible el kit de desarrollo *software* para plataformas multinúcleo (MCSDK) [MCS]. Además, CCS permite, en versiones anteriores a la v6, realizar simulaciones de las aplicaciones desarrolladas.

Figura 36. Diagrama del flujo de trabajo con Code Composer Studio.

Para el desarrollo de esta tesis se ha utilizado la versión 2.01 del paquete MCSDK, en la Tabla 16 se presenta una relación de los módulos adicionales que se han utilizado y que están incluidos en el paquete, así como las versiones de SYS/BIOS y XDCTools compatibles entre ellos.

Acrónimo	Significado	Utilidad	Versión utilizada
CSL	<i>Chip Support Library</i>	Permite el acceso a recursos <i>hardware</i> de bajo nivel del sistema	CSL está integrado en el paquete PDK utilizado
IPC	<i>Inter-Processor Communication Development kit</i>	Facilita los protocolos de transferencia de datos entre procesadores	3.22.02.11
XDCTools (RTSC)	<i>Real-Time Software Components</i>	Facilita el acceso a los recursos del resto de paquetes instalados	3.25.06.96
OMP	<i>Open Multi-Processing</i>	Da soporte a la API de OpenMP	2.01.17.00

PDK	<i>Platform Development Kit</i>	Contiene los ficheros fuente, ejemplos y documentación de CSL y LLDs ³²	2.01.03.07
SYS/BIOS	--	--	6.37.03.30

Tabla 16. Módulos utilizados del paquete MCSDK de Texas Instruments.

La utilización de los distintos módulos RTSC dentro de un proyecto se indica de la siguiente manera: en las propiedades del proyecto se deben seleccionar los módulos deseados, esto entrega al compilador las rutas a los ficheros y bibliotecas que componen el paquete. Además, aquellos módulos que requieran algún tipo de configuración específica se deben instanciar y configurar en un fichero script de extensión *.cfg. En el ANEXO I – Archivos de configuración Anexo I se puede consultar el archivo cfg diseñado en la fase de desarrollo de la tesis.

7.3. Otras plataformas utilizadas en el trabajo de investigación

A lo largo del trabajo de investigación realizado durante la tesis se han utilizado otras arquitecturas multinúcleo distintas a los Multi-DSP, esto se ha hecho con el objetivo de validar el correcto funcionamiento de las aplicaciones desarrolladas sobre distintas plataformas. Además, los resultados obtenidos sobre varias plataformas permiten realizar comparativas de los datos obtenidos con respecto a las implementaciones desarrolladas por otros grupos de investigación. Con el objetivo de cubrir tres de las principales plataformas en uso a día de hoy se escogieron, además de la plataforma DSP, dos sistemas con procesadores tipo GPPs. El proceso de compilación, y la extracción de resultados sobre estas plataformas se han realizado utilizando compiladores nativos al sistema, por lo que no se ha utilizado el entorno CCS en estos casos.

Por un lado, se ha utilizado una plataforma basada en procesadores de ARM (ver apartado 3.3). Este tipo de procesadores han visto considerablemente aumentada su utilización en terminales de telefonía móvil o sistemas multimedia portátiles. Esto se debe a su bajo consumo energético y a su reducido, precio pero una potencia de procesamiento elevada. Por otro lado, se ha utilizado un ordenador personal de altas prestaciones integrado por un procesador Intel multinúcleo (ver apartado 3.2).

La plataforma GPP tipo ARM elegida fue la Odroid U3 [ODR]. Esta plataforma está formada por un procesador Exynos 4412 [EXY], el cual dispone de cuatro núcleos ARM Cortex A-9 en su arquitectura a 1,7 GHz. En la Figura 37 se muestra una fotografía de la plataforma, con sus elementos más significativos numerados.

³² Low Level Drivers (LLDs), Drivers de bajo nivel.

Figura 37. Fotografía de la plataforma Odroid U3.

Los elementos principales que integran la plataforma Odroid U3 son un conector micro HDMI (1), un conector USB-OTG (2), conector de alimentación a 5V (3), conectores USB 2.0 (4), disipador de calor montado sobre el procesador Exynos (5) y un conector Ethernet (6). Todos estos dispositivos de conexonado permiten al desarrollador probar de forma rápida diversas aplicaciones multimedia. La Figura 38 muestra su diagrama de bloques interno. Como se puede observar, el procesador incluye, además de los cuatro núcleos Cortex A-9, una memoria caché de nivel 2, un controlador de memoria externa DDR2, un procesador gráfico (GPU) Mali-400MP, periféricos diversos y varios conectores de comunicaciones (I2C, UART, etc.), y conexiones multimedia (HDMI, audio, etc.).

Figura 38. Diagrama de bloques de la plataforma Odroid U3.

En la mayoría de GPPs, al contrario que en plataformas tipo Multi-DSP (ver apartado 7.1), la coherencia de los datos almacenados en los distintos niveles de memoria se garantiza, de forma automática, vía *hardware*, mediante sistemas tipo *Snoopy Control Unit* (SCU). Dichos sistemas se sirven de una serie de *flags* de aviso localizados en cada línea de la memoria caché que indican si los datos almacenados son válidos, compartidos o deben actualizarse.

El ordenador personal utilizado es un GPP de altas prestaciones de la familia Intel Core i7. La arquitectura de este procesador es muy similar a la explicada en el apartado 3.2 (Figura 17), y sus principales características son su elevada potencia de cálculo, y la disponibilidad de memorias más rápidas y de mayor tamaño que las que se encuentran en las plataformas basadas en GPPs para procesadores empotrados o en DSPs. En este sentido destaca la memoria caché de nivel 3 de 8 MB compartida por los cuatro núcleos.

En la Tabla 17 se resumen las principales características de las plataformas utilizadas durante el desarrollo de la tesis.

Plataforma / Procesador	Núcleos	Frec. (GHz)	Mem L1 (KB)	Mem L2 (KB)	Otras memorias	Memoria externa y frecuencia de acceso
TMDSEVM6678 / TMS320C6678	8	0.80~1.25	32	512	4 MB compartida	DDR3 @ 667MHz
PC / Intel Core i7 950	4	3.07	32	1024	8 MB caché L3 compartida	DDR3 @ 1066 MHz
Odroid U3 / Exynos 4412	4	1.76	32	1024	-	DDR2 @ 880MHz

Tabla 17. Relación de procesadores utilizados y sus características.

En todos los casos las plataformas elegidas se corresponden con sistemas multiprocesador, esto permite estudiar el efecto de la paralelización de secciones de código entre los distintos procesadores, pudiendo comparar así las diferencias entre las ganancias obtenidas de dicha paralelización sobre las distintas plataformas. En el capítulo 10 se presentan los resultados obtenidos para distintas implementaciones de decodificadores de vídeo sobre los tres tipos de plataformas: las basadas en GPPs de altas prestaciones, las basadas en GPPs para sistemas empotrados y las basadas en DSPs.

7.4. Tarjetas de desarrollo KeyStone II: plataformas heterogéneas

Muy recientemente, los fabricantes de SoCs han comenzado a desarrollar y comercializar plataformas heterogéneas, es decir, aquellas que integran procesadores de más de un tipo en el mismo SoC. Estas plataformas son la respuesta a la necesidad de integrar en un mismo sistema las ventajas que puedan aportar distintos tipos de procesadores, aumentando considerablemente la potencia computacional del sistema. Esto se consigue distribuyendo entre los distintos procesadores la ejecución de los algoritmos de una aplicación.

El desarrollo de este tipo de plataformas es la evolución lógica de las plataformas multinúcleo homogéneas, como el Multi-DSP TMS320C6678. Sin embargo, su programación supone un reto aún mayor que el de la programación de plataformas homogéneas. La complejidad inherente al desarrollo del soporte *software* necesario para varios tipos de procesador en un mismo sistema, así como la gestión de

la transferencia de información o de la sincronización, requieren de nuevas herramientas y metodologías de programación.

Si bien este tipo de plataformas han tenido su salida al mercado de forma paralela al desarrollo de esta tesis doctoral, y su estudio queda fuera de los objetivos de la misma, las posibilidades que pueden ofrecer merecían ser analizadas.

La plataforma de desarrollo EVMK2H [K2H] de Advantech [ADV] se basa en la arquitectura *Keystone II* de Texas Instruments y, en su gama más alta, en el SoC 66AK2H14 [AK2H]. Este SoC integra en un mismo chip cuatro núcleos ARM Cortex-A 15 [C15] y un procesador TMS320C6678 con ocho núcleos DSP, que salvo algunas diferencias mínimas, como el tamaño de la memoria L2, es muy similar al utilizado como plataforma Multi-DSP de referencia en esta tesis. La Tabla 18 resume las características de los procesadores que integran el SoC, y la Figura 39 muestra su diagrama de bloques interno.

Procesador	Núcleos	Frecuencia máx. (GHz)	Caché L1	Caché L2	Mem. Compartida interna	Mem. Externa y frecuencia de acceso
ARM Cortex-A 15	4	1.4	32 KB	4 MB	6 MB	2 GB DDR3 @ 800MHz
TMS320C6678	8	1.2	32 KB	1 MB		

Tabla 18. Resumen de características de los procesadores del SoC 66AK2H14.

Figura 39. Diagrama de bloques del SoC 66AK2H14 [AK2H].

Una de las herramientas que permiten abordar la programación de este tipo de plataformas es OpenCL, ya introducida en el apartado 6.7.2. OpenCL se sirve de una API para gestionar los recursos de la plataforma heterogénea y el lenguaje de programación OpenCL C para el desarrollo de *software*. De esta forma se da una solución de alto nivel que permite aprovechar la ejecución en paralelo sobre los distintos procesadores. Además, OpenCL puede utilizarse sobre CCS a través del conjunto de paquetes incluidos en el kit MCSDK-HPC (*High Performance Computing*) y disponible en [MHPC]. En este sentido, se están llevando a cabo pruebas preliminares, en el laboratorio del GDEM, para analizar la posibilidad de paralelizar la ejecución de decodificadores HEVC-RVC con este tipo de plataformas (ver capítulo 13).

7.5. Resumen

A lo largo de este capítulo se han descrito las particularidades y principales características de las distintas plataformas que servirán para la prueba, extracción de resultados y verificación de la metodología propuesta en este trabajo de investigación.

Se ha puesto especial énfasis en la descripción de la plataforma TMDSEVM6678L y del procesador TMS320C6678, ya que con ellos se ha realizado la mayor parte del trabajo experimental de la tesis.

Además de las características de la plataforma se ha descrito el entorno de trabajo, el cual tiene como elemento central el IDE Code Composer Studio. Esta herramienta facilita la programación, volcado y depuración del código de las aplicaciones desarrolladas.

También se han presentado las particularidades de dos plataformas multinúcleo basadas en GPPs de altas prestaciones y en GPPs para sistemas empujados, respectivamente. Ambas plataformas se han utilizado para verificar la funcionalidad de la metodología de diseño propuesta para DSPs así como para comparar los resultados obtenidos.

Finalmente, se mencionan las plataformas basadas en procesadores heterogéneos que permitirán dar continuidad a la investigación en metodologías de implementación en el futuro.

8. Entorno de desarrollo de Orcc para RVC-CAL

En este capítulo se presenta el entorno de desarrollo basado en el compilador *Open RVC-CAL compiler* (Orcc), que es la herramienta que permite generar código fuente a partir de especificaciones RVC-CAL.

En primer lugar se realiza una breve introducción al compilador (apartado 8.1), explicando brevemente su origen, su propósito y el contexto en el que se enmarca. En el apartado 8.2 se presenta el entorno de trabajo con Orcc, los requisitos necesarios para su funcionamiento y los ficheros (código fuente) que genera como resultado de la compilación. En el apartado 8.3 se detallan los pormenores del funcionamiento interno del compilador, con especial énfasis en cómo se lleva a cabo la generación del código fuente, ya que en esta tesis se han introducido modificaciones en esta fase de compilación. Por último, el apartado 8.4 resume la utilización de código optimizado de forma complementaria al código fuente generado por Orcc.

8.1. Introducción a Open RVC-CAL compiler

Open RVC-CAL compiler fue desarrollado en el año 2010 dentro de la tesis doctoral de Mattieu Wipliez [Wip10] en el INSA de Rennes a partir del compilador Cal2C [RWR⁺08], como una infraestructura de compilación para programas basados en flujos de datos y específicamente pensado para trabajar con RVC-CAL. Desde la versión inicial hasta el día de hoy se han realizado numerosas actualizaciones del *software* que han ampliado y mejorado la funcionalidad inicial. Orcc es una herramienta de código abierto utilizado por distintos grupos de investigación, y que en la actualidad se administra a través de su sitio web en *Sourceforge*³³ [ORC]. El principal objetivo de Orcc es dar soporte a la infraestructura que permite la generación de código fuente en distintos lenguajes de programación³⁴, como C, Java o HDLs, a partir de descripciones RVC-CAL.

³³ En el sitio web de Orcc en Sourceforge se encuentran los tutoriales que guían al usuario en el proceso de instalación del entorno de trabajo además del acceso y descarga a las aplicaciones desarrolladas en RVC-CAL.

³⁴ De forma complementaria a la generación de código fuente en los lenguajes de programación clásicos, diferentes proyectos de investigación han creado *back-ends* (ver apartado 8.3.2) que generan código experimental; así, en la fecha de redacción de esta tesis doctoral se mantienen operativos los siguientes *back-ends* en Orcc: C, HLS, Jade, LLVM, PREESM, Promela, Simulator, TTA, Xronos y DAL.

El funcionamiento de Orcc se basa en la transformación de programas descritos mediante flujos de datos en Representaciones Intermedias (IR, ver apartado 8.3.1) para facilitar el análisis, la transformación y la generación de código fuente. Este proceso se divide en tres fases bien diferenciadas: *front-end*, *middle-end* y *back-end*, las cuales se explican en el apartado 8.3. Sin embargo, es sobre la última fase, el *back-end*, en la que se pondrá el máximo detalle, dado que es clave en el proceso de automatización de la metodología de diseño que se propone en esta tesis.

El lenguaje de programación C ha sido el utilizado durante toda la fase experimental de la tesis, por lo que se utilizará el *back-end* C de Orcc a lo largo de este capítulo para explicar los pormenores del funcionamiento del compilador.

8.2. Entorno de trabajo de Orcc

El entorno de trabajo de Orcc se apoya sobre una serie de aplicaciones *software* que dan soporte al desarrollador para generar el código fuente en alguno de los lenguajes de programación disponibles en los *back-ends*. El objetivo es reducir significativamente el tiempo de desarrollo, prueba y puesta en marcha de aplicaciones basadas en descripciones RVC-CAL sobre distintas plataformas.

Por un lado Orcc permite utilizar el compilador de una forma transparente para la generación de código fuente a partir de una versión *release*³⁵ que se descarga en forma de *plugin* en el IDE de Eclipse. Por otro lado también se pueden llevar a cabo modificaciones sobre la propia infraestructura del compilador, generar el código fuente en base a dichas modificaciones y comprobar su efecto. Los subapartados siguientes, 8.2.1, 8.2.2 y 8.2.3, se ocupan de la primera opción de trabajo, el apartado 8.3 de la segunda.

8.2.1. Requisitos del entorno de trabajo de Orcc

Orcc funciona sobre el entorno del IDE Eclipse [ECL], el cual requiere de *Java Run time Environment*³⁶ (JRE) [JRE] para funcionar. En las versiones más recientes de Eclipse³⁷ vienen incluidos los *plugins* de Xtext [XTE] y Graphiti [GRA], los cuales dan soporte tanto a la edición del código RVC-CAL como a la representación y modificación de los diagramas que representan a los actores y los flujos de datos que los interconectan en una aplicación.

Como se explicó en el apartado 2.3.5 existen en el repositorio distintas implementaciones basadas en flujos de datos para este entorno, tanto de descodificadores de vídeo como de otro tipo de aplicaciones. Éstas deben descargarse e importarse en un *workspace* de Eclipse. Una vez hecho esto el compilador de Orcc podrá generar el código fuente relativo a la aplicación deseada.

³⁵ Se llaman versiones *release* a aquellas cuyo correcto funcionamiento ha sido comprobado y se encuentran disponibles en un repositorio.

³⁶ *Java Runtime Environment* es un paquete *software* que permite la ejecución de programas escritos en Java mediante una máquina virtual.

³⁷ Versiones de Eclipse superiores a la 4.3.

8.2.2. Código fuente generado por Orcc

El código fuente generado por Orcc depende del *back-end* que se seleccione para la última fase de la secuencia de operaciones del compilador. En función del *back-end* seleccionado el código fuente, así como las bibliotecas generadas pueden variar. A continuación se explica el resultado de la ejecución del *back-end* C, que ha sido el utilizado en este trabajo de investigación.

El código generado se organiza en dos grupos, en primer lugar una carpeta fuente (*source*) que contiene el código C de los actores que implementan la funcionalidad de la aplicación. Se generará un fichero C por cada actor incluido en el diagrama. Además en esta carpeta se generan dos ficheros adicionales:

- Un fichero con extensión *xcf*³⁸ el cual contiene el listado de todos los actores. Este fichero puede modificarse para realizar una asignación estática de la ejecución de los actores a los núcleos en una plataforma multinúcleo, tal y como se describió en el apartado 4.5.3.
- Un archivo *Top_n_main*³⁹ que contiene las declaraciones de las FIFOs, las estructuras de los actores y conexiones, y la función principal o *main*. Las estructuras de los actores contienen el nombre del actor, las funciones de inicialización y las funciones a ser llamadas por el planificador, entre otras. Las estructuras de conexiones contienen la información de todas las conexiones entre todos los actores, origen y destino de la conexión, y medidas del número de *tokens* transmitidos.

Junto a la carpeta *source* se genera una segunda carpeta (*libs*) que contiene las bibliotecas que dan soporte al funcionamiento de la aplicación, esta carpeta contiene a su vez tres carpetas correspondientes al código de las tres bibliotecas necesarias, cuyas funcionalidades se describen a continuación:

- *Orcc-runtime*: en esta carpeta se incluyen los ficheros *.c* y *.h* que gestionan los aspectos relativos al control de la ejecución de los actores. Aquí se incluye la creación de los hilos de ejecución, la asociación a estos de los posibles planificadores de actores para cada procesador en un entorno multinúcleo y la lectura de los parámetros de entrada de la aplicación. Además, esta biblioteca incluye, en el archivo *mapping.c*, la tarea *agent_routine* cuyo código implementa las distribuciones dinámicas de actores a núcleos (ver apartado 4.5.4).
- *Orcc-native*: la biblioteca native permite acceder a los recursos nativos del sistema operativo instalado en la plataforma objetivo, tales como hilos de ejecución, elementos de sincronización, *mutex* o semáforos.

³⁸ Los ficheros *xcf* son archivos tipo texto con formato xml.

³⁹ El nombre del archivo es *Top_<nombre del gráfico compilado>_main.c*

- *Roxml*: la biblioteca *roxml* permite leer y escribir sobre los distintos ficheros con formato xml⁴⁰ utilizados durante la ejecución de la aplicación.

Dado el elevado número de ficheros generados por Orcc (ficheros con el código fuente de los actores y ficheros de las bibliotecas) se debe utilizar una herramienta de pre-compilación como CMake⁴¹ [CMA]. De esta forma se puede generar fácilmente un archivo *makefile*, que permita realizar el proceso de compilación de una forma sencilla.

8.2.3. Flujo de trabajo con Orcc

A continuación se explica el flujo de trabajo con Orcc. La Figura 40 proporciona un resumen que puede servir como guía durante la explicación.

Figura 40. Diagrama del flujo de trabajo en el entorno de Orcc.

Una vez se ha diseñado o descargado una aplicación basada en flujos de datos DPN, y en concreto bajo el estándar RVC, se ejecuta el compilador de Orcc que generará el código fuente de los actores y de las bibliotecas necesarias.

Si la plataforma de trabajo es tipo GPP y dispone de un SO de altas prestaciones se debe utilizar la herramienta CMake para generar el fichero *makefile*, lo que permite compilar el código fuente y obtener el ejecutable de la aplicación.

⁴⁰ Los archivos xml son archivos tipo texto, utilizados en las aplicaciones basadas en Orcc para la creación y modificación de mapeos estáticos de actores, así como para la escritura de los mapeos generados dinámicamente.

⁴¹ CMake es un *software* de código abierto multi-plataforma que permite gestionar la compilación de proyectos *software* para diferentes plataformas.

En un entorno multinúcleo el fichero *xcf* puede pasarse como parámetro de entrada al ejecutable con una distribución de la carga de los actores. Este fichero puede modificarse sin necesidad de recompilar la aplicación, pudiendo así realizar pruebas utilizando las mejores distribuciones de carga de una forma rápida y sencilla.

8.3. Infraestructura interna de Orcc

Como se ha introducido anteriormente, la infraestructura interna de Orcc está dividida en tres fases: *front-end*, *middle-end* y *back-end*. A continuación se hará una descripción general de todo el proceso. La descripción detallada de las operaciones que se llevan a cabo en las dos primeras fases, que no resulta relevante para la tesis, están bien descritas en [Wip10].

8.3.1. Descripción general

En la primera fase, el *front-end*, se realiza una transformación del código RVC-CAL de los actores a una Representación Intermedia; este paso requiere la realización de un análisis sintáctico y un análisis de los tipos de las variables utilizadas. La IR de Orcc es una representación de bajo nivel de la estructura y la semántica de un programa basado en una descripción de flujos de datos. Se realiza por tanto una descripción representativa de cada uno de los actores que integran la aplicación, dicha representación se hace en base al formato *JavaScript Object Notation* (JSON). Cada descripción JSON contiene, para cada actor, el nombre, los parámetros correspondientes y una lista de las acciones del mismo. Además, la FSM interna a los actores se traduce en una lista de transiciones *estado-a-estado* (*state-to-state*). Por último, se incluye la descripción correspondiente a las prioridades como una lista de etiquetas de las acciones correspondientes.

En la segunda fase, o *middle-end*, se clasifican y normalizan las IR de actores, tras lo cual se analiza e inserta la información relativa a la interconexión y transferencia de datos entre ellos, que como se ha descrito anteriormente se implementa mediante FIFOs unidireccionales.

Finalmente, en la tercera fase, o *back-end*, el código fuente de salida del compilador se genera a partir de la IR de los actores y de la descripción de sus interconexiones (red *xdf* de conexionado entre los actores). La Figura 41 resume el flujo de operaciones, las entradas y las salidas de cada una de las tres fases descritas.

Figura 41. Diagrama de la infraestructura del proceso de compilación de Orcc.

8.3.2. Los *back-ends* de Orcc

La última fase de la infraestructura del compilador Orcc se lleva a cabo mediante alguno de los *back-ends* existentes, que genera código para un determinado lenguaje de programación (C, LLVM, etc.). Sin embargo, el código que implementa el *back-end* no está escrito en estos lenguajes. El proceso de generación de código es distinto para cada *back-end* dadas las diferencias entre cada lenguaje de programación, como el grado de expresividad o la posibilidad de utilizarse sobre distintos tipos de plataformas pero, en todos los casos, se realizan las siguientes acciones:

- 1) Generación del código de los actores. En primer lugar se realiza una transformación de la IR, mediante la que se obtienen las instrucciones especificadas en la IR en el lenguaje objetivo. Tras esto se imprime directamente el código fuente mediante unas plantillas escritas en Xtend⁴² [XTN].
- 2) Generación del código de red. Comenzando desde el nivel más alto de la jerarquía de red se sustituyen los parámetros que la forman por valores concretos, acoplando las transmisiones de datos entre las unidades funcionales y generando el código para cada una de ellas.

Como se observa en la Figura 42, la generación del código de cada uno de los actores se controla mediante la plantilla *Instance Printer*, y el código del fichero *Top_n_main* (descrito en el apartado 8.2.2) mediante la plantilla *Network Printer*. El código de cada uno los archivos que componen las bibliotecas (*runtime*, *native* y *roxml*) está directamente incluido en el *back-end* en lenguaje C.

⁴² Xtend es un lenguaje de programación de alto nivel muy similar a Java. Originalmente Orcc utilizaba *String Template* [Par06], éste permitía imprimir o generar código a partir de plantillas diseñadas mediante una serie de reglas, el código generado de esta forma no tiene secciones ocultas y puede contener bucles (como bucles *for*).

 Figura 42. Diagrama de módulos internos del *back-end* C.

Como se analizará en el apartado 9.9, tanto las plantillas del *back-end*, como el código de las bibliotecas, pueden editarse para modificar el código generado y adaptarlo a la plataforma de trabajo deseada. En los apartados 9.9.3 y 9.9.4 se describen las modificaciones introducidas sobre distintas partes del código de las plantillas *Instance Printer* y *Network Pprinter* necesarias para crear un nuevo *back-end*.

8.4. Utilización de módulos optimizados desde el código fuente generado por Orcc

Las herramientas que controlan la generación automática del código a partir de las descripciones RVC-CAL permiten la incorporación de secciones de código nativo, escrito para la plataforma sobre la que se va a realizar la implementación. Esta técnica permite incrementar el rendimiento en términos de tiempo de ejecución del código generado.

La idea de utilizar módulos optimizados de OpenHEVC fue presentada por investigadores del INSA de Rennes en [JRS⁺14]. La solución propuesta se basa en mantener la generación del mismo código por parte de Orcc e incluir un *flag* en cada una de las funciones optimizadas disponibles. En caso de estar la optimización disponible, se realiza una llamada a la función de la biblioteca correspondiente de OpenHEVC en lugar de ejecutar el código nativo generado por Orcc. La Figura 43 muestra el código mediante el cual se selecciona la utilización de una función propia de OpenHEVC o nativa generada por Orcc.

```

#ifdef OPENHEVC_ENABLE
//Llamada a la función de la biblioteca de OpenHEVC
#else
//Código generado automáticamente por Orcc
#endif

```

Figura 43. Sección de código que permite la utilización de funciones optimizadas de OpenHEVC sobre código generado con Orcc.

Para dar soporte a las funciones optimizadas se ha diseñado un *wrapper* (envoltura o capa intermedia) en la biblioteca de Orcc *native* (ver apartado 8.2.2). La función del *wrapper* es enlazar correctamente los parámetros de entrada y salida de las funciones nativas de Orcc con los necesarios en aquellas funciones que se van a utilizar de OpenHEVC, la Figura 44 ilustra su funcionamiento.

Figura 44. Esquema de utilización de módulos optimizados de OpenHEVC sobre RVC.

En la Tabla 19 se muestra a modo de ejemplo el código correspondiente a la función *put_hevc_epel_cal()* del actor *InterPred_Inter_U* (ver apartado 10.1.2) del descodificador HEVC-RVC YUV. En este ejemplo, el código fuente generado por Orcc llamará a la función *put_hevc_epel_orcc()*, que es una función definida en OpenHEVC, si la macro *OPENHEVC_ENABLE* ha sido definida; en caso contrario se ejecutará el código especificado en RVC-CAL. Para que pueda ejecutarse la función de OpenHEVC es necesario realizar una adaptación de parámetros, como se muestra en la segunda fila de la Tabla 19, donde además se llama a la función nativa correspondiente de OpenHEVC *hevcDsp.put_hevc_epel()*. Dado que en función del número de bits utilizados en la codificación puede haber distintas implementaciones de una misma función⁴³, en el fichero *hevcdsp.c* se realiza una nueva llamada a la función final correspondiente, que en este caso es *put_hevc_epel_h()*.

Función optimizable del actor *InterPred_U*

```

static void HevcDecoder_InterPred_Inter_u_put_hevc_epel_cal(...) {
    #if defined(OPENHEVC_ENABLE)
        put_hevc_epel_orcc(predSamples, listIdx, refSamples, predCuSize, width,
height, mx, my);
    #else
        //Código generado por Orcc a partir de RVC-CAL.
    #endif // defined(OPENHEVC_ENABLE)
}

```

Fichero *openhevc_sse.c* (orcc-native)

```

void put_hevc_epel_orcc(i16 _dst[2][64*64], u8 listIdx, u8 _src[71*71], u8
srcstride, u8 _width, u8 _height, i32 mx, i32 my)
{
    //Adaptación de parámetros entre las funciones
    u8 *src = &_src[1+1*srcstride];
    i16 *dst = _dst[listIdx];
    u8 width = _width + 1;
    u8 height = _height + 1;
    int idx = LUT_PEL_FUNC[width];
    //Llamada a la función nativa de OpenHEVC
    hevcDsp.put_hevc_epel[idx][!!my][!!mx](dst, width, src, srcstride,
height, mx, my, width);
}

```

⁴³ Esto se debe a que las técnicas de optimización dependen del tamaño de las variables utilizadas, existiendo distintas soluciones en función del número de bits utilizados en la codificación.

Fichero hevcdsp.c (OpenHEVC)
<pre>#undef EPEL_FUNCS #define EPEL_FUNCS(depth) \ PEL_FUNC(put_hevc_epel, 0, 0, put_hevc_pel_pixels, depth); \ PEL_FUNC(put_hevc_epel, 0, 1, put_hevc_epel_h, depth); \ PEL_FUNC(put_hevc_epel, 1, 0, put_hevc_epel_v, depth); \ PEL_FUNC(put_hevc_epel, 1, 1, put_hevc_epel_hv, depth)</pre>
Fichero hevcdsp_template.c (OpenHEVC)
<pre>static void FUNC(put_hevc_epel_h)(int16_t *dst, ptrdiff_t dststride, uint8_t *_src, ptrdiff_t _srcstride, int height, intptr_t mx, intptr_t my, int width) { //Código nativo de OpenHEVC que implementa la función }</pre>

Tabla 19. Ejemplo de funcionamiento del esquema del *wrapper* entre OpenHEVC y RVC.

8.5. Resumen

A lo largo de este capítulo se ha detallado el entorno de trabajo del compilador Orcc, así como sus particularidades y comportamiento a la hora de generar código fuente a partir de descripciones RVC-CAL.

A partir de una descripción RVC-CAL, Orcc genera de manera automática el código fuente correspondiente en alguno de los lenguajes soportados, de acuerdo con el *back-end* previamente seleccionado. Los *back-ends* pueden ser modificados para influir en la generación del código fuente y adaptarlo a distintos entornos. Este proceso de generación de código fuente se lleva a cabo mediante plantillas escritas en Xtend.

En este trabajo de investigación se ha trabajado principalmente sobre el *back-end* C, encargado de controlar la generación de código fuente en este lenguaje de programación. Para cada aplicación RVC-CAL Orcc genera un fichero C con el código de cada uno de los actores que implementan la aplicación, además de las bibliotecas necesarias para gestionar el acceso a los recursos nativos del sistema y la ejecución de los actores.

Por último, y como se introdujo en el apartado 4.5.3, la implementación de un diagrama de flujos de datos y de los actores que integran la funcionalidad se realiza en sistemas con un SO sobre todos los procesadores, mediante la creación de hilos o *threads* que ejecutan secuencialmente los actores mapeados en un entorno multiprocesador. Sin embargo no todas las arquitecturas multiprocesador soportan este tipo de hilos de ejecución. Además, de forma general los *threads* realizan tareas y se ejecutan en paralelo, sin embargo los accesos a memoria y a los recursos del sistema son gestionados de forma transparente para el programador vía *hardware* o por un sistema operativo de altas prestaciones como puedan ser Linux o Windows.

CUARTA PARTE: Diseño de descodificadores de vídeo sobre plataformas Multi-DSP

En esta cuarta parte de la memoria se aborda el diseño de descodificadores de vídeo sobre plataformas Multi-DSP.

Esta parte se divide en dos capítulos, en el primero de ellos (capítulo 9) se presenta la metodología propuesta para el diseño de descodificadores de vídeo sobre plataformas multinúcleo, así como la automatización del procedimiento en un nuevo *back-end* de Orcc. Seguidamente, en el capítulo 10, se presentan los resultados obtenidos en las distintas pruebas realizadas sobre las tres plataformas utilizadas en la fase de desarrollo descritas en el apartado 7.3. De esta forma se pueden extraer conclusiones sobre la metodología propuesta.

9. Metodología de diseño

En este capítulo se propone una metodología de diseño para descodificadores de vídeo basados en chips multinúcleo. El desarrollo de esta metodología se ha basado en la experiencia acumulada durante el proceso de diseño de un descodificador HEVC sobre una plataforma Multi-DSP. En el apartado 9.1 se describe cómo este proceso ha servido para identificar los problemas generales que debe resolver el diseñador y obtener las soluciones que, a la postre, se han empleado para desarrollar la metodología propuesta. En los apartados del 9.2 al 9.8 se describe el detalle de los problemas identificados, las soluciones adoptadas y cómo éstas pueden generalizarse a otros casos de diseño. Finalmente, en el apartado 9.9 se explica cómo este conjunto de soluciones se ha integrado en una herramienta que permite abordar el diseño de descodificadores de vídeo con chips multinúcleo de manera automática.

9.1. Introducción

El método de trabajo que ha permitido el desarrollo de la metodología de diseño para descodificadores de vídeo basados en chips multinúcleo que se propone en la tesis está basado en la realización experimental de descodificadores de vídeo HEVC a partir de descripciones RVC-CAL, utilizando como tecnología objetivo chips multinúcleo. Si bien el planteamiento inicial era realizar una sola implementación basada en un chip Multi-DSP, durante el desarrollo de la tesis han ido surgiendo diferentes versiones de descodificadores HEVC RVC, cuya utilización ha enriquecido considerablemente el proceso de desarrollo de la metodología de diseño. También a lo largo de la tesis se ha visto la conveniencia, y en algunos casos la oportunidad, de utilizar otros chips multinúcleo basados en tecnología DSP o GPP. De modo que, finalmente, la fase experimental ha resultado más dilatada de lo esperado, pero también más enriquecedora. En la siguiente tabla se resumen las diferentes versiones RVC-CAL de los descodificadores HEVC implementados, así como las plataformas que se han utilizado durante su desarrollo. La tabla incorpora también una referencia al apartado de esta memoria en la que se realiza una descripción de los descodificadores, los chips y las plataformas.

	TMDSEVM6678 (apartado 7.1)	ARM – Odroid U3 (apartado 7.3)	PC – I7 (apartado 7.3)	EVMK2H (apartado 7.4)
RVC REF (apartado 10.1.1)	✓ (apartado 10.4.1)	✓ (apartado 10.4.2)	✓ (apartado 10.4.2)	
RVC YUV (apartado 10.1.2)	✓ (apartado 10.4.1)	✓ (apartado 10.4.2)	✓ (apartado 10.4.2)	
RVC FB3 (apartado 10.1.3)			Verificación (apartado 10.4.3)	Verificación (apartado 10.4.3)

Tabla 20. Relación de plataformas y descodificadores implementados durante el diseño de la metodología.

Durante esta fase experimental se han detectado las siguientes áreas de interés desde el punto de vista metodológico:

- Generación automática de código. La generación automática de código fuente en lenguaje C a partir de descripciones RVC-CAL está bien resuelta cuando la arquitectura objetivo es un GPP. Para tecnología DSP no resulta factible la utilización directa del *back-end* de C por diversos motivos.
- Paralelización del código para arquitecturas multinúcleo. El esquema de paralelización del código que implementa el *back-end* C de Orcc se basa en la premisa de que existe un sistema operativo que gestiona la ejecución paralela de las diferentes secciones del código. Esta presunción no es lo suficientemente general y, desde luego, no se cumple habitualmente en las plataformas basadas en chips Multi-DSP.
- Gestión del flujo de información y coherencia de las memorias caché. En los sistemas basados en flujo de datos, el rendimiento de las implementaciones depende

en gran medida de la eficiencia con la que se implementen las comunicaciones de datos. Esta eficiencia puede incrementarse con un uso adecuado de las memorias caché. Generalmente, las aplicaciones corren sobre de un sistema operativo que se encarga de gestionar la coherencia entre los diferentes niveles de caché, cuando no se garantiza vía *hardware* mediante una unidad SCU; pero esto no ocurre sobre todas las plataformas.

- Mapeo de algoritmos a núcleos. En algoritmos con un elevado nivel de paralelismo el mapeo de módulos a núcleos debe ser necesariamente automático. Los algoritmos de mapeo diseñados para GPPs pueden ser utilizados también para plataformas basadas en DSP, sin embargo se debe adaptar su implementación a este tipo de plataformas y al resto de modificaciones llevadas a cabo en la metodología, como su uso con OpenMP.
- Control del consumo de energía. En las aplicaciones que utilizan chips multinúcleo el consumo se incrementa con el número de núcleos utilizados. Las soluciones de diseño deben reducir el consumo cuando esto sea posible. También deberían permitir el establecimiento de soluciones de compromiso en las que se intercambie el consumo por prestaciones.
- Utilización de módulos optimizados. Cualquier implementación *software* es susceptible de optimizarse con el fin de mejorar el rendimiento de la misma al aprovechar las ventajas de una arquitectura *hardware* concreta. Sin embargo, los procesos de optimización implican largos tiempos de desarrollo difíciles de integrar en una metodología de trabajo multiplataforma. En este sentido se debe diseñar una infraestructura que permita utilizar secciones de código optimizado para una plataforma concreta de la forma lo más transparente posible al resto de la metodología de diseño.
- Personalización de las aplicaciones para las plataformas/chips objetivo. Cuando se trabaja con tecnología DSP, el mapeo de una misma aplicación en diferentes plataformas/chips requiere el uso de configuraciones (recursos *hardware*, memorias, paquetes RTSC, etc.) diferentes para optimizar la ejecución.

En los apartados 9.2 a 9.8 se entra en el detalle de cada una de las áreas de interés que se han enumerado. En cada apartado se realiza un análisis de la problemática que se plantea, se detalla la solución o las soluciones propuestas y se discute acerca de cómo estas soluciones pueden generalizarse a otros casos.

El conjunto de soluciones aportadas define una serie de pasos, una metodología, que puede aplicarse al diseño de decodificadores de vídeo a partir de descripciones RVC-CAL sobre diferentes plataformas/chips multinúcleo. Este conjunto de pasos se ha formalizado en un *back-end* Orcc, de manera que es posible realizar implementaciones de manera automática, evitando así el lento proceso de migración manual de código fuente. La descripción de este proceso de automatización se ha realizado en el apartado 9.9.

9.2. Generación automática de código

En primer lugar se abordan las modificaciones que se deben realizar sobre el código generado por Orcc para que éste pueda compilarse en la plataforma objetivo, y en el caso de la plataforma Multi-DSP en el IDE de CCS.

9.2.1. Planteamiento del problema

Como se ya se ha explicado en el apartado 8.2.2, el compilador Orcc genera el código fuente que implementa la funcionalidad de los descodificadores RVC mediante una serie de *back-ends*. Este código se estructura en varias carpetas y ficheros que deben compilarse en la plataforma objetivo, o mediante el compilador cruzado correspondiente. Sin embargo, no todas las funciones o recursos originalmente incluidos en el código fuente son susceptibles de poder ser compilados para las plataformas objetivo que se han empleado en esta tesis.

9.2.2. Solución propuesta

Dado que el código fuente original es compatible con plataformas tipo GPP con soporte para *threads*, en primer lugar se han identificado todas las secciones de código incompatibles con el entorno de las plataformas Multi-DSP. Para cada una de estos casos particulares se ha buscado una solución compatible y se han realizado las modificaciones oportunas. Esto permitirá, posteriormente, discriminar qué secciones de código deben omitirse del proceso de compilación en función de la plataforma objetivo.

Además, se desea que la solución propuesta sea operativa, independientemente de la plataforma objetivo. Esto implica que la solución deberá incluir en el código fuente todas las funciones o bibliotecas necesarias, y se activarán o desactivarán, en cada caso, las secciones de código implicadas para que el compilador pueda generar con éxito el fichero ejecutable del descodificador. Este criterio es extensible al resto de planteamientos que se hacen a lo largo de este capítulo.

9.2.2.1. Análisis del código generado del fichero Top

El fichero Top_<<implementación>>_main.c (en adelante simplemente Top) contiene la declaración de las estructuras de los actores, de las comunicaciones entre estos y la función *main*. La referencia <<implementación>> en el nombre del fichero se refiere al nombre de la descripción (fichero *.xdf) de uno de los proyectos RVC-CAL para Orcc al cual hace referencia y que engloba toda la jerarquía de la aplicación implementada. Todas las modificaciones se han resumido en la Tabla 21, donde se comparan con la versión original generada por Orcc.

Original	Modificado
Inclusión de bibliotecas de Orcc	Inclusión de bibliotecas de Orcc Inclusión de bibliotecas propias de la plataforma
Declaración de FIFOs	Declaración de FIFOs en memoria compartida FIFOs alineadas a tamaño de línea de caché
Función Main Parámetros de entrada	Función Main Configuración tamaño de las cachés Parámetros de entrada

Tabla 21. Resumen de modificaciones realizadas en el fichero Top generado por Orcc.

En primer lugar se han incluido los ficheros de cabecera necesarios en este archivo, tanto a la biblioteca de OpenMP (apartado 9.3) como al fichero `csl_xmcAux.h` del paquete CSL del RTOS. Esto es necesario para gestionar, posteriormente, todos los aspectos relativos a las memorias caché L1 y L2.

Seguidamente, son declaradas cada una de las estructuras de las FIFOs y de sus variables (punteros de lectura/escritura y contenido). Todos los parámetros deben ubicarse, siempre que sea posible, en memoria compartida MSM para evitar una bajada del rendimiento. Además, cada uno de los campos debe estar alineado a tamaño línea de caché (128 B para el procesador TMS320C6678) para evitar que se realicen operaciones de *writeback/invalidate* (WB/INV) sobre variables de forma involuntaria (los aspectos relacionados con la gestión del flujo de información y la gestión de las memorias caché se detallan en el apartado 9.4).

Por último, en la función *main*, se fija el tamaño de las cachés mediante las instrucciones descritas en la Figura 45.

```

CACHE_setL1Psize(CACHE_L1_32KCACHE);
CACHE_setL1Dsize(CACHE_L1_32KCACHE);
CACHE_setL2Size(CACHE_256KCACHE);

```

Figura 45. Instrucciones para definir el tamaño de las cachés L1 y L2.

9.2.2.2. Análisis del código de las bibliotecas generado por Orcc

Como se expuso en el apartado 8.2.2, el compilador de Orcc genera tres carpetas que contienen los ficheros fuente de las bibliotecas *runtime*, *native* y *roxml*. Estos ficheros también deben ser modificados puntualmente para poder compilarse en el entorno CCS. Tras el proceso de compilación cada conjunto generará el correspondiente archivo de biblioteca (*.lib o *.a) que se enlazará al proyecto principal, en el que se encuentra el fichero Top y los relativos a la funcionalidad de los actores del decodificador.

9.2.2.2.1. Modificación de la biblioteca *Orcc-runtime*

En esta biblioteca se encuentran, entre otros, los siguientes ficheros que deben modificarse:

- *fifo.h*: en este fichero se encuentra la declaración de las estructuras de las FIFOs. Se ha modificado el valor de la constante `CACHELINE_SIZE`, que hace referencia al tamaño de línea de caché de la plataforma utilizada, de 64⁴⁴ a 128.
- *scheduler.c*: es en este fichero donde se encuentra el código relativo al *launcher* (lanzador), es decir, la función que crea los hilos de ejecución sobre los que corren las rutinas de los planificadores, las cuales llaman a ejecución a todos los actores mapeados en cada núcleo. Al código de este fichero se le irán añadiendo sucesivas modificaciones que complementan la funcionalidad del descodificador sobre distintas plataformas (apartados 9.3.2, 9.4.2.2, 9.5.2.1 y 9.6.2.4).
- *util.c*: en este fichero se gestionan los parámetros de entrada que configuran el proceso de descodificación. Todos los parámetros se copian en la estructura *opt*, para ello se hace uso de la función *strdup()*, la cual no existe en las bibliotecas de SYS/BIOS, por lo que se sustituye por la función *strcpy()*. El cometido de ambas funciones es realizar la copia de un *array* de caracteres.

9.2.2.2.2. Modificación de la biblioteca *Orcc-native*

En la biblioteca *native* se encuentran los ficheros que permiten hacer uso de algunas bibliotecas o recursos nativos del SO de la plataforma. Por este motivo también se encuentran distintos archivos que permiten hacer uso de optimizaciones mediante algunas funciones de OpenHEVC y/o instrucciones SSE para plataformas x86, así como de la utilización de un *display* mediante la biblioteca SDL (ver apartado 5.1). En el caso de un entorno DSP los siguientes archivos deben, a priori, excluirse de la compilación del proyecto:

- *display_sdl.c* - *display_sdl2.c* - *ffmpeg_util.c* y *.h*
- *framerate_sdl.c* - *openhevc_sse.c* y *.h* - *audio_sdl.c*
- Todos los ficheros que hagan referencia a *hevc_xx_.c*

Además, se ha modificado parcialmente el código de los siguientes ficheros:

- *thread.h*: en la sección de código referente a los tipos de plataforma (WIN32, APPLE...) se debe sustituir la inclusión del archivo *semaphore.h* por *sem.h*, acorde con el nombre de este tipo de recurso en SYS/BIOS.
- *native.c* y *openhevc_sse.c*: en caso de utilizar secciones de código optimizadas de OpenHEVC se deberán activar los *flags* correspondientes (ver apartado 9.7).

9.2.2.2.3. Modificación de la biblioteca *Orcc-roxml*

En distintos ficheros de esta biblioteca se hace uso de las funciones *strdup()* y *fdopen()* que no existen en el SYS/BIOS de Texas Instruments, en todos los casos deben sustituirse por las funciones *strcpy()* y *fopen()* respectivamente. Además deben comentarse o excluirse todas las inclusiones del fichero *fcntl.h*, el cual tampoco está disponible y se emplea en el manejo de ficheros.

⁴⁴ El tamaño de línea de cache de 64B es típico de arquitecturas x86.

9.2.2.3. Modificaciones sobre el código generado de los actores

Dentro de lo posible siempre será recomendable no modificar manualmente el código que implementa la funcionalidad de los actores para mantener la interoperatividad del código fuente, asegurando así su funcionamiento en distintas plataformas. En el apartado 2.3.4 ya se detalló el funcionamiento interno de los actores.

En un primer acercamiento a la migración del código de los actores al entorno Multi-DSP únicamente ha sido necesario excluir el uso de las funciones encargadas de la representación por pantalla del vídeo descodificado. Dichas funciones se encuentran en el actor *display* y son todas aquellas en cuyo nombre se encuentre el identificador “YUV”, a excepción de la función *displayYUV_getNbFrames()* que se utiliza para contabilizar el número de imágenes descodificadas. Esta función será necesaria para calcular el rendimiento de la descodificación en FPS y para saber si es necesario implementar un nuevo mapeo en el contexto de mapeo dinámico (apartado 9.5).

9.2.3. Generalización

La solución adoptada para mantener la compatibilidad entre las plataformas que pueden hacer uso de las distintas bibliotecas se basa en que las modificaciones introducidas deben acotarse con directivas del tipo *ifdef/endif*, haciendo referencia a la definición o no de una macro que indique el entorno de trabajo. Este tipo de directivas permiten habilitar y deshabilitar secciones de código antes de que se produzca la compilación del proyecto. De esta manera se han definido dos macros globales de aviso (*flags*) a todo el proyecto ⁴⁵: `OPENMP_ENABLE` (ver apartado 9.3) y `MDSP_ENABLE`. La primera habilita el uso de OpenMP y omite el uso de *threads*, la segunda activa aquellas secciones de código necesarias para que el descodificador pueda funcionar sobre una plataforma Multi-DSP.

Sin embargo, la activación de la macro `MDSP_ENABLE` no puede realizarse de la misma forma al no existir compatibilidad entre el IDE de CCS y CMake. En su lugar, y cuando se vaya a trabajar sobre una plataforma Multi-DSP, ambas macros se han definido como constantes entre los parámetros del compilador utilizados en CCS, integrándolas en el fichero *.cproject* de configuración del proyecto como muestra la Figura 46.

```
(...)  
<option id="com.ti.ccstudio.buildDefinitions.C6000_7.4.compilerID.  
DEFINE.2116070978" name="Pre-define NAME (--define, -D)"  
107ectores107s="com.ti.ccstudio.buildDefinitions.C6000_7.4.compilerID.DEFINE  
" valueType="definedSymbols">  
  <listOptionValue builtIn="false" value="OPENMP_ENABLE"/>  
  <listOptionValue builtIn="false" value="MDSP_ENABLE"/>  
</option>  
(...)
```

Figura 46. Activación de las constantes globales para entorno Multi-DSP.

⁴⁵ El proyecto se refiere al conjunto de ficheros fuente y bibliotecas generados por Orcc y que deben ser compilados.

Se debe tener en cuenta que las secciones dependientes de la activación de MDSP_ENABLE están específicamente diseñadas para funcionar en el entorno de trabajo CCS de las plataformas de Texas Instruments, en caso de querer trabajar en otro entorno únicamente será necesario adaptar las instrucciones acotadas por la macro MDSP_ENABLE o aquellas llamadas a funciones específicas de la plataforma objetivo.

9.3. Paralelización del código para arquitecturas multinúcleo

Como se ha detallado en el capítulo 6, la API de OpenMP sirve como herramienta para la paralelización de secciones de código en entornos multinúcleo sobre arquitecturas con memoria compartida. En este sentido OpenMP es compatible con las tres plataformas multinúcleo elegidas para la realización de pruebas (un GPP de altas prestaciones, un GPP tipo ARM y la plataforma Multi-DSP) entre las muchas disponibles en el mercado actualmente.

9.3.1. Planteamiento del problema

Como ya se introdujo en el apartado 9.2.2, el código fuente generado por Orcc utiliza llamadas a funciones de la biblioteca *threads* no disponibles para plataformas tipo Multi-DSP. La gestión de la ejecución en paralelo de secciones de código debe utilizar entonces alternativas compatibles con la plataforma objetivo cuyo soporte esté garantizado por el fabricante de la misma.

Ya en el apartado 6.7 se presentaron distintas alternativas a la API de OpenMP para la implementación de secciones de código cuya ejecución es susceptible de paralelizarse en plataformas Multi-DSP, sin embargo no todas ofrecen una solución válida para integrarse en una metodología de diseño. Además, la compatibilidad en la utilización de distintos recursos de sincronización, como semáforos o *mutex*, debe asegurarse para el procesador objetivo; manteniendo de igual forma su interoperatividad entre las tres plataformas escogidas.

9.3.2. Solución propuesta

La solución se basa en realizar las modificaciones necesarias sobre el código fuente generado por Orcc con dos objetivos: (1) gestionar la ejecución multinúcleo en la plataforma Multi-DSP, y (2) disponer de una solución basada en hilos OpenMP en lugar de *threads* para plataformas GPP. Esto es posible gracias a que se ha observado que todas las modificaciones necesarias para plataformas GPP también son necesarias para la plataforma Multi-DSP. De esta forma, aquellas modificaciones que sustituyen el uso de *threads* por OpenMP se activarán mediante la macro OPENMP_ENABLE, y las extensiones para plataformas Multi-DSP quedarán supeditadas, además, a la macro MDSP_ENABLE. En este sentido debe tenerse en cuenta que el objetivo de esta metodología es que permita que el código fuente generado sea compatible con los tres tipos de plataformas contemplados en esta tesis, y no sólo las plataformas Multi-DSP de Texas Instruments.

Se ha llevado a cabo un análisis del código fuente con el objetivo de identificar todas las llamadas a hilos de ejecución (*threads*), que serán sustituidas por regiones paralelas OpenMP. El análisis del código fuente se ha realizado siguiendo la misma organización de carpetas y directorios descrita en el apartado 8.2.2.

Todas las modificaciones necesarias para realizar la implementación del decodificador basado en OpenMP afectan a las bibliotecas de Orcc. En su mayoría estos cambios atañen al uso de funciones de sincronización, como semáforos, o de otros recursos nativos del sistema como los hilos de ejecución. Así, en los archivos *scheduler.c* y *.h*, y *mapping.c* y *.h* se ha unificado la inclusión del fichero *thread.h*, como se muestra en el extracto de código de la Figura 47.

```
#if defined(THREADS_ENABLE) || defined(OPENMP_ENABLE)
#include "thread.h"
#endif
```

Figura 47. Modificación de la inclusión del fichero *thread.h*.

El uso de las funciones con semáforos se define desde el mismo archivo *thread.h* de la biblioteca *native*. Es necesario redefinir cada función descrita en Orcc con su equivalente en la biblioteca de OpenMP, por ello se añade la sección de código descrita en la Figura 48.

```
#ifdef OPENMP_ENABLE
#include <omp.h>

#define orcc_semaphore_create(lock, number) omp_init_lock(&lock)
#define orcc_semaphore_wait(lock) omp_set_lock(&lock)
#define orcc_semaphore_set(lock) omp_unset_lock(&lock)
#define orcc_semaphore_destroy(lock) omp_destroy_lock(&lock)
#define orcc_semaphore_t omp_lock_t
#else
```

Figura 48. Redefinición de funciones para el manejo de semáforos con OpenMP.

Además, siempre que se vaya a hacer uso de alguna de estas funciones condicionadas a la macro *THREADS_ENABLE* se deberá incluir la definición de *OPENMP_ENABLE*, de forma similar a como se mostró en la Figura 47. Sin embargo, los mayores cambios afectan a la creación de hilos y de las rutinas de los planificadores que ejecutan los actores. En la Tabla 22 se muestra el código del lanzador (*launcher*), el cual implementa esta parte del programa. Debe tenerse en cuenta que el número de procesadores a utilizar en la decodificación (*nb_threads*) se introduce como parámetro de entrada (*-c <nº_hilos>*) al inicio del programa. En el caso de utilizar OpenMP se debe sumar un hilo de ejecución adicional al pasar este parámetro a la función *omp_set_num_threads()*, este hilo extra ejecutará la rutina de remapeo en los casos en los cuales se utilicen distribuciones dinámicas de actores.

Threads	OpenMP
<pre> void launcher(options_t *opt, network_t *network) { //Carga de mapeo estático int nb_threads = opt->nb_processors; //Número de hilos a utilizar #ifdef THREADS_ENABLE // Se declaran los hilos de ejecución #endif /* THREADS_ENABLE */ //Se declaran los parámetros de los planificadores //Se declaran los parámetros de la rutina de remapeo #ifdef THREADS_ENABLE for(i=0 ; I < nb_threads; i++){ //Se crean los hilos de ejecución y //se asocian a los procesadores } //Se crea la rutina de remapeo #endif #else //Si no hay threads ni OpenMP //Llamada a una única rutina del planificador (con un solo núcleo) #endif } </pre>	
	<pre> #ifdef OPENMP_ENABLE omp_set_num_threads(nb_threads+1); #endif /* OPENMP_ENABLE */ #ifdef OPENMP_ENABLE #pragma omp parallel { #pragma omp for nowait for(i=0 ; I < nb_threads; i++){ //Llamada a las rutinas del //planificador para cada procesador } } //Fin de región paralela </pre>

Tabla 22. Código del lanzador que arranca el proceso de descodificación.

9.3.3. Generalización

La solución planteada anteriormente es válida para cualquiera de las plataformas utilizadas en esta tesis doctoral, y complementa la propuesta del apartado anterior, 9.2.3. La solución propuesta es fácilmente extensible a cualquier plataforma con soporte OpenMP.

Además, la activación de la macro `OPENMP_ENABLE` se ha introducido en el fichero *CmakeLists.txt* común a todos los directorios del proyecto (bibliotecas y código fuente de los actores), como muestra el código de la Figura 49. De esta forma al ejecutar la herramienta CMake la constante `OPENMP_ENABLE` estará activa en todos los ficheros sin necesidad de habilitar su uso de forma independiente en cada fichero.

```

# Allow the use of OpenMP instead of Threads
set(USE_OPENMP True)

```

Figura 49. Activación de OpenMP en el fichero CmakeLists.txt.

Un objetivo fundamental de todas las modificaciones que se están presentando es no alterar, en la medida de lo posible, la filosofía del funcionamiento de las aplicaciones que se basan en especificaciones RVC-CAL y Orcc. En este sentido, OpenMP no afecta de forma significativa al comportamiento general del descodificador. En la Figura 59 del apartado 9.5.3 se presenta el diagrama completo del funcionamiento global de la aplicación, ya sea con *threads* u OpenMP.

9.4. Gestión del flujo de información y coherencia de caché

A lo largo de esta memoria se han hecho varias referencias a la problemática que supone gestionar los posibles errores al acceder a la información compartida en un escenario con varios procesadores. En el apartado 7.1.3 se presentaba un primer caso sencillo donde dos procesadores accedían a la misma variable y se mostró cómo el dato leído por uno de ellos no era el esperado. También se ha explicado que en arquitecturas GPP esto se soluciona de forma automática, bien por el propio SO o bien por el *hardware* del sistema.

9.4.1. Planteamiento del problema

En la plataforma TMS320C6678 (y generalmente en plataformas Multi-DSP) la coherencia de memoria entre las memorias L1 y L2 se asegura vía *hardware*, pero esto no ocurre entre la memoria L2 y las memorias compartidas MSM y DDR3, quedando su correcto funcionamiento bajo la responsabilidad del programador. En las implementaciones basadas en especificaciones RVC-CAL y Orcc, los datos compartidos entre núcleos, y por tanto aquellos cuya coherencia en sus valores debe ser asegurada, se encuentran únicamente en las FIFOs.

Las FIFOs utilizadas en este tipo de aplicaciones disponen de tres campos: puntero de escritura (Wr), punteros de lectura (Rd) y datos o información transmitida (contenido o *payload*). En cada FIFO habrá un único puntero de escritura (sólo un actor puede poner datos en la FIFO), pero puede haber múltiples punteros de lectura (varios actores pueden leer de una misma FIFO). En la Figura 50 se muestra una FIFO que comunica los actores 1 y 2, con sus correspondientes punteros de lectura/escritura. En este ejemplo el actor 1 ha escrito una cantidad de datos en la FIFO (datos escritos en color marrón), sin llegar a llenar la capacidad de la FIFO (espacio libre en color verde), y el actor 2 ha leído parte de esos datos (datos ya leídos en color verde claro). Cuando el espacio libre se acabe el actor 1 comenzará a escribir nuevos datos sobre los ya leídos por el actor 2. Si no quedase espacio libre el actor 1 quedaría a la espera de que el actor 2 leyera datos de la FIFO, liberando espacio.

Figura 50. Ejemplo del proceso de lectura/escritura mediante una FIFO y dos actores.

Cada vez que un actor escribe *tokens* en la FIFO, éstos se almacenan en un *buffer* circular de tamaño fijo (típicamente 16 kB ó 32 kB⁴⁶), y actualiza el valor del puntero de escritura, indicando la posición hasta la cual ha escrito nueva información. Los actores lectores saben que hay nuevos datos calculando la diferencia entre su puntero de lectura y el de escritura, así se conoce el número exacto de nuevos *tokens* disponibles. Una vez el actor lector ha procesado la nueva información disponible actualiza el valor de su puntero de lectura para que indique la posición del último dato leído, si ha leído todos entonces el valor del puntero de escritura y de lectura serán iguales. Cada actor lector realizará esta operación de forma independiente. El actor que escribe *tokens* en la FIFO verificará, en primer lugar, si hay espacio disponible, observando hasta dónde han consumido datos todos los actores lectores mediante la comprobación de sus punteros de lectura.

Mediante este procedimiento se asegura que no se producen sobrescrituras de datos, que varios actores pueden recoger datos de una misma FIFO sin entrar en conflicto y que un actor puede producir *tokens* siempre que haya espacio disponible. Estos factores ofrecen la ventaja fundamental de no requerir ningún tipo de sistema de sincronización en la transmisión de datos.

La Figura 51 ilustra un ejemplo de la problemática aquí planteada. Por un lado, el actor 1 está mapeado en el núcleo 0 y escribe datos en la FIFO 1, al mismo tiempo el actor 4, mapeado en el núcleo 1, lee de esta misma FIFO. Cada vez que el actor 1 produzca nuevos datos, éstos se escribirán en la FIFO 1 (paso 1) y se actualizará el puntero de escritura (paso 2). Al mismo tiempo el actor 4 toma los nuevos datos disponibles (paso 3) y actualiza su puntero de lectura (paso 4). Por otra parte, los actores 2 y 3 se ejecutan también en el núcleo 0, y comparten la FIFO 2. El actor 2 escribe datos en la FIFO 2 (paso 5) y actualiza el puntero de escritura (paso 6), mientras que el actor 3 lee los datos de la misma FIFO (paso 7) y actualiza su puntero de lectura (paso 8).

⁴⁶ El tamaño mínimo de las FIFOs viene definido por las particularidades del código que implementa el descodificador, y su valor típico es de 8 kB. Al definir FIFOs de menor tamaño las necesidades de memoria serán menores, pero se realizarán más operaciones de lectura/escritura, lo que tendrá un impacto negativo en el rendimiento. De esta forma se debe establecer un valor de compromiso para el tamaño de las FIFOs.

Figura 51. Diagrama del acceso de los actores a las FIFOs.

En el primer ejemplo (actores 1 y 4), si no se lleva a cabo ninguna acción específica, todo el procesamiento descrito se lleva a cabo en las memorias cachés de cada núcleo. El problema que esto supone es que, en el ejemplo presentado, el actor 1 está poniendo datos en la FIFO 1, pero esta operación se realiza localmente en sus memorias cachés, el actor 4 no se percatará de esta situación, es más, el actor 4 también podría leer una copia de los datos de la FIFO 1 de su propia caché. De esta forma el procesamiento no continuará hasta que, por otros motivos, no se actualice la memoria local de cada núcleo.

Sin embargo, esta situación no presenta problemas en el segundo ejemplo (en la transmisión de datos entre los actores 2 y 3) ya que ambos están mapeados en el mismo núcleo, sus memorias cachés son por tanto las mismas y los datos estarán siempre actualizados y disponibles entre ellos.

9.4.2. Solución propuesta

Una vez descritos los factores que pueden afectar en el mantenimiento de la coherencia de datos entre los distintos niveles de memoria, se procede a implementar una solución válida para toda la metodología basada en especificaciones RVC-CAL y Orcc. Además, esta solución tendrá en cuenta la posible utilización de algoritmos para realizar mapeos dinámicos de actores en tiempo de ejecución (apartado 9.5). En estos casos el emplazamiento de uno o varios actores en los distintos núcleos podrá variar de forma dinámica, en esos casos la gestión de las FIFOs que comuniquen con los actores afectados deberá ajustarse también de forma dinámica.

En aplicaciones basadas en flujos de datos, cuyo código fuente ha sido generado por el compilador Orcc, e implementadas sobre plataformas multinúcleo, un número variable de actores se ejecutarán en alguno de los núcleos disponibles, y toda la información compartida entre éstos se transmitirá por las FIFOs que los enlazan. De esta forma, si se asegura que tanto los datos almacenados en las FIFO, como las copias

locales de éstas en las memorias L1 y L2 de cada núcleo están actualizados, la aplicación no debería presentar problemas de funcionamiento.

Una primera solución a las posibles incoherencias de datos entre las memorias caché y compartida (interna o externa) consistiría en la realización de una operación de WB/INV (ver apartado 7.1.3) de toda la memoria caché tras las ejecución de un actor. Sin embargo esta solución resulta muy poco eficiente, ya que desalojaría de las memorias caché gran cantidad de información válida, reduciendo considerablemente las ventajas ofrecidas por este tipo de memorias. Además, en numerosas ocasiones se llamará a ejecución a actores que no tengan información a procesar, por lo que se procedería a actualizar, de forma innecesaria, información no modificada.

Una solución más eficiente debería invalidar los datos de una FIFO que compartan dos o más actores mapeados en núcleos distintos sólo cuando se haya producido una modificación, y debería actualizar en memoria compartida (operación de WB) sólo los datos modificados fruto de un procesado en el uno de los núcleos afectados, para que éstos nuevos datos estén actualizados para el resto. Además, se deberá evitar realizar operaciones de WB/INV innecesarias que reduzcan el rendimiento de las memorias.

En conclusión, cada vez que un actor escriba datos en una FIFO compartida por dos núcleos se deberá hacer una operación WB, tanto del puntero de escritura, como de los datos a compartir. Cada vez que un actor lea una FIFO, deberá, previamente, invalidar (INV) el puntero de escritura, para saber si se han escrito nuevos datos, y si es así deberá además, invalidar el contenido de la FIFO para actualizar los datos almacenados. Cada vez que un actor haya leído datos, y por tanto deba actualizar su puntero de lectura, tendrá que hacer un WB para que el actor escritor pueda saber la posición de los punteros de lectura y calcular el espacio disponible en la FIFO.

En primer lugar parece oportuno que para cada FIFO utilizada por cada uno de los actores exista un *flag* que indique la necesidad, o no, de realizar operaciones de WB/INV. Dicho *flag* estará activo en los casos en los que el núcleo que ejecuta el actor escritor, y al menos uno de los actores lectores no sea el mismo.

Por lo tanto, y dado el siguiente caso: tres actores (A, B y C) leen de una misma FIFO en la que escribe el actor D. Los actores A, B y D están mapeados en el núcleo 1, mientras que el actor C lo está en el núcleo 2, con los correspondientes cuatro *flags* de aviso: Inv_A, Inv_B, Inv_C y Inv_D. En este caso los *flags* Inv_A e Inv_B deberán quedar a 0, indicando que no es necesario realizar operaciones de WB/INV sobre las FIFOs de esos actores ya que comparten núcleo con el actor escritor. Pero, los *flags* Inv_C e Inv_D deberán estar a 1, ya que están mapeados en núcleos diferentes. En este escenario resulta necesario conocer el número (o identificador) del núcleo en que se encuentran mapeados todos y cada uno de los actores. Conociendo esta información, y comparándola con la de todos los actores, se podrán calcular los valores de los *flags*.

9.4.2.1. Rediseño de las FIFOs

Con el fin de no introducir estructuras de datos excesivamente complejas y globales, se han definido dos nuevos campos en las estructuras de las FIFOs, las cuales se incluyen en el archivo *fifo.h* de la biblioteca *orcc-native*. Un campo indicará el núcleo en el que el actor (escritor o lector) de esa FIFO está mapeado (*core_maps[i]*, *Cm_n*), y otro campo contendrá los *flags* de aviso para las operaciones de WB/INV (*invalidate[i]*, *Inv_n*). La Figura 52 muestra la nueva estructura propuesta de las FIFOs (derecha) contrastada con la original (izquierda). Debe tenerse en cuenta que siempre que se realiza una operación de WB/INV ésta se realiza sobre una sección de la memoria, cuyo tamaño mínimo es el de la línea de caché, que en la plataforma TMS320C6678 es de 128 B. Si se mantuviese la estructura original de la FIFO, Figura 52 (izquierda), se correría el riesgo de borrar o actualizar datos incorrectos de los punteros de escritura o lectura. En este sentido, cada uno de los campos que integran la FIFO deben emplazarse en líneas de tamaño de caché independientes, asegurando así que siempre que se realice una operación de WB/INV ésta se efectúe exclusivamente sobre el campo deseado. Para implementar esta característica se debe añadir un espacio vacío (*padding*) que complete los 128 B de cada línea, como se muestra en la Figura 52 (derecha).

Figura 52. Comparación entre la estructura de las FIFOs original (izquierda) y la modificada (derecha).

9.4.2.2. Inicialización de parámetros

Una vez se ha rediseñado la estructura de las FIFOs se debe diseñar el proceso mediante el cual se calcula si es necesario activar los *flags* de aviso. Este proceso se deberá ejecutar al inicio de la aplicación, tras cargar el mapeo de actores, en caso de distribuciones estáticas, y cada vez que se realice un nuevo mapeo, en el caso de distribuciones dinámicas de actores.

En la Figura 53 se muestran los pasos a dar para inicializar correctamente los nuevos parámetros introducidos en las FIFOs. Seguidamente, se detalla el procedimiento.

Figura 53. Diagrama de trabajo durante la inicialización de parámetros de las FIFOs.

- 1) Se guardan los valores de los núcleos en los que cada actor se encuentra mapeado. Para ello se lee el valor del identificador de cada núcleo y se guarda en el correspondiente campo `Cm_n`. Tras esta operación se debe efectuar una actualización del nuevo valor en la memoria compartida mediante un WB. El código de la Figura 54 muestra un caso de ejemplo con el actor escritor⁴⁷ de la FIFO `DBFilter_DBF_u_FiltSample`.

```
DBFilter_DBF_u_FiltSample->core_maps[0] = omp_get_thread_num();
CACHE_wbInvL2(&dec0_DBFFilter_DBF_u_FiltSample->core_maps[0],4,CACHE_WAIT) ;
```

Figura 54. Sección de código que implementa el registro del número de núcleo.

- 2) Seguidamente, los actores que escriben en alguna FIFO pasan a comprobar si es necesario o no activar los *flags* de WB/INV. En primer lugar se pone por defecto el valor del *flag* del actor que escribe a 0, este se modificará cuando se dé el primer caso en el que el núcleo de un lector y el suyo no coincidan. Tras esto se procede a comparar uno a uno los núcleos en los que están mapeados los lectores con el núcleo del escritor, cada valor que se va leyendo debe actualizarse previamente, para asegurarse de que es el correcto. Si lector y escritor están mapeados en el mismo núcleo, el *flag* de aviso se pone a 0, en caso contrario ambos *flags*, el del lector y el del escritor se ponen 1. Cuando se han evaluado todos los lectores se hace un WB/INV de cada uno de ellos para actualizar los nuevos datos en la memoria compartida.
- 3) Por último comienza el proceso normal de ejecución de actores (comienza la descodificación). En este punto se evalúa si es la primera vez que se ejecuta el planificador, de ser el caso se invalidan todos los *flags* de las FIFOs que comunican con ese actor para asegurar que los valores se actualicen de acuerdo con las posibles modificaciones realizadas en el paso anterior.

9.4.2.3. Sincronización del procedimiento

Dentro del procedimiento descrito en el apartado anterior debe asegurarse que antes de que comience el segundo paso todos los núcleos han finalizado el primer paso.

⁴⁷ El primer campo del array `core_maps` (`Cm_n`) corresponde al actor escritor.

Y de forma similar debe asegurarse que antes de comenzar el paso 3 todos los núcleos han terminado el paso 2. Esto es necesario para que la información acerca de en qué núcleo está mapeado cada actor y la información de los *flags* de aviso está actualizada en la memoria compartida antes de que cada núcleo acceda a ella.

Dicha sincronización se puede llevar fácilmente a cabo mediante el uso de semáforos, cada vez que un núcleo llega a un punto de sincronización manda el aviso al semáforo correspondiente, una vez se hayan recibido un número de señales iguales al número de núcleos utilizados en el mapeo, el núcleo correspondiente podrá continuar la ejecución.

9.4.2.4. *Buffers de prefetch* y efectos sobre el vídeo

Como se introdujo en el apartado 7.1.3, la plataforma TMS320C6678 dispone, para cada núcleo, de un *prefetch buffer*, donde se almacena de forma adelantada información antes de que el procesador la utilice. Sin embargo, cada vez que ejecuta una instrucción para invalidar datos en caché, este *buffer* no se ve afectado. Esta situación puede ocasionar que información desactualizada quede almacenada en el *buffer de prefetch* y sea utilizada por el procesador, ocasionando errores en la decodificación de las imágenes, como se muestra en Figura 55.

Figura 55. Artefactos (en verde) producidos por la no-invalidación de los *buffers de prefetch*.

Para asegurar que todos los datos procesados están actualizados, los *buffers de prefetch* deben ser igualmente invalidados cada vez que se ejecute una instrucción de invalidar datos en caché. Esto se realiza mediante la ejecución del código que se muestra en la Figura 56. El efecto que este borrado puede tener sobre el rendimiento del decodificador ha sido estudiado, resultando que su efecto sobre la eficiencia del procesamiento es despreciable (<1% sobre el rendimiento global). Este factor deberá tenerse en cuenta al utilizar otras plataformas, en cuyo caso se recomienda acudir a la información facilitada por cada fabricante acerca de la gestión de la coherencia de la información de la memoria.

```
XPFCMD = 1;
```

Figura 56. Instrucción para efectuar un borrado del *buffer* de prefetch.

9.4.2.5. Modificación sobre el código de los actores

Finalmente, y tras presentar la problemática existente en la gestión de las FIFOs, y haber sentado las bases de una solución, se debe escribir, o ampliar, el código que implemente la nueva metodología para la gestión de la información y mantener la coherencia entre los niveles de las memorias de la plataforma.

Cada actor integra cuatro funciones prediseñadas para la gestión de los datos de entrada/salida a través de las FIFOs, en la Tabla 23 se listan y describen estas funciones. Los contenidos de las FIFOs se leen y escriben directamente cuando cada uno de los actores trabaja con ellos siempre que, respectivamente, se haya comprobado que hay datos nuevos disponibles o hay espacio libre para escribir.

Función	Funcionalidad
read_<<fifo>>	Se comprueba si hay nuevos datos en la FIFO de entrada
write_<<fifo>>	Se comprueba si hay espacio disponible en la FIFO de salida
read_end_<<fifo>>	Se actualiza el valor del puntero de lectura
write_end_<<fifo>>	Se actualiza el valor del puntero de escritura y de los contenidos nuevos.

Tabla 23. Funciones para la gestión de las FIFOs por parte de los actores.

Estas funciones se deben modificar para incluir las especificaciones descritas a lo largo de este apartado. Además, se debe resolver de forma explícita la gestión de los datos en estas funciones, esto es necesario para evitar tener que introducir modificaciones sobre el código que implementa las diversas funcionalidades de los actores. Como se describió en el apartado 8.3.2, el código que implementa esta funcionalidad está específicamente descrito en la plantilla *Network Printer* del *back-end*.

Teniendo en cuenta todo lo anterior y la solución a la problemática ofrecida al final del apartado 9.4.1, se ha rediseñado el código de las cuatro funciones, de forma eficiente y asegurando la coherencia de los datos almacenados en la memoria, como se muestra en el pseudocódigo de la Tabla 24.

read_<<fifo>>	write_<<fifo>>
<pre> if(Inv_1) //si flag de aviso INV(Wr_1) if(Wr_1-Rd_1) //¿Nuevos datos? INV(contenido_FIFO1) Rd_local* = Rd_1 </pre>	<pre> if(Rd_local - Rd_1) Rd_1 = Rd_local if(Inv_1) WB(Rd_1) </pre>
read_end_<<fifo>>	write_end_<<fifo>>
<pre> if(Inv_1) for(i=1 hasta num_lectores) INV(Rd_1) Wr_local* = Wr_1 </pre>	<pre> if(Wr_local - Wr_1) Wr_1 = Wr_local if(Inv_1) WB(contenido_FIFO1) WB(Wr_1) </pre>

(*) Las variables *Wr/Rd_local* se refieren a las copias locales de los valores de los punteros de lectura/escritura que permiten a los actores gestionar la transferencia de información.

Tabla 24. Pseudocódigo de las nuevas funciones de gestión de FIFOs en los actores.

9.4.3. Generalización

La solución propuesta en este apartado para la gestión del flujo de información y el mantenimiento de la coherencia de datos entre los distintos niveles de memoria es extensible a cualquier plataforma o entorno en el que no se asegure de forma automática dicha gestión. Para ello se deberán implementar todos los pasos descritos anteriormente, a excepción del 9.4.2.4 que es aplicable únicamente a aquellas arquitecturas con *buffers* de *prefetch*. Sólo deberán utilizarse las funciones propias del entorno para realizar las operaciones de WB/INV, en lugar de las utilizadas en esta tesis y que son propias de Texas Instruments para sus plataformas DSP⁴⁸.

9.5. Mapeo de algoritmos a núcleos

La tarea de distribuir de forma eficiente el procesamiento de los distintos actores que integran un descodificador entre los núcleos disponibles en una plataforma no es trivial. Distintos aspectos como la carga computacional, la cantidad de información transferida entre los núcleos y la dependencia de datos entre unidades funcionales deben ser tenidos en cuenta.

9.5.1. Planteamiento del problema

En el apartado 4.5.4 se introdujeron los conceptos sobre los que se apoyan los nuevos algoritmos que permiten realizar distribuciones dinámicas de actores en tiempo de ejecución para aplicaciones basadas en especificaciones RVC-CAL y Orcc. Esta funcionalidad resulta más ventajosa cuanto mayor es el número de actores que integran una aplicación y/o mayor es el número de núcleos disponibles en la plataforma. En este sentido, y de forma similar a las características descritas a lo largo de este capítulo, se debe realizar la migración e integración, según las necesidades de las plataformas, de las secciones de código encargadas de calcular y aplicar mapeos dinámicos de actores en tiempo de ejecución.

También existe la posibilidad de realizar distribuciones de actores a núcleos de forma manual y estática. Si bien esta opción es menos ventajosa que la ofrecida por los mapeos dinámicos, sí puede resultar de interés cuando el programador tiene la experiencia suficiente como para realizar una distribución de forma manual, o también puede emplearse con la intención de realizar una evaluación específica del rendimiento de la aplicación sobre un entorno multinúcleo.

Sin embargo, ninguna de las dos opciones está implementada para plataformas Multi-DSP. La integración en el código de ambas soluciones, de forma particularizada para cada aplicación, conllevaría un excesivo coste de desarrollo, teniendo en cuenta el elevado número de FIFOs y actores implicados en este tipo de aplicaciones. Además tampoco existe una solución al manejo de distribuciones multinúcleo, tanto estáticas como dinámicas, con directivas OpenMP en lugar de con *threads*.

⁴⁸ En el caso de las plataformas de la familia C66 de Texas Instruments las definiciones de las funciones que realizan operaciones de WB/INV se encuentran en el fichero `csl_xmcAux.h`.

Por último, y del mismo modo que con anteriores aspectos, la adaptación del código fuente debe mantener las bases de funcionamiento originales. Así, la utilización de mapeos dinámicos no debe afectar al resto de funcionalidades descritas anteriormente, y debe, por ejemplo, complementarse con el método propuesto para el mantenimiento de la coherencia entre los diferentes niveles de memoria. A partir de esta propuesta, el funcionamiento de las aplicaciones seguirá el diagrama de la Figura 57.

Figura 57. Diagrama de flujo trabajando con mapeos dinámicos o estáticos.

9.5.2. Solución propuesta

De la misma forma que en los apartados anteriores, en este caso se ha llevado a cabo un análisis de las secciones de código fuente encargadas de implementar la funcionalidad del mapeo de actores a núcleos, tanto estáticos como dinámicos. Una vez identificadas todas las secciones susceptibles de sufrir modificaciones, se han buscado las soluciones oportunas, y mediante el uso de construcciones `#ifdef` - `#else` - `#endif`, asociadas a la definición de la constante `MDSP_ENABLE`, se ha incluido el código necesario.

La implementación de mapeos estáticos no requiere de desarrollo adicional a las soluciones ya aportadas en los apartados 9.3 y 9.4.2.2). Sin embargo, sí se ha llevado a cabo un análisis del código que implementa el mapeo dinámico de actores y se aportan las soluciones necesarias en este sentido.

9.5.2.1. Análisis y modificaciones sobre el código fuente

Como se detalló en el apartado 4.5.4, la implementación de mapeos dinámicos se basa en la toma de medidas de distintos parámetros de la ejecución de los actores. En este sentido la primera tarea de migración del código consiste en adecuar la toma de medidas a los recursos internos de la plataforma Multi-DSP. Mientras el código fuente original se sirve de funciones nativas del SO para la toma de medidas de tiempo sobre plataformas GPP, como la función *getticks()*, en la plataforma Multi-DSP se utiliza un *timer* de 64 bits.

Durante la ejecución en bucle del planificador se miden distintos parámetros, tales como los ciclos de reloj consumidos por cada actor o la carga de las comunicaciones sobre las FIFOs. En la Tabla 25 se resumen todos los parámetros medidos.

Parámetros	Utilidad
my_actor->ticks	Número total de ciclos de reloj consumidos por un actor
my_actor->misses	Número de veces que el actor ha sido llamado a ejecución pero no ha disparado ninguna acción.
My_actor->switches	Número de veces totales que se ha llamado a ejecución al actor.
Rate <<nombre_FIFO>>	Ratio de utilización de la FIFO (<i>tokens</i> transmitidos).

Tabla 25. Parámetros registrados para el cálculo de mapeos dinámicos.

El *timer* utilizado mide ciclos de reloj del sistema (CLKs), y tiene la particularidad de no poder pararse ni reiniciarse una vez arrancado. Debido a esto todos los tiempos se calcularán de forma diferencial entre dos puntos de lectura del valor del *timer*. El proceso de toma de medidas se presenta en la Figura 58.

```

while (1) {
    my_actor = sched_get_next_schedulable(sched); //Próximo actor a ejecutar
    if(my_actor != NULL){
        #ifdef MDSP_ENABLE
            tick_in = CSL_tscRead();                //Lectura del timer (DSP)
        #else
            tick_in = getticks();                    //Lectura reloj (GPP)
        #endif
        ...
        //Ejecución actor
        ...
        if (si.num_firings != 0) { //Si el actor ha disparado alguna acción
            #ifdef MDSP_ENABLE
                tick_out = CSL_tscRead();            //Lectura del timer (DSP)
                diff_tick = tick_out - tick_in;       //Tiempo de ejecución del actor
            #else
                tick_out = getticks();                //Lectura reloj (GPP)
                diff_tick = elapsed(tick_out, tick_in);
            #endif
            //Se acumula el tiempo de ejecución del actor (DSP y GPP)
            ...
            my_actor->ticks += diff_tick;
        } else {
            my_actor->misses++;                       //Si no se ejecutado ninguna acción
        }
        my_actor->switches++;                          //Transiciones del actor
    } }

```

Figura 58. Toma de medidas durante la ejecución del planificador.

También es necesario que el actor *Source* esté siempre mapeado en el núcleo 0. Este actor implementa la lectura del fichero de vídeo codificado y procede a transmitir por la FIFO única que lo comunica con el *Parser* (ver apartado 10.1.1) los paquetes que contienen las tramas a descodificar. Por defecto, al comenzar el proceso de descodificación, el actor *Source* se encuentra mapeado en el núcleo 0, y durante la inicialización del mismo el acceso al fichero de vídeo codificado queda configurado. Si se cambia la ubicación, se produce un error en la primera lectura de fichero desde el nuevo núcleo donde se haya mapeado este actor. Sin embargo, en el caso de utilizar mapeos dinámicos resulta necesario obligar a que este actor quede en el núcleo 0. Para ello se ha modificado ligeramente la función *set_mapping_from_partition* del archivo *mapping.c* que forma parte de la biblioteca *orcc-runtime*. Esta función codifica la nueva distribución de actores calculada por los algoritmos de mapeo en los correspondientes nuevos parámetros de los planificadores.

9.5.2.2. Compilación de la biblioteca METIS

Como se introdujo en el apartado 4.5.4, existen distintos algoritmos para realizar la repartición de actores entre los núcleos a partir de los datos registrados en tiempo de ejecución. Varios algoritmos de mapeo están implementados a partir de la biblioteca METIS⁴⁹ [MET], la cual se encuentra fácilmente disponible y compilable para plataformas GPP pero no para DSPs. En este sentido se ha procedido a descargar el

⁴⁹ Dado que METIS se ha utilizado en esta tesis doctoral únicamente como herramienta auxiliar no se explicará en detalle.

código fuente de METIS para, posteriormente, realizar su migración al entorno DSP, compilarlo y poder utilizarlo en la fase de pruebas.

La biblioteca METIS que implementa los algoritmos de mapeo MR, MKEC y MKCV (ver apartado 4.5.4) utilizados en este entorno de trabajo se integra a su vez por dos bibliotecas libGK y lib_metis. En libGK se incluyen rutinas necesarias en lib_metis para la utilización de recursos nativos del sistema, como *timers* o hilos de ejecución⁵⁰, y funciones matemáticas complejas, como el cálculo de logaritmos en base 2 y manejo de matrices complejas. La biblioteca lib_metis integra los algoritmos que implementan las distintas funcionalidades de METIS. Dado que la ejecución de algunos de estos algoritmos puede suponer un coste computacional elevado, algunas secciones de código se encuentran ya paralelizadas mediante directivas OpenMP, lo que permite mejorar el rendimiento del algoritmo.

OpenMP está mayoritariamente presente en las funciones utilizadas para manejar matrices CSR, en estos casos se hace uso de la directiva *omp for* y *omp parallel if*. Si la versión utilizada de OpenMP no soportase alguna de las directivas utilizadas por METIS las secciones de código afectadas deberán excluirse de compilación, dejando la ejecución de dichas secciones sin paralelismo. Además, sobre el código de la versión de METIS v5.1 utilizada ha sido necesario modificar el uso de otras funciones que típicamente hacen uso de recursos de SOs como Linux o Windows, y que no se encuentran disponibles para RTOS en plataformas Multi-DSP de Texas Instruments. En la Tabla 26 se citan las funciones que han sido modificadas, o excluidas de compilación cuando su omisión no afecta al correcto funcionamiento del algoritmo de METIS.

Función	Alternativa implementada
strerror(); Getrusage();	Excluidas de compilación. Se presentará mensaje por pantalla en caso de error.
stat();	Se utiliza en la comprobación de estado de ficheros y/o directorios, por lo que no es necesaria en este caso. Se excluye de la compilación y se comprueba que no afecta al funcionamiento.

Tabla 26. Modificación de funciones en las bibliotecas de METIS.

9.5.3. Generalización

Las soluciones aportadas para el mapeo de actores a núcleos, tanto de forma estática como dinámica, se han implementado de tal forma que pueden utilizarse sobre cualquier plataforma con soporte para OpenMP, además de estar particularizadas para plataformas Multi-DSP. En este sentido, la Figura 59 muestra el diagrama de funcionamiento comparativo entre el código fuente original con *threads* y la solución para mapeos dinámicos con OpenMP. Como se puede observar el flujo de funcionamiento del descodificador no se ve afectado.

⁵⁰ En [MET] se encuentra disponible una versión OpenMP de METIS.

Figura 59. Diagrama del funcionamiento general del descodificador con *threads* y OpenMP.

El proceso de generalización pasa fundamentalmente por adaptar a la plataforma objetivo el proceso de toma de medidas detallado en la Figura 58. Dado que se ha utilizado el mismo procedimiento *#ifdef* - *#endif* seguido en los apartados anteriores para la migración del código fuente se facilita la adaptación del mismo a una plataforma distinta de las contempladas en esta tesis.

Por otra parte el proceso de migración de la biblioteca METIS a un entorno distinto de los aquí contemplados no puede garantizarse. Esto es debido principalmente a la gran cantidad de ficheros y líneas de código que integran esta biblioteca, un análisis

en profundidad del código fuente de la biblioteca queda fuera de los objetivos de esta tesis.

9.6. Control de consumo

El consumo de energía de los sistemas empotrados o de aquellos sistemas que van a ser utilizados en dispositivos cuya fuente de alimentación son baterías es un factor que no puede ser ignorado y que se debe tratar de reducir en la medida de lo posible. Si bien este aspecto no forma parte de los objetivos principales de esta tesis doctoral, sí se ha realizado un acercamiento a la temática mediante el estudio de posibles mejoras del consumo energético del procesador TMS320C6678.

9.6.1. Planteamiento del problema

La implementación de metodologías que permitan hacer un uso eficiente de los recursos energéticos sobre sistemas empotrados cuya fuente de alimentación es una batería es cada vez más necesaria. Sin embargo, las diferencias del *hardware* que integran las distintas plataformas multinúcleo dificulta la tarea de desarrollar una metodología multiplataforma.

La mayor parte del consumo energético del procesador Multi-DSP proviene de la alimentación de los núcleos del procesador y de la velocidad de funcionamiento de los mismos. Por esta razón las técnicas utilizadas se deben centrar en reducir el consumo producido por los núcleos del procesador. Sin embargo, el procesador TMS320C6678 se encuentra integrado en la plataforma TMDSEVM6678, la cual dispone de otros dispositivos, como una FPGA, interfaces Ethernet y de comunicación serie, diodos LED, etc. Dado que ni la plataforma TMDSEVM6678, ni el procesador TMS320C6678 disponen de puntos de medida sólo será posible muestrear el consumo global de la plataforma. Así, se ha realizado un modelo del consumo de la plataforma, mediante el cual se puedan establecer los escenarios de consumo óptimo (en función del número de procesadores y frecuencia de reloj) en cada momento. La aplicación del descodificador deberá ser capaz de desplazarse dinámicamente de un escenario a otro en función de las necesidades.

9.6.2. Solución propuesta

El desarrollo de un modelo de consumo permite identificar, en tiempo real, de una forma rápida y sencilla la mejor configuración de los escenarios de funcionamiento del sistema para unas prestaciones dadas. Teniendo en cuenta que la plataforma dispone de ocho núcleos de procesamiento que pueden utilizarse de forma simultánea para la descodificación de vídeo, resulta de gran utilidad conocer qué cantidad de núcleos y frecuencia de funcionamiento del sistema es suficiente para procesar un vídeo, y a partir del cual se podría prescindir del resto de unidades de procesamiento y/o modificar la frecuencia del reloj del sistema para reducir el consumo energético.

Las plataformas de la familia KeyStone del fabricante Texas Instruments ofrecen diversos modos de configuración que permiten modificar el consumo energético de los

dispositivos. En función de las necesidades del sistema final se pueden desconectar periféricos, apagar aquellos núcleos del procesador que no sean necesarios, variar la frecuencia del reloj del sistema y modificar la frecuencia de funcionamiento de la memoria externa DDR3.

Para la alimentación de la plataforma y la toma de medidas se ha utilizado una fuente de alimentación marca Agilent modelo 66321D [AGI], conectada a un PC donde mediante un *software* específico se puede realizar un muestreo, tanto de la corriente, como del voltaje suministrados a la plataforma. El periodo de muestreo puede establecerse entre 5 milisegundos y 20 milisegundos.

En la elaboración del modelo de consumo han influido por tanto diversos aspectos que tendrán en cuenta por una parte las características del vídeo, y su impacto en el ratio de decodificación, y por otra a los parámetros de consumo modificables de la plataforma. Se han utilizado dos técnicas que permiten modificar dichos parámetros pudiendo adaptarse de forma dinámica las necesidades del descodificador: (1) la entrada en modo *idle* de los procesadores (apartado 9.6.2.1) y (2) el cambio de la frecuencia de reloj (apartado 9.6.2.2). En base a estas dos técnicas se define el modelo de consumo que, posteriormente, podrá integrarse en el código del descodificador (apartado 9.6.2.3) para estimar el consumo en tiempo real y tomar decisiones que permitan reducir el consumo energético de la plataforma.

9.6.2.1. Modo IDLE

Si bien los núcleos que no vayan a ser utilizados pueden apagarse completamente, reduciendo por tanto su consumo energético a cero, esta acción resulta poco práctica, ya que para encenderlos en caso de que fueran a utilizarse es necesario reiniciar completamente el sistema. Uno de los objetivos de la metodología propuesta en este trabajo es dotar al sistema de cierto grado de adaptación dinámica a los requisitos de procesamiento en cada momento. Como se ha expuesto en el apartado 9.5, al utilizar configuraciones de mapeos dinámicos, y durante la decodificación de una secuencia de vídeo, el número de núcleos utilizados puede variar de forma dinámica. En este sentido resulta necesario poder apagar o poner en *standby* a aquellos núcleos que en un momento dado no son necesarios, pero que puedan reactivarse de forma instantánea en caso de que aumente el número de núcleos necesario.

El procesador TMS320C6678 dispone, entre otros, del modo de funcionamiento *idle* (ocioso). Cuando un núcleo entra en modo *idle* pasa a estar a efectos prácticos desconectado, sin embargo, puede ser fácilmente reactivado mediante el uso de alguna de las interrupciones del sistema. De forma aproximada, la diferencia de potencia requerida por un núcleo en modo *idle* o funcionando a pleno rendimiento es de 450 milivatios.

Para poner en modo *idle* a alguno de los núcleos de la plataforma es necesario realizar la siguiente serie de acciones:

- 1) Habilitar el modo *idle* mediante la activación del campo MEGPD, el cual se encuentra en el registro PDCCMD, tal y como se muestra en las instrucciones de la Figura 60.
- 2) Habilitar las interrupciones⁵¹ que se vayan a utilizar para sacar de modo *idle* a los núcleos que pasen a este modo, también se deben deshabilitar el resto de interrupciones.
- 3) Ejecutar la instrucción de entrada en modo *idle*, como se muestra al final de la Figura 60.

```
#define REG(x) (*((volatile Uint32 *)(x)))
#define PDCCMD_REG      0x01810000
#define PDCCMD_IDLE     0x00010000
(...)
REG(PDCCMD_REG) |= PDCCMD_IDLE;
(...)
asm("idle");
```

Figura 60. Instrucciones para activar el modo *idle*.

Siguiendo los pasos anteriormente descritos resulta sencillo poner en modo *idle* a aquellos núcleos que no vayan a ser utilizados de forma indefinida (hasta que se reinicie el procesador o se interrumpa el modo *idle*). En el caso concreto de un decodificador de vídeo basado en RVC, los núcleos que no tengan mapeado ningún actor en su planificador podrán ponerse de forma indefinida en modo *idle*, si se están utilizando mapeos estáticos, o hasta el próximo mapeo calculado en el caso de utilizar mapeos dinámicos. De esta forma el diagrama de funcionamiento incluyendo esta nueva funcionalidad queda como se muestra en la Figura 61.

⁵¹ La gestión de las interrupciones del procesador TMS320C6678 es compleja y dispone de distintos vectores configurables. En el Anexo V se facilita la sección de código utilizada en la implementación de las pruebas realizadas para la gestión de las interrupciones.

Figura 61. Diagrama de funcionamiento con modo *idle* de un descodificador basado en RVC.

En aquellos casos en los que se desee sacar uno o varios núcleos del modo *idle* es necesario realizar la configuración de la interrupción encargada de tal propósito, previamente habilitada como se describía en el paso 2 del índice anterior. En la Figura 62 se presenta el código que implementa la solicitud de la interrupción encargada de sacar, desde el núcleo 0, del modo *idle* a los núcleos 1 y 2.

```

if(MultiProc_self()==0){           //Si es el núcleo 0
    CSL_IPC_genGEMInterrupt(1,0); //Interrupción desde el núcleo 0 al núcleo 1
    for(i = 0; i < 10000; i++); //Espera
    CSL_IPC_genGEMInterrupt(2,0); //Interrupción desde el núcleo 0 al núcleo 2
    for(I = 0; I < 10000; i++); //Espera
}
  
```

Figura 62. Código de llamada a las interrupciones para sacar del modo *idle* a dos núcleos.

9.6.2.2. Cambio de la frecuencia de reloj

El otro factor para el control del consumo energético introducido es la modificación de la frecuencia del reloj de los núcleos. El reloj del sistema es común a todos los núcleos y su modificación afecta por igual a todos los que se estén utilizando o vayan a utilizarse. Además, la frecuencia de funcionamiento puede modificarse de forma dinámica, sin necesidad de reiniciar el sistema o detener la ejecución del programa, por lo que resulta muy útil a la hora de adaptarse a posibles circunstancias cambiantes en los requisitos de la aplicación en ejecución. La frecuencia del sistema se genera mediante un lazo de seguimiento de fase (o *Phase-Locked Loop* – PLL), cuyos registros son accesibles y pueden modificarse, variando así la frecuencia de funcionamiento.

La Figura 63 muestra el diagrama de bloques del circuito mediante el que se generan las señales de reloj de los núcleos del procesador TMS320C6678. Como se puede observar, la señal de reloj SYSCLK1 que finalmente conecta con los núcleos del procesador es el resultado de la modificación de una fuente de reloj (CORECLK) inicial mediante los parámetros que hayan sido establecidos para el PLL y su controlador. La Fórmula 3 permite hallar el valor de la señal de reloj SYSCLK a partir del diagrama del PLL. La información detallada acerca del PLL, así como del resto de registros de configuración aquí mencionados, puede encontrarse en la documentación facilitada por el fabricante del procesador [C6678].

Figura 63. Diagrama del circuito para la generación de las señales de reloj del procesador TMS320C6678.

$$SYSCLK_x = CORECLK * \frac{PLLM + 1}{(DIV + 1) * (PLLD + 1)}$$

Fórmula 3. Fórmula para calcular el valor de la señal SYSCLK.

La Tabla 27 muestra los valores que deben cargarse en los registros del PLL para tres frecuencias de funcionamiento.

Frecuencia de 0,8 GHz	Frecuencia de 1,0 GHz	Frecuencia de 1,2 GHz
MAINPLLCTL0=0x07000000; PLL1_PLLM=0x0F;	MAINPLLCTL0=0x09000000; PLL1_PLLM=0x13;	MAINPLLCTL0=0x0B000000; PLL1_PLLM=0x17;

Tabla 27. Instrucciones para la modificación de la frecuencia del reloj.

Para poder modificar los registros del PLL es necesario desbloquear previamente los registros KICK⁵², tal y como se muestra en las instrucciones de la Figura 64.

```
#define REG(x) (*((volatile Uint32 *) (x)))
#define KICK0 0x02620038 //Definición y posición de KICK0
#define KICK1 0x0262003C //Definición y posición de KICK1
#define KICK0_UNLOCK (0x83E70B13) //Valor de desbloqueo de KICK0
#define KICK1_UNLOCK (0x95A4F1E0) //Valor de desbloqueo de KICK1
#define KICK_LOCK 0 //Valor de bloqueo de KICKs
```

⁵² El procedimiento de bloqueo/desbloqueo de los registros KICK asegura que no se realizará ninguna modificación indeseada sobre los registros MMR (*Memory-Mapped Registers* – Registros mapeados en memoria), que contienen la configuración del PLL.

```

REG(KICK0) |= KICK0_UNLOCK;    //Desbloqueo de KICK0
(...)                          //Cambio de frecuencia
REG(KICK0) = KICK0_LOCK;       //Bloqueo de KICK0

```

Figura 64. Instrucciones para el bloqueo/desbloqueo de los registros KICK.

9.6.2.3. Modelo de consumo

Una vez han sido detallados los procedimientos para modificar el consumo energético de la plataforma se procede a caracterizar el consumo de la misma. Para ello se introducen en el código las secciones de código que permiten poner al procesador en todas las situaciones contempladas, es decir 3 frecuencias de trabajo distintas y 5 distribuciones estáticas de actores a núcleos (para 1, 2, 4, 6 y 7 núcleos⁵³), mientras los núcleos sin actores asignados entran en modo *idle*.

Dado que no existen puntos de medida sobre la plataforma, el consumo energético se calcula de forma diferencial. Mediante la fuente de alimentación Agilent se monitoriza el proceso de descodificación completo y se identifican los distintos escenarios. La Figura 65 muestra la variación del consumo de corriente desde el encendido de la plataforma TMDSEVM6678 hasta el inicio del proceso de descodificación de una secuencia de vídeo. Cada uno de los recuadros rojos se corresponde con distintas situaciones (A~F) que se explican a continuación.

Figura 65. Consumo de corriente durante el proceso de arranque de la plataforma TMS320C6678.

⁵³ Sólo se utilizan hasta 7 núcleos en previsión de necesitar un octavo núcleo para la ejecución de la *agent_routine* (ver apartado 4.5.4)

- A. Encendido de la plataforma y transitorio de estabilización del consumo.
- B. Consumo de la plataforma en vacío (I_0), sin conexión del emulador a los núcleos.
- C. Proceso de encendido y carga del código en cada uno de los núcleos del procesador.
- D. Procesador/es a la espera de arrancar el programa cargado (I_1) y los núcleos arrancados.
- E. Inicio del programa, lectura de ficheros de mapeo estático y modificación de la frecuencia de reloj a 0,8 GHz, por defecto es 1,0 GHz (I_2).
- F. Descodificación del vídeo, primero con un núcleo, luego con dos y finalmente con tres. En todos los casos el resto de núcleos pasan a modo *idle*. Las oscilaciones en el consumo se deben a la ejecución de las instrucciones de lectura del fichero del vídeo codificado⁵⁴ (I_d).

La Fórmula 4 muestra las relaciones utilizadas para calcular los valores de interés a la hora de caracterizar el consumo del procesador TMS320C6678.

$$\begin{aligned} \text{Diferencia de consumo de un núcleo trabajando a 1 GHz vs 0,8 GHz} &\rightarrow I_2 - I_1 \\ \text{Consumo del procesador durante la decodificación} &\rightarrow I_d - I_0 \end{aligned}$$

Fórmula 4. Cálculo diferencial de consumos a partir de las medidas realizadas durante el proceso de arranque.

La variación del consumo respecto de la frecuencia de funcionamiento del sistema y del número de núcleos, para cada uno de los escenarios contemplados, se presenta en la gráfica de la Figura 66. En este caso se muestra la corriente demandada por el procesador TMS320C6678 en amperios (A) (eje Y), para tres frecuencias de reloj y un número variable de núcleos a pleno funcionamiento (eje X). Los núcleos sin actores mapeados se encuentran en modo *idle*.

⁵⁴ Cada vez que el decodificador lee del fichero donde se encuentra el vídeo codificado el procesador interrumpe su ejecución.

Figura 66. Evolución de la corriente demandada por la plataforma TMDSEVM6678 expresada en amperios, para distintas frecuencias de reloj y número de núcleos.

Como se puede observar en la Figura 66, el consumo derivado de una mayor frecuencia de reloj aumenta con el número de núcleos que se estén utilizando. Por ejemplo, para un solo núcleo la diferencia de corriente al configurar una frecuencia de 0,8 GHz ó 1,2 GHz es ligeramente inferior a 50 mA, pero al utilizar seis núcleos dicho margen se ve incrementado hasta los 110 mA. Dado que la alimentación de la plataforma es de 12 voltios (V), la diferencia de potencia requerida para este último caso es superior a los 1,32 vatios (W). Además, la evolución del consumo sigue una tendencia lineal en función del número de núcleos utilizados, lo que permite interpolar los valores para 3 y 5 núcleos, en el caso de ser necesario.

La Figura 67 presenta la evolución del consumo energético en el mismo escenario de la Figura 66, pero desglosado por frecuencias y número de núcleos utilizados durante la decodificación.

Figura 67. Corriente consumida por el procesador TMS320C6678 para distintos escenarios de frecuencia y número de núcleos.

Una vez se han caracterizado los consumos para cada uno de los escenarios se calcula la potencia demandada tanto por el procesador como por la plataforma⁵⁵. La potencia se calcula como el producto de la corriente demanda en cada momento por la tensión de alimentación, que es constante a 12V. Estos valores se introducen en el código fuente mediante dos matrices de valores, como se muestra en la Figura 68. Estos valores serán utilizados para establecer el punto de consumo energético óptimo durante la decodificación.

```
// Matriz de valores de consumo en W de la plataforma en función de
// la frecuencia y el número de núcleos.
Const double l33ecto_consumo_target[8][4]={
//Freq 0    0.8    1    1.2 GHz
{0,    0,    0,    0},//para 0 núcleos
{0, 12.696, 13.068, 13.452},//para 1 núcleo + remapeador
{0, 13.080, 13.464, 13.884},//para 2 núcleos + remapeador
{0, 13.356, 13.830, 14.334},//para 3 núcleos + remapeador
{0, 13.632, 14.196, 14.784},//para 4 núcleos + remapeador
{0, 13.920, 14.556, 15.192},//para 5 núcleos + remapeador
{0, 14.208, 14.916, 15.600},//para 6 núcleos + remapeador
{0, 14.496, 15.276, 16.020},//para 7 núcleos + remapeador
};

// Matriz de valores de consumo en W del procesador en función de
// la frecuencia y el número de núcleos.
Const l33ector tabla_consumo_nucleos[8][4]={
//Freq 0    0.8    1    1.2 GHz
{0,    0,    0,    0},//para 0 núcleos
```

⁵⁵ Los datos de consumo se han caracterizado mediante la decodificación de la secuencia BQMall QP 32 que sirve como referencia. Esto es posible ya que aunque existen pequeñas diferencias en el consumo energético entre unas secuencias y otras, éstas no alteran el modelo obtenido.

```

{0, 3.228, 3.600, 3.984}, //para 1 núcleo + remapeador
{0, 3.612, 3.996, 4.416}, //para 2 núcleos + remapeador
{0, 3.888, 4.362, 4.866}, //para 3 núcleos + remapeador
{0, 4.164, 4.728, 5.316}, //para 4 núcleos + remapeador
{0, 4.452, 5.088, 5.724}, //para 5 núcleos + remapeador
{0, 4.740, 5.448, 6.132}, //para 6 núcleos + remapeador
{0, 5.028, 5.808, 6.552}, //para 7 núcleos + remapeador
};

```

Figura 68. Declaración de matrices con los escenarios de consumo.

Como se puede observar a partir de los datos obtenidos, los cambios en la frecuencia de trabajo conllevan variaciones en el consumo menores que las producidas por cambios en el número de núcleos utilizados. De esta forma se utilizarán las modificaciones en el número de núcleos activos como ajuste grueso del consumo y las variaciones de la frecuencia de trabajo como ajuste fino del mismo.

9.6.2.4. Integración de las técnicas de reducción de consumo en el código fuente

Todos los aspectos relativos a la modificación del consumo energético evaluados en este apartado han sido integrados en el código fuente del descodificador siguiendo las mismas premisas de interoperatividad aplicados en los apartados anteriores de la metodología. Dado que el código del algoritmo que implementa el control de consumo está particularizado para la plataforma TMS320C6678, las nuevas funcionalidades quedarán supeditadas a la activación de una nueva macro PCRED_ENABLE, de forma similar a como se han utilizado hasta ahora OPENMP_ENABLE y MDSP_ENABLE.

El diagrama de la Figura 69 esquematiza el funcionamiento del descodificador atendiendo a las medidas para el control del consumo energético. A partir del algoritmo desarrollado el objetivo será mantener una tasa de decodificación (FPS) lo más estable y constante posible, para ello se modificarán dinámicamente tanto las distribuciones de actores (número de núcleos) como la frecuencia de trabajo del procesador. Como se puede observar, cada cierto tiempo (o número n de imágenes decodificadas) establecido por el usuario se realiza una consulta. Esta consulta evalúa la tasa de imágenes decodificadas por segundo, y si es necesario provoca una petición de cambio en la configuración.

Figura 69. Diagrama de funcionamiento del proceso de decodificación con control de consumo energético.

Para realizar un cambio (frecuencia o mapeo) todos los núcleos deben interrumpir las tareas en ejecución y sincronizarse antes de llevar a cabo cualquier modificación. Entonces, el algoritmo evalúa la diferencia entre la tasa de decodificación actual y la establecida por el usuario. En función de la diferencia existente se tomarán las siguientes medidas y se continúa con la decodificación:

- Si la diferencia es inferior al 15%⁵⁶: realiza un cambio de la frecuencia de funcionamiento (siempre que éste sea posible por no estar ya al máximo o mínimo).
- Si la diferencia es superior al 15%, o si no se puede modificar la frecuencia por estar ya al máximo o mínimo para un número de núcleos dado: realiza un cambio del número de núcleos o ambos, y por tanto un nuevo mapeo.

La integración del código que implementa las técnicas para la reducción del consumo energético se ha llevado a cabo prestando especial atención al tipo de tareas desempeñadas por cada núcleo. De esta forma se identifican los siguientes tipos de núcleos según su funcionalidad:

- Núcleo principal (siempre será el núcleo 0): éste es el núcleo que inicia el proceso de decodificación y que ejecuta las directivas de OpenMP encargadas de iniciar el procesamiento paralelizado. Además, es sobre la rutina del planificador de este núcleo donde se lleva a cabo la comprobación del número de imágenes decodificadas. Aprovechando esta circunstancia se incluye en la misma sección de código la consulta que evalúa si se dan las circunstancias para cambiar la frecuencia de trabajo y/o el número de núcleos. Por último, éste núcleo es el encargado de provocar las interrupciones, en caso de ser necesario, encargadas de sacar del modo *idle* al resto de núcleos.
- Núcleo remapeador (rutina *agent*): éste es el núcleo encargado de la ejecución de la *agent_routine* (ver apartado 4.5.4), que implementa el procedimiento para el cálculo

⁵⁶ Los márgenes establecidos han sido fijados de forma aproximada para validar el correcto funcionamiento de las técnicas de reducción de consumo energético.

y aplicación de mapeos dinámicos. Aprovechando esta circunstancia se incluye en el código de la misma rutina la información relativa al modelo de consumo de la plataforma necesario para establecer la configuración de trabajo óptima en cada momento, en función de las tablas del modelo de consumo mostradas en la Figura 68.

- Núcleo/s descodificador/es: el funcionamiento del resto de núcleos no se ve alterado por la parte de control de consumo energético. Esto facilita la migración a otros entornos.

De esta manera la solución adoptada aporta una serie de criterios que permiten llevar a cabo reconfiguraciones dinámicas del descodificador, y de los recursos utilizados, en tiempo real.

9.6.3. Generalización

Todo el código fuente que implementa la solución propuesta puede ser reutilizado para otras plataformas distintas de la aquí empleada, únicamente será necesario adaptar las instrucciones que ponen los núcleos en modo *idle* y que varían la frecuencia de trabajo a las propias del entorno de trabajo de la plataforma objetivo. Esto implicaría modificar únicamente las secciones de código correspondientes a este apartado en los ficheros *scheduler.c* y *mapping.c* de la biblioteca *runtime*.

Algunas plataformas actuales disponen de mecanismos de reducción del consumo energético propios, así como sistemas de medida del consumo en tiempo real; en estos casos se deberá acudir a la información del fabricante para implementar las técnicas que ofrezcan un uso más eficiente de los recursos energéticos de la plataforma. Sin embargo, y aún en éstos supuestos, tanto el algoritmo desarrollado, como las técnicas para modificar la frecuencia de trabajo y el número de núcleos siguen siendo válidas.

9.7. Mejora del rendimiento mediante módulos optimizados

El código fuente de cualquier aplicación es susceptible de optimizarse para obtener mejoras en el rendimiento de la misma. Sin embargo, los procesos de optimización de código requieren de tiempos de desarrollo elevados y de programadores con experiencia en las arquitecturas objetivo para las que se esté desarrollando la aplicación.

En este sentido, se ha tenido en cuenta en este trabajo de investigación, la inclusión en la metodología de diseño, la utilización de módulos cuyo código ha sido optimizado, sin que dicha integración afecte al resto de la metodología.

9.7.1. Planteamiento del problema

La utilización de módulos de código optimizado está integrada en la generación con Orcc de código fuente para plataformas tipo GPP, tanto ARM como para arquitecturas x86. Sin embargo, no ocurre así para plataformas tipo DSP.

Como se ha descrito en el apartado 8.4 existen dos niveles de mejora para las plataformas GPP, (1) el que utiliza secciones de código escrito en C del descodificador OpenHEVC (capítulo 5) para implementaciones de descodificadores RVC, y (2) la utilización de instrucciones SIMD para arquitecturas compatibles con el mismo. Ambos niveles se encuentran combinados, ya que la mayoría de optimizaciones que utilizan instrucciones SIMD están integradas en el código de OpenHEVC.

Sin embargo, ninguno de los dos niveles de mejora mencionados es compatible con plataformas DSP. El primero (1) requiere de una versión del descodificador migrada y compilada para la plataforma DSP sobre la que se desee trabajar. En cuanto al segundo (2), el repertorio de instrucciones no existe para plataformas DSP, en su lugar podrían utilizarse instrucciones en ensamblador específicas de estas arquitecturas escritas directamente por el programador para que aprovecharan las ventajas de la arquitectura DSP.

Además, la utilización de ambas optimizaciones debe ser activada por el usuario durante el proceso de pre-compilación mediante las macros específicas (OPENHEVC y USE_SSE) disponibles al utilizar CMake (ver apartado 8.2.3). Como se ha mencionado anteriormente, el flujo de trabajo sobre plataformas tipo DSP no contempla la utilización de herramientas como CMake, por lo que deberá ofrecerse una solución a este problema.

9.7.2. Solución propuesta

En el apartado 8.4 se describió el procedimiento que permite utilizar secciones de código optimizado para una arquitectura específica sobre implementaciones basadas en código fuente generado por Orcc. Este procedimiento permite mantener el flujo de trabajo intacto al tiempo que no afecta a la funcionalidad de la solución sobre otras plataformas incompatibles con las optimizaciones desarrolladas. Esta solución permite conseguir mejoras importantes en el rendimiento de las implementaciones aprovechando las características de las arquitecturas de las plataformas objetivo.

En este sentido, la primera solución pasa por migrar el código de la implementación del descodificador OpenHEVC al entorno de la plataforma Multi-DSP TMDSEVM6678. Este proceso de migración permite por un lado disponer de una solución funcional de un descodificador HEVC escrita en lenguaje C sobre la misma plataforma Multi-DSP que se utilizará como referencia; y por otro lado permite aprovechar determinadas secciones del nuevo código migrado sobre las implementaciones basadas en RVC y Orcc. A continuación se detallan los pasos seguidos durante el proceso de migración del código de OpenHEVC:

- 1) En primer lugar se han deshabilitado las macros que se muestran en la Tabla 28, y cuya activación no procede sobre plataformas DSP. Para ello se han definido con valor 0 en el fichero de configuración común a todo el proyecto *config.h*.

Macros deshabilitadas	Justificación
ARCH_x86	Sólo para arquitecturas tipo x86.
HAVE_SSEn	Sólo para procesadores Intel con repertorio SSE.
HAVE_PTHREADS	Los <i>threads</i> no están soportados en este DSP.
HAVE_ATOMICS_GCC	No hay operaciones atómicas al no haber hilos.
HAVE_LLINT y HAVE_LLINTF	No existen funciones de redondeo para este procesador.
HAVE_ATOMIC_NATIVE	No hay operaciones atómicas al no haber hilos.
HAVE_GETTIMEOFDAY	Requiere de la biblioteca <i>time</i> , no disponible sobre el DSP.
HAVE_SCHED_GETAFFINITY	Sólo para entornos multinúcleo con <i>threads</i> .
HAVE_THREADS	Los <i>threads</i> no están soportados en este DSP.
HAVE_UNISTD_H	Sólo para POSIX.
HAVE_USLEEP	Sólo para permitir función <i>usleep()</i> con <i>threads</i> .

Tabla 28. Relación de macros deshabilitadas en el entorno Multi-DSP.

- 2) Se ha excluido de compilación la relación de directorios y ficheros detallados en la Tabla 29.

Carpeta o fichero excluido	Justificación
libavcodec/arm	Este directorio contiene ficheros para plataformas tipo ARM exclusivamente.
libavcodec/x86	Este directorio contiene ficheros para plataformas tipo x86 exclusivamente.
libavutil/pthread.c libavutil/pthread_frame.c libavutil/pthread_slice.c	Estos ficheros implementan la ejecución paralelizada mediante <i>threads</i> .
Compat	Excluida por contener ficheros relativos al uso de <i>threads</i> .
main/sdl.c main/sdl2.c	Funciones relativas a las bibliotecas SDL, no procede sobre plataformas DSP.

Tabla 29. Relación de carpetas y ficheros excluidos de compilación.

- 3) Con el fin de no alterar la estructura del código original se han creado las siguientes funciones vacías, es decir sin código, en el fichero *libavutil/utls.c*:

- *ff_thread_await_il_progress()*
- *ff_thread_report_il_status()*
- *ff_thread_il_progress()*

- 4) Se ha sustituido la macro *DECLARE_ALIGNED* por la directiva *#pragma DATA_ALIGN*, ambas utilizadas para alinear en memoria variables y *buffers*. Esto se ha realizado en los archivos *libavutil/mem.h*, *libavcodec/hevcdsp.c* y *libavcodec/hevc.c*.
- 5) Para la medida del rendimiento del descodificador en número de imágenes descodificadas por segundo se ha utilizado un *timer* del DSP de 64 bits. El código de la Figura 70 muestra las líneas de código encargadas de calcular la diferencia en

ciclos de reloj transcurridos durante la decodificación de una imagen. Para utilizar las funciones relativas al *timer* es necesario incluir en el proyecto CCS el paquete CSL (ver apartado 7.2), e incluir en el archivo el fichero de cabecera *cs_l_rsc.h*.

```
(...)
tick_in = CSL_tscRead();           //Lectura del timer (DSP)
got_picture=libOpenHEVCDecode(...); //Llamada a la función del decodificador

if (got_picture > 0){               //¿Imagen decodificada?
    tick_out = CSL_tscRead();       //Lectura del timer (DSP)
    elapsed = tick_out - tick_in;   //Se calcula la diferencia
    FPS = nb_pictures/(elapsed/freq_DSP); //Se calcula el ratio de FPS
}

(...)
```

Figura 70. Sección de código que mide el número de FPS decodificados.

Una vez se ha realizado la migración del código de OpenHEVC ha sido necesario introducir una serie de pequeñas modificaciones sobre la biblioteca *native* generada por Orcc, ya que es a través de esta biblioteca donde el código fuente generado por Orcc accede a las funciones optimizadas. La activación o desactivación de todas las secciones de código relacionadas con la utilización de OpenHEVC quedan supeditadas a la macro `OPENHEVC_ENABLE`, ya incluida en el *back-end* C, y para aquellas propias de la plataforma DSP a la macro `MDSP_ENABLE`.

Ha sido necesario habilitar la compilación del archivo *openhevc_sse.c*, para disponer de la llamada a la función *ff_hevc_dsp_init()*. Esto permitirá activar la funcionalidad del *wrapper* descrito en el apartado 8.4, para las funciones optimizadas para la plataforma DSP. El código que implementa las funciones optimizadas se encuentra en el archivo *hevcdsp_template.c* integrado en el código de OpenHEVC.

El proceso de optimización sobre una arquitectura concreta es largo y queda fuera de los objetivos de esta tesis [Pes11]. Por ello únicamente se han integrado mejoras puntuales sobre algunos actores del decodificador RVC YUV, dado que el objetivo es la verificación del correcto funcionamiento de la solución propuesta. La Tabla 30 muestra la relación de actores y funciones sobre los que se ha llevado a cabo alguna optimización.

Actor (HevcDecoder_)	Función optimizada
DBFilter_DBF_u	hevc_h_loop_filter_chroma_orcc
DBFilter_DBF_v	hevc_v_loop_filter_chroma_orcc
DBFilter_DBF_y	hevc_h_loop_filter_luma_orcc hevc_v_loop_filter_luma_orcc
InterPred_Inter_u	put_hevc_epel_orcc
InterPred_Inter_v	
InterPred_Inter_y	put_hevc_qpel_orcc

Tabla 30. Relación de actores y funciones optimizadas.

Dado el gran esfuerzo necesario para llevar a cabo un proceso de optimización de código fuente, la elección de los módulos a mejorar debe realizarse en base a un

análisis mediante el cual se identifiquen aquellas secciones de código cuya ejecución penaliza en mayor medida el rendimiento de la aplicación, o limita la paralelización de la ejecución del código en un entorno multinúcleo.

9.7.3. Generalización

Como se ha descrito al inicio de este apartado, la forma mediante la cual el código fuente generado por Orcc accede al código optimizado, cuando éste se encuentre disponible, es independiente de la plataforma objetivo. De esta forma se asegura la compatibilidad del procedimiento independientemente del entorno de trabajo.

Por otra parte, las mejoras que quieran llevarse a cabo sobre aquellos módulos especialmente susceptibles de ser optimizados serán responsabilidad del desarrollador.

9.8. Personalización de las aplicaciones para las plataformas objetivo

Cualquier aplicación desarrollada en un entorno de trabajo específico requiere, en la mayoría de los casos, del establecimiento de una configuración determinada, mediante la que se indique al compilador los parámetros necesarios para la utilización de periféricos (como *timers*), otros recursos específicos de la plataforma, gestión de interrupciones o la configuración de las memorias del sistema, como memoria compartida MSM, memoria externa DDR, etc.

9.8.1. Planteamiento del problema

A la hora de trabajar con plataformas tipo Multi-DSP, en particular con el procesador TMS320C6678 utilizado en esta tesis doctoral, y al contrario que con las plataformas tipo GPP, el programador debe establecer la configuración de ciertos recursos *hardware* tales como mapas de memoria o periféricos. El procedimiento para la configuración de este tipo de parámetros depende de cada entorno de trabajo y por consiguiente de cada fabricante. Además, la utilización de módulos específicos en procesadores de la familia C66 de TI, como OpenMP, requiere de modificaciones adicionales sobre la configuración que permitan al código hacer uso de los recursos propios de la API.

9.8.2. Solución propuesta

Se ha llevado a cabo un minucioso estudio de todos los aspectos relativos tanto a la configuración de los recursos propios de la plataforma Multi-DSP de Texas Instruments empleada, como de la configuración del entorno de trabajo y de los módulos utilizados. Dicho estudio ha permitido establecer una configuración por defecto válida para el código fuente generado por Orcc, incluyendo las consideraciones de los apartados anteriores (9.2 a 9.7), y que deberá ampliarse en caso de ser necesario para dar soporte a nuevos aspectos que se quieran añadir a la metodología de diseño. Además, esta solución sería válida para plataformas similares a las utilizadas en esta tesis.

Por lo tanto, en este apartado se describe la configuración tanto del entorno de trabajo de CCS compuesto por un *workspace* y los proyectos que sean necesarios, como de los parámetros de la plataforma Multi-DSP TMS320C6678. La configuración descrita a continuación es una propuesta acorde con los requerimientos del decodificador de vídeo HEVC utilizado.

La configuración de todos los parámetros que conforman el entorno de trabajo se compone de tres partes bien diferenciadas:

- 1) Configuración del mapa de memoria de la plataforma. El IDE de CCS dispone de distintas configuraciones prediseñadas para todas las plataformas a las que da soporte. Esta configuración debe ser comprobada, y en su caso ajustada, a las necesidades de la aplicación en desarrollo. La selección de la plataforma se realiza en el apartado de Propiedades→General→RTSC de cada proyecto. Además, CCS permite al programador diseñar sus propias configuraciones mediante la herramienta disponible en *Tools* → *RTSC Tools* → *Platform* → *New*, La Figura 71 muestra una captura de esta utilidad.

The screenshot shows the 'Device Details' and 'Device Memory' configuration windows in CCS. The 'Device Details' window has fields for 'Device Name' (TMS320C6678), 'Device Family' (c6000), and 'Clock Speed (MHz)' (1000), with an 'Import...' button. The 'Device Memory' window contains a table of memory sections, cache settings, and a 'Customize Memory' checkbox.

Name	Base	Length	Space	Access
MSMCSRAM	0x0c000000	0x00400000	code/data	RWX
L1DSRAM	0x00f00000	0x00000000	data	RW
L1PSRAM	0x00e00000	0x00000000	code	RWX

Cache settings: L2 Cache: 0k, L1D Cache: 32k, L1P Cache: 32k. ☒ Customize Memory.

Name	Base	Length	Space	Access
DDR3	0x80000000	0x20000000	code/data	RWX

Memory Sections: Code Memory: L2SRAM, Data Memory: L2SRAM, Stack Memory: L2SRAM.

Figura 71. Captura de la herramienta de CCS para configurar nuevas plataformas.

- 2) Configuración del SYS/BIOS y de los paquetes RTSC utilizados. Por un lado los paquetes RTSC utilizados deben indicarse en las propiedades del proyecto para que estén accesibles posteriormente, y en caso de ser necesario se modificarán los parámetros necesarios en el fichero de configuración *config.cfg*. En el Anexo I se puede consular en detalle la configuración utilizada durante la fase de pruebas de la metodología propuesta.

- 3) Finalmente, y para poder validar las implementaciones es necesario cargar el ejecutable en la plataforma, y en su caso para poder depurar el código, será necesario establecer la configuración del emulador y de la conexión con la plataforma de prueba. En este sentido, se debe indicar el emulador que se va utilizar para establecer la conexión entre el PC donde se ejecuta el IDE y la plataforma DSP. Todo ello se lleva a cabo en el fichero *platform.ccxml*. En este punto también se deben facilitar las propiedades básicas de la plataforma, como se explica a continuación.

9.8.2.1. Configuración de la plataforma multi-DSP

En primer lugar se establece el mapa de memoria de la plataforma de acuerdo con los requisitos de la aplicación. Las distintas memorias de la plataforma TMS320C6678 se distribuyen como se muestra en la Tabla 31.

Memoria	Tamaño	Dirección inicial	Dirección final	Consideraciones
L1D	32 kB	0x00E00000	0x00E08000	Individual de cada núcleo
L1P	32 kB	0x00F00000	0x00F08000	
L2	512 kB	0x00800000	0x00880000	
MSMCSRAM	4 MB	0x0C000000	0x0C3FFFFF	Compartida entre los 8 núcleos
DDR3	512 MB	0x80000000	0x9FFFFFFF	

Tabla 31. Mapa de memoria de la plataforma TMS320C6678.

Como se introdujo en el apartado 7.1.2, las memorias L1 y L2 son locales a cada uno de los núcleos de la plataforma, y las memorias MSM y DDR3 son compartidas por los ocho procesadores. L1 y L2 pueden configurarse como memorias caché, SRAM o un combinado. Así, se configuran las memorias L1D y L1P como caché en su totalidad y la memoria L2 se divide en dos, una mitad como caché y la otra como SRAM. Es necesario dejar un espacio como SRAM en la memoria L2 para alojar la pila de programa, además también podrán alojarse aquí variables privadas cuya utilización sea muy frecuente, pudiendo mejorar así el rendimiento, dado que la velocidad de transferencia de datos de esta memoria es muy superior a la de las memorias compartidas (ver apartado 7.1.2). La Tabla 32 muestra el mapa de memoria una vez configurado.

Memoria	Tamaño	Dirección inicial	Dirección final	Consideraciones
L1D	32 kB	--	--	Todo caché
L1P	32 kB	--	--	Todo caché
L2 caché	256 kB	--	--	
L2 SRAM	256 kB	0x00800000	0x00840000	
MSMCSRAM	4 MB	0x0C000000	0x0C3FFFFF	
DDR3	512 MB	0x80000000	0x9FFFFFFF	

Tabla 32. Modificación del mapa de memoria de la plataforma TMS320C6678.

En función del tamaño del programa y de la cantidad de espacio requerido para el emplazamiento de las variables se debe establecer la configuración por defecto para alojar el código del programa y las variables. En este caso, la configuración se realiza para que el código del programa se aloje en memoria compartida MSM, quedando así

accesible por todos los núcleos, y las variables en memoria DDR3. Si bien sería recomendable que dichas variables se alojasen en memorias de mayor velocidad que la DDR3, la necesidad de utilizar *buffers* de gran tamaño para la descodificación de imágenes hace imposible ubicar todas las variables en otra memoria que no sea la DDR3. Sin embargo, de forma puntual podrán ubicarse algunas variables en memoria L2SRAM o MSM. Esto se puede realizar mediante el uso de la directiva *#pragma DATA_SECTION*, como se muestra en el código ejemplo de Figura 72, donde el *array* *predSamples_u* se ubica en memoria MSM.

```
#pragma DATA_SECTION (predSamples_u, ".mem_MSMC")
static i16 predSamples_u[2][4096];
```

Figura 72. Alojamiento de un array en memoria MSM.

Una vez se ha configurado el mapa de memoria deseado se procede a incluir en las propiedades del proyecto los paquetes RTSC utilizados. Como se detalló en el apartado 7.2, en este caso se hace uso de los paquetes CSL, IPC y OpenMP.

El resto de la configuración se realiza manualmente⁵⁷ introduciendo comandos en el fichero *config.cfg*. En este último caso cada vez que se modifique alguno de los parámetros vía interfaz CCS escribirá en el archivo de configuración los comandos correspondientes. En caso contrario, es el programador el que debe introducir o modificar los comandos de este fichero manualmente. En la Tabla 33 se muestran algunos de los comandos utilizados y su funcionalidad, en el Anexo I se facilita la configuración completa utilizada.

Comando	Utilidad
<code>var Cache=xdc.useModule('ti.sysbios.hal.Cache');</code>	Habilita el módulo de manejo de cachés
<code>var Timer=xdc.useModule('ti.sysbios.hal.Timer');</code>	Habilita el módulo para el manejo de los <i>timers</i>
<code>program.sectMap[".text"]=new Program.SectionSpec(); program.sectMap[".text"].loadSegment="MSMCSRAM";</code>	Obliga a que la sección (<i>section</i>) <i>text</i> (programa) se ubique en MSMCSRAM
<code>program.sectMap[".NC"]=new Program.SectionSpec(); program.sectMap[".NC"].loadSegment= "OMP_MSMC_NC_VIRT";</code>	Crea un nuevo conjunto de datos llamado NC en memoria virtual no cacheable (ver apartado 9.8.2.2).
<code>Program.stack = 0x20000;</code>	Configura un tamaño de pila de 0x20000

Tabla 33. Comandos de configuración del fichero *cfg*.

No se ha encontrado un listado que resuma todos los comandos y distintos parámetros que pueden incluirse en este fichero. De forma empírica, mediante el uso de aplicaciones de ejemplo facilitadas por el fabricante y la interfaz de usuario de CCS, se ha establecido una configuración válida para cumplir con los objetivos de este trabajo de investigación (ver Anexo I).

⁵⁷ En algunas versiones de CCS hay disponible una interfaz gráfica para realizar la configuración de los distintos módulos.

El último aspecto de la configuración se refiere a la conexión vía emulador entre el PC donde se ejecuta el IDE de CCS y la plataforma de pruebas. Esta configuración se realiza en un fichero de extensión *platform.ccxml*, donde se indica la plataforma que se va a utilizar, el modelo del emulador, el número de procesadores sobre los que se desea cargar el código implementado y si éstos deben actuar como maestros o esclavos, entre otros parámetros que la interfaz pone a disposición del desarrollador⁵⁸.

De forma opcional, y en función de las necesidades de la plataforma, se puede enlazar un fichero de configuración *.GEL al fichero ccxml. Este fichero incluye información básica sobre la configuración de la plataforma, como el uso de memorias caché, velocidad del reloj del sistema o modos de acceso a memoria externa. Una vez establecida la comunicación con la plataforma para depurar un programa, CCS configurará, antes de realizar el volcado del código del programa, el fichero GEL en la plataforma.

9.8.2.2. Configuración con OpenMP

A la hora de trabajar con la API de OpenMP se deben realizar algunas modificaciones en la configuración de los proyectos de CCS. En primer lugar el código del programa debe alojarse en memoria compartida para que sea visible y accesible por parte de todos los núcleos. Además, se debe configurar una región de memoria no cacheable en la misma memoria compartida. Dado que en la plataforma TMS320C6678 la memoria MSM es cacheable en su totalidad, y no puede modificarse esta configuración, se debe crear una memoria virtual en una región cuya configuración pueda ser modificada. En este caso se fija el mapa de memoria que se muestra en la Tabla 34 a partir del mapa de memoria inicial mostrado en la Tabla 32.

Memoria	Tamaño	Dirección inicial	Dirección final	Consideraciones
L1D	32 kB	--	--	Todo caché
L1P	32 kB	--	--	Todo caché
L2 caché	256 kB	--	--	
L2 SRAM	256 kB	0x00800000	0x00840000	
MSMCSRAM	3968 kB	0x0C020000	0x0C3FFFFF	Memoria
DDR3	512 MB	0x80000000	0x9FFFFFFF	compartida
OMP_MSMC_NC_PHY	128 kB	0x0C000000	0x0C01FFFF	Reservado
OMP_MSMC_NC_VIRT	128 kB	0xA0000000	0xA001FFFF	OpenMP

Tabla 34. Mapa de memoria con OpenMP.

También es necesario incluir en el fichero de configuración *config.cfg* aquellos parámetros que definan las propiedades deseadas de OpenMP, algunos de los cuales fijan el número de núcleos a utilizar, o el tamaño de las *heap*⁵⁹ locales y compartidas. La Tabla 35 resume algunos de los parámetros de configuración más destacables, y en el

⁵⁸ En las versiones anteriores a la v6.0, la utilizada en esta tesis es la v6.1, de CCS también era posible realizar una simulación sin necesidad de disponer de la plataforma de desarrollo, esto era posible mediante la configuración del modo *simulator* en el fichero ccxml.

⁵⁹ Se deben realizar asignaciones de memoria para aquellas rutinas que en tiempo de ejecución requieran de espacio en memoria para su funcionamiento, como ocurre al utilizar funciones *malloc()*.

Anexo I se facilita el archivo de configuración completo. De forma similar a como se expuso en el apartado anterior (ver apartado 9.8.2.1), no ha sido posible encontrar un documento que resuma los distintos parámetros de configuración de OpenMP sobre el fichero *config.cfg* en CCS. La configuración diseñada y establecida es fruto de la experimentación a partir de ficheros de configuración disponibles en distintos ejemplos facilitados por el fabricante.

Comando	Utilidad
<code>var ompSettings=xdc.useModule("ti.runtime.openmp.Settings"); ompSettings.usingRtsc = true ;</code>	Habilita el módulo de configuración de OpenMP
<code>var OpenMP = xdc.useModule('ti.runtime.ompbios.OpenMP');</code>	Habilita el módulo de OpenMP
<code>OpenMP.masterCoreIdx = 0;</code>	Establece el hilo de ejecución principal
<code>OpenMP.numCores = 8;</code>	Establece el número máximo de núcleos utilizados por OpenMP
<code>var HeapOMP=xdc.useModule('ti.runtime.ompbios.HeapOMP'); var localHeapSize = 0x8000; var sharedHeapSize = 0x800000;</code>	Habilitación y configuración de la <i>heap</i> privada y compartida con OpenMP

Tabla 35. Parámetros de configuración de OpenMP en el fichero *cfg*.

9.8.3. Generalización

El procedimiento descrito anteriormente es compatible con la mayoría de plataformas Multi-DSP de Texas Instruments, ya que todas sus plataformas comparten el mismo IDE. En el caso de que la plataforma objetivo cambie, únicamente será necesario realizar pequeños ajustes de la configuración, como pueda ser el mapa de memorias utilizado, esto se realiza mediante la herramienta de RTSC para la edición de las propiedades de la plataforma.

En el caso de que la plataforma objetivo sea de un fabricante distinto se recomienda repetir el mismo orden aquí seguido, de forma que se asegure que todos los aspectos quedan cubiertos e incluidos en la configuración. Pero será necesario adaptar los comandos de configuración a los especificados por el fabricante para cada plataforma.

En aquellas plataformas con sistemas operativos capaces de realizar estas tareas de forma automática, cuya configuración ya haya sido establecida o no está en manos del programador, no será necesario llevar a cabo este paso de la metodología. Como es el caso de ambas plataformas GPP utilizadas en esta tesis doctoral.

9.9. Automatización del proceso de diseño

En el apartado 8.3 se presentaron los fundamentos de la infraestructura de trabajo de Orcc que permiten la generación de código fuente. Sobre estos principios se

ha realizado un nuevo *back-end* que abarca todos los aspectos incluidos en la metodología descrita en este capítulo. Esto permite la generación de código fuente en lenguaje de programación C, tanto para la plataforma Multi-DSP TMS320C6678, como para los GPP con soporte OpenMP utilizados.

El trabajo de diseño de un nuevo *back-end* supone la culminación de uno de los hitos principales de esta tesis doctoral. Además, se ha diseñado de forma lo suficientemente flexible para que pueda ser utilizado para generar código fuente compilable para otras plataformas tanto Multi-DSP, como tipo GPP, con soporte para OpenMP.

Como se ha detallado a lo largo de esta memoria, el lenguaje de programación utilizado en todo momento es C. De esta manera, el *back-end* C ya existente sirve como punto de partida para el diseño del nuevo *back-end* DSP C. Dado que dentro de la infraestructura de Orcc, el *back-end* es la parte encargada en último término de la generación de código fuente, el trabajo de diseño se ha centrado exclusivamente en esta parte de la infraestructura, manteniendo las fases de *front-end* y *middle-end* en su forma original.

El código que implementa el compilador de Orcc está integrado por una serie de proyectos, los cuales contienen los ficheros con el código que lleva a cabo las distintas funcionalidades del compilador. Estos proyectos deben importarse en un *workspace* del IDE de Eclipse para ser modificados como se muestra en la Figura 73. En el caso de este trabajo la mayoría de las modificaciones se han llevado a cabo sobre el proyecto *net.sf.orcc.backends*.

Figura 73. Proyectos del compilador Orcc importados en Eclipse.

9.9.1. Definición del nuevo *back-end* DSP

El primer paso para la definición del nuevo *back-end* DSP ha sido proceder a replicar el conjunto de carpetas y ficheros que integran el *back-end* C, que sirve como punto de partida. El proyecto sobre el que se ha trabajado en todo momento es

net.sf.orcc.backends, dentro del mismo se encuentran, por un lado, y en la carpeta *src*, los ficheros con las plantillas encargadas de controlar la generación de código, además de los ficheros *.java* que controlan la ejecución del compilador. Por otro lado, y dentro de la carpeta *runtime*, se encuentran los directorios, para cada *back-end*, con los archivos de biblioteca necesarios para compilar el código fuente generado, en el caso del nuevo *back-end*, las bibliotecas necesarias son: *orcc-runtime*, *orcc-native* y *roxml*.

Es en el directorio donde se encuentran las bibliotecas nativas de Orcc donde deben incluirse los archivos necesarios para la correcta generación del código fuente para la plataforma DSP. Así, se incluye el archivo de configuración de la plataforma DSP *config.cfg*, el archivo de configuración *config.h* y los archivos de biblioteca que integran las funciones para el manejo de las operaciones con cachés *cache.h* y *cache.c*.

Dado que el código fuente de las bibliotecas nativas de Orcc se encuentra disponible de forma directa en el *back-end*, y no se realiza ninguna transformación del mismo mediante ninguna de las plantillas, las modificaciones detalladas en los apartados 9.2.2.2 y 9.4.2 se introducirán directamente sobre los ficheros de las bibliotecas en el nuevo *back-end*.

La generación del fichero de configuración de la conexión con la plataforma (*platform.ccxml*) y los ficheros con las propiedades de cada proyecto para el IDE CCS, se lleva a cabo mediante clases modificadas en los ficheros de configuración del *back-end*, como se describe en el apartado 9.9.2.

En todos los directorios y carpetas del nuevo *back-end* se han modificado las rutas de aquellos archivos y funciones propios del nuevo diseño, utilizando la notación del *back-end* C (*net.sf.orcc.backends.c*) seguido de la extensión *.dsp*, resultando *net.sf.orcc.backends.c.dsp*.

Desde el punto de vista de la interfaz de Orcc, a la hora de compilar una especificación RVC-CAL con el nuevo *back-end* no se produce ningún cambio, únicamente se deberá seleccionar en el menú desplegable el nuevo *back-end*, como se muestra en la Figura 74. También se ha añadido un nuevo menú desplegable que contiene el listado de plataformas DSP a las que el *back-end* puede dar soporte, permitiendo cambiar fácilmente de plataforma.

Figura 74. Aspecto de la ventana de opciones del compilador Orcc sobre Eclipse.

9.9.2. Generación de los ficheros de configuración

El funcionamiento global del *back-end* se controla desde el fichero *DSPBackend.java*. Desde este fichero se llama a la extracción de los archivos de biblioteca contenidos en los directorios correspondientes, mediante la función *doLibrariesExtraction()*, esta función ha sido modificada para incluir los nuevos ficheros de configuración incluidos en el directorio de bibliotecas, como se muestra en la Figura 75.

```

Protected Result doLibrariesExtraction() {
    final Result result = FileManager.extract("/runtime/DSP/README.txt", outputPath);
    //Ya existentes:
    result.merge(FileManager.extract("/runtime/C/libs/orcc-native", outputPath));
    result.merge(FileManager.extract("/runtime/C/libs/orcc-runtime", outputPath));
    result.merge(FileManager.extract("/runtime/C/libs/roxml", outputPath));
    //Nuevos:
    result.merge(FileManager.extract("/runtime/DSP/config/config.cfg", srcPath));
    result.merge(FileManager.extract("/runtime/DSP/config/config.h", srcPath));
    //Nuevos:
    result.merge(FileManager.extract("/runtime/DSP/libs/dsp-native/cache.c", srcPath));
    result.merge(FileManager.extract("/runtime/DSP/libs/dsp-native/cache.h", srcPath));
    result.merge(FileManager.extract("/runtime/DSP/libs/dsp-native/measures.c", srcPath));
    result.merge(FileManager.extract("/runtime/DSP/libs/dsp-native/measures.h", srcPath));

    return result;
}

```


Figura 75. Sección de código encargada de la generación de los ficheros de configuración.

Además, mediante la función *doAdditionalGeneration()* (ver Figura 76) se realizan las llamadas a las funciones encargadas de controlar la generación del resto de ficheros de configuración necesarios para la gestión de los proyectos del IDE CCS.

Las propiedades de cada proyecto CCS quedan definidas mediante tres ficheros de configuración: *.project*, *.cproject* y *ccsproject*. En el fichero *.project* se almacena información relativa al nombre del proyecto, y al *plugin* de Eclipse utilizado. El fichero *.cproject* contiene los detalles sobre el compilador cruzado a utilizar, su versión y opciones de compilación. También almacena información sobre los paquetes CCS utilizados en el proyecto. Finalmente, el fichero *.ccsproject* guarda una relación general de información sobre la plataforma, el compilador o el emulador, entre otros detalles.

Dado que debe haber un proyecto CCS por cada biblioteca (*orcc-runtime*, *orcc-native* y *roxml*), y otro adicional para el ejecutable se deben generar los respectivos ficheros de configuración para cada proyecto. También se debe añadir en este momento la llamada correspondiente a la generación del archivo de configuración *.ccxml* que controla las propiedades de la conexión entre la estación de trabajo (PC) y la plataforma DSP. Esta funcionalidad queda implementada en la clase *ccsProjectPrinter*, tal y como se muestra en la Figura 76.

```

Protected Result doAdditionalGeneration(Network network) {
    final Result result = Result.newInstance();
    final Mapping mapper = new Mapping(network, mapping);

    result.merge(FilesManager.writeFile(mapper.getContentFile(),
srcPath, network.getSimpleName() + ".xcf"));

    String libOrccNativePath = outputPath + File.separator + "orcc-native";
    String libOrccRuntimePath = outputPath + File.separator + "orcc-runtime";
    String libRoxmlPath = outputPath + File.separator + "roxml";

    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCcxml(dsp), srcPath, dsp.getDevice() +
".ccxml"));

    result.merge(FilesManager.writeFile(ccsProjectPrinter.getProject(network.getSimpleName()),
srcPath, ".project"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getProject("orcc-native"),
libOrccNativePath, ".project"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getProject("orcc-runtime"),
libOrccRuntimePath, ".project"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getProject("roxml"), libRoxmlPath,
".project"));

    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCProject(dsp, network.getSimpleName()),
srcPath, ".cproject"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCProject(dsp, "orcc-native"),
libOrccNativePath, ".cproject"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCProject(dsp, "orcc-runtime"),
libOrccRuntimePath, ".cproject"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCProject(dsp, "roxml"), libRoxmlPath,
".cproject"));

    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCcsProject(dsp, true), srcPath,
".ccsproject"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCcsProject(dsp, false),
libOrccNativePath, ".ccsproject"));
    result.merge(FilesManager.writeFile(ccsProjectPrinter.getCcsProject(dsp, false),
libOrccRuntimePath, ".ccsproject"));
}

```

```

result.merge(FileManager.writeFile(ccsProjectPrinter.getCcsProject(dsp, false), libRoxmlPath,
".ccsproject"));

    return result;
}

```

Figura 76. Sección de código encargada de la gestión de bibliotecas y archivos en el *back-end*.

9.9.3. Modificación de la plantilla *Network Printer*

Como se introdujo en el apartado 8.3.2, la plantilla *Network Printer* es la encargada de generar el código fuente del archivo `Top_n_main.c`, en el que se encuentra la función *main* del programa, así como la declaración de las variables globales, las FIFOs y demás estructuras de datos comunes a todos los actores. Dentro del diseño del nuevo *back-end* DSP es necesario realizar modificaciones en esta plantilla, en concreto las descritas en el apartado 9.2.2.1.

Como se muestra en la Tabla 36, esta plantilla se compone de las siguientes funciones encargadas de la generación de una sección de código concreta, a continuación se detallan las modificaciones realizadas sobre las distintas funciones originalmente descritas.

Funciones	Funcionalidades originales	Modificaciones
getNetworkFileContent	Crea la cabecera del archivo.	Se deben añadir las inclusiones a las bibliotecas propias de la plataforma
	Llama a la función <code>allocateFifos</code> (se detalla a continuación).	
	Llama a la función <code>assignFifo</code> (se detalla a continuación).	
	Crea las cabeceras de las funciones comunes de los actores.	Se deben incluir las nuevas funciones comunes, como <code>sectores()</code> e <code>initializecores()</code> .
	Crea las estructuras de los actores.	
	Crea las estructuras de las comunicaciones.	
	Genera el código de la función <i>main</i> .	Se incluye la configuración del tamaño de las cachés. Se incluye, en caso de utilizarse, la configuración de la frecuencia de trabajo (ver apartado 9.6.2.2).
allocateFifos	Genera el código relativo a la declaración de cada una de las FIFOs.	Se incluyen aquí las sentencias para ubicar las FIFOs en memoria compartida y alinear los campos que las componen.
assignFifo	Genera el código que asigna a cada nueva FIFO los punteros mediante los cuales los actores transmiten información.	

Tabla 36. Relación de modificaciones introducidas en la plantilla *Network Printer*.

Mediante la modificación de la plantilla *Network Printer* sólo el archivo fuente `Top_n_main.c` se verá afectado. Esta plantilla también generará el código, de acuerdo a las nuevas modificaciones, para cada una de las FIFOs descritas en el gráfico *xdif* de la especificación RVC-CAL dada. Esto ocurre de acuerdo con lo descrito en la clase `allocateFifos` y `assignFifo`. La Tabla 37 muestra la sección del código modificada y

encargada de la declaración y configuración de las FIFOs. Así, y a modo de ejemplo, la Tabla 38 muestra la comparativa entre el código fuente generado para las FIFOs 0 y 1, al utilizar el *back-end* C original y el nuevo *back-end* DSP.

```
def protected allocateFifo(Connection conn, int nbReaders) """
DECLARE_FIFO(«conn.sourcePort.type.doSwitch», «if (conn.size != null) conn.size else "SIZE"»,
«conn.<Object>getValueAsObject("idNoBcast")», «nbReaders»)
"""

def protected allocateFifo(Connection conn, int nbReaders) """
#pragma DATA_SECTION (invalidar_«conn.<Object>getValueAsObject("idNoBcast")», ".aux_mem");
#pragma DATA_SECTION (fifo_«conn.<Object>getValueAsObject("idNoBcast")», ".aux_mem");
#pragma DATA_SECTION (read_inds_«conn.<Object>getValueAsObject("idNoBcast")», ".aux_mem");
#pragma DATA_SECTION (array_«conn.<Object>getValueAsObject("idNoBcast")», ".aux_mem");
#pragma DATA_ALIGN (fifo_«conn.<Object>getValueAsObject("idNoBcast")», 128);
#pragma DATA_ALIGN (array_«conn.<Object>getValueAsObject("idNoBcast")», 128);
#pragma DATA_ALIGN (read_inds_«conn.<Object>getValueAsObject("idNoBcast")», 128);
#pragma DATA_ALIGN (core_maps_«conn.<Object>getValueAsObject("idNoBcast")», 128);
#pragma DATA_ALIGN (invalidar_«conn.<Object>getValueAsObject("idNoBcast")», 128);
DECLARE_FIFO(«conn.sourcePort.type.doSwitch», «if (conn.size != null) conn.size else "SIZE"»,
«conn.<Object>getValueAsObject("idNoBcast")», «nbReaders»)
"""
```

Tabla 37. Sección de código encargado de la declaración y configuración de las FIFOs.

Sección de código fuente generado de acuerdo con el <i>back-end</i> C original	Sección de código fuente generado en base al nuevo <i>back-end</i> DSP
DECLARE_FIFO(u8,SIZE,0,1)	<pre>#pragma DATA_SECTION (invalidar_0, ".aux_mem"); #pragma DATA_SECTION (fifo_0, ".aux_mem"); #pragma DATA_SECTION (read_inds_0, ".aux_mem"); #pragma DATA_SECTION (array_0, ".aux_mem"); #pragma DATA_ALIGN (fifo_0, 128); #pragma DATA_ALIGN (array_0, 128); #pragma DATA_ALIGN (read_inds_0, 128); #pragma DATA_ALIGN (core_maps_0, 128); #pragma DATA_ALIGN (invalidar_0, 128); DECLARE_FIFO(u8, SIZE, 0, 1)</pre>
DECLARE_FIFO(u8,SIZE,1,11)	<pre>#pragma DATA_SECTION (invalidar_1, ".aux_mem"); #pragma DATA_SECTION (fifo_1, ".aux_mem"); #pragma DATA_SECTION (read_inds_1, ".aux_mem"); #pragma DATA_SECTION (array_1, ".aux_mem"); #pragma DATA_ALIGN (fifo_1, 128); #pragma DATA_ALIGN (array_1, 128); #pragma DATA_ALIGN (read_inds_1, 128); #pragma DATA_ALIGN (core_maps_1, 128); #pragma DATA_ALIGN (invalidar_1, 128); DECLARE_FIFO(u8, SIZE, 1, 11)</pre>

Tabla 38. Comparativa entre el código generado relativo a la declaración de FIFOs entre los *back-ends* C y DSP.

Como se puede observar, en primer lugar se define que los distintos campos que integran la FIFO se ubiquen en la región de memoria denominada “aux_mem” (memoria compartida interna MSM), el margen de direcciones físicas en memoria que puede ocupar esta región se establece en el archivo de configuración *config.cfg*. Seguidamente, se determina que todos los campos de la FIFO queden alineados en memoria a razón de 128 B, es decir, el tamaño de línea de caché para el procesador TMS320C6678. Esta configuración debe llevarse a cabo antes de realizar la llamada a la macro que finalmente declara la FIFO. El conjunto de instrucciones debe aparecer, en el

código fuente, en el orden mostrado en la Tabla 38 para cada una de las FIFOs que integren la aplicación, esto se realiza de forma automática por la plantilla del *back-end*.

9.9.4. Modificación de la plantilla *Instance Printer*

Como se introdujo en el apartado 8.3, la plantilla *Instance Printer* es la encargada de generar el código fuente de cada uno de los actores involucrados en la aplicación. Dentro del diseño del nuevo *back-end* DSP es necesario realizar modificaciones en esta plantilla, en concreto las descritas en el apartado 9.4.2.

Como se muestra en la Tabla 39, esta plantilla se compone de las siguientes funciones encargadas de la generación de una sección de código concreta. A continuación se detallan las modificaciones realizadas sobre las distintas funciones originalmente descritas. Debido al elevado número de funciones que componen esta plantilla sólo se incluyen aquí aquellas de mayor relevancia o que han sido modificadas.

Funciones	Funcionalidades originales	Adiciones y/o Modificaciones
getFileContent	Crea la cabecera del archivo.	
	Genera las llamadas a las bibliotecas necesarias.	Se deben hacer las inclusiones de las bibliotecas propias de la plataforma.
	Se crean las variables locales que alojarán los <i>flags</i> de invalidación.	
	Genera las variables de control de las FIFOs.	
	Genera el código de la FSM.	
	Se llama a R/W ⁶⁰ TokensFunctions.	
	Se llama a InitializeR/WcoreFunctions.	
	Se llama a proc/action.print	
	Se llama a printInitialize	
	Se llama a printInitializeCores	
	Se llama a printSetCores	
	Se llama a printWbInv	
	Se llama a printActorScheduler.	
PrintActorScheduler	Genera el código del planificador interno del actor, el cual está sujeto a la FSM.	Se incluye, al inicio del planificador, la actualización de los <i>flags</i> de invalidación.
PrintInitializeCores	Genera, para cada FIFO de salida, el código que evalúa si es necesario activar los <i>flags</i> de invalidación.	
PrintSetCores	Registra, para cada entrada y salida de las FIFOs, el número de procesador donde están mapeados los actores que las utilizan.	
PrintWbInv	Llamada a la actualización de <i>flags</i> de invalidar tras un remapeo	
PrintInitialize	Llama a las funciones de inicialización de cada uno de los actores.	
R/WtokensFunction	Escribe las funciones de control de escritura y lectura de FIFOs de acuerdo a las especificaciones dadas en el apartado 9.4.2.5.	

Tabla 39. Relación de funcionalidades modificadas de la plantilla *Instance Printer*.

A modo de ejemplo, la Tabla 40 ilustra la relación entre una modificación introducida en la plantilla y el código fuente generado. En este caso se muestra la parte de la plantilla encargada de generar, para cada uno de los actores, el código relativo a la función que analiza dónde están mapeados los actores que comparten las FIFOs, y en

⁶⁰ ReadTokensFunctions, WriteTokensFunctions (Funciones de lectura/escritura de *tokens*).

caso de ser distintos procesadores activar el *flag* de invalidar. Al comienzo de la función se asegura que el valor del número de núcleo está actualizado mediante una operación de *invalidate* de la sección correspondiente de la caché. Del mismo modo, para cada modificación introducida en el valor de los *flags* de invalidar se realiza una operación de *writeback* que asegura que el nuevo valor se actualiza correctamente en la memoria compartida.

Código insertado en la plantilla <i>Instance Printer</i>	Código generado en la FIFO de salida del actor <i>Source</i>
<pre>def protected printInitializeCores() ''' «inline»void «entityName»_initializecores(schedinfo_t *si) { int I; «FOR port : actor.outputs.notNative» «port.fullName»->invalidar[0] = 0; for(i=0;i<(1+NUM_READERS_«port.name»);i++){ CACHE_invL2(&«port.fullName»- >core_maps[i], 1, CACHE_WAIT); XPFCMD = 1; if(«port.fullName»->core_maps[0] != «port.fullName»->core_maps[i]){ «port.fullName»->invalidar[0]= 1; «port.fullName»->invalidar[i]= 1; }else{ «port.fullName»->invalidar[i]= 0; } } for(i=0;i<(1+NUM_READERS_«port.name»);i++){ CACHE_wbInvL2(&«port.fullName»- >invalidar[i], 1, CACHE_WAIT); XPFCMD = 1; } «ENDFOR» return; }'''</pre>	<pre>void Source_initializecores(schedinfo_t *si) { int I; Source_0->invalidar[0] = 0; for(i=0;i<(1+NUM_READERS_0); i++){ CACHE_invL2(&Source_0 ->core_maps[i],1, CACHE_WAIT); XPFCMD = 1; if(Source_0->core_maps[0] != Source_0->core_maps[i]){ Source_0->invalidar[0]= 1; Source_0->invalidar[i]= 1; }else{ Source_0->invalidar[i]= 0; } } for(i=0;i<(1+NUM_READERS_0); i++){ CACHE_wbInvL2(&Source_0- >invalidar[i], 1, CACHE_WAIT); XPFCMD = 1; } return; }</pre>

Tabla 40. Relación entre parte del código insertado en la plantilla y el generado posteriormente.

9.10. Resumen

A lo largo de este capítulo se han descrito cada uno de los apartados que integran la metodología de diseño propuesta. Esta metodología de diseño permite aprovechar ventajas tales como el mantenimiento del diseño modular de RVC, o la implementación basada en flujos de datos de decodificadores de vídeo, al tiempo que permite ampliar las funcionalidades del código generado mediante modificaciones en el código que implementa el *back-end* de Orcc.

En la metodología propuesta se han abordado todos los aspectos relacionados con la generación automática de código para plataformas con soporte OpenMP, al tiempo que se garantiza la compatibilidad del código con distintas arquitecturas sin necesidad de llevar a cabo modificaciones sobre el código fuente. Además, se ha propuesto un algoritmo *software* para el mantenimiento de la coherencia entre los

distintos niveles de memoria durante las transferencias de datos en un entorno multinúcleo.

La mencionada metodología de diseño se ha sintetizado a partir de la experiencia adquirida durante la realización experimental de descodificadores de vídeo HEVC a partir de descripciones RVC-CAL utilizando como tecnología objetivo chips multinúcleo. Los aspectos abordados en la metodología son los siguientes:

- Generación automática de código.
- Paralelización del código para arquitecturas multinúcleo.
- Personalización de las aplicaciones para las plataformas/chips objetivo.
- Gestión del flujo de información y coherencia de las memorias caché.
- Mapeo de algoritmos a núcleos.
- Control del consumo de energía.
- Utilización de módulos optimizados.

Las soluciones de los problemas planteados en las distintas áreas de interés son suficientemente generalizables, lo que permite su utilización para la implementación de distintas aplicaciones basadas en descripciones RVC-CAL sobre múltiples plataformas multinúcleo.

Finalmente, se ha diseñado un nuevo *back-end* de Orcc que implementa la generación automática de código fuente en base a esta metodología compilable para plataformas tipo DSP o GPP con soporte OpenMP.

10. Implementación de descodificadores HEVC

En este capítulo se presentan las distintas versiones del descodificador HEVC utilizadas para la verificación de la metodología de diseño propuesta en el capítulo 9. Además, se detallan los resultados obtenidos para cada implementación y para cada una de las plataformas utilizadas.

En primer lugar (apartado 10.1) se presentan las tres ⁶¹ versiones del descodificador HEVC utilizadas para validar la metodología propuesta a lo largo de todo el trabajo de investigación. A continuación (apartado 10.2), se presenta el banco de pruebas utilizado para la realización de todos los tests. Seguidamente, en el apartado 10.3 se realiza un estudio sobre el paralelismo de las distintas implementaciones sobre plataformas Multi-DSP. Finalmente, en el apartado 10.4, se presentan los resultados de las distintas implementaciones sobre las tres plataformas utilizadas; en este apartado también se detallan los resultados sobre las técnicas para la reducción de consumo energético en plataformas Multi-DSP.

10.1. El descodificador HEVC-RVC

Para la realización de este trabajo de investigación se han utilizado principalmente tres implementaciones de descodificadores HEVC, todas ellas de código abierto, disponibles en el repositorio de aplicaciones RVC-CAL para Orcc [OAPP]. Sin embargo, al inicio del desarrollo de la tesis doctoral se realizó una implementación a partir del código de referencia (HM) del estándar HEVC. Ésta es una versión funcional sin optimizaciones específicas y cuyo objetivo principal es estar disponible para aquellos desarrolladores que deseen trabajar y comprender el funcionamiento de las herramientas incluidas en el nuevo estándar. Los detalles de esta implementación están disponibles en el Anexo IV.

A lo largo de las siguientes subsecciones se detallan las particularidades de cada una de las implementaciones utilizadas en la fase de pruebas de este trabajo.

⁶¹ En este punto se hace mención de las tres versiones HEVC-RVC utilizadas. Sin embargo, el código de referencia del estándar HM también ha sido migrado al entorno DSP, en el Anexo IV se detalla esta implementación. Además se ha realizado en el laboratorio del GDEM una quinta implementación a partir del código de OpenHEVC sobre una plataforma Multi-DSP, pero que en este trabajo sólo ha sido utilizada de forma parcial.

10.1.1. Descodificador HEVC RVC de referencia

La primera versión de un descodificador HEVC implementada con el estándar RVC se desarrolló durante el año 2013, estando disponible en el repositorio una versión completa⁶² a finales de ese año. Esta versión del descodificador está disponible en el repositorio de aplicaciones para RVC-CAL en el directorio `src.org.sc29.wg11.mpeg.part2` del proyecto RVC. Dado que ésta es la primera versión de un descodificador HEVC, implementada mediante un diagrama de flujos de datos entre distintas unidades funcionales en RVC, se ha denominado como versión de *referencia* (en adelante HEVC-RVC REF).

La Figura 77 muestra el nivel más alto de la jerarquía del diagrama de flujo que implementa el descodificador. Este nivel se compone de dos actores y una FU, presentes en todas las implementaciones de descodificadores de vídeo en RVC: los actores *Source* y *display*, y la FU *HevcDecoder*, que a su vez comprende el resto de FUs que integran la funcionalidad del descodificador. El principio de funcionamiento de todas las implementaciones basadas en RVC se basa en que el actor *Source* lee el *bitstream* de un fichero de entrada, el cual es descodificado por *HevcDecoder*. Según las imágenes descodificadas van estando disponibles el actor *display* las presenta por pantalla.

Figura 77. Diagrama del nivel más alto de la jerarquía del descodificador HEVC-RVC de referencia.

Dada la envergadura del diagrama compuesto de FUs, actores y flujos de datos que integran el descodificador, el esquema completo del mismo está disponible en el Anexo II. En la Tabla 41 se presenta la correspondencia entre los principales bloques funcionales del descodificador HEVC y las FUs correspondientes al nivel más alto en la jerarquía de la descripción RVC-CAL del descodificador (*HevcDecoder*). Los acrónimos que aparecen en esta tabla serán utilizados en el resto del capítulo.

Bloques funcionales	Acrónimo	FUs en el modelo RVC-CAL
Descodificación de entropía	ED	Algo_parser
Transformada inversa y cuantificación	ITQ	xIT
Predicción intra	IP	IntraPrediction
Compensación de movimiento (Predicción inter)	EP	InterPrediction, GenerateInfo
Buffer de imágenes descodificadas	BF	DecodingPictureBuffer
Filtro anti-bloques	DF	DBFilter
Filtro SAO	SAO	SAO

⁶² Las versiones previas de HEVC RVC no incluían las etapas de filtrado anti-bloques ni SAO.

Gestión de CUs	CU	SelectCU
----------------	----	----------

Tabla 41. Relación de bloques funcionales del decodificador HEVC-RVC.

En el mismo proyecto RVC donde se encuentra esta versión del decodificador están disponibles otras implementaciones similares con ligeras variaciones respecto de la utilizada en este trabajo. Por ejemplo, está disponible una versión que incluye la comprobación de errores mediante *Message-Digest Algorithm 5* (MD5) [MD5].

La FU HevcDecoder está a su vez compuesta, de forma jerárquica, por otras 38 FUs y 39 actores. En la Tabla 42 se resumen las características de las FUs *source* y *Display*, así como las de las FUs que integran el HevcDecoder.

FU	Descripción			
	Jerárquica	Nº de FUs	Nº de actores	Funcionalidad
Source	no	-	1	Captura del <i>bitstream</i> a decodificar
Algo_parser	no	-	1	Descodifica el <i>bitstream</i>
xIT	sí	26	21	Transformada inversa
IntraPrediction	no	-	1	Predicción Intra
InterPrediction	no	-	1	Predicción Inter
SelectCU	no	-	1	Reconstrucción de la imagen
GenerateInfo	sí	3	2	Obtiene información para la predicción inter
DecodingPictureBuffer	no	-	1	<i>Buffer</i> de imágenes
DBFilter	sí	3	2	Filtro anti-bloques
SAO	no	-	1	Filtro SAO
Display	no	-	1	Representa las imágenes decodificadas

Tabla 42. Características de las FUs principales que integran el decodificador HEVC-RVC.

10.1.2. Decodificador HEVC RVC YUV

La versión del decodificador HEVC-RVC YUV se desarrolla con el ánimo de mejorar el grado de paralelismo de la versión HEVC-RVC REF. En este caso, y al igual que en distintas versiones de otros decodificadores de vídeo, se divide el procesamiento de las imágenes por sus componentes de luminancia (Y) y crominancias (U y V). De esta manera aquellos actores que en la versión de referencia producían un cuello de botella debido a su elevada carga computacional, ahora se dividen en tres actores diferentes, los cuales pueden mapearse en distintos núcleos, obteniendo así un aprovechamiento más eficiente de los recursos en un entorno multinúcleo.

El nivel más alto de la jerarquía de esta implementación se presenta en la Figura 78, en el Anexo II está disponible el diagrama completo de la FU HevcDecoder.

Figura 78. Diagrama del nivel más alto de la jerarquía del decodificador HEVC-RVC YUV.

La FU *HevcDecoder* esta a su vez compuesta, de forma jerárquica, por otras 31 FUs y 24 actores. Las características de las FUs *Source* y *Display*, junto con las que integran *HevcDecoder*, se listan en la Tabla 43. Si bien el número de actores de esta implementación es menor que el número de los que integraban la versión de referencia, el reparto es más eficiente en este caso, ya que la descomposición funcional se ha realizado en los actores con más carga computacional, mientras que en la versión de referencia la única FU descompuesta en distintos actores era la *xIT*.

FU	Descripción			
	Jerárquica	Nº de FUs	Nº de actores	Funcionalidad
Source	no	-	1	Captura del <i>bitstream</i> a decodificar
Algo_parser	no	-	1	Descodifica el <i>bitstream</i>
xIT	no	-	1	Transformada inversa
IntraPrediction	sí	4	3 (Y, U y V)	Predicción intra por componentes
InterPrediction	sí	4	3 (Y, U y V)	Predicción inter por componentes
SelectCU	sí	4	3 (Y, U y V)	Reconstrucción de la imagen por componentes
QPGen	no	-	1	Analiza el factor de cuantificación
GenerateInfo	sí	3	2	Obtiene información para la predicción inter
DecodingPicture Buffer	sí	4	3 (Y, U y V)	Almacén de imágenes por componentes
DBFilter	sí	5	4 (GenerateBitstream + Y, U y V)	Filtro anti-bloques por componentes
SAO	sí	4	3 (Y, U y V)	Filtro SAO por componentes
Display	no	-	1	Representa las imágenes decodificadas

Tabla 43. Características de las principales FUs que integran el decodificador HEVC-RVC YUV.

La Figura 79 muestra el diagrama simplificado de esta versión del descodificador. Los bloques en verde son aquellos donde el procesamiento se realiza de forma separada por componentes Y, U y V, y el índice sobre las flechas el número de FIFOs que comunican cada bloque.

Figura 79. Diagrama simplificado del descodificador HEVC-RVC YUV.

La ventaja que presenta esta implementación es que facilita la distribución estática de los actores entre los núcleos de la plataforma objetivo. Por ejemplo, un primer acercamiento a una buena distribución de la carga computacional sobre cuatro núcleos será aquella que reparta en tres núcleos el procesamiento de cada componente Y, U y V, dejando el procesamiento de los actores restantes en el cuarto núcleo.

10.1.3. Descodificador HEVC RVC FBn

La última versión del descodificador HEVC RVC es la llamada *Frame-Based* (FB), la cual permite decodificar secuencias de vídeo paralelizando a nivel de imágenes. Existen tres subversiones de HEVC RVC FB: FB2, FB3 y FB4. Cada una de ellas está compuesta por 2, 3 ó 4 descodificadores respectivamente, que pueden procesar ese número de imágenes de forma simultánea.

El principio de funcionamiento de esta implementación se basa en replicar todos los bloques funcionales que implementan un descodificador RVC HEVC YUV, cada uno de los cuales realizará el procesamiento de las imágenes de forma independiente. Inicialmente será necesario disponer de un bloque funcional que se encargue de la entrega sincronizada de las imágenes codificadas a cada uno de los descodificadores. En la parte final, otro bloque funcional se encargará de almacenar las imágenes ya decodificadas y de resolver los posibles problemas de sincronización entre los descodificadores antes de presentarlas por pantalla.

De esta forma, la versión HEVC RVC FB3 está compuesta por cinco FUs principales, como muestra el diagrama simplificado de la Figura 80: tres FUs de decodificación (1) dec0, (2) dec1 y (3) dec2, cada una de las cuales contiene la mayoría de los actores que implementan el descodificador HEVC RVC YUV, salvo los actores *Source*, *Display* y *DPB_{y, u, v}*, (4) una FU *DpbInfo*, que a su vez contiene el actor

MainParser y la FU *GenMVInfo*, compuesta a su vez por los actores *GenerateRefList* y *MvComponentPred*, y (5) una FU llamada *DPB* integrada por los actores *DPB_y*, *DPB_u* y *DPB_v* que almacenan las imágenes por sus componentes YUV descodificadas por los tres descodificadores. El descodificador se completa con tres actores más: un actor *Source*, que lee el *stream* de vídeo a descodificar, un actor *SelectParser* que sincroniza la descodificación de cada uno de los tres descodificadores, y un actor *Display* que muestra por pantalla la secuencia de vídeo descodificada. En el Anexo II se muestra el diagrama de la versión HEVC RVC FB3 en su nivel más alto de la jerarquía.

Figura 80. Diagrama simplificado del nivel más alto de la jerarquía del descodificador HEVC-RVC FB3.

10.2. Banco de pruebas y secuencias utilizadas

Para la realización de todas las pruebas sobre las distintas plataformas se han seleccionado cuatro secuencias clase C (832x480 píxeles), *BasketballDrill* (DB), *BQMall* (BM), *PartyScene* (PS) y *RaceHorses* (RH), todas ellas disponibles en [SHM]. Además, en función de cada prueba se han utilizado dos configuraciones diferentes para cada una de las cuatro secuencias: *All-Intra* (AI) y *Low-Delay* (LD). Finalmente, para cada configuración y secuencia se han elegido dos valores intermedios del factor de cuantificación QP: 27 y 32. Los acrónimos aquí definidos se utilizarán a lo largo de todo el capítulo. La Tabla 44 presenta los datos de régimen binario y PSNR para cada secuencia.

Configuración	Nombre (acrónimo)	Imágenes	QP	Régimen binario (kbps)	PSNR (dB)
<i>All-Intra</i> (AI)	Basketball Drill (BD)	500	27	11072	38,3
			32	5957	35,3
	BQMall (BM)	600	27	13673	39,0
			32	8056	35,9
	PartyScene (PS)	500	27	26615	36,9
			32	15869	33,0
	RaceHorses (RH)	300	27	8806	38,9
			32	5004	35,5
<i>Low-Delay</i> (LD)	Basketball Drill (BD)	500	27	1705	37,0
			32	822	34,1
	BQMall (BM)	600	27	1856	37,2
			32	910	34,3
	PartyScene (PS)	500	27	3391	34,5
			32	1493	31,0
	RaceHorses (RH)	300	27	2225	36,2
			32	987	33,1

Tabla 44. Listado y características de las secuencias de prueba utilizadas.

La utilización sistemática de esta serie de secuencias a lo largo de las distintas pruebas permite realizar comparativas entre las diferentes implementaciones de forma sencilla y clara. De forma puntual se han utilizado otras secuencias con características diferentes, en estos casos se indicará claramente la necesidad de realizar dichas pruebas.

10.3. Análisis del factor de mejora

Uno de los indicadores que comúnmente se utiliza en la evaluación de las implementaciones basadas en arquitecturas multinúcleo es el cociente entre la velocidad alcanzada por la implementación multinúcleo y la que puede obtenerse empleando un solo núcleo. En el caso que nos ocupa las implementaciones son descodificadores de vídeo y el factor de mejora, o *speedup* (S) se calcula como el cociente entre la velocidad media de descodificación, en imágenes por segundo (FPS), de la implementación multinúcleo con relación a la implementación con un solo núcleo. En las pruebas que se recogen en el apartado 10.4 la validación de la metodología de diseño se realiza, por una parte, constatando la correcta funcionalidad de los descodificadores que se han implementado y, por otra, analizando los factores de mejora que se obtienen con diferente número de núcleos.

Con relación al factor de mejora, resulta de interés analizar los aspectos de los que depende, con objeto de poder realizar una estimación cabal de cuál es el factor esperable en cada implementación y tener así una idea acerca de la bondad de los resultados obtenidos en cada diseño. El factor de mejora depende de diversos aspectos, algunos de los más relevantes son el grado de paralelización del código que integra la aplicación, la dependencia entre los datos a procesar, el número de núcleos disponibles en la plataforma objetivo o el impacto de las comunicaciones entre los bloques funcionales de la aplicación.

En este apartado se realiza un análisis del paralelismo de las distintas implementaciones HEVC-RVC utilizadas para verificar la metodología de diseño.

El objetivo de este análisis es extraer, de forma aproximada, los distintos factores de mejora esperables en las implementaciones multinúcleo acometidas. El estudio en profundidad del paralelismo de cada una de las implementaciones, y para cada una de las plataformas utilizadas, excede los objetivos de este trabajo, por lo tanto los índices de mejora aquí estimados deben considerarse como una aproximación razonable de los resultados que cabría esperar de un reparto óptimo de las FUs que integran los descodificadores entre los núcleos de procesamiento.

Para realizar este estudio se ha analizado la carga computacional, en tiempo de ejecución, de las FUs que integran las distintas versiones de los descodificadores HEVC-RVC. En base a estos datos resulta sencillo facilitar una estimación aproximada del factor de mejora esperable. Sin embargo, dicha estimación no tiene en cuenta la posible dependencia de datos durante el procesamiento paralelizado, ni tampoco las penalizaciones por transferencias de datos entre núcleos. Así, se puede concluir que aquellas implementaciones que ofrezcan resultados cercanos a los valores estimados habrán alcanzado grados de paralelización muy satisfactorios.

10.3.1. Cálculo del factor de mejora

Una primera aproximación al cálculo del factor de mejora esperable consiste en analizar la carga computacional de todas las FUs que integran una aplicación, e identificar el actor cuya carga computacional es mayor ($\%CargaActorBott$); este actor supone un cuello de botella (*bottleneck*) a efectos de paralelismo, ya que una vez que se consigan ejecutar el resto de actores, de menor carga computacional, de forma simultánea a éste se habrá alcanzado el máximo factor de mejora. Además, si se identifican actores cuya ejecución vaya a realizarse forzosamente en serie al resto de FUs, su tanto por ciento de carga computacional ($\%CargaActorSerie$) debe tenerse en cuenta en la estimación⁶³. A esta aproximación se le ha dado el nombre de *CargaActorBott* (en adelante CAB) y el factor de mejora se calcula como se muestra en la Fórmula 5:

$$S_{CAB} = \frac{1}{(\%CargaActorBottleneck + \%CargaActorSerie)}$$

Fórmula 5. Estimación del factor de mejora máximo.

Sin embargo, esta aproximación no tiene en cuenta el número de núcleos utilizados. Una aproximación más útil que sí tiene en cuenta el número de núcleos utilizados es la Ley de Amdahl [Amd67]:

$$T' = T \cdot \left((1 - P) + \frac{P}{N} \right)$$

Fórmula 6. Cálculo del tiempo de mejora según la Ley de Amdahl.

⁶³ En la mayoría de descodificadores de vídeo la ejecución de la descodificación de entropía (ED) se ejecuta en serie al resto de bloques.

donde T' es el tiempo de ejecución al aplicar la paralelización, T es el tiempo de ejecución en serie, P es la proporción de tiempo (en %) utilizado por el código paralelizable y N el índice de mejora⁶⁴ introducido en el algoritmo. La Fórmula 6 puede reescribirse según la Fórmula 7 para calcular de manera explícita el factor de mejora:

$$S = \frac{T}{T'} = \frac{1}{(1 - P) + \frac{P}{N}}$$

Fórmula 7. Cálculo del factor de mejora según la Ley de Amdahl.

Como se puede observar la Ley de Amdahl no toma en consideración posibles mejoras en la ejecución de la sección de código serie en una plataforma multinúcleo, y únicamente considera el impacto sobre el factor de mejora de la sección paralelizable de los algoritmos. Sin embargo, sí es posible que el tiempo de ejecución de una sección de código no paralelizable mejore su rendimiento al utilizar varios procesadores al aprovechar ventajas de una arquitectura multinúcleo⁶⁵. A modo de ejemplo en la Figura 81 se presenta el % de la carga computacional requerida para la ejecución de las diferentes FUs de un decodificador HEVC-RVC YUV cuando decodifica la secuencia *BQMall QP 32*, con configuración *All-Intra*, y utilizando 1 y 7 núcleos⁶⁶.

Figura 81. Variación de la carga computacional de las distintas FUs al utilizar 1 y 7 núcleos con la versión HEVC-RVC YUV.

Como se puede observar en la Figura 81, el reparto de la carga computacional es diferente al ejecutar el decodificador con un núcleo y con siete núcleos. En este

⁶⁴ En este caso el índice de mejora puede considerarse igual al número de núcleos utilizados.

⁶⁵ Un ejemplo de dichas ventajas es que cada uno de los núcleos adicionales dispongan de sus propias memorias caché, como es el caso de los procesadores utilizados en la fase de verificación de esta tesis.

⁶⁶ Los datos de esta prueba se han obtenido de las pruebas de verificación detalladas en el apartado 10.4.1. Se ha escogido para esta prueba una configuración *All-Intra*, ya que es con este tipo de secuencias donde se espera que el factor de mejora pueda superar a las estimaciones realizadas con las otras dos aproximaciones al no haber una elevada dependencia de datos.

sentido, la variación más relevante es la reducción del 22% al 16% de la carga total representada por la descodificación de entropía (ED), la cual supone el actor limitante del paralelismo para esta implementación (ver apartado 10.1.2).

Para tener en cuenta estas posibles mejoras puede utilizarse la Ley de Gustafson [Gus88], que ofrece una aproximación más optimista que la Ley de Amdahl; la Fórmula 8 muestra el cálculo del factor de mejora según esta aproximación:

$$S = N - (1 - P) * (N - 1)$$

Fórmula 8. Cálculo del factor de mejora según la Ley de Gustafson.

donde N y P tienen el mismo significado que en las fórmulas anteriores.

La Figura 82 ilustra las diferencias entre las tres aproximaciones mencionadas para el cálculo del factor de mejora. En la figura se muestra el factor de mejora calculado para los tres casos y un número de núcleos entre 1 y 20. Los factores de mejora se han calculado con tres valores distintos del tanto por ciento del código de la aplicación que es paralelizable (50%, 75% y 90%)⁶⁷.

Figura 82. Factores de mejora calculados para un número de núcleos utilizados y distintos porcentajes de carga paralelizable.

Como se puede observar, la estimación del factor de mejora que se obtiene al aplicar la Ley de Gustafson pronostica una mejora en el tiempo de ejecución muy elevada. Este tipo de estimaciones optimistas sólo podrán hacerse realidad cuando la

⁶⁷ En este ejemplo se ha considerado que el tanto por ciento de la ejecución serie es cero.

dependencia de datos durante la ejecución simultánea de un algoritmo sea muy reducida, al tiempo que el procesador multinúcleo donde se realiza dicha prueba disponga de ventajas tanto *hardware* como *software* que permitan reducir el impacto de la carga computacional de las secciones de código no paralelizable.

10.3.2. Estimación del factor de mejora de las implementaciones HEVC-RVC con el procesador TMS320C6678

En este apartado se realiza la estimación del factor de mejora para las implementaciones de descodificadores HEVC-RVC basadas en las versiones HEVC-RVC REF y HEVC-RVC YUV⁶⁸.

Para la realización de estas estimaciones se ha procedido a descodificar cada una de las secuencias listadas en la Tabla 44 (ver apartado 10.2) con un solo núcleo, sin optimizaciones y midiendo el número de ciclos de reloj empleados en la ejecución de cada uno de los actores que integran cada versión del descodificador. Los detalles de las pruebas realizadas se presentan en los apartados 10.3.2.1 para la versión HEVC-RVC REF y 10.3.2.2 para la versión HEVC-RVC YUV, luego en el apartado 10.3.2.3 se calculan los factores de mejora estimados en base a estos datos. Los factores de mejora estimados se compararán posteriormente (ver apartado 11.2.2) con los resultados obtenidos en las pruebas de verificación, los cuales se detallan en el apartado 10.4.

10.3.2.1. Reparto de la carga computacional con la versión HEVC-RVC REF

Esta primera versión de un descodificador HEVC basado en RVC tiene un grado de paralelismo reducido debido principalmente a que las FUs que soportan una mayor carga computacional no se encuentran a su vez divididas en FUs más simples. Esto provoca cuellos de botella que impiden alcanzar grados de mejora elevados cuando se utilizan varios procesadores. La Figura 83 muestra el reparto de la carga computacional, por FUs, medida para esta implementación. Los datos de la Figura 83 (a) se corresponden con la carga computacional media resultante de descodificar todas las secuencias *Low-Delay* del banco de pruebas, y los datos de la Figura 83 (b) se corresponden con la carga computacional obtenida de la descodificación de secuencias *All-Intra*.

⁶⁸ No se ha evaluado el factor de mejora de la versión HEVC-RVC FB3 dado que las pruebas llevadas a cabo con esta versión son parciales y complementarias al resto de la verificación realizada (ver apartado 10.4.3).

Figura 83. Reparto de la carga computacional por FUs del descodificador HEVC-RVC REF sobre la plataforma TMDSEVM320C6678.

Como se puede observar, al descodificar secuencias con configuración *Low-Delay* (a), el actor encargado de la predicción Inter, soporta una carga computacional que ronda el 25% de la carga computacional total. Esto limita inicialmente el factor de mejora a 4, según la primera aproximación (CAB) realizada en el apartado anterior, no importando el número de núcleos utilizados en el proceso de descodificación. Respecto a las secuencias con configuración *All-Intra* se observa como la descodificación de entropía (ED) supone un 28% de la carga total, aproximadamente. En este sentido, el factor de mejora máximo esperable sería ligeramente inferior a 4; si bien la descodificación de este tipo de secuencias implica una dependencia de datos entre actores reducida al no haber predicción Inter, lo que se espera incremente el factor de mejora con varios núcleos.

10.3.2.2. Reparto de la carga computacional con la versión HEVC-RVC YUV

Como ya se introdujo en el apartado 10.1.2, la versión HEVC-RVC YUV permite paralelizar el procesamiento de las imágenes codificadas según las componentes de color Y, U y V, pudiendo obtener así factores de mejora superiores a los esperables con la versión HEVC-RVC REF.

La Figura 84 muestra el reparto de la carga computacional medio de las FUs que integran el descodificador HEVC-RVC YUV, al ejecutar el mismo sobre la plataforma Multi-DSP y descodificar secuencias del banco de test (ver apartado 10.2) con configuración *Low-Delay*. La Figura 84 (izquierda) muestra, en la circunferencia externa, la carga de cada bloque funcional con todas sus componentes agrupadas, en lo que sería una distribución según el diseño de la versión HEVC-RVC REF, y en la circunferencia interna la carga de cada una de las componentes para cada bloque (por las componentes Y, U y V). Como se puede observar, la carga de aquellos módulos más pesados ahora se ve repartida en tres actores, cada uno de los cuales se encargará de procesar cada una de las componentes. De esta manera queda una distribución de la carga más repartida entre las FUs, lo que permite un mayor número de combinaciones

en los mapeos de actores entre los núcleos de la plataforma. Sin embargo, y como se remarca en la Figura 84 (derecha), la carga computacional de los actores *PictureBuffer Y* (BF y) e *InterPrediction Y* (EP y) representa el 15% y 12% del total, respectivamente. En este punto se debe tener en cuenta que además estos actores tienen un volumen de transferencia de datos entre ellos muy elevado, por lo que mapearlos en distintos núcleos puede conllevar una penalización considerable que haga desmerecer la ejecución simultánea de los mismos.

Figura 84. Reparto de la carga computacional media por FUs del descodificador HEVC-RVC YUV al decodificar las secuencias *Low-Delay* con la plataforma TMDSEVM320C6678.

Tomando la aproximación para estimar el factor de mejora CAB, se pueden calcular dos factores de mejora, uno optimista, en el que se mapean en núcleos separados los actores *PictureBuffer_Y* e *InterPrediction_Y* sin que esto suponga una penalización considerable por la comunicación entre ellos, y otro pesimista en el que estos actores deben mapearse de forma conjunta en el mismo núcleo. Además, en esta estimación se incluye la carga computacional de la descodificación de entropía (ED) como % de carga serie. Así resultan los factores de mejora máximos calculados en la Fórmula 9:

$$S_{CAB_{pesimista}} = \frac{1}{(\%carga_{BF_y} + \%carga_{EP_y} + \%carga_{ED(serie)})} = \frac{1}{15\% + 12\% + 8\%} = 2.85$$

$$S_{CAB_{optimista}} = \frac{1}{(\%carga_{BF_y} + \%carga_{ED(serie)})} = \frac{1}{15\% + 8\%} = 4.34$$

Fórmula 9. Estimación del factor de mejora con la versión HEVC-RVC YUV y secuencias *Low-Delay*.

Si bien la versión del descodificador HEVC-RVC YUV mejora considerablemente el grado de paralelismo esperable con respecto a la de referencia, éste sigue siendo reducido para plataformas con un número de núcleos elevado. Este hecho se debe principalmente a dos factores:

- 1) La imposibilidad de paralelizar la ejecución del *Parser*. La descodificación de entropía se debe ejecutar en serie con la del resto de los actores.
- 2) La elevada carga computacional de la predicción Inter Y y del *buffer* de imágenes Y, que suponen, conjuntamente, un 27% de la carga total.
- 3) La elevada tasa de tráfico de datos entre la predicción Inter Y y el *buffer* de imágenes Y produce que en ocasiones no resulte ventajoso ejecutarlos paralelamente en distintos núcleos.

Por último, la Figura 85 muestra los datos relativos a la descodificación de secuencias *All-Intra* con esta versión del descodificador. Como se puede observar, y de la misma forma que se detalló para la versión HEVC-RVC REF, el bloque de la descodificación de entropía es el de mayor carga computacional, suponiendo un 25,6% del total.

Figura 85. Reparto de la carga computacional por FUs del descodificador HEVC-RVC YUV al descodificar secuencias *All-Intra* sobre la plataforma TMDSEVM320C6678.

10.3.2.3. Factores de mejora calculados con los diferentes estimadores

En la Tabla 45 se presentan y resumen las estimaciones de los factores de mejora máximos calculados para las dos implementaciones analizadas con el procesador TMS320C6678. Dichos factores de mejora se han calculado para cada una de las tres aproximaciones presentadas en el apartado 10.3.1. Los valores calculados servirán para conocer de forma aproximada si el rendimiento conseguido al utilizar varios núcleos está próximo al límite, y por lo tanto si el grado de paralelización conseguido se acerca a su valor óptimo estimado. Se ha considerado que el número de núcleos utilizados es

$N=7$ a la hora de calcular el factor de mejora según las leyes de Amdahl (Amd.) y Gustafson (Gus.).

Versión HEVC-RVC	Tipo de secuencias	Actor limitante	% actor limitante	% actor serie	$S_{\text{máx}}$		
					CAB	Amd.	Gus.
REF	LD _{optimista}	EP	24,9%	7,0%	3,1	2,8	5,5
	LD _{pesimista}	EP+BF	45,4%	7,0%	1,9	1,9	4,2
	AI	ED	19,4%	28,4%	2,1	3,2	5,8
YUV	LD _{optimista}	DPB _y	15,0%	8,0%	4,3	3,6	6,1
	LD _{pesimista}	EP _y +DPB _y	27,0%	8,0%	2,8	2,7	5,3
	AI	ED	9,4 %	25,6%	2,8	4,5	5,4

Tabla 45. Comparativa de los respectivos factores de mejora máximos esperados para las distintas implementaciones.

Como se puede observar en la Tabla 45, en la mayoría de los casos la estimación del factor de mejora obtenido con la Ley de Amdahl es el más conservador, y los obtenidos con la Ley de Gustafson los más optimistas. Es importante reseñar que los datos mostrados en esta tabla son estimaciones realizadas a partir de los datos obtenidos de la decodificación de las secuencias de prueba utilizando un único núcleo. En este sentido, es de esperar que el reparto de la carga computacional sufra variaciones al pasar a una ejecución simultánea sobre varios núcleos, produciendo modificaciones en los factores de mejora.

10.4. Pruebas de verificación de la metodología de diseño

La metodología de diseño propuesta en el capítulo 9 ha sido verificada mediante la implementación de tres versiones distintas de decodificadores HEVC-RVC para cuatro plataformas distintas; en esta sección se presentan los resultados de dichas implementaciones. En estas pruebas de verificación se han tenido en cuenta todos los aspectos contemplados en la metodología de diseño, a excepción de las técnicas para la reducción del consumo de energía descritas en el apartado 9.6, cuyos resultados se presentan de forma independiente en el apartado 10.4.4.

Se ha diseñado un banco de pruebas homogéneo en el que se prueba con las secuencias listadas en la Tabla 44 (ver apartado 10.2) utilizando la estrategia de remapeo dinámico MKEC (ver apartado 4.5.4) cada 50 imágenes ($-r\ 50 -a$). Con esta configuración se han probado las versiones HEVC-RVC REF y HEVC-RVC YUV sobre las plataformas Multi-DSP, Odroid y PC. Para la versión HEVC-RVC FB3, al tratarse de la última versión desarrollada y teniendo en cuenta las limitaciones especificadas en el apartado 10.1.3, se ha realizado un conjunto de pruebas más reducido. La Tabla 46 resume los decodificadores que se han implementado, las plataformas sobre las que se han realizado las pruebas y los apartados donde se detallan los resultados de cada una de ellas.

Plataforma Descodificador	TMDSEVM6678	Odroid U3	PC – I7	EVMK2H
HEVC-RVC REF	(apartado 10.4.1)	(apartado 10.4.2)	(apartado 10.4.2)	
HEVC-RVC YUV	(apartado 10.4.1)	(apartado 10.4.2)	(apartado 10.4.2)	
HEVC-RVC FB3			(apartado 10.4.3)	(apartado 10.4.3)

Tabla 46. Relación de plataformas y descodificadores implementados durante el diseño de la metodología.

10.4.1. Pruebas realizadas con la plataforma Multi-DSP TMS320C6678

En primer lugar se van a presentar los resultados relativos a las implementaciones realizadas con la metodología propuesta sobre el procesador Multi-DSP TMS320C6678. Se han descodificado las cuatro secuencias descritas en el banco de pruebas, con dos configuraciones distintas (*Low-Delay* y *All-Intra*) y dos factores de cuantificación QP (27 y 32). En las gráficas se muestra el número de FPS medio al utilizar 1, 2, 4, 6 y 7 núcleos. Los datos para 3 y 5 núcleos se han omitido ya que los valores obtenidos no modifican la tendencia de los resultados en ningún caso. Además, se han realizado las mismas pruebas utilizando optimizaciones del código fuente generado por Orcc, principalmente sobre la predicción Inter, tal y como se detalló en el apartado 9.7. La gráfica de la Figura 86 muestra el resumen de los resultados obtenidos con la implementación del descodificador RVC-HEVC REF para secuencias *Low-Delay*. Estos datos se encuentran desglosados por factor QP y la utilización o no de optimizaciones (OPT/N_OPT). Los factores de mejora o *speedups* (S) se muestran como columnas agrupadas para número de núcleos utilizado en la descodificación.

Figura 86. Factor de mejora y FPS con HEVC-RVC REF con y sin módulos optimizados y secuencias *Low-Delay* sobre la plataforma TMDSEVM6678, estrategia de mapeo MKEC.

Como se puede observar en la Figura 86, la evolución del factor de mejora alcanza su máximo con 7 núcleos para esta versión del descodificador. Tanto para 6, como para 7 núcleos el factor de mejora medio sin optimizaciones es de aproximadamente 2 veces con respecto a un núcleo. Como se indicó en el apartado 10.3.2.1 esta versión ofrece un paralelismo limitado por la forma en que se han implementado los actores que integran el descodificador. Las optimizaciones llevadas a cabo en el procesamiento de la predicción Inter permiten obtener una mejora media aproximada del 15%, logrando un factor de mejora de 2,2, al reducir la carga computacional del actor más pesado (Predicción Inter), y por tanto el que más limita el grado de paralelismo de esta implementación. En el Anexo III se realiza un análisis pormenorizado del efecto que estas optimizaciones tienen sobre el rendimiento de cada uno de los actores.

En la Figura 87 se presentan los resultados de descodificar secuencias *All-Intra* con la misma versión del descodificador (HEVC-RVC REF). En este caso no se muestran datos relativos a la utilización de módulos optimizados debido a que su impacto es mínimo, ya que la carga computacional de los actores encargados del procesamiento de la predicción Inter es residual.

Figura 87. Factor de mejora y FPS con HEVC-RVC REF y secuencias *All-Intra* sobre la plataforma TMDSEVM6678, estrategia de mapeo MKEC.

Como se puede observar en la Figura 87, el factor de mejora obtenido es considerablemente mayor que el conseguido con secuencias *Low-Delay*; superando, con 6 y 7 núcleos, una mejora de 3,5 veces con respecto a un núcleo. Esto se debe a la reducida dependencia de datos durante la descodificación existente en el procesado de

secuencias *All-Intra*. Por último, se puede observar cómo con 6 núcleos se ha alcanzado el máximo factor de mejora para esta configuración, viéndose incluso reducido al utilizar 7 núcleos. Este descenso en el rendimiento con un número mayor de núcleos se debe a la dificultad para encontrar una distribución de la carga computacional capaz de mejorar el rendimiento obtenido con 6. En este punto se hace difícil paralelizar la ejecución de más actores, al tiempo que las comunicaciones entre ellos se ven penalizadas por encontrarse en núcleos distintos.

Las limitaciones que la versión HEVC-RVC REF presenta en el factor de mejora máximo se ven reducidas en la versión HEVC-RVC YUV. Como se detalló en el apartado 10.1.2, esta versión permite la paralelización del proceso de descodificación en las componentes de luminancia y crominancias del vídeo, lo que permite obtener mejoras más significativas del rendimiento al aumentar el número efectivo de núcleos utilizados.

La Figura 88 muestra los resultados obtenidos para esta versión del descodificador sobre la plataforma Multi-DSP al descodificar secuencias con configuración *Low-Delay*. En este caso los datos se muestran de la misma forma que en el caso anterior, es decir, desglosados por la utilización o no de optimizaciones (OPT/N_OPT) y de forma independiente para cada factor de cuantificación QP.

Figura 88. Factor de mejora y FPS con HEVC-RVC YUV con y sin módulos optimizados y secuencias *Low-Delay* sobre la plataforma TMDSEVM6678, estrategia de mapeo dinámico MKEC.

Como se puede observar en la Figura 88, el factor de mejora es 3 al utilizar 7 núcleos con respecto al rendimiento obtenido con un único núcleo. También se puede observar cómo el factor de mejora se acerca a su máximo con 7 núcleos al producirse un incremento reducido al pasar de 6 a 7 núcleos. Por último, y al igual que ocurre con la

versión HEVC-RVC REF, las optimizaciones introducidas permiten obtener una mejora aproximada del 15% en el rendimiento global.

La Figura 89 muestra los resultados obtenidos al decodificar secuencias *All-Intra* con la versión HEVC-RVC YUV. Los resultados se muestran de la misma forma que en el caso correspondiente a la versión HEVC-RVC REF.

Figura 89. Factor de mejora y FPS con HEVC-RVC YUV y secuencias *All-Intra* sobre la plataforma TMDSEVM6678, estrategia de mapeo dinámico MKEC.

Del mismo modo que ocurre en los resultados con secuencias *Low-Delay*, el factor de mejora obtenido en este caso es considerablemente superior al conseguido con la versión HEVC-RVC REF, superando el 5 con 7 núcleos, con respecto al rendimiento obtenido con un único núcleo. Además, en este caso, la tendencia de los resultados con 6 y 7 núcleos hace pensar que el máximo de mejora no ha sido alcanzado y que podría verse incrementado con un número de núcleos mayor que 7.

10.4.2. Verificación sobre plataformas GPP

Cada una de las implementaciones que se han llevado a cabo, descritas en el apartado 10.1, se han probado igualmente sobre otras dos plataformas, un PC y una plataforma de desarrollo para sistemas empotrados. Esto se ha hecho con el objetivo de, por un lado, poder realizar comparativas de los resultados obtenidos entre la plataforma Multi-DSP y las otras plataformas, y por el otro de asegurar la correcta funcionalidad sobre distintas plataformas de las versiones basadas en OpenMP para el aprovechamiento de un entorno con varios núcleos.

El PC utilizado para estas pruebas es un Intel Core i7 950 [I7] de cuatro núcleos sobre el que se ha instalado la distribución 12.04 LTS de 64 bits de Ubuntu [UBU], y la plataforma de desarrollo para sistemas empotrados es una Odroid U3 (ODR) con un procesador Exynos 4412 donde se ha instalado una distribución Linux Xubuntu para Odroid versión 3.8.13 [ODR], también con cuatro núcleos en su arquitectura. Los

detalles de todas las plataformas utilizadas en este trabajo de investigación se presentaron en el apartado 7.3.

A la hora de tomar en consideración todos los resultados a continuación deben tenerse en cuenta las diferencias considerables que se dan entre los tres tipos de arquitecturas. Si bien es cierto que la frecuencia del reloj del sistema es posiblemente el factor más determinante en el rendimiento conseguido para cada implementación, otros factores como el tamaño de las memorias caché, la velocidad de transferencia entre los distintos niveles de memoria, o la velocidad de acceso a memoria externa (DDR2 o DDR3), también deben tenerse en cuenta.

Se ha llevado a cabo un conjunto de pruebas análogo a las realizadas con la plataforma Multi-DSP: se han descodificado las cuatro secuencias descritas en el banco de pruebas, con dos configuraciones distintas (*Low-Delay* y *All-Intra*) y dos QPs (27 y 32). En las gráficas se muestra el número de FPS medio al utilizar 1, 2, 3 y 4 núcleos. La gráfica de la Figura 91 muestra el resumen de los resultados obtenidos con la implementación del descodificador RVC-HEVC REF para secuencias *Low-Delay* con el PC, y la gráfica de la Figura 91 con la plataforma Odroid. En ambos casos los datos se encuentran desglosados por factor QP para cada plataforma GPP. Los factores de mejora o *speedups* (S) se muestran como columnas agrupadas para número de núcleos utilizado en la descodificación.

Figura 90. Factor de mejora y FPS con HEVC-RVC REF y secuencias *Low-Delay* sobre la plataforma PC, estrategia de mapeo MKEC.

Figura 91. Factor de mejora y FPS con HEVC-RVC REF y secuencias *Low-Delay* sobre la plataforma Odroid, estrategia de mapeo MKEC.

Como se puede observar en la Figura 91, la evolución del factor de mejora alcanza su máximo con 3 núcleos para esta versión del descodificador en la plataforma Odroid, no ocurriendo lo mismo sobre el PC (Figura 90), que logra una mejora mayor. En ambos casos el factor de mejora alcanzado no supera una ganancia de 1,8 con respecto al rendimiento con un único núcleo; pero como se detalló en el apartado 10.3.2.1 esta versión ofrece un paralelismo limitado.

Seguidamente se presentan los resultados de descodificar secuencias *All-Intra* con la misma versión del descodificador (HEVC-RVC REF). En este caso, y al igual que ocurrió sobre la plataforma Multi-DSP (ver apartado 10.4.1), el factor de mejora obtenido es considerablemente mayor que el conseguido con secuencias *Low-Delay*. El factor de mejora con el PC (Figura 92) llega a 1,9 con 4 núcleos, y con la plataforma Odroid se alcanza una mejora de 2,5 (Figura 93). Sin embargo, se puede observar como la tendencia no da muestras de haber llegado al máximo de mejora en ninguno de los dos casos.

Figura 92. Factor de mejora y FPS con HEVC-RVC REF y secuencias *All-Intra* sobre la plataforma PC, estrategia de mapeo MKEC.

Figura 93. Factor de mejora y FPS con HEVC-RVC REF y secuencias *All-Intra* sobre la plataforma Odroid, estrategia de mapeo MKEC.

Como ya se explicó en apartados anteriores (ver apartados 10.1.2 y 10.3.2.1), las limitaciones que la versión HEVC-RVC REF presenta en el factor de mejora máximo se ven reducidas en la versión HEVC-RVC YUV. Así, la Figura 94 muestra los resultados obtenidos para esta versión del descodificador con el PC, y la Figura 95 con la plataforma Odoird al descodificar secuencias con configuración *Low-Delay*.

Figura 94. Factor de mejora y FPS con HEVC-RVC YUV y secuencias *Low-Delay* sobre la plataforma PC, estrategia de mapeo MKEC.

Figura 95. Factor de mejora y FPS con HEVC-RVC YUV y secuencias *Low-Delay* sobre la plataforma Odroid, estrategia de mapeo MKEC.

Como se puede observar en la Figura 94, el factor de mejora en este caso alcanza las 2 veces al utilizar 4 núcleos con respecto al rendimiento obtenido con un único núcleo tanto en la plataforma PC como en la Odroid (Figura 95). También se puede observar cómo el factor de mejora tiende a su máximo con 4 núcleos al utilizar la Odroid, y en menor medida sobre el PC.

La Figura 96 muestra los resultados obtenidos de descodificar secuencias *All-Intra* con la versión HEVC-RVC YUV con el PC, y la Figura 97 con la plataforma Odroid. Los resultados se muestran de la misma forma que en el caso correspondiente a la versión HEVC-RVC REF.

Figura 96. Factor de mejora y FPS con HEVC-RVC YUV y secuencias *All-Intra* sobre la plataforma PC, estrategia de mapeo MKEC.

Figura 97. Factor de mejora y FPS con HEVC-RVC YUV y secuencias *All-Intra* sobre la plataforma Odroid, estrategia de mapeo MKEC.

De la misma forma que ocurre con los resultados para secuencias *Low-Delay*, el factor de mejora obtenido en este caso es considerablemente superior al conseguido con la versión HEVC-RVC REF, superando las 2,5 veces de mejora con 3 núcleos con respecto al rendimiento obtenido con un único núcleo en ambos casos. Sin embargo, la tendencia de los resultados con 4 núcleos hace pensar que el máximo de mejora ha sido alcanzado ya con 3 núcleos.

Adicionalmente al conjunto de pruebas que se han descrito, se ha considerado de utilidad comparar el rendimiento ofrecido por el código generado a partir del *back-end* C original (el cual utiliza *threads*) y el nuevo *back-end* OpenMP. Los resultados más relevantes de esta comparación se resumen en la Tabla 47 [CFG16]. En esta tabla se muestran los valores medios de rendimiento, en número de imágenes por segundo

codificadas (FPS), correspondientes a decodificar las secuencias *Low-Delay* del mismo banco de pruebas descrito en el apartado 10.2 para el decodificador HEVC-RVC YUV, utilizando el código fuente generado por ambos, el *back-end* C original y el nuevo *back-end* basado en OpenMP. Se han sombreado en azul aquellos resultados en los que el rendimiento obtenido a partir del *back-end* OpenMP supera al *back-end* C original.

Plataforma	QP	Rendimiento en FPS por número de núcleos			
		1	2	3	4
PC (<i>back-end</i> OpenMP)	27	27,8	49,0	55,7	59,2
	32	33,4	57,6	72,6	80,6
PC (<i>back-end</i> C threads)	27	26,8	48,5	55,4	63,5
	32	33,2	57,1	69,9	79,2
ODR (<i>back-end</i> OpenMP)	27	7,6	13,9	15,1	17,1
	32	9,2	16,2	19,4	21,7
ODR (<i>back-end</i> C threads)	27	7,4	13,9	15,3	17,1
	32	8,7	16,4	19,7	21,1

Tabla 47. Comparativa del rendimiento (en FPS) de las implementaciones con OpenMP y threads del decodificador HEVC-RVC sobre plataformas PC y Odroid para varios núcleos.

Las versiones con OpenMP ofrecen mejores datos en 10 resultados (sombreado azul), las versiones con *threads* presentan mejores datos en 4 resultados (sombreado rojo), y ambas implementaciones ofrecen resultados idénticos en 2 tests (sin sombreado). En ningún caso la diferencia supera el 7%. Por lo tanto los resultados obtenidos para ambas implementaciones y en ambas plataformas son muy similares, lo que permite confirmar que la sobrecarga introducida por OpenMP es despreciable y que ésta es una solución competitiva con respecto a la original.

10.4.3. Verificación con la versión HEVC-RVC FB3

De forma complementaria a las pruebas realizadas con las versiones HEVC-RVC REF y HEVC-RVC YUV, se ha utilizado también el nuevo *back-end*, que automatiza la metodología de diseño descrita en el capítulo 9, para realizar una implementación de la versión HEVC-RVC FB3. La intención de estas pruebas es ampliar el proceso de verificación, con una tercera versión del decodificador y sobre otra plataforma Multi-DSP, en concreto, la plataforma EVMK2H (ver apartado 7.4). También se han realizado las mismas pruebas sobre la plataforma PC⁶⁹. Debido a las limitaciones de memoria de las plataformas Multi-DSP, se ha realizado un conjunto de pruebas más reducido. En particular, se han utilizado solamente imágenes de la secuencia BQSquare (432x280 píxeles, clase D) con un factor de cuantificación QP 32 y configuración *All-Intra*.

⁶⁹ No se han realizado pruebas de esta versión con la plataforma Odroid debido a un problema de incompatibilidad de versiones de FFMPEG.

10.4.3.1. Implementación y resultados de la versión HEVC RVC FB3 sobre la plataforma Multi-DSP

HEVC-RVC FB3 es la implementación más reciente de un decodificador HEVC realizada con el estándar RVC. Está compuesta por un total de 69 actores y 212 FIFOs. Si bien su diseño permite conseguir una mayor paralelización de la ejecución de los actores en un entorno multinúcleo, también supone un reto a la hora de llevar a cabo la implementación sobre una plataforma con recursos de memoria reducidos, como es el caso de la plataforma Multi-DSP. En este sentido, debe tenerse en cuenta que esta implementación está compuesta por tres decodificadores completos YUV, cada uno de ellos compuesto por sus respectivos actores, además del resto de FUs descritas en el apartado 10.1.3. De esta forma se consigue paralelizar el procesamiento tanto a nivel de imágenes como a nivel de componentes YUV.

El principal problema surge a la hora de ubicar en memoria compartida las 212 FIFOs. Cada FIFO suele configurarse con un tamaño de entre 16 kB y 32 kB (ver apartado 9.4.1), más el espacio necesario para el resto de campos que la integran, como los punteros de lectura/escritura. Este hecho implica que serán necesarios, al menos, entre 4 MB y 7 MB únicamente para las FIFOs. Como ya se introdujo en el apartado 7.1, la plataforma TMS320C6678 dispone de 4 MB de memoria compartida, que, además, debe alojar el código de programa y algunos recursos de OpenMP (ver apartado 9.3). La plataforma EVMK2H (ver apartado 7.4), dispone de una memoria compartida de 6 MB; por este motivo, y para poder llevar a cabo las pruebas deseadas es necesario configurar un tamaño de FIFO de 8 kB. De esta forma hay espacio suficiente para alojar todas las FIFOs en memoria compartida, más el código del programa y los demás recursos necesarios. Mediante esta configuración se ha realizado una prueba de concepto del decodificador HEVC-RVC FB3 con la metodología de diseño propuesta. Los siguientes parámetros del *back-end* han tenido que ser modificados para ser adaptados a la plataforma EVMK2H:

- 1) Utilización del fichero de configuración GEL propio de la plataforma.
- 2) Modificación del fichero *platform.ccxml* de configuración del emulador, ya que la plataforma EVMK2H dispone de un emulador modelo XDS200.
- 3) Modificación del mapa de memoria, tal y como se detalla en la Tabla 48.

Memoria	Tamaño	Dirección inicial	Dirección final	Consideraciones
L1D	32 KB	--	--	Todo caché
L1P	32 KB	--	--	Todo caché
L2 caché	512 KB	--	--	
L2 SRAM	512 KB	0x00800000	0x00880000	
MSMCSRAM	6016 KB	0x0C020000	0x0C5FFFFFFF	Memoria
DDR3	512 MB	0xA0000000	0xBFFFFFFF	compartida
OMP_MSMC_NC_PHY	128 KB	0x0C000000	0x0C01FFFF	Reservado
OMP_MSMC_NC_VIRT	128 KB	0xE0000000	0xC001FFFF	OpenMP

Tabla 48. Modificación del mapa de memoria de la plataforma EVMK2H.

En la Tabla 49 se muestran los resultados obtenidos sobre la plataforma EVMK2H con dos estrategias de mapeo dinámico: MKEC y QM (ver apartado 4.5.4). Aquí se muestran, para cada estrategia, la media de FPS resultantes de la decodificación de la secuencia BQSquare (432x280 píxeles, clase D) con un factor de cuantificación QP 32 y configuración *All-Intra*. Se ha establecido, al igual que en las pruebas anteriores, un remapeo cada 50 imágenes (*-r 50 -a*). Además, para cada estrategia se ha calculado el factor de mejora, obtenido con respecto a la utilización de un único núcleo. En dicha tabla se puede observar cómo los resultados obtenidos con la estrategia MKEC son ligeramente mejores que con la estrategia QM, esto se debe a que la primera de ellas está diseñada para buscar un mejor reparto en función del volumen de las comunicaciones, factor especialmente limitante en las plataformas tipo Multi-DSP, y más concretamente en esta.

Estrategia	FPS por número de núcleos					Factor de mejora			
	1	2	4	6	7	2	4	6	7
MKEC	3,9	6,5	9,5	11,7	12,1	1,7	2,4	3,0	3,1
QM	3,9	5,8	9,8	11,5	11,7	1,5	2,5	2,9	3,0

Tabla 49. Rendimiento medio, en FPS, y factor de ganancia con distintas estrategias de mapeo dinámico sobre la plataforma EVMK2H.

Como se puede observar, los resultados conseguidos por cualquiera de las dos estrategias, en términos de factor de mejora de rendimiento, son razonables pero no superan los obtenidos con la versión HEVC-RVC YUV. Esto es debido a que ha sido necesario reducir significativamente el tamaño de las FIFOs para poder implementar esta versión del decodificador HEVC-RVC. Esta reducción origina que el número de operaciones de lectura/escritura, y por consiguiente de instrucciones WB/INV, aumente proporcionalmente, limiando así el factor de mejora que se puede obtener con un número de núcleos elevado.

En la Figura 98 se muestra, de forma detallada, la evolución de la tasa de decodificación entre 1 y 7 núcleos, junto con los respectivos factores de ganancia realizando una reconfiguración cada 50 imágenes⁷⁰.

⁷⁰ En las gráficas correspondientes a las otras implementaciones no se han presentado resultados tan detallados para reducir la complejidad de la representación.

Figura 98. Evolución del rendimiento durante la decodificación de una secuencia *All-Intra* entre 1 y 4 núcleos y factor de ganancia obtenido cada 50 imágenes sobre la plataforma EVMK2H, estrategia MKEC.

10.4.3.2. Implementación y resultados de la versión HEVC RVC FB3 sobre la plataforma GPP de altas prestaciones

La implementación de la versión FB3 del descodificador HEVC-RVC sobre un GPP de altas prestaciones (PC) se ha realizado siguiendo los mismos pasos que en el resto de implementaciones.

Aunque el entorno ideal para la realización de estas pruebas sería una plataforma con un número de núcleos elevado⁷¹, se ha llevado a cabo sobre el mismo PC empleado en el resto de pruebas. Dado que en este caso no se dan las limitaciones de memoria descritas en el caso de las plataformas Multi-DSP no ha sido necesario reducir a 8 kB el tamaño de las FIFOs.

En este caso también se han utilizado dos estrategias de mapeo dinámico: MKEC y QM (ver apartado 4.5.4). La Tabla 50 muestra los resultados obtenidos. Aquí se muestran, para cada estrategia, la media de FPS resultantes de la decodificación de la secuencia *All-Intra BQMall* con QP 32 completa. Al igual que en el resto de pruebas, se realiza una reconfiguración cada 50 imágenes; esto implica que cada 50 *frames* se realizará una nueva distribución de actores entre los núcleos. Además, para cada estrategia se ha calculado el factor de mejora, obtenido respecto al número de FPS cuando solo se utiliza un núcleo.

⁷¹ Las plataformas GPP utilizadas disponen de 4 núcleos en su arquitectura, las pruebas mostrarían resultados más relevantes sobre una plataforma con al menos 6 u 8 núcleos.

Estrategia	FPS por número de núcleos				Factor de mejora		
	1	2	3	4	2	3	4
MKEC	2,8	5,1	8,1	10,1	1,8	2,9	3,6
QM	2,8	5,2	7,8	9,8	1,9	2,8	3,5

Tabla 50. Rendimiento medio, en FPS, y factor de ganancia con distintas estrategias de mapeo dinámico.

Como se puede observar los resultados conseguidos por cualquiera de las dos estrategias, en términos de factor de ganancia, son razonables ya que están próximos al máximo para cada número de núcleos utilizado. Siendo aproximadamente de 1,9 con dos núcleos, 2,9 con tres y 3,5 con cuatro.

En la Figura 99 se muestra la evolución de la tasa de decodificación usando entre 1 y 4 núcleos, junto con los respectivos factores de ganancia cada 50 imágenes decodificadas.

Figura 99. Evolución del rendimiento durante la decodificación de una secuencia All-Intra empleando entre 1 y 4 núcleos y factor de ganancia obtenido cada 50 imágenes sobre una plataforma GPP de altas prestaciones, estrategia MKEC.

10.4.4. Reducción del consumo de energía sobre la plataforma TMS320C6678

Para finalizar la fase de verificación de la metodología de diseño propuesta se han llevado a cabo pruebas complementarias que permiten analizar el impacto del algoritmo diseñado para la reducción del consumo energético sobre plataformas Multi-DSP, y en concreto con el procesador TMS320C6678. Dicho algoritmo se centra en el aprovechamiento de las ventajas ofrecidas por las especificaciones basadas en RVC-CAL, así como en la utilización de reconfiguraciones dinámicas de actores, como

se detalló en el apartado 9.6. Cabe destacar que éste es uno de los últimos aspectos introducidos en la metodología de diseño, y por lo tanto los detalles de su integración en la misma, y los resultados obtenidos no han sido aún publicados. Por este motivo a continuación se muestran los detalles de una única prueba de validación de este aspecto de la metodología.

En primer lugar se ha caracterizado el consumo del procesador para distintos escenarios (ver apartado 9.7) durante el proceso de descodificación. Para la realización de estas pruebas se ha utilizado en todo momento la versión HEVC-RVC YUV y secuencias *Low-Delay*. La Figura 100 muestra el total de la energía consumida en Julios⁷² por el procesador TMS320C6678 para la descodificación completa de la secuencia *BQMall QP 32* con distintas configuraciones de frecuencia de reloj y número de núcleos utilizados. Como se puede observar, el mínimo de consumo se da al utilizar 6 núcleos y una frecuencia de reloj de 1,2 GHz; esto se debe a que la limitada mejora en el rendimiento (en FPS) que se obtiene al pasar de 6 a 7 núcleos no compensa el incremento de energía demandada por un núcleo más.

Figura 100. Energía consumida en el proceso de descodificación por el procesador TMS320C6678 para distintos escenarios de consumo con la versión HEVC-RVC YUV.

En este punto podría concluirse que en todos los casos⁷³ será más beneficioso descodificar el vídeo con las máximas capacidades del procesador, ya que consumirá menos energía y el proceso de descodificación finalizará antes. Sin embargo, debe tenerse en cuenta que el objetivo es que un descodificar sea capaz de procesar imágenes a una velocidad más o menos constante y cercana a la tasa de reproducción (normalmente entre 24 y 60 FPS). Si el descodificador procesa imágenes a más velocidad de la establecida para su correcta reproducción dichas imágenes pasan a un

⁷² 1 Julio = 1 Vatio por segundo.

⁷³ Estas pruebas se han llevado a cabo con la versión HEVC-RVC YUV del descodificador, con otras versiones el modelo podría cambiar.

buffer temporal a la espera de presentarse por pantalla. La utilización de este tipo de *buffers* siempre es necesaria, pero cuanto menor sea su tamaño menor será el espacio en memoria ocupado. Además, podría darse la situación en la que dicho almacén se llenase de imágenes, en este caso se procedería a eliminar imágenes ya decodificadas, omitiendo su representación por pantalla y produciendo efectos indeseables en la reproducción del vídeo.

Tomando en consideración lo anterior, se ha diseñado el algoritmo descrito en el apartado 9.6.2, mediante el cual el usuario fija una tasa de decodificación (en FPS), y el programa modificará dinámicamente tanto la frecuencia de reloj del procesador como el número de núcleos utilizados, para intentar decodificar a dicha tasa. Con el objetivo de validar el funcionamiento de este algoritmo se han concatenado tres secuencias de vídeo con resoluciones y factores de cuantificación distintos: *RaceHorses* 832x480 QP 32 (300 imágenes), *BlowingBubbles* 416x240 QP 22 (500 imágenes) y *BasketballDrill* 832x480 QP 32 (500 imágenes). Mediante la decodificación de la concatenación de varias secuencias con características diferentes se pretende obligar a que el algoritmo realice los cambios pertinentes en los recursos *hardware* especificados. La Tabla 51 muestra el escenario de consumo establecido por el algoritmo diseñado cada 50 imágenes decodificadas, estrategia de mapeo MKEC. En este caso el objetivo de FPS establecido es de 12.

Imágenes decodificadas	FPS	Escenario
0-50	2,6	1 núcleo @ 0,8 GHz
50-100	12,6	7 núcleos @ 1 GHz
100-150	11,8	7 núcleos @ 1,2 GHz
150-200	13,6	7 núcleos @ 1 GHz
200-250	12,1	7 núcleos @ 1,2 GHz
250-300 (fin de la primera secuencia)	13,6	7 núcleos @ 1,2 GHz
300-350	25,6	7 núcleos @ 1 GHz
350-400	14,2	2 núcleos @ 0,8 GHz
400-450	11,3	1 núcleo @ 1 GHz
450-500	9,7	1 núcleo @ 1 GHz
500-550	13,8	2 núcleos @ 0,8 GHz
550-600	10,1	1 núcleo @ 1 GHz
600-650	13,8	2 núcleos @ 0,8 GHz
650-700	10,7	1 núcleo @ 1 GHz
700-750	13,6	2 núcleos @ 0,8 GHz
750-800 (fin de la segunda secuencia)	9,2	1 núcleo @ 1 GHz
800-850	5,9	2 núcleos @ 0,8 GHz
850-900	13,6	7 núcleos @ 1 GHz
900~1300	~12,5	6 núcleos @ 1,2 GHz

Tabla 51. Escenarios de consumo seleccionados en cada momento durante la decodificación de varias secuencias concatenadas.

Como se puede observar, inicialmente la decodificación se inicia con un núcleo a 0,8 GHz (punto de menor rendimiento), y tras la decodificación de las primeras 50 imágenes se estima necesario utilizar más núcleos (7 núcleos a 1 GHz en este caso) para

alcanzar los 12 FPS establecidos para la descodificación. Este modo se mantiene estable, dándose pequeñas modificaciones de la frecuencia de trabajo entre 1 y 1,2 GHz, durante la descodificación de la primera secuencia (300 primeras imágenes). En ese momento se inicia la descodificación de la segunda secuencia, de menor resolución y por tanto requerirá menos potencia de cálculo para su descodificación a 12 FPS. Como se puede observar el algoritmo reduce de 7 a 2 núcleos utilizados en la descodificación tras percatarse (entre las 350 y 400 imágenes descodificadas) del cambio de situación. La configuración establecida sufre ligeras modificaciones (cambia de 1 a 2 núcleos utilizados y de 1 a 0,8 GHz varias veces) a lo largo de la descodificación de la segunda secuencia. Por último, se inicia la descodificación de la última secuencia, que vuelve a tener una resolución de 832x480, por lo que es necesario recuperar un escenario de mayor rendimiento (7 núcleos tras las 900 imágenes descodificadas). La descodificación continúa estable con 6 núcleos a 1,2 GHz hasta el final del vídeo.

La Figura 101 muestra de forma gráfica el proceso de descodificación detallado anteriormente. Aquí se muestra la evolución de la descodificación en FPS (línea azul), el número de FPS objetivo (12 FPS en naranja), y las transiciones de secuencias (línea vertical verde punteada).

Figura 101. Descodificación de varias secuencias concatenadas y evolución del rendimiento.

De esta forma se hace un uso eficiente de las capacidades de cómputo del procesador, utilizando en cada momento únicamente el número de núcleos, y trabajando a una frecuencia de reloj determinada, necesarios para descodificar una secuencia a una tasa constante.

Para valorar la reducción del consumo energético al utilizar el algoritmo implementado se ha medido el consumo total de todo el proceso de la descodificación

(caso 1). Éste se ha comparado con el consumo resultante de decodificar la misma secuencia en el escenario de máximo rendimiento (caso 2: 6 núcleos a 1,2 GHz). Como se puede observar en los resultados de la Tabla 52, el caso 2 presenta un menor consumo energético, debido a que es capaz de decodificar toda la secuencia en menos tiempo aún utilizando en todo momento más núcleos y a una frecuencia mayor que el caso 1.

Consumo del procesador caso 1	Tiempo total decodificación caso 1 (t_1)	Consumo del procesador caso 2	Tiempo total decodificación caso 2 (t_2)
640 J	123,1 s	572 J	72,2 s

Tabla 52. Comparativa del consumo energético y tiempo de decodificación de dos escenarios.

Sin embargo, el hecho de decodificar un vídeo a mayor velocidad de la deseada (12 FPS en este caso) implica que el procesador detendrá la decodificación, para mantener el ratio de imágenes por segundo especificado en la codificación. En los momentos en los que el procesador detenga la decodificación el consumo de energía puede reducirse, pero de forma limitada. Así, se ha estimado un consumo reducido para dichos periodos (consumo reducido: 6 núcleos a 0,8 GHz, potencia demandada 4,74 W), siendo t_3 la suma de todos ellos. Por ello se puede recalcular el consumo total del caso 2 como se muestra en la Fórmula 10:

$$\begin{aligned}
 \text{Consumo Total}_{\text{caso2}} &= \text{Consumo}_{\text{decodificación}} + t_3 * \text{Consumo}_{\text{mínimo}} \\
 t_3 &= \frac{\text{imágenes}_{\text{secuencia}}}{FP_{\text{establecido}}} - t_2 = \frac{1300 \text{ imágenes}}{12 \text{ FPS}} - 72,2 = 36,1 \\
 \text{Consumo Total}_{\text{caso}} &= 572 + 36,1 * 4,74 = 743,1 \text{ J} \\
 \text{Ahorro conseguido} &= \text{Consumo Total}_{\text{escenario2}} - \text{Consumo Total}_{\text{caso1}} = \\
 &743,1 - 640 = 103,1 \text{ J} \quad (14\% \text{ de reducción})
 \end{aligned}$$

Fórmula 10. Cálculo del consumo total de energía para ambos casos y comparativa.

Como se puede observar, una vez recalculado el consumo real de ambos casos se comprueba que se ha reducido el consumo energético aproximadamente un 14% con respecto al escenario de mayor rendimiento (6 núcleos a 1,2 GHz). También se debe tener en cuenta que la demostración aquí presentada se corresponde a una prueba de concepto, quedando pendiente la evaluación del algoritmo con un mayor número de secuencias de prueba y la implementación de posibles mejoras que permitan incrementar la reducción del consumo energético.

10.5. Resumen

A lo largo de este capítulo se han presentado, en primer lugar, las distintas versiones de decodificadores HEVC-RVC utilizadas durante la fase de desarrollo y verificación de la metodología propuesta en esta tesis doctoral. También se han detallado las características de las secuencias incluidas en el banco de pruebas.

Seguidamente, se ha llevado a cabo un análisis del paralelismo de las distintas versiones de las implementaciones basadas en descodificadores HEVC-RVC. Para ello, se ha analizado en detalle la carga computacional de las distintas FUs que integran los descodificadores y se han estimado los factores de ganancia que cabría obtener al utilizar varios núcleos durante la descodificación de las secuencias de vídeo.

Una vez presentadas las distintas versiones de los descodificadores HEVC-RVC, el banco de pruebas y el análisis del paralelismo de dichas versiones, se han mostrado los resultados de la implementación de dos versiones de los descodificadores sobre tres plataformas multinúcleo diferentes. Las implementaciones se han realizado utilizando el compilador de libre distribución Orcc, utilizando un nuevo *back-end* que permite automatizar la metodología de diseño propuesta en la tesis. Las pruebas realizadas incluyen la descodificación de varias secuencias *All-Intra* y *Low-Delay* sobre las tres plataformas multinúcleo y utilizando las dos versiones de los descodificadores. Adicionalmente, se ha realizado un conjunto de pruebas más limitado con una tercera versión del descodificador utilizando dos plataformas. Los resultados de las pruebas demuestran que la metodología de diseño propuesta permite llevar a cabo diseños de descodificadores RVC-CAL sobre plataformas basadas en procesadores multinúcleo tipo DSP y GPP de manera rápida y flexible, y con buenos factores de mejora en relación con las prestaciones obtenidas por las implementaciones mononúcleo.

Por último también se han evaluado, con prometedores resultados, las técnicas propuestas para la reducción del consumo energético sobre plataformas multinúcleo.

QUINTA PARTE: Resultados, aportaciones y líneas de trabajo futuro

En esta quinta y última parte de la memoria se presentan los resultados, aportaciones, publicaciones y líneas de trabajo futuro fruto del desarrollo de la presente tesis doctoral.

En primer lugar, en el capítulo 11, se recuerdan los objetivos de la tesis detallados en el capítulo 1; también se presentan los distintos resultados y aportaciones surgidos del trabajo de investigación realizado. En el capítulo 12 se enumeran las publicaciones científicas a las que ha dado lugar el trabajo realizado; en este capítulo también se mencionan otros méritos y publicaciones realizados durante el desarrollo de la tesis doctoral. Finalmente, en el capítulo 13, se enumeran las líneas de trabajo futuro que se proponen a partir de la investigación realizada.

11. Resultados y aportaciones

En este capítulo se resumen los objetivos de la tesis y se presentan los resultados obtenidos y las aportaciones que representan.

11.1. Objetivos de la tesis

En el capítulo 1 de esta memoria se presentaron los objetivos de la tesis doctoral, así como su relación en el marco de las actividades que viene desarrollando el GDEM. Así, y a modo de recordatorio, el principal objetivo de esta tesis doctoral es la formalización de una metodología de trabajo que facilite el diseño de descodificadores de vídeo sobre plataformas multinúcleo basadas en tecnología DSP. Esto se ha llevado a cabo mediante la creación de una herramienta que permite automatizar las tareas de

diseño. Dicha metodología ha sido validada mediante la implementación de distintas versiones de descodificadores de vídeo compatibles con el estándar HEVC.

Asociados al objetivo principal se han definido una serie de objetivos secundarios, que persiguen dotar a la metodología propuesta de una mayor funcionalidad e interoperabilidad entre distintas plataformas.

- Dotar al sistema desarrollado de la compatibilidad necesaria para poder trabajar sobre plataformas del tipo GPP.
- Evaluar la integración de algoritmos de mapeo dinámico de las entidades funcionales entre los núcleos disponibles en cada plataforma.
- Realizar un estudio sobre el consumo energético de la plataforma Multi-DSP durante el proceso de decodificación de vídeo.

11.2. Resultados y aportaciones de la tesis doctoral

A lo largo de este apartado se van a resumir los distintos resultados alcanzados a partir de los objetivos definidos al comienzo de la tesis doctoral, resaltando las aportaciones más relevantes.

La Figura 102 resume en un diagrama las publicaciones científicas fruto del trabajo de investigación realizado (en verde), los hitos de mayor relevancia en la concesión de los objetivos (en azul) y los distintos Proyectos Fin de Grado llevados a cabo en relación con la tesis doctoral (en rojo). Los detalles de las publicaciones y proyectos realizados se presentan en el capítulo siguiente de esta memoria (ver capítulo 12).

Figura 102. Diagrama con la evolución del trabajo realizado por el doctorando, publicaciones y trabajos relacionados con la investigación.

11.2.1. Metodología de diseño

Los Procesadores Digitales de Señal han estado ligados a la implementación de aplicaciones multimedia desde su origen, y su utilización en diversas plataformas tecnológicas está hoy en día plenamente extendida.

En el capítulo 3 se presentó un análisis sobre el estado del arte en la utilización de diversas tecnologías para la implementación de decodificadores de vídeo digital. En este sentido diversos trabajos realizados por distintos grupos de investigación publicados en la literatura científica han sido analizados; sin embargo, no se ha encontrado ninguna referencia a una metodología que englobe todo el proceso de diseño sobre plataformas Multi-DSP. Las únicas referencias encontradas sobre metodologías para el desarrollo de este tipo de aplicaciones sobre distintas plataformas son las relativas al trabajo realizado con el estándar RVC, principalmente por parte de los grupos IETR y GDEM.

La metodología de diseño presentada en este trabajo es la principal contribución de esta tesis doctoral. Dicha metodología contempla no sólo el diseño de decodificadores de vídeo sobre plataformas Multi-DSP, sino que abarca distintas arquitecturas gracias a la utilización de OpenMP. Durante el desarrollo de la metodología se han propuesto soluciones a cada uno de los aspectos relevantes del proceso de diseño y se ha estudiado cómo estas soluciones pueden generalizarse de forma que la metodología de diseño pueda aplicarse fácilmente sobre distintos tipos de plataformas multinúcleo.

Por último, cabe destacar que la metodología desarrollada queda definida dentro del conjunto de herramientas de Orcc, plenamente compatible con el estándar RVC. Este hecho aporta a su vez una estructura de trabajo dinámica, sobre la que se pueden llevar tanto actualizaciones como ampliaciones de forma sencilla, al tiempo que está disponible para el resto de la comunidad científica.

11.2.2. Resultados de las implementaciones

En el capítulo 10 se presentaron las particularidades de las distintas versiones de los decodificadores HEVC-RVC utilizados para verificar la metodología de diseño propuesta. Utilizando dicha metodología se han llevado implementaciones de tres versiones de decodificadores que han sido verificadas sobre cuatro plataformas multinúcleo, de las cuales dos integran procesadores Multi-DSP y otras dos GPPs. Los resultados muestran que la metodología de diseño propuesta permite llevar a cabo diseños de decodificadores RVC-CAL sobre plataformas basadas en procesadores multinúcleo basados en DSPs y GPPs de manera rápida y flexible, y con buenos factores de mejora en relación con las prestaciones obtenidas por las implementaciones mononúcleo.

En lo que respecta a las implementaciones con el procesador Multi-DSP TMS320C6678, principal plataforma de trabajo de esta tesis, se ha llevado a cabo distintas estimaciones del factor de ganancia en base al reparto de la carga

computacional de las FUs. La Tabla 53 muestra una comparativa de los factores de mejora conseguidos y estimados según las aproximaciones detalladas en el apartado 10.3.

Versión HEVC-RVC	Tipo de secuencias	S _{máx}			S _{máx} conseguido
		CAB	Amd.	Gus.	
REF	LD _{optimista}	3,1	2,8	5,5	2,0
	LD _{pesimista}	1,9	1,9	4,2	
	AI	2,1	3,2	5,8	3,7
YUV	LD _{optimista}	4,3	3,6	6,1	3,1
	LD _{pesimista}	2,8	2,7	5,3	
	AI	2,8	4,5	5,4	5,2

Tabla 53. Comparativa del factor de mejora conseguido para las distintas implementaciones con el procesador TMS320C6678.

Como se puede observar, los factores de ganancia obtenidos con secuencias *All-Intra* (AI) superan las estimaciones realizadas en base a la Ley de Amdahl y a la aproximación CAB en función del reparto de la carga computacional, aproximándose a los valores calculados según la Ley de Gustafson. Esto es principalmente debido a la baja dependencia de datos durante la descodificación de las imágenes, en el apartado 10.3.1 ya se hizo un análisis sobre la variación de la carga computacional de las FUs al utilizar varios núcleos. Por otro lado, los factores de mejora conseguidos con secuencias *Low-Delay* son considerablemente inferiores a los anteriores. Sin embargo, estos resultados son razonables ya que se debe tener en cuenta que para la descodificación de este tipo de secuencias es necesario descodificar, y almacenar previamente, varias imágenes para poder realizar el procesado de la predicción inter; esto implica elevadas dependencias de datos entre los actores que integran los descodificadores. Pero, aún teniendo esto último en cuenta, cabe destacar que el factor de mejora de 3,1 conseguido con la versión HEVC-RVC YUV, está muy próximo a la mejora de 3,6 veces con respecto a un núcleo, máximo estimado en base a la Ley de Amdahl.

Estos resultados ponen de manifiesto la eficiencia de la solución implementada e integrada en la metodología propuesta, principalmente en lo que se refiere a la gestión del flujo de información mediante FIFOs y el mantenimiento de la coherencia entre los distintos niveles de memoria en plataformas Multi-DSP (ver apartado 9.4).

De forma complementaria, la metodología propuesta ha sido verificada utilizando dos plataformas de desarrollo basadas en diferentes GPPs. Si bien los procesadores utilizados sólo disponen de cuatro núcleos en su arquitectura, los resultados obtenidos han servido no solo para verificar el correcto funcionamiento de la solución presentada, además, muestran cómo, en términos de factor de mejora, los resultados obtenidos con la plataforma Multi-DSP son comparables a los obtenidos con plataformas en las que la gestión del flujo de la información se lleva a cabo de forma automática (ver apartado 7.3). La Figura 103 muestra una comparativa general de los factores de ganancia obtenidos para cada plataforma (DSP, PC y Odroid), descodificador (REF y YUV) y tipo de secuencia (AI y LD). En esta figura se han

incluido, de forma orientativa, los factores de mejora calculados según las tres aproximaciones presentadas en el apartado 10.3, para un código cuya parte paralelizable representa un 75% de la carga total.

Figura 103. Comparativa general de los resultados obtenidos con tres plataformas multinúcleo.

Por último, cabe destacar que la tasa de imágenes descodificadas por parte de la plataforma DSP respecto de las plataformas GPP es considerablemente menor. Esto se debe claramente a las grandes diferencias entre las arquitecturas de cada plataforma. Pero debe tenerse en cuenta que no era objetivo de este trabajo incidir en la mejora del rendimiento de las aplicaciones. En este sentido se ha propuesto la utilización de módulo optimizados (ver apartado 9.7) que permitan mejorar el rendimiento sobre la plataforma DSP, al tiempo que se empleen los distintos recursos, propios de este tipo de arquitecturas, para incrementar la velocidad de la descodificación.

11.2.3. Utilización de distribuciones dinámicas de actores

Uno de los objetivos planteados durante el desarrollo de la tesis doctoral se basa en la adaptación de un sistema que permita la realización automática de distribuciones dinámicas de las unidades funcionales que integran las aplicaciones basadas en el

estándar RVC, obteniendo una reconfiguración dinámica en tiempo real muy útil en situaciones con un número de Fus y/o núcleos elevado.

Para ello, en el *back-end* de Orcc que automatiza la metodología de diseño se ha integrado infraestructura necesaria para la generación y aplicación de mapeos dinámicos de actores. Esto se realiza mediante la toma de distintas medidas en tiempo real, que posteriormente son utilizadas para estimar distintos repartos de las unidades funcionales, teniendo en cuenta así aspectos como la carga computacional y/o el volumen de las comunicaciones entre los actores.

Además, actualmente se están desarrollando plataformas que integran en su arquitectura varias decenas de núcleos de procesamiento; estas plataformas serían susceptibles de utilizarse en el procesamiento de algoritmos especialmente complejos. La gestión del reparto de la carga computacional en este tipo de situaciones supone un reto para el que las distribuciones dinámicas pueden ser una buena solución.

Existen diversas estrategias de mapeo dinámico, que permiten obtener distintos resultados en función de las características de la plataforma. Además, de esta manera se dota al sistema de una herramienta que le permite adaptarse, en tiempo de ejecución, a aquellos cambios que puedan darse durante el proceso de descodificación. Asimismo, y dado que el código fuente que implementa esta funcionalidad es abierto, sería posible modificar, o incluso diseñar, los algoritmos existentes para adaptarlos a las particularidades de cada plataforma; de esta manera podrían obtenerse distribuciones automáticas con un mejor aprovechamiento de los distintos núcleos disponibles.

11.2.4. Reducción del consumo energético

La gestión del consumo energético en terminales móviles, cuya fuente de alimentación es una batería, es un factor clave en el desarrollo de aplicaciones multimedia. Por ello, dentro de la metodología de diseño, se ha previsto la implementación de técnicas para la reducción del consumo energético en plataformas multinúcleo. En este caso se ha desarrollado un algoritmo, integrado en el código del descodificador, que permite hacer un uso eficiente de los recursos *hardware* (número de núcleos utilizados y frecuencia de funcionamiento).

El algoritmo desarrollado permite variar tanto la frecuencia de funcionamiento de los procesadores, como el número de núcleos utilizados en la descodificación, con el fin de hacer funcionar al descodificador a una tasa de FPS fija y establecida por el usuario. Las pruebas preliminares de las técnicas desarrolladas muestran ahorros significativos (superiores al 13%) en la energía consumida durante la descodificación de varias secuencias concatenadas con diferentes características.

12. Publicaciones y méritos

En este capítulo se presentan las publicaciones y otros resultados relacionados con el trabajo de investigación desarrollado durante la tesis doctoral. En el apartado 12.1 se presentan los trabajos que han sido publicados tanto en revistas como en comunicaciones a congresos. En el apartado 12.2 se resumen otros resultados relacionados con el trabajo de investigación. Finalmente, en el apartado 12.3, se identifican algunos resultados susceptibles de publicación en el futuro.

12.1. Trabajos publicados en relación con la tesis

Los resultados obtenidos en esta tesis doctoral han dado lugar a cuatro publicaciones en revistas con índice de impacto y cinco comunicaciones en congresos internacionales con revisión por pares. Las publicaciones en revistas se resumen a continuación:

- Fernando Pescador, Miguel Chavarrias, Matías J. Garrido, Eduardo Juárez, César Sanz. “Complexity analysis of an HEVC decoder based on a digital signal processor”. *IEEE Transactions on Consumer Electronics*, vol. 59, no. 2, pp. 391-399. Mayo 2013. [PCG⁺13].

En este trabajo se presenta un análisis detallado de la implementación de un descodificador HEVC basado en el código de referencia HM v9.0 sobre una plataforma DSP. Mediante un completo banco de pruebas se presentan los resultados de carga computacional de cada uno de los bloques funcionales que integran el descodificador, así como el rendimiento global del descodificador en FPS para distintas secuencias y configuraciones. Esto sirvió como punto de partida para caracterizar un descodificador HEVC con la única implementación disponible en aquel momento.

- Miguel Chavarrias, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, Mickaël Raulet. “A DSP-Based HEVC decoder implementation using an actor language dataflow model”. *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 839-847. Noviembre 2013. [CPG⁺13].

En este segundo artículo se presenta la primera implementación de un descodificador HEVC-RVC sobre una plataforma DSP. Como resultado de esta implementación se ofrecen los datos de carga computacional de cada una de las Fus que

integran el descodificador y los datos de rendimiento en FPS con un núcleo. *Este artículo recibió el premio Chester Shall* al tercer mejor trabajo (ver apartado 12.2.4).

- Miguel Chavarriás, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, César Sanz. “A multicore DSP HEVC decoder using an actor based dataflow model and OpenMP”. *IEEE Transactions on Consumer Electronics*, vol. 61, no. 2, pp. 236-244. Mayo 2015. [CPG⁺15b].

En este tercer trabajo se presenta la primera implementación de un descodificador HEVC-RVC sobre una plataforma multi-DSP haciendo uso simultáneo de varios núcleos para la descodificación de vídeo. En él se presentan datos sobre la mejora del rendimiento, que ronda el 70% con dos núcleos, y sobre la tasa de descodificación en FPS. Así mismo se realiza un estudio sobre la idoneidad de las Fus a mapear en cada uno de los núcleos de la plataforma.

- Miguel Chavarriás, Fernando Pescador, Matías J. Garrido, Alexandre Sánchez, César Sanz. “Design of Multicore HEVC Decoders Using Actor-based Dataflow Models and OpenMP”. *IEEE Transactions on Consumer Electronics*, vol. 62, no. 3, pp. 325-333. Agosto 2016. [CPG⁺16b].

En este artículo se presenta el diseño de un nuevo *back-end C*, mediante el cual se permite la generación de código fuente C a partir de especificaciones RVC-CAL, el cual sustituye el uso de *threads* por OpenMP. Además se integran las modificaciones necesarias para automatizar la migración del código fuente a una plataforma Multi-DSP y gestionar los distintos niveles de memoria de una forma eficiente. Este nuevo diseño se ha utilizado para realizar una implementación del descodificador HEVC-RVC en su versión YUV sobre una plataforma Multi-DSP. Los resultados de esta implementación se han comparado con los obtenidos sobre una plataforma ARM y un PC.

A continuación se listan las comunicaciones a congresos internacionales:

- Miguel Chavarriás, Fernando Pescador, Eduardo Juárez, Matías J. Garrido. “An automatic tool for the static distribution of actors in RVC-CAL based multicore designs”. *Conference on Design of Circuits and Integrated Circuits DCIS 2014*, pp. 1-6. 26-28 Noviembre 2014. [CPJ⁺14].

En esta primera comunicación a un congreso internacional se publicó una herramienta diseñada para calcular distribuciones de actores en entornos multinúcleo a partir de distintas medidas tomadas en tiempo real. Las pruebas de dicha herramienta se realizaron utilizando el código fuente de HEVC-RVC.

- Pablo Rodríguez, Fernando Balseiro, Miguel Chavarriás, Fernando Pescador, Matías J. Garrido. “A DSP-based HEVC decoder implementation using RVC-CAL and native OpenHEVC code”. *International Symposium on Consumer Electronics ISCE 2015*, pp. 1-2. 24-26 Junio 2015. [RBC⁺15].

En esta segunda comunicación a un congreso internacional se presentó la optimización, sobre una plataforma DSP, de una implementación HEVC-RVC con módulos escritos en C del *software* de OpenHEVC. Los resultados de este diseño muestran una mejora global del rendimiento superior al 10%.

- Miguel Chavarrías, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, César Sanz. “A multicore DSP HEVC decoder using an actor-based dataflow model”. *IEEE International Conference on Consumer Electronics ICCE 2015*, pp. 370-371. 9-12 Enero 2015. [CPG⁺15].

En esta comunicación a un congreso internacional se presenta la primera implementación de un decodificador HEVC-RVC sobre una plataforma Multi-DSP. Para ello se hace uso de la API de OpenMP. En esta primera implementación se realizan pruebas sobre uno y dos núcleos, este trabajo se amplía en [CPG⁺15b].

- Miguel Chavarrías, Fernando Pescador, Matías J. Garrido, Maxime Pelcat, Eduardo Juárez. “Design of multicore HEVC decoders using actor-based dataflow models and OpenMP”. *IEEE International Conference on Consumer Electronics ICCE 2016*, pp. 287-288. 7-11 Enero 2016. [CPG⁺16].

En este trabajo se presentan los primeros resultados del diseño del nuevo *back-end* DSP/OpenMP. El mismo se utiliza para realizar una implementación del decodificador HEVC-RVC sobre distintas plataformas (PC, ARM y Multi-DSP). En [CPG⁺16b] este trabajo se amplía optimizando la gestión de las transferencias de datos mediante las FIFOs.

- Fernando Pescador, Miguel Chavarrías, Matías J. Garrido, Javier Malagón, César Sanz. “Real-time HEVC decoding with OpenHEVC and OpenMP”. *IEEE International Conference on Consumer Electronics ICCE 2017*, pp. 370-371. Enero 2017. [PCG⁺17].

En esta comunicación a congreso internacional se presenta la implementación del decodificador OpenHEVC sobre una plataforma Multi-DSP. En este caso se ha utilizado OpenMP para gestionar la paralelización del proceso de decodificación, que en el caso de OpenHEVC permite simultanear el procesamiento a nivel de *slice*.

12.2. Otros resultados relacionados con la tesis

En este apartado se presentan otros resultados y méritos relacionados con el trabajo de investigación desarrollado: el desarrollo de la colaboración internacional existente entre el grupo GDEM y el IETR, la dirección de distintos proyectos de fin de grado por parte del doctorando y las becas y premios recibidos durante el desarrollo de la tesis.

12.2.1. Proyectos de investigación

El trabajo desarrollado durante esta tesis doctoral ha estado estrechamente relacionado con tres proyectos de investigación de los que el grupo GDEM ha formado parte.

- H2B2VS – *HEVC Hybrid Broadcast Video Services* [H2B]. El principal objetivo de este proyecto es investigar en la distribución híbrida de programas y servicios de televisión sobre redes heterogéneas, *Broadband* y *Broadcast*, utilizando el estándar HEVC.
- MR-UHDTV – *Mixed Reality over Ultra High Definition Television* [MRU]. El objetivo de este proyecto es contribuir en el desarrollo de técnicas de procesamiento combinado de vídeo UHD con realidad mixta⁷⁴.
- IVME – *Immersive Visual Media Environments* [IVME]. El objetivo de este proyecto es contribuir en el desarrollo de técnicas de procesamiento de vídeo digital omnidireccional, que permitan ofrecer nuevas percepciones inmersivas de visualización e interacción por parte de los usuarios.

12.2.2. Colaboración y estancia en el INSA de Rennes

Como se explicaba al comienzo de esta memoria (ver capítulo 1), la relación de colaboración entre el laboratorio IETR del INSA de Rennes (Francia) y el GDEM de la UPM ha permitido llevar a cabo distintos proyectos y estancias por parte de numerosos investigadores de ambos centros. En este sentido, a lo largo de esta tesis doctoral se han estrechado lazos y se han realizado publicaciones conjuntas en revistas y congresos internacionales [CPG⁺13], [CPG⁺16] y [CPG⁺16b].

Cabe destacar que el doctorando realizó una estancia en el laboratorio del IETR del INSA de Rennes (Francia) entre el 12 de febrero de 2015 y el 22 de mayo de 2015, el profesor responsable de la estancia fue el Dr. Maxime Pelcat.

Los objetivos marcados para dicha estancia eran el desarrollo, o la adquisición de los conocimientos necesarios que permitieran diseñar un nuevo *back-end* de Orcc que permitiera la generación de código fuente para plataformas Multi-DSP con la API de OpenMP a partir de especificaciones RVC-CAL. En segundo lugar también se propuso como objetivo estudiar la optimización del rendimiento de los descodificadores de vídeo basados en RVC-CAL mediante la utilización de módulos OpenHEVC (ver capítulo 5).

Además de los objetivos marcados para esta estancia de investigación también se estudiaron los métodos de mapeos dinámicos (ver apartado 4.5.4). Las secciones de código encargadas de la implementación de dichos mapeos fueron integradas con éxito en la infraestructura de los nuevos *back-ends* diseñados.

⁷⁴ La realidad mixta es la combinación de las técnicas de realidad virtual y realidad aumentada.

12.2.3. Dirección de Proyectos Fin de Grado

En relación con algunos de los hitos que se han desarrollado durante el trabajo de investigación asociado a la tesis doctoral se han llevado a cabo, por parte de estudiantes en Ingeniería de Telecomunicaciones de la ETSIST, diferentes trabajos que han dado lugar a los respectivos Proyectos Fin de Carrera o Proyectos Fin de Grado. El doctorando ha tutorizado el siguiente Proyecto Fin de Grado:

- Javier Adrián Barriobero. “Control de consumo de un decodificador RVC_HEVC sobre plataforma multi DSP”. Fecha de lectura: 17-2-17 en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Calificación: 10 (SB).

A continuación se citan aquellos Proyectos Fin de Grado que han sido dirigidos por el doctorando:

- Rafael Pons Castellanos. “Aplicación automatizar la asignación de procesos RVC a los núcleos de un sistema multiprocesador”. Fecha de lectura: 15-7-14 en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Tutor: Fernando Pescador del Oso, calificación: 9 (SB).
- Pablo Rodríguez Nava. “Métodos de programación para la plataforma EVMK2H”. Fecha de lectura: 17-7-15, en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Tutor: Fernando Pescador del Oso, calificación: 10 (MH).
- Luis Martín Sanz. “Reducción del consumo del dispositivo TMS320C6678”. Fecha de lectura 19-5-2016, en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Tutor: Fernando Pescador del Oso, calificación: 7,5 (N).
- Álvaro Bueno Sáez. “Evaluación del rendimiento de decodificadores de vídeo con OpenMP sobre diferentes plataformas”. Fecha de lectura: 11-4-16, en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Tutor: Fernando Pescador del Oso, calificación: 9 (SB).

Dentro del mismo marco de investigación en el que se encuadra la presente tesis doctoral se han desarrollado otros trabajos asociados a la finalización de distintos grados de estudios universitarios. Estos trabajos guardan, al igual que los citados anteriormente, una estrecha relación con el trabajo de investigación desarrollado en esta tesis doctoral y han sido tutelados por uno de los directores de tesis:

- Daniel Gómez-Lobo García. “Methodologies for implement video decoders over multiprocessor platforms”. Trabajo Fin de Máster (MISSSI). Fecha de lectura: 18-7-2013, en el Departamento de Sistemas Electrónicos de Control de la EUITT-UPM. Tutor: Fernando Pescador del Oso, calificación 10 (SB).
- Fernando Javier Balseiro Lago. “Integración de HEVC-RVC y OpenHEVC sobre la plataforma MultiDSP C6678”. Trabajo Fin de Grado. Fecha de lectura: 10-7-2015, en el Departamento de Ingeniería Telemática y Electrónica de la ETSIST-UPM. Tutor: Fernando Pescador del Oso, calificación: 10 (SB).

Además, actualmente el doctorando está tutorizando los siguientes Proyectos Fin de Grado, cuyos anteproyectos han sido aprobados por la Comisión Académica de Grado, estando todos ellos en un estado de desarrollo avanzado:

- Julio Antonio Treviño Páez. “Diseño de una metodología para la comunicación entre los núcleos de la plataforma heterogénea 66AK2H”.
- Pablo Cano Megías. “Integración de FVC y OpenHEVC sobre la plataforma Multi-DSP TMS320C6678”.
- Estefanía Camuñas Galán. “Diseño de un banco de pruebas para evaluar el rendimiento en descodificadores HEVC sobre la plataforma Multi-DSP TMS320C6678”.

12.2.4. Becas y premios obtenidos

Durante el desarrollo de la tesis también se obtuvieron los siguientes premios y becas:

- IEEE Consumer Electronics Society 2014, premio Chester Sall al tercer mejor artículo. Por la publicación titulada “*A DSP-Based HEVC Decoder Implementation Using an Actor Language Dataflow Model*” publicado en el *IEEE Transactions on Consumer Electronics*, volumen 59, No. 4, noviembre 2013, pp. 839-847.
- Beca “*soutien à la mobilité des doctorants – 1^{er} semestre 2015*” concedida por Rennes Métropole (Francia), para la realización de estancias por parte de doctorandos en las universidades de Rennes. Duración de la ayuda: 3 meses.

12.3. Resultados susceptibles de publicación en el futuro

Algunos de los trabajos de investigación relacionados con la metodología de diseño propuesta, tales como la utilización de mapeos dinámicos de actores sobre plataformas Multi-DSP (apartado 9.5) o la utilización de técnicas para la reducción de consumo energético (apartado 9.6), son susceptibles de publicación. Dado que estos trabajos han sido los últimos en ser abordados y su verificación se ha llevado a cabo de forma paralela a la redacción de este documento, aún queda pendiente un proceso de análisis, selección y toma de decisiones acerca de cuáles son los contenidos publicables y el tipo de publicación al que pueden enviarse.

13. Líneas de trabajo futuro

Para finalizar la presente memoria de tesis doctoral, se presentan las distintas líneas de trabajos futuros que se proponen a partir del trabajo ya realizado. Algunas de estas tareas ya han sido puestas en marcha por parte del grupo GDEM a partir de los hitos alcanzados en esta tesis doctoral.

Se definen tres líneas de investigación o trabajos futuros relacionados con la temática de la tesis:

1) Ampliación de la metodología diseñada – plataformas heterogéneas

En el capítulo 7 se hizo una breve introducción a las nuevas plataformas heterogéneas, es decir, aquellas cuya arquitectura se compone de distintos tipos de procesadores. Este tipo de plataformas ofrecen una serie de ventajas y posibilidades mayores que las ofrecidas por las plataformas homogéneas. Sin embargo, su programación supone un reto proporcional a las posibilidades ofrecidas.

En este sentido se abrió una línea de investigación en el GDEM para estudiar las posibilidades de este tipo de plataformas. El trabajo inicial fue desarrollado como parte del Proyecto Fin de Grado de Pablo Rodríguez durante el año 2015 [Rod15]. Hoy en día se encuentran en desarrollo, dentro del propio GDEM, una tesis doctoral y un Proyecto Fin de Grado en esta temática. Los principales objetivos de ambos trabajos son la utilización simultánea de los distintos procesadores que integran una plataforma heterogénea durante la descodificación de vídeo digital.

Con el ánimo de ofrecer, a medio plazo, una solución de diseño basado en una metodología similar a la aquí defendida, se ha comenzado con el proceso de migración del código de descodificadores HEVC-RVC.

Además, gran parte del trabajo desarrollado tiene como soporte tecnológico la API de OpenMP, que si bien ha demostrado ser una solución factible para la programación en entornos multinúcleo, presenta diversas limitaciones en entornos basados en plataformas heterogéneas⁷⁵. En este sentido, OpenCL es, a día de hoy, una herramienta clave en la programación de plataformas heterogéneas. El trabajo

⁷⁵ A partir de la versión v4.0 OpenMP incluye algunas funcionalidades para la paralelización de código sobre plataformas heterogéneas.

desarrollado en esta tesis doctoral, y principalmente la automatización del proceso de diseño (ver apartado 9.9), es susceptible de ampliarse para ofrecer una solución rápida y eficaz que permita superar los retos que la programación de plataformas heterogéneas supone.

2) Exploración e integración de futuros estándares de codificación de vídeo digital – *Future Video Coding*

Como ya se introdujo en el capítulo 2, las herramientas y estándares de codificación de vídeo digital están en continuo desarrollo. Actualmente se ha comenzado a trabajar en las bases del que podría ser el estándar sucesor de HEVC, y que a día de hoy se conoce como *Future Video Coding* (FVC). Su desarrollo está siendo liderado por MPEG y si bien se encuentra en una fase muy inicial, no deben perderse de vista las posibles mejoras que surjan de su puesta en marcha.

Además, parte del desarrollo de nuevas propuestas para el diseño del nuevo estándar está siendo liderado por miembros del IETR del INSA de Rennes. Fruto de la ya existente colaboración entre el IETR y el GDEM se está colaborando activamente en la implementación y prueba de nuevas técnicas de codificación. Actualmente, se está desarrollando un Proyecto Fin de Grado en el GDEM cuyo objetivo principal es evaluar el rendimiento de algunas de las nuevas propuestas sobre una plataforma Multi-DSP.

3) Extensión y verificación de la metodología propuesta a distintas aplicaciones basadas en flujos de datos.

Ya en el apartado 2.3, y también a lo largo del capítulo 9, se ha mencionado que si bien el objetivo principal de RVC y Orcc es, respectivamente, la especificación y generación de código fuente para la implementación de descodificadores de vídeo, también pueden utilizarse en el desarrollo de otros tipos de aplicaciones. Así, la metodología de diseño propuesta también es susceptible de ser utilizada con otros objetivos.

En este sentido, se propone la evaluación de la metodología propuesta para el desarrollo de otro tipo de aplicaciones. Se propone como punto de partida la implementación de algunas de las aplicaciones, distintas de los descodificadores de vídeo, disponibles en el propio repositorio de Orcc [ORC] para valorar las posibilidades de esta propuesta. En el apartado 2.3.5 se listaron algunas de estas aplicaciones.

BIBLIOGRAFÍA

- [3DN] Manual de referencia de instrucciones 3Dnow, disponible en la URL: <http://support.amd.com/TechDocs/21928.pdf>
- [4EV] Sitio web de 4EVER, URL: <http://www.4ever-project.com/accueil/>
- [ADV] Sitio web de Advantech. URL: <http://www.advantech.com/>
- [AGI] Guía de usuario de la fuente de alimentación Agilent 66321D. URL: <http://cp.literature.agilent.com/litweb/pdf/5964-8184.pdf>
- [AK2H] Descripción y documentación técnica del procesador 66AK2H, URL: <http://www.ti.com/lit/ds/symlink/66ak2h14.pdf>
- [ALM] Proyecto ALMA. Disponible en http://cordis.europa.eu/project/rcn/99828_es.html
- [AMD] Sitio web de AMD. URL : <http://www.amd.com/es-xl>
- [Amd67] G.M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. Conferencia del American Federation of Information Processing Societies (AFIPS). Abr. 1967, pp. 483-485.
- [ANA] Listado de plataformas DSP de Analog Devices, URL : <http://www.analog.com/en/products/processors-dsp/all-dsp-processors.html>
- [ARM] Sitio web de ARM, URL : <http://ir.arm.com/phoenix.zhtml?c=197211&p=irol-irhome>
- [ASIG12] M. Ali, E. Stotzer, F. D. Igual, R. A. van de Geijn, “Level-3 BLAS on the TI C6678 Multi-core DSP”. IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Oct. 2012, pp. 179-156.
- [Bal15] Fernando Javier Balseiro. “Integración de HEVC-RVC y OpenHEVC sobre la plataforma multiDSP C6678”. Trabajo Fin de Grado leído en el Departamento de ingeniería Telemática y Electrónica de la E.T.S.I.S. de Telecomunicación de la U.P.M. Julio 2015.
- [BBA⁺15] N. Bahri, N. Belhadj, M. Ali Ben Ayed, N. Masmoundi, T. Grandpierre, M. Akil. “Real-time H264/AVC high definition video encoder on a multicore DSP TMS320C6678”. International Conference on Computer Vision and Image Analysis Applications (ICCVIA). Ene. 2015, pp. 1-6.
- [BBEG14] M. Bahtat, S. Belkouch, P. Elleaume, P. le Gall. “Efficient

- implementation of a complete multi-beam radar coherent-processing on a telecom SoC". International Radar Conference. Oct. 2014, pp. 1-6.
- [BBSF12] F. Bossen, B. Bross, K. Sühring, D. Flynn "HEVC Complexity and Implementation Analysis", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1685-1696, Dic. 2012
- [BEJ⁺09] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, "Overview of the MPEG reconfigurable video coding framework," Journal of Signal Processing Systems, 2009.
- [BEP⁺08] Shuvra S. Bhattacharyya, Johan Eker, Carl Von Platen, Marco Mattavelli Gordon Brebner, örn W. Janneck, Mickaël Raulet "OpenDF – A Dataflow Toolset for Reconfigurable *Hardware* and Multicore Systems" ACM SIGARCH Computer Architecture News, Volume 36 Issue 5, December 2008.
- [BLRP14]. M. Bariani, P. Lambruschini, M. Raggio, L. Pezzoni. "An Optimized *Software* Implementation of the HEVC/H.265 Video Decoder". IEEE 11th Consumer Communications and Networking Conference (CCNC), ene. 2014, pp. 77-82.
- [BOR⁺14] T. Bruckschloegl, O. Oey, M. Rueckauer, T. Stripf, J. Becker. "A Hierarchical Architecture Description for Flexible Multicore System Simulation". Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2014), pp. 40-52.
- [C15] ARM Cortex-A15, URL:
<http://www.arm.com/products/processors/cortex-a/cortex-a15.php>
- [C6472] Descripción y documentación técnica del procesador TMS320C6472 facilitada por el fabricante, disponible en la URL :
<http://www.ti.com/product/TMS320C6472>
- [C66] Relación de procesadores de la familia C66 de Texas Instruments, URL :
http://www.ti.com/lstds/ti/processors/dsp/c6000_dsp/c66x/overview.page
- [C6678] Descripción y documentación técnica del procesador TMS320C6678 facilitada por el fabricante, disponible en la URL :
<http://www.ti.com/product/TMS320C6678>
- [CAL] CAL Language Report – Specification of the CAL actor language. Johan Eker y Jörn W. Janneck. Language version 1.0. UCB/ERL M03/48, Universidad de California en Berkeley. Diciembre de 2003.
- [CCS] Sitio web del IDE Code Composer Sturudio de Texas Instruments. URL :
<http://www.ti.com/tool/ccstudio>

- [Cha12] Miguel Chavarrias. “Exploración de aplicaciones RVC empleando un procesador ARM multinúcleo”. Trabajo Fin de Máster leído en el Departamento de Sistemas Electrónicos y de Control de la E.U.I.T. de Telecomunicación de la U.P.M. Julio 2012.
- [CHP11] G. Clare, F. Henry, and S. Pateux, “Wavefront parallel processing for HEVC Encoding and Decoding,” in document JCTVC-F274. Torino, Italy, July 2011.
- [CMA] Sitio web de Cmake, documentación y descargas. URL: <https://cmake.org/>
- [COM] Sitio web de Compaan, URL: <http://compaandesign.com/>
- [CUDA] Sitio web de CUDA, URL : http://www.nvidia.com/object/cuda_home_new.html
- [DJL⁺06] D. Deli, A. Jebson, Y. Liao, N. C. Paver, S. J. Strazdus. Multiply-accumulate (MAC) Unit for Single-instruction/multiple-data (SIMD) Instructions. Intel Corporation, assignee. Patente EE.UU. 7107305 B2. 12 de septiembre de 2006.
- [DM6437] Descripción y documentación técnica del procesador TMS320DM6437 facilitada por el fabricante, disponible en la URL : <http://www.ti.com/product/TMS320DM6437>
- [DPN⁺13] K. Desnos, M. Pelcat, J.F. Nezan, S.S. Bhattacharyya, S. Aridhi. “Pimm: Parameterized and interfaced data flow meta-model for mpsoes runtime reconfiguration”. SAMOS XIII, 2013, pp. 41-48.
- [DSY⁺14] Y. Duan, J. Sun, L. Yan, K. Chen, Z. Guo. “Novel Efficient HEVC Decoding Solution on General-Purpose Processors”. IEEE Transactions on Multimedia. Vol. 16, No. 7, Nov. 2014.
- [ECL] Sitio web de Eclipse, información y descargas. URL: <https://eclipse.org/>
- [EPFL] Sitio web de la EPFL, URL: <https://www.epfl.ch/index.en.html>
- [EUR] Proyecto EUROTILE, URL: http://euretile.roma1.infn.it/mediawiki/index.php/Main_Page
- [EXY] Manual de usuario del procesador Exynos 4412 de Samsung, URL: http://www.samsung.com/global/business/semiconductor/file/product/Exynos_4_Quad_User_Manual_Public_REV1.00-0.pdf
- [FFM] Sitio web de FFMPEG – descargas, URL: <https://ffmpeg.org/download.html>
- [FRA] Sitio web para la descarga del código de referencia del estándar HEVC

a través de la página del Fraunhofer, URL :

<https://hevc.hhi.fraunhofer.de/>

- [Gar04] Matías J. Garrido. “Arquitectura versátil para la codificación de vídeo multi-estándar : aportaciones metodológicas para el diseño de sistemas reutilizables y sistemas en un chip”. Tesis doctoral leída en el Departamento de Ingeniería Electrónica de la E.T.S.I. de Telecomunicación de la U.P.M. Septiembre 2004.
- [GCV⁺13] G. Goulas, C. Gogos, C. Valouxis, P. Alefragis, N. Voros. “Coarse Grained Parallelism Optimization for Multicore Architectures : The ALMA Project Approach”. Reconfigurable Computing : Architectures, Tools and Applications. Volume 7806 of the series Lecture Notes in Computer Science pp 235-236.
- [GRA] Sitio web de Graphiti. Información y descargas. URL: <https://eclipse.org/graphiti/>
- [Gus88] J. L. Gustafson. “Reevaluating Amdahl’s Law”. Communications of the ACM, may. 1988, vol. 31. No. 5, pp. 532-533.
- [H2B] European Celtic-Plus project H2B2VS, URL: <https://www.celticplus.eu/project-h2b2vs/>
- [HDN⁺12] J. Heulot, K. Desnos, J.-F. Nezan, M. Pelcat, M. Raulet, H. Yviquel, P.-L. Lagalaye, J.-C. Le Lann. “An experimental toolchain based on high-level data flow models of computation for heterogeneous MPSoC”. Design and Architectures for Signal and Image Processing (DASIP), 2012.
- [HEAP] Sitio web del proyecto HEAP de la UE: http://cordis.europa.eu/project/rcn/93254_en.html
- [HM] Repositorio descargas del *software* de referencia de HEVC HM. URL: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/
- [HM9] *Software* de referencia de HEVC HM90: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/branches/HM-9.0-dev/
- [HP] Sitio web de Hewlett-Packard, URL: <http://www8.hp.com/es/es/home.html>
- [HRD14] W. Hamidouche, M. Raulet, O. Déforges. “Multi-Core *Software* Architecture for the Scalable HEVC Decoder”. IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP) 2014, pp. 7545-7549.
- [HRD14b] W. Hamidouche, M. Raulet, O. Déforges. “Parallel SHVC decoder: implementation and analysis”. IEEE International Conference on

- Multimedia and Expo (ICME), 2014.
- [HRD16] W. Hamidouche, M. Raullet, O. Déforges. “4K Real-Time and Parallel *Software* Video Decoder for Multilayer HEVC Extensions”. IEEE Transactions on Circuits and Systems for Video Technology. Vol. 26, No. 1, Ene. 2016, pp. 169-180.
- [I7] Intel i7-950, URL: http://ark.intel.com/es-es/products/37150/Intel-Core-i7-950-Processor-8M-Cache-3_06-GHz-4_80-GTs-Intel-QPI
- [IET14] IETR. Sitio web del proyecto Preesm en GitHub, Junio de 2014. URL: <https://github.com/preesm/preesm>
- [IETR] Sitio web del IETR: <https://www.ietr.fr/?lang=fr>
- [INSA] Sitio web del INSA de Rennes: <https://www.insa-rennes.fr/>
- [INT] Sitio web de Intel:
<http://www.intel.es/content/www/es/es/homepage.html>
- [IRISA] Sitio web del IRISA: <https://www.irisa.fr/>
- [ISO] Sitio web de la ISO, URL: <http://www.iso.org/iso/home.html>
- [ISO03] Advanced Video Coding for Generic Audio-Visual Services, ITU-T Rec. H.264 e ISO/IEC 14496-10 (AVC), ITU-T e ISO/IEC JTC 1, May. 2003.
- [ISO08] ISO/IEC 23001-5. Information technology – MPEG systems technologies – Part 5: *Bitstream* Syntax Description Language, 2008.
- [ISO09] ISO/IEC 23001-4. MPEG systems technologies Part 4: *Códec* Configuration Representation, 2009.
- [ISO10] ISO/IEC 23002-4. Information technology – MPEG video technologies – Part 4: Video tool library, 2010.
- [ISO11] ISO/IEC 23001-4, MPEG systems technologies – Part 4: *Códec* Configuration Representation (2011).
- [ISO12] ISO/IEC-JTC1/SC29/WG11 and ITU-T-SG16, “Joint Call for Proposals on Scalable Video Coding Extensions of High Efficiency Video Coding (HEVC),” in ISO/IEC JTC 1/SC 29/WG11 (MPEG) Doc. N12957 or ITU-T SG 16 Doc. VCEG-AS90. Stockholm, Sweden, July 2012.
- [ISO15] ISO/IEC 23008-2. Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High efficiency video coding, 2015.
- [ISO94] Generic Coding of Moving Pictures and Associated Audio Information-

Part 2: video, ITU-T Rec. H.262 e ISO/IEC 13818-2 (MPEG 2 Video), ITU-T e ISO/IEC JTC 1, Nov. 1994.

- [ITU] Buscador de normas y recomendaciones de la ITU. Sitio web: <http://www.itu.int/en/ITU-T/Pages/default.aspx>
- [ITU15] Series H: Audiovisual and multimedia systems. Infrastructure of audiovisual services – Coding of moving video. H.265 (04/2015). Recommendation ITU-T
- [IVME]
- [JCT] Sitio web del consorcio JCT-VC : <http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jctvc.aspx>
- [JRE] Sitio web de Oracle, documentación y descargas de Java. URL: <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
- [JRR⁺13] E. Juárez, R. Ren, M. Raulet, J. Wei, M.J. Garrido, C. Sanz y F. Pescador. “Performance Monitoring for Energy Estimation in RVC-CAL Descriptions”. ISO/IEC JTC1/SC29/WG11 MPEG2013/M31243. Ginebra, Suiza, Octubre 2013.
- [JRS⁺14] K. Jerbi, D. Renzi, D. De Saint Jorre, H. Yviquel, M. Raulet, C. Alberti, M. Mattavelli. “Development and optimization of high level dataflow programs: The HEVC decoder design case”. 48th Asiomar Conference on Signals, Systems and Computers, Nov. 2014, pp. 2155-2159.
- [JRW⁺13] E. Juárez, R. Ren, J. Wei, M. Raulet, M. J. Garrido, C. Sanz y F. Pescador. “Proposal of a Decoder Energy Management Scheme with RVC”. ISO/IEC JTC1/SC29/WG11 MPEG2013/M28171. Ginebra, Suiza, Enero 2013.
- [JYS⁺16] K. Jerbi, H. Yviquel, A. Sanchez, D. Renzi, D. De Saint Jorre, C. Alberti, M. Mattavelli, M. Raulet. “On the Development and Optimization of HEVC Video Decoders Using High-Level Dataflow Modeling”. Journal of Signal Processing Systems, Springer, 2016, <10.1007/s11265-016-1113-x>. <hal-01298595>
- [K2H] Advantech Co., Ltd., “EVMK2HX Evaluation Modules”. URL: <http://www2.advantech.com/Support/TI-EVM/EVMK2HX.aspx>
- [Kah74] G. Kahn. “The semantics of a simple language for parallel programming”. Information processing 74, 1974.
- [KBS⁺10] R. Koenig, L. Bauer, T. Stripf, M. Shafique, W. Ahmed, J. Becker, J. Henkel. “KAHRISMA: a novel hypermorphic reconfigurable-instruction-set multi-grained-array architecture”. Design, Automation Test in Europe Conference Exhibition (DATE), 2010, pp. 819–824.

- [KHR] Sitio web de Khronos, URL: <https://www.khronos.org/>
- [KRD00] B. Kienhuis, E. Rijkema, E. F. Deprettere. “Compaan: deriving process networks from matlab for embedded signal processing architectures”. Proceedings of the Eighth International Workshop on *Hardware/Software Codesing*, 2000, pp. 13-17.
- [LAV] Libav – Open source audio and video processing tools. URL: <https://libav.org/>
- [LKR10] K. Lakshmanan, S. Kato, R. Rajkumar. “Scheduling Parallel Real-Time Tasks on Multicore Processors”. 31st IEEE Real-time Systems Symposium (RTSS). Nov.-dic. 2010, pp. 259-268.
- [LLP+13] L. Lavagno, M. Lazarescu, I. Papaefstathiou, A. Brokalakis, J. Walters, B. Kienhuis, F. Schäfer. “HEAP: A Highly Efficient Adaptive multi-Processor framework”. *Microprocessors and Microsystems*, vol. 37 (8), nov. 2013, pp. 1050-1062.
- [LLP⁺13] L. Lavagno, M. T. Lazarescu, I. Papaefstathiou, A. Brokalakis, J. Walters, B. Kienhuis, F. Schäfer. “HEAP: A Highly Efficient Adaptive multi-Processor Framework”. Elsevier – *Microprocessors and Microsystems*, 37(8)-C, pp.1050-1062. Nov. 2013.
- [LM87] E.A. Lee, D.G. Messerschmitt. “Static scheduling of synchronous data flow programs for digital signal processing”. *IEEE Transactions on Computers*, 36(1):24-35. Ene. 1987.
- [LP95] E. A. Lee and T. M. Parks. “Data flow process networks”. *Proceedings of the IEEE* 83 (5), 1995.
- [LSL⁺15] S. Lee, J. Song, W. Lee, D. Kim, J. Kim, S. Lee. “DSP Based Programmable FHD HEVC Decoder”. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, mar. 2015, pp. 972-973.
- [Mar16] Luis Martín Sanz. “Reducción del consumo del dispositivo TMS320C6678”. Trabajo Fin de Grado leído en el Departamento de ingeniería Telemática y Electrónica de la E.T.S.I.S. de Telecomunicación de la U.P.M. Mayo 2016.
- [MCS] Kit MCSDK de Texas Instruments, documentación y descargas, URL: <http://www.ti.com/tool/bioslinuxmcsdk>
- [MD5] T. A. Berson “Differential Cryptanalysis Mod 2^{32} with Applications to MD5”. *Advances in Cryptology EUROCRYPT. Lectures Notes in Computer Sciences*. Vol. 658. Pp. 71–80. 1993.
- [MET] METIS – *Serial Graph Partitioning and Fill-reducing Matrix Ordering*. URL: <http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>

- [MHPC] MCSDK-HPC disponible en la URL: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk_hpc/latest/index_FDS.html
- [MMS94] J. Meseguer, U. Montanari, V. Sassone. “On the Model of Computation of Place/Transition Petri Nets”. 15th International Conference on Application and Theory of Petri Nets 1994, LNCS N. 815.
- [MOT] Sitio web de Motorola, URL: <https://www.motorola.es/home>
- [MPEG] The Moving Picture Experts Group. Sitio web: <http://mpeg.chiariglione.org/>
- [MPEGS] The Moving Picture Experts Group. Buscador de estándares, sitio web: <http://mpeg.chiariglione.org/standards>
- [MRJ13] M. Mattavelli, M. Raulet, J. W. Janneck. MPEG reconfigurable video coding. Editores S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, J. Takala, Handbook of Signal Processing Systems, pp. 281-314. Springer, Nueva York, EE.UU, 2013.
- [MRU] Proyecto MR-UHDTV, financiado por el MINECO. Duración 2014-2016. URL del sitio web del proyecto en el CITSEM: <https://www.citsem.upm.es/index.php/es/proyectos-es?view=project&task=show&id=54>
- [MSW03] D. Marpe, H. Schwarz, T. Wiegand. “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard”. IEEE Transactions on Circuits Systems and Video Technologies. Vol. 13 (7), pp. 620-636. Jul. 2003.
- [NBP⁺14] E. Nogues, R. Berrada, M. Pelcat, D. Menard, E. Raffin. “A DVFS Based HEVC Decoder for Energy-Efficient *Software* Implementation on Embedded Processors”. IEEE International Conference on Multimedia and Expo (ICME) 2015, pp 1-6.
- [NMQC14] Nandamuri, A. M. Malik, A. Qawasmeh, B. M. Chapman. “Power and Energy Footprint of OpenMP Programs Using OpenMP Runtime API”. Energy Efficient Supercomputing Workshop (E2SC), nov. 2014, pp. 79-88.
- [OAPP] Repositorio en Github de las aplicaciones para Orcc: <https://github.com/orcc/orc-apps>
- [OCL] Sitio web de OpenCL, URL: <https://www.khronos.org/opencl/>
- [ODR] Plataforma Odroid U3, URL: http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275

- [OHE] Sitio web de OpenHEVC, URL: <http://openhevc.insa-rennes.fr/>
- [OHEVC] Código fuente de OpenHEVC disponible en github, URL: <https://github.com/OpenHEVC/openHEVC>
- [OMAP] Sitio web de Texas Instruments, dispositivos OMAP, URL: http://www.ti.com/llds/ti/processors/dsp/media_processors/omap/products.page#
- [OMP] Sitio web de OpenMP, URL: <http://openmp.org/wp/>
- [OMPb] Resumen y tutorials de OpenMP, URL: <https://computing.llnl.gov/tutorials/openMP/>
- [OMPC] Especificación de la versión 4.0 de OpenMP, julio de 2013. Disponible en: <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>
- [Ope03] “OpenMP Shared Memory Parallel Programming”. International Workshop on OpenMP – Applications and Tools, WOMPAT 2003, Toronto, Canadá, Junio de 2003. Editado por Michael J.Voss y publicado por Springer.
- [ORC] Sitio web de Orcc: <http://orcc.sourceforge.net/>
- [OSS⁺12] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, “Comparison of the coding efficiency of video coding standards Including High Efficiency Video Coding (HEVC)”. IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1668–1683, Dec. 2012.
- [PAPN12] M. Pelcat, S. Aridhi, J. Piat, and J.-F. Nezan. “Physical Layer Multi-Core Prototyping: A Data flow-Based Approach for LTE eNodeB”. Springer, 2012.
- [Par06] T. Parr. “A functional language for generating structured text”. URL: <http://www.cs.usfca.edu/~parr/papers/ST.pdf>, 2006.
- [PCC] PccMuTe: Power Consumption Control in Multimedia Terminals. Ministerio de Ciencia e Innovación, TEC2009-14672-C02-01.
- [PDH⁺14] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi. “Data flow-based rapid prototyping for multicore DSP systems”. Informe técnico PREESM/2014-05TR01, IETR, INSA-Rennes, 2014.
- [Pes11] Fernando Pescador. “Contribución a las metodologías de optimización del tiempo de ejecución de algoritmos de decodificación de vídeo sobre DSPs.”. Tesis doctoral leída en el Departamento de Sistemas Electrónicos y de Control de la E.U.I.T. de Telecomunicación de la U.P.M. Julio 2011.

- [PGJS13] F. Pescador, M. Garrido, E. Juárez, C. Sanz. “On an Implementation of HEVC Video Decoders with DSP Technology”. IEEE International Conference on Consumer Electronics (ICCE), ene. 2013, pp. 121-122.
- [PGS⁺06] F. Pescador, M. J. Garrido, C. Sanz, E. Juárez, D. Samper, R. Antoniello. “MPEG-4 SP/ASP decoder for a DSP-based Multi-format IP Set-Top-Box”. 32 Annual Conference on IEEE Industrial Electronics (IECON), Nov. 2006, pp. 3397-3402.
- [PRP⁺08] J. Piat, M. Raulet, M. Pelcat, P. Mu, and O. Déforges, “An extensible framework for fast prototyping of multiprocessor dataflow applications”, IDT08: Proceedings of the 3rd International Design and Test Workshop, Dic. 2008.
- [PSB14] W. Plishker, N. Sane, Shuvra S. Bhattacharyya. “A generalized scheduling approach for dynamic dataflow applications”. Design, Automation & Test in Europe Conference & Exhibition, Abr. 2009, pp. 111-116.
- [PSJ⁺12] F. Pescador, E. Seisdedos, E. Juárez, M. Garrido, C. Sanz. “Real Time H.264/SVC Compliant Video Client Based on a SoC”. 16th IEEE International Symposium on Consumer Electronics (ISCE), Jun. 2012, pp. 1-3.
- [RCS⁺13] M. Raulet, G. Cocherel, D. de Saint Jorre, M. Mattavelli y E. Juárez. “RVC description of an HEVC low delay decoder”. ISO/IEC JTC1/SC29/WG11 MPEG2013/M28209. Ginebra, Suiza, Enero 2013.
- [REC] Recore Systems, Procesador DSP Xentium, año 2012. URL: <http://www.recoresystems.com/technology/>
- [RK12] N. Ros, S. Kaxiras. “Complexity-effective multicore coherence”. PACT '12 Proceedings of the 21st international conference on Parallel architectures and compilation techniques, pp. 241-252. Septiembre 2012.
- [RNH⁺16] E. Raffin, E. Nogues, W. Hamidouche, S. Tomperi, M. Pelcat, D. Menard. “Low Power HEVC Software Decoder for Mobile Devices”. Journal of Real-Time Image Processing, Springer Verlag, 2016, 12 (2), pp. 495-507.
- [Rod15] Pablo Rodríguez Nava. “Métodos de programación para la plataforma EVMK2H”. Trabajo Fin de Grado leído en el Departamento de ingeniería Telemática y Electrónica de la E.T.S.I.S. de Telecomunicación de la U.P.M. Julio 2015.
- [RWR⁺08] G. Roquier, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, D. B. Parlour. “Automatic *software* synthesis of dataflow program: An mpeg-4 simple profile decoder case study”. IEEE Workshop on Signal

- Processing Systems, 2008 SIPS'08. Washington, D.C., EE.UU. 2008.
- [Sam84] H. Samet, "The quadtree and related hierarchical data structures", *Comput. Survey*, vol. 16, no. 2, pp. 187–260, Jun. 1984.
- [San98] César Sanz. "Arquitectura VLSI para la estimación de movimiento en codificación de imágenes". Tesis doctoral leída en el Departamento de Ingeniería Electrónica de la E.T.S.I. de Telecomunicación de la U.P.M. Marzo 1998.
- [SBCO⁺13] J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, C. A. Segall, A. Vetro, "Standardized extensions of High Efficiency Video Coding (HEVC)," *IEEE J. Sel. Topics Signal Process.* Vol. 7, no. 6, pp. 1001–1016. Diciembre 2013.
- [SCI] Sitio web de Scilab, URL: <http://www.scilab.org/>
- [SGT10] S. Stuijk, A.H. Ghamarian, B.D. Theelen, M.C.W. Geilen and T. Basten. "FSM-based SADF". Publicación interna dentro del Proyecto FP7 IST MNEMEE, 2008-2010.
- [SHM] Repositorio de secuencias de prueba de HM:
<http://ftp.kw.bbc.co.uk/hevc/hm-10.0-anchors/bitstreams/>
- [SKB11] T. Stripf, R. Koenig, J. Becker. "A novel ADL-based compiler-centric *software* framework for reconfigurable mixed-ISA processors". *International Conference on Embedded Computer Systems (SAMOS)*, 2011, pp. 157–164.
- [SOHW12] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, Thomas Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 22, No. 12, pp. 1649-1668, Dic. 2012.
- [SPR0C] Guía de usuario SPRUGW0C para TMS320C66. Disponible en la URL: <http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf>
- [SPR3P] Guía de usuario SPRUEX3P de SYS/BIOS v6.45. Disponible en la URL: <http://www.ti.com/lit/ug/spruex3p/spruex3p.pdf>
- [SPR7A] Guía de usuario SPRUGW7A memoria MSM en arquitecturas KeyStone. Disponible en la URL: <http://www.ti.com/lit/ug/sprugw7a/sprugw7a.pdf>
- [SSE] Listado de instrucciones SSE de Intel, disponible en la URL: <http://software.intel.com/sites/landingpage/IntrinsicsGuide/#>
- [Str13] T. Stripf et Al. "Compiling Scilab to high performance embedded multicore systems". *Microprocessors and Microsystems*, Nov. 2013, Vol. 37, Issue 8, Part C, pp.1033-1049

- [SYS] Texas Instruments SYS/BIOS Real-Time Kernel. URL: <http://www.ti.com/tool/sysbios>
- [TI] Listado de plataformas DSP de Texas Instruments, URL: <http://www.ti.com/lstds/ti/processors/dsp/overview.page>
- [TIWP] Texas Instruments *White Paper, Accelerate multicore application development with KeyStone software*. URL: <http://www.ti.com/lit/wp/spry231/spry231.pdf>
- [TMS0] TMS320C6678 Texas Instruments Evaluation Modules. URL: <http://www.ti.com/tool/tmdsevm6678>
- [TMS1] TMS320C6474 Texas Instruments Evaluation Modules. URL: <http://www.ti.com/tool/tmdsevm6474>
- [TWC⁺14] N. Tech, K. Wegner, Y. Chen, M. Hannuksela, J. Boyce. MV-HEVC Draft Text 9, 214arallel JCT3V-I1002, Sapporo, Japan, Jul. 2014.
- [UBU] Sitio web de Ubutnu, URL: <https://www.ubuntu.com/>
- [VLC] Sitio web de VLC, URL: <http://www.videolan.org/vlc/index.html>
- [Wip10] Matthieu Wipliez. "Compilation infrastructure for dataflow programs. Modeling and Simulation". INSA de Rennes, 2010. Inglés. <tel-00598914>
- [XDC] Versiones de XDCTools de Texas Instruments y descarga, URL: http://software-dl.ti.com/dsp/dsp_public_sw/sdo_sb/targetcontent/rtsc/
- [XDS] Blackhawk XDS560v2, URL: <http://www.blackhawk-dsp.com/products/XDS560v2.aspx>
- [XTE] Sitio web de Xtext. Información y descargas. URL: <https://eclipse.org/Xtext/>
- [XTN] Sitio web de Xtend, URL: <http://www.eclipse.org/xtend/>
- [YCR⁺14] H. Yviquel, E. Casseau, M. Raulet, P. Jääskeläinen, J. Takala. "Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms". International Symposium on Image and Signal Processing and Analysis (ISPA), Sep. 2014, pp. 732-737.
- [YDSG12] L. Yan, Y. Duan, J. Sun, Z. Guo. "Implementation of HEVC Decoder on x86 Processors with SIMD Optimization". IEEE Visual communications and Image Processing (VCIP) 2012, pp. 1-6.
- [YSJ⁺14] H. Yviquel, A. Sanchez, P. Jääskeläinen, J. Takala, M. Raulet, E. Casseau. "Efficient *Software* Synthesis of Dynamic Dataflow

Programs". IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), May. 2014, pp. 4998-4992.

ACRÓNIMOS

	<i>English / Français</i>	<i>Castellano</i>
ADF	<i>Atomic DataFlow</i>	<i>Flujo de Datos Atómico</i>
ADL	<i>Architecture Description Language</i>	<i>Lenguaje de Descripción de Arquitectura</i>
ADM	<i>Abstract Decoder Models</i>	<i>Modelos Abstractos de Descodificadores</i>
AI	<i>All-Intra</i>	<i>Todo Intra</i>
ALMA	<i>Architecture oriented Parallelization for high performance embedded Multicore systems using scilab</i>	<i>Paralelización orientada a la arquitectura para sistemas de altas prestaciones utilizando scilab</i>
API	<i>Application Programming Interface</i>	<i>Interfaz de Programación de Aplicaciones</i>
ARB	<i>Architecture Review Board</i>	<i>Junta de Revisión de la Arquitectura</i>
ARM	<i>Advanced Risc Machine</i>	<i>Maquina Risc Avanzada</i>
AVC	<i>Advanced Video Coding</i>	<i>Codificación de Vídeo Avanzada</i>
BD	<i>Basketball Drill (Sequence)</i>	<i>Basketball Drill (Secuencia)</i>
BM	<i>BQMall (Sequence)</i>	<i>BQMall (Secuencia)</i>
CABAC	<i>Context-Adaptive Binary Arithmetic Coding</i>	<i>Codificación Aritmética Binaria Adaptable al Contexto</i>
CAVLC	<i>Context-Adaptive Variable Length Coding</i>	<i>Codificación de Longitud Variable Adaptable al Contexto</i>
CB	<i>Coding Block</i>	<i>Bloque de Codificación</i>
CCS	<i>Code Composer Studio</i>	<i>Estudio de Composición de Código</i>
CISC	<i>Complex Instruction Set Computing</i>	<i>Computadora con Conjunto de Instrucciones Complejas</i>
CPU	<i>Central Processing Unit</i>	<i>Unidad Central de Procesamiento</i>
CTB	<i>Coding Tree Block</i>	<i>Bloque de Codificación en Árbol</i>
CTU	<i>Coding Tree Unit</i>	<i>Unidad de Codificación en Árbol</i>
CU	<i>Coding Unit</i>	<i>Unidad de Codificación</i>
DCT	<i>Discrete Cosine Transform</i>	<i>Transformada Discreta del Coseno</i>
DD/DD	<i>Data Driven/Data Demand</i>	<i>Entrega de Datos/Demanda de Datos</i>
DDR3	<i>Double Data Rate type 3</i>	<i>Transmisión de Datos Dual de tipo 3</i>
DISSSI	<i>PhD. On Engineering in Sístems and Services for the Information Society</i>	<i>Doctorado en Ingeniería de Sistemas y Servicios para la Sociedad de la Información</i>
DPN	<i>Data-flow Process Networks</i>	<i>Redes de Procesos de Flujos de Datos</i>
DTE	<i>Telematic and Electronic Engineering Department</i>	<i>Departamento de Ingeniería Telemática y Electrónica</i>
DSP	<i>Digital Signal Processor</i>	<i>Procesador Digital de Señal</i>
DSSF	<i>Deterministic SDF with Shared FIFOs</i>	<i>SDF Determinístico con FIFOs compartidas</i>
ED	<i>Entropy Decoding</i>	<i>Descodificación de Entropía</i>
EMIF	<i>External Memory InterFace</i>	<i>Interfaz de Memoria Externa</i>

EPFL	<i>École Polytechnique Fédérale de Lausanne</i>	<i>Escuela Politécnica Federal de Lausanne</i>
ETSIST	<i>School of Telecommunications Systems and Engineering</i>	<i>Escuela Técnica Superior de Ingeniería de Sistemas de Telecomunicación</i>
EUITT	<i>School of Technical Telecommunication Engineering</i>	<i>Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones</i>
FIFO	<i>First In – First Out</i>	<i>Primero dentro – Primero Fuera</i>
FPGA	<i>Field Programmable Gate Array</i>	<i>Conjunto de Puertas Programables</i>
FSM	<i>Finite State Machine</i>	<i>Máquina de Estados Finitos</i>
FU	<i>Functional Unit</i>	<i>Unidad Funcional</i>
FVC	<i>Future Video Coding</i>	<i>Codificación de Vídeo Futura</i>
GDEM	<i>Electronic and Microelectronic Design Group</i>	<i>Grupo de Diseño Electrónico y Microelectrónico</i>
GPU	<i>Graphics Processing Unit</i>	<i>Unidad de Procesamiento Gráfico</i>
H2B2VS	<i>HEVC Hybrid Broadcast Video Services</i>	<i>Servicios de Difusión de Vídeo HEVC Híbrido</i>
HD	<i>High Definition</i>	<i>Alta Definición</i>
HDL	<i>Hardware Description Language</i>	<i>Lenguaje de Descripción del Hardware</i>
HEAP	<i>Highly Efficient Adaptive multi-Processor framework</i>	<i>Marco multi-procesador adaptativo altamente eficiente</i>
HEVC	<i>High Efficiency Video Coding</i>	<i>Codificación de Vídeo de Alta Eficiencia</i>
IBSDF	<i>Interface-Based hierarchical Synchronous Data-Flow</i>	<i>Flujo de Datos Síncrono Basado en Interfaz Jerárquica</i>
IDCT	<i>Inverse Discrete Cosine Transform</i>	<i>Transformada Discreta Inversa del Coseno</i>
IEC	<i>International Electrotechnical Commission</i>	<i>Comisión Internacional de Electrotecnia</i>
IETR	<i>Institut d'Electronique et de Télécommunications de Rennes</i>	<i>Instituto de Electrónica y de Telecomunicaciones de Rennes</i>
INSA	<i>Institut National des Sciences Appliquées</i>	<i>Instituto Nacional de Ciencias Aplicadas</i>
INV	<i>Invalidate</i>	<i>Invalidar</i>
IP	<i>Intra Prediction</i>	<i>Predicción Intra</i>
IRISA	<i>Institut de Recherche en Informatique et Systèmes Aléatoires</i>	<i>Instituto de Investigación en Informática y Sistemas Aleatorios</i>
ISO	<i>International Organization for Standardization</i>	<i>Organización Internacional para la Estandarización</i>
ITU	<i>International Telecommunications Union</i>	<i>Unión Internacional de Telecomunicaciones</i>
JSON	<i>JavaScript Object Notation</i>	<i>Notación de Objetos JavaScript</i>
KPN	<i>Kahn Process Network</i>	<i>Redes de Procesos Kahn</i>
LLVM	<i>Low Level Virtual Machine</i>	<i>Máquina Virtual de Bajo Nivel</i>
MAC	<i>Multiply-Accumulate</i>	<i>Multiplicar-Acumular</i>
MAR	<i>Memory Address Registers</i>	<i>Registros de Direcciones de Memoria</i>
MCSDK	<i>Multicore Software Development Kit</i>	<i>Kit de Desarrollo Software Multinúcleo</i>

MCSDK-HPC	<i>Multicore Software Development Kit – High Performance Computing</i>	<i>Kit de Desarrollo Software Multinúcleo – Computación de Altas Prestaciones</i>
MD5	<i>Message-Digest Algorithm 5</i>	<i>Algoritmo 5 de Resumen del Mensaje</i>
MISSSI	<i>MSc on Engineering in Systems and Services for the Information Society</i>	<i>Máster en Ingeniería de Sistemas y Servicios para la Sociedad de la Información</i>
MKCV	<i>METIS K-way graph partition (Communication Volume)</i>	<i>Partición de Gráficos según METIS K (Volumen de Comunicaciones)</i>
MKEC	<i>METIS K-way graph partition (Edge-Cut)</i>	<i>Partición de Gráficos según METIS K (Peso de las Comunicaciones)</i>
MMR	<i>Memory-Mapped Registers</i>	<i>Registros Mapeados en Memoria</i>
MoC	<i>Model of Computation</i>	<i>Modelo de Computación</i>
MPEG	<i>Moving Picture Experts Group</i>	<i>Grupo de Expertos de Imágenes en Movimiento</i>
MR	<i>METIS Recursive</i>	<i>METIS Recursivo</i>
MR-UHDTV	<i>Mixed Reality over Ultra High Definition Television</i>	<i>Realidad Mixta sobre Televisión de Ultra Alta Definición</i>
MSM	<i>Multicore Shared Memory</i>	<i>Memoria Compartida Multinúcleo</i>
MSMC	<i>Multicore Shared Memory Controller</i>	<i>Controlador de Memoria Compartida Multinúcleo</i>
MV-HEVC	<i>Multiview HEVC</i>	<i>HEVC Multivista</i>
NAL	<i>Network Abstraction Layer</i>	<i>Capa de Abstracción de Red</i>
OpenCL	<i>Open Computing Language</i>	<i>Lenguaje de Computación Abierto</i>
OpenMP	<i>Open Multi-Processing</i>	<i>Multi-Procesado Abierto</i>
Orcc	<i>Open RVC-CAL Compiler</i>	<i>Compilador Abierto RVC-CAL</i>
OT	<i>Others</i>	<i>Otros</i>
PC	<i>Personal Computer</i>	<i>Ordenador Personal</i>
PiSDF	<i>Parametrized & Interfaced Synchronous Data-Flow</i>	<i>Flujo de Datos Parametrizado y de Interfaz Síncrona</i>
PLL	<i>Phase-Locked Loop</i>	<i>Bucle de Fase Cerrado</i>
PPS	<i>Picture Parameter Set</i>	<i>Conjunto de Parámetros de Imagen</i>
PS	<i>PartyScene (Sequence)</i>	<i>PartyScene (Secuencia)</i>
PSDF	<i>Parametrical Synchronous Data-Flow</i>	<i>Flujo de Datos Paramétrico Síncrono</i>
PSNR	<i>Peak Signal-to-Noise Ratio</i>	<i>Relación Señal-Ruido</i>
PU	<i>Processing Unit</i>	<i>Unidad de Procesamiento</i>
QM	<i>Quick Mapping</i>	<i>Mapeo Rápido</i>
QP	<i>Quantization Parameter</i>	<i>Parámetro de Cuantificación</i>
RAM	<i>Random Access Memory</i>	<i>Acceso a Memoria Aleatorio</i>
RGB	<i>Red Green Blue</i>	<i>Rojo Verde Azul</i>
RH	<i>RaceHorses (Sequence)</i>	<i>RaceHorses (Secuencia)</i>
RISC	<i>Reduced Instruction Set Computing</i>	<i>Computadora con Conjunto de Instrucciones Reducido</i>
RR	<i>Round Robin</i>	<i>Operación por Turnos</i>
RVC	<i>Reconfigurable Video Coding</i>	<i>Codificación de Vídeo Reconfigurable</i>
SADF	<i>Scenario-Aware Data-Flow</i>	<i>Flujo de Datos de Escenario Consciente</i>

SAO	<i>Sample Adaptative Offset</i>	<i>Offset de Muestra Adaptativa</i>
SANLP	<i>Static Affine Nested Loop Programs</i>	<i>Programas Anidados en Bucle Afines y Estáticos</i>
SCU	<i>Snoopy Control Unit</i>	<i>Unidad de Control Fisgona</i>
SDF	<i>Synchronous Data Flow</i>	<i>Flujo de Datos Síncrono</i>
SDL	<i>Simple DirectMedia Layer</i>	<i>Capa Simple de DirectMedia</i>
SHVC	<i>Scalable HEVC</i>	<i>HEVC Escalable</i>
SIMD	<i>Single Instruction Multiple Data</i>	<i>Múltiples Datos Una Instrucción</i>
SMP	<i>Symmetric Multi-Processing</i>	<i>Multi Procesado Simétrico</i>
SNR	<i>Signal to Noise Ratio</i>	<i>Ratio de Señal-Ruido</i>
SO	<i>Operating System</i>	<i>Sistema Operativo</i>
SoC	<i>System on Chip</i>	<i>Sistema en el Chip</i>
SPDF	<i>Schedulable Parametric Data-Flow</i>	<i>Flujo de Datos Paramétrico Planificable</i>
SPS	<i>Sequence Parameter Set</i>	<i>Conjunto de Parámetros de Secuencia</i>
SRAM	<i>Static Random Access Memory</i>	<i>Memoria de Acceso Aleatorio Estático</i>
ST	<i>String Template</i>	<i>Plantilla de Cadena</i>
TTA	<i>Transport Triggered Architecture</i>	<i>Transporte de Arquitectura Automático</i>
TU	<i>Transform Unit</i>	<i>Unidad de Transformación</i>
UE	<i>European Union</i>	<i>Unión Europea</i>
UHD	<i>Ultra High Definition</i>	<i>Ultra Alta Definición</i>
UPM	<i>Polytechnic University of Madrid</i>	<i>Universidad Politécnica de Madrid</i>
USB	<i>Universal Serial Bus</i>	<i>Bus Serie Universal</i>
VHDL	<i>VHSIC Hardware Description Language</i>	<i>Lenguaje de Descripción del Hardware VHSIC</i>
VLC	<i>Variable Length Code</i>	<i>Código de Longitud Variable</i>
VPS	<i>Video Parameter Set</i>	<i>Conjunto de Parámetros del Vídeo</i>
WB	<i>Write-Back</i>	<i>“Respuesta” / Escritura hacia atrás</i>
WLB	<i>Weighted Load Balancing</i>	<i>Equilibrio Ponderado de la Carga</i>
XMC	<i>Extended Memory Controller</i>	<i>Controlador de Memoria Extendida</i>
YUV	<i>Y, U, V (Referred to Luminance and Chrominance)</i>	<i>Y, U, V (Referido a Luminancia y Crominancias)</i>

ANEXO I – Archivos de configuración

1) Archivo de configuración de la plataforma Multi-DSP *Platform_configuration.cfg*

```
// SYS/BIOS modules activation
var program = xdc.useModule('xdc.cfg.Program');
var BIOS = xdc.useModule('ti.sysbios.BIOS');
var Cache = xdc.useModule('ti.sysbios.hal.Cache');
var ti_sysbios_family_c66_Cache = xdc.useModule('ti.sysbios.family.c66.Cache');
var HeapMem = xdc.useModule('ti.sysbios.heaps.HeapMem');
var xdc_runtime_Memory = xdc.useModule('xdc.runtime.Memory');
var ti_sysbios_knl_Clock = xdc.useModule('ti.sysbios.knl.Clock');
var Timer = xdc.useModule('ti.sysbios.hal.Timer');
var Timestamp = xdc.useModule('xdc.runtime.Timestamp');

// Place big variables and buffers in DDR3
program.sectMap[".far"] = new Program.SectionSpec();
program.sectMap[".far"].loadSegment = "DDR3";

// These sections must be placed in local core memory
program.sectMap[".args"] = new Program.SectionSpec();
program.sectMap[".cio"] = new Program.SectionSpec();
program.sectMap[".args"].loadSegment = "L2SRAM";
program.sectMap[".cio"].loadSegment = "L2SRAM";

// Program code in MSM
program.sectMap[".text"] = new Program.SectionSpec();
program.sectMap[".text"].loadSegment = "MSMCSRAM";

// User memory sections - optimizations & no cacheable variables
program.sectMap[".aux_mem"] = new Program.SectionSpec();
program.sectMap[".aux_mem"].loadSegment = "MSMCSRAM";
program.sectMap[".aux_mem_2"] = new Program.SectionSpec();
program.sectMap[".aux_mem_2"].loadSegment = "L2SRAM";
program.sectMap[".NC"] = new Program.SectionSpec();
program.sectMap[".NC"].loadSegment = "OMP_MSMC_NC_VIRT";

// Size the default stack and place it in L2SRAM
program.stack = 0x20000;
xdc_runtime_Memory.defaultHeapSize = 20000;

// OpenMP configuration
var ompSettings = xdc.useModule("ti.runtime.openmp.Settings");
ompSettings.usingRtsc = true ;

if (ompSettings.usingRtsc){
var OpenMP = xdc.useModule('ti.runtime.ompbios.OpenMP');
OpenMP.masterCoreIdx = 0;
// Num Cores
OpenMP.numCores = 8;

var ddr3 = Program.cpu.memoryMap["DDR3"];
var msmc = Program.cpu.memoryMap["MSMCSRAM"];
var msmcNcVirt = Program.cpu.memoryMap["OMP_MSMC_NC_VIRT"];
var msmcNcPhy = Program.cpu.memoryMap["OMP_MSMC_NC_PHY"];

OpenMP.msmcBase = msmc.base;
OpenMP.msmcSize = msmc.len;

OpenMP.msmcNoCacheVirtualBase = msmcNcVirt.base;
OpenMP.msmcNoCacheVirtualSize = msmcNcVirt.len;
```

```
OpenMP.msmcNoCachePhysicalBase = msmcNcPhy.base;

OpenMP.ddrBase = ddr3.base;
OpenMP.ddrSize = ddr3.len;

var HeapOMP = xdc.useModule('ti.runtime.ompbios.HeapOMP');

var sharedRegionId = 0;
var localHeapSize = 0x8000;

// 64K per core for stack +64K for other mallocs, now set for 8 cores
var sharedHeapSize=0x800000;

//Configure a SharedRegion with a heap in MSMC memory region
var SharedRegion = xdc.useModule('ti.sdo.ipc.SharedRegion');
SharedRegion.setEntryMeta(sharedRegionId, {base:ddr3.base,len:sharedHeapSize,ownerProcId:
0, cacheEnable:true,createHeap:true,isValid:true,name:"DDR3_SR0",});
HeapOMP.configure(sharedRegionId,localHeapSize);

var Reset = xdc.useModule('xdc.runtime.Reset');
Reset.fxns.$add('&__TI_omp_reset_rtsc_mode');

var Startup = xdc.useModule('xdc.runtime.Startup');
Startup.lastFxns.$add('&__TI_omp_initialize_rtsc_mode');

}else{
    program.heap = sharedHeapSize;
}
```

ANEXO II – Diagramas de los descodificadores RVC-HEVC

1) Diagrama del descodificador HEVC-RVC REF:

2) Diagrama del decodificador HEVC-RVC YUV:

3) Diagrama del descodificador HEVC-RVC FB3:

ANEXO III – Análisis de la utilización de módulos OpenHEVC sobre la plataforma Multi-DSP

Como se introdujo en el capítulo 5 y en apartado 8.4, existe la posibilidad de aprovechar ciertas secciones de código nativo del decodificador OpenHEVC sobre implementaciones basadas en especificaciones RVC-CAL. En estos casos el código fuente generado por Orcc ejecutará las funciones de la biblioteca de OpenHEVC, en lugar de las nativas a la especificación RVC-CAL.

Cabe destacar que no todos los bloques funciones disponen de módulos o funciones optimizadas. Los esfuerzos en este sentido se han centrado en aquellos módulos con mayor carga computacional o de mayor interés por su funcionalidad. Específicamente se han introducido mejoras considerables en el procesamiento de la predicción Inter.

Sin embargo, la funcionalidad del *buffer* de imágenes integrada en los actores DPB_n en las implementaciones HEVC-RVC no puede ser optimizada utilizando este método. Esto se debe a que dicha funcionalidad no se encuentra integrada en el código de OpenHEVC, no pudiendo desligar ciertas funciones para su utilización de forma independiente sobre otras implementaciones. Este hecho no tiene especial relevancia en aquellas implementaciones destinadas a funcionar sobre plataformas tipo GPP en las que el peso de estos actores es significativo pero no limitante, pero sí supone una merma en la mejora del rendimiento en las implementaciones sobre plataformas tipo DSP. Esto es debido a que en los actores DPB_n se producen importantes movimientos de datos de matrices de gran tamaño (correspondientes a los propios *buffers* de imágenes), estos datos se encuentran por lo general ubicados en memoria externa, por lo que su acceso y modificación es más lento, produciendo el consiguiente cuello de botella. Además, el hecho de reducir la carga computacional de algunos actores hace que se incremente la de los no optimizados. Esto afectará negativamente a la hora de realizar mejores distribuciones de actores entre varios núcleos cuando se desee trabajar sobre un entorno multinúcleo.

III.1. Análisis de la variación en el rendimiento sobre la plataforma DSP

Una vez se realizó la migración del código de OpenHEVC al entorno DSP y se realizaron las modificaciones detalladas en el apartado 9.7, se llevó a cabo un estudio pormenorizado del impacto resultante de la utilización de secciones de código OpenHEVC sobre la implementación HEVC-RVC en su versión YUV.

Dado que el código de OpenHEVC está especialmente diseñado para plataformas tipo GPP no todas las funcionalidades originales son susceptibles de aplicarse sobre el entorno DSP. En este sentido se utilizaron mejoras sobre los actores DBF_Y y EP_n. En algunos casos, y principalmente sobre la predicción Inter (EP) se pudieron llevar a cabo mejoras significativas del rendimiento al explotar algunas de las ventajas de la arquitectura DSP.

En la Tabla 54 se muestran los resultados obtenidos del análisis realizado sobre la plataforma DSP para la secuencia completa BQMall con QP 32, y en la Tabla 55 para la misma secuencia con QP 27. En ambas tablas se muestra un listado de todos los actores que integran el descodificador HEVC-RVC YUV, y en la segunda columna (Ciclos_v1) el número de ciclos de reloj del sistema consumidos en la ejecución de cada uno de ellos. En la tercera columna (Ciclos_v2) se muestran los ciclos de reloj consumidos por la implementación HEVC-RVC YUV con módulos de OpenHEVC. La diferencia entre ambas implementaciones se muestra en la cuarta columna. En las columnas quinta y sexta se muestra el porcentaje de la carga de cada uno de los actores respecto del total de su implementación. Por último, en la séptima columna se muestra la influencia sobre el rendimiento global de la diferencia introducida.

Actor	Ciclos_v1 (x10 ⁸)	Ciclos_v2 (x10 ⁸)	Variación	% del actor sobre el total (v1)	% del actor sobre el total (v2)	Influencia sobre el total
Source	0,05	0,05	0,44%	0,05%	0,06%	0,00%
Display	3,15	3,15	-0,03%	2,94%	3,36%	0,00%
Parser	6,22	6,23	-0,09%	5,80%	6,64%	-0,01%
ITQ	2,38	2,38	-0,26%	2,22%	2,54%	-0,01%
QP_Gen	0,27	0,27	0,40%	0,26%	0,29%	0,00%
GenInfo_GrList	0,002	0,002	1,21%	0,00%	0,00%	0,00%
GenInfo_MvCP	2,34	2,33	0,33%	2,18%	2,49%	0,01%
DBF_GB	2,24	2,23	0,29%	2,09%	2,38%	0,01%
DBF_Y	9,26	8,30	10,39%	8,63%	8,85%	0,90%
DBF_U	3,79	3,79	0,15%	3,54%	4,04%	0,01%
DBF_V	3,79	3,79	0,08%	3,53%	4,04%	0,00%
IP_Y	0,69	0,69	-1,04%	0,64%	0,74%	-0,01%
IP_U	0,53	0,55	-3,60%	0,49%	0,58%	-0,02%
IP_V	0,52	0,55	-5,35%	0,48%	0,58%	-0,03%
EP_Y	12,55	7,62	39,23%	11,69%	8,12%	4,59%
EP_U	6,13	2,40	60,95%	5,72%	2,55%	3,48%
EP_V	6,12	2,29	62,49%	5,70%	2,44%	3,56%
SAO_Y	5,26	5,22	0,74%	4,90%	5,56%	0,04%
SAO_U	1,08	1,07	0,77%	1,01%	1,14%	0,01%
SAO_V	1,08	1,07	0,79%	1,01%	1,14%	0,01%
CU_Y	7,34	7,32	0,23%	6,84%	7,80%	0,02%
CU_U	4,07	4,08	-0,14%	3,80%	4,35%	-0,01%
CU_V	3,86	3,87	-0,11%	3,60%	4,12%	0,00%

DPB_Y	15,78	15,79	-0,07%	14,71%	16,83%	-0,01%
DPB_U	4,39	4,39	0,01%	4,09%	4,68%	0,00%
DPB_V	4,38	4,37	0,16%	4,08%	4,66%	0,01%
TOTAL	107,29	93,83	12,55%			

Tabla 54. Variación del rendimiento por actores del decodificador HEVC-RVC YUV (v1) al utilizar módulos de OpenHEVC (v2) con una secuencia de QP 32.

Como se puede observar en la Tabla 54 los actores relativos a la predicción Inter (EP) son los que presentan una mejora mayor en su rendimiento, llegando al 62% en el caso de EP_V, con un impacto del 3.56% de mejora sobre el total del proceso de decodificación. Ciertos bloques funcionales presentan ligeras bajadas de rendimiento, principalmente la predicción Intra (IP), debidas a la redistribución de la carga computacional entre los actores que integran el decodificador. En cualquier caso en ningún caso la reducción en el rendimiento de estos actores supera el 0.03%, por lo que se considera despreciable en términos de rendimiento global.

En la última fila de ambas tablas se presenta la mejora global de utilizar módulos optimizados de OpenHEVC, siendo ésta del 12,55% para la secuencia con QP 32 y del 11,98% para la secuencia con QP 27.

Actor	Ciclos_v1 (x10 ⁸)	Ciclos_v2 (x10 ⁸)	Variación	% del actor sobre el total (v1)	% del actor sobre el total (v2)	Influencia sobre el total
Source	0,10	0,09	0,38%	0,08%	0,09%	0,00%
Display	3,15	3,15	-0,03%	2,58%	2,93%	0,00%
Parser	11,20	11,21	-0,11%	9,16%	10,42%	-0,01%
ITQ	3,05	3,06	-0,31%	2,49%	2,84%	-0,01%
QP_Gen	0,33	0,33	0,33%	0,27%	0,31%	0,00%
GenInfoGrfList	0,002	0,002	1,59%	0,00%	0,00%	0,00%
GenInfo_MvCP	3,33	3,32	0,34%	2,72%	3,08%	0,01%
DBF_GB	2,83	2,82	0,32%	2,32%	2,62%	0,01%
DBF_Y	9,76	8,59	11,98%	7,98%	7,98%	0,96%
DBF_U	3,81	3,80	0,12%	3,11%	3,53%	0,00%
DBF_V	3,80	3,80	0,06%	3,11%	3,53%	0,00%
IP_Y	0,93	0,94	-0,40%	0,76%	0,87%	0,00%
IP_U	0,73	0,76	-3,46%	0,60%	0,70%	-0,02%
IP_V	0,72	0,75	-5,29%	0,58%	0,70%	-0,03%
EP_Y	13,40	8,10	39,51%	10,95%	7,53%	4,33%
EP_U	6,87	2,85	58,49%	5,62%	2,65%	3,29%
EP_V	6,85	2,70	60,53%	5,60%	2,51%	3,39%
SAO_Y	6,35	6,29	0,86%	5,19%	5,84%	0,04%
SAO_U	1,16	1,15	1,04%	0,95%	1,07%	0,01%
SAO_V	1,16	1,15	1,07%	0,95%	1,07%	0,01%
CU_Y	7,66	7,63	0,34%	6,26%	7,09%	0,02%
CU_U	4,37	4,37	-0,18%	3,57%	4,06%	-0,01%

CU_V	4,16	4,16	-0,15%	3,40%	3,87%	-0,01%
DPB_Y	16,64	16,65	-0,09%	13,60%	15,47%	-0,01%
DPB_U	4,99	4,99	0,00%	4,08%	4,64%	0,00%
DPB_V	4,98	4,97	0,15%	4,07%	4,62%	0,01%
TOTAL	122,31	107,67	11,98%			

Tabla 55. Variación del rendimiento por actores del descodificador HEVC-RVC YUV (v1) al utilizar módulos de OpenHEVC (v2) con una secuencia de QP 27.

Si bien un incremento sobre el rendimiento global en torno al 12% supone una mejora considerable cabe destacar que no todos los bloques funcionales susceptibles de optimizarse han sido modificados. En este sentido el objetivo de esta parte del trabajo desarrollado es la validación de esta herramienta para llevar a cabo ulteriores optimizaciones sobre las implementaciones desarrolladas, y no la explotación de toda la capacidad de mejora ofrecida por las plataformas DSP.

ANEXO IV – Implementación de un descodificador con el código de referencia de HEVC

En cada uno de los estándares de codificación de vídeo se distribuyen distintas versiones del código fuente que implementa las funcionalidades básicas del estándar. Estas versiones sirven como punto de partida a los fabricantes y desarrolladores que desean trabajar con el nuevo estándar. En el caso de HEVC todas las versiones del código de referencia pueden descargarse del enlace [HM]. Al comienzo de esta tesis doctoral estaba en desarrollo la versión HM 10, por lo que se utilizó la versión HM 9.0 [HM9] para la realización de las primeras pruebas y la primera migración de código fuente a un entorno basado en DSP.

La versión HM 9.0 está escrita en código C++ e incluye tanto un codificador como un descodificador. El descodificador se compone de más de 500 métodos agrupados en más de 30 clases. En la Figura 104 se facilita el diagrama con la estructura jerárquica y las distintas dependencias entre las clases más importantes y sus métodos.

Figura 104. Distribución jerárquica de las clases utilizadas por el descodificador HM 9.0.

En la Figura 105 se presenta el diagrama de flujo simplificado del descodificador HM 9.0. En primer lugar el *bitstream* se lee de archivo, se analiza el tipo NAL, si ésta es de tipo VPS, SPS o PPS se extraen los parámetros correspondientes, en caso contrario la NAL contendrá un fragmento de imagen a descodificar. El procedimiento que se sigue a continuación es el propio de la descodificación bajo el estándar HEVC descrito en el apartado 2.2. De forma adicional, en el modo *skip* no se lleva a cabo ni la cuantificación ni la transformada inversa. En este caso el descodificador utiliza la información de las CBs descodificadas previamente. Este flujo de trabajo es muy similar a los respectivos flujos seguidos por otras implementaciones de descodificadores HEVC.

Figura 105. Diagrama de flujo del funcionamiento del decodificador HM v9.0.

Dado que código de referencia de HM está escrito en C++ se deben llevar a cabo algunas modificaciones durante el proceso de migración al entorno DSP que permitan disponer de una versión funcional para este tipo de plataformas. A continuación se describen las modificaciones realizadas más relevantes:

- 1) Utilización de la biblioteca *string* en lugar de *memory*.
- 2) Redefinición de la función *find* incluida en la clase *TcomIterator* para realizar la adaptación de los parámetros de entrada
- 3) Redefinición de la función utilizada para el cálculo del valor absoluto *abs* e inclusión de la biblioteca *math*.

- 4) Redefinición de algunas clases de C++ para adaptar las variables no soportadas por el compilador C++ del entorno DSP.
- 5) La definición del tipo booleano (*boolean*) en el decodificador y en el RTOS es incompatible. Se han creado distintas definiciones para cada uno de ellos.
- 6) Se han eliminado las funciones encargadas de llevar a cabo las medidas de rendimiento con timers para PC. En su lugar se han incluido las propias del entorno DSP.
- 7) Se ha desarrollado la funcionalidad de la función *strdup* al no estar disponible para el entorno DSP.
- 8) Se ha configurado el formato de salida del compilador en modo ELF en lugar de COFF. Esto es necesario para mantener algunas de las propiedades del código C++.

La Figura 106 muestra el banco de pruebas utilizado para medir el rendimiento de esta implementación. En primer lugar se crea un hilo de ejecución encargado de la tarea de decodificación, éste lee de un archivo la secuencia codificada, realiza el procesamiento y, finalmente, guarda en un archivo de extensión *.yuv las imágenes decodificadas. A esta tarea se le ha asignado una pila (*stack*) de 16MB y una *heap* de 24MB.

Figura 106. Banco de pruebas utilizado para la toma de medidas con las distintas versiones de los decodificadores implementados.

Las pruebas llevadas a cabo con esta primera implementación se han realizado utilizando el procesador TMS320DM6437 de Texas Instruments [DM6437], el cual tiene un reloj de sistema de 594 MHz. Las memorias caché del DSP se han configurado al máximo de su tamaño, es decir, 32KB para datos y programa en la L1 y 128KB en la caché L2. Se han utilizado los timers propios del procesador para la toma de medidas de rendimiento y caracterización de los bloques funcionales del decodificador. En todos los casos se han decodificado todos los frames de las secuencias utilizadas.

La Tabla 56 muestra el número medio de ciclos de reloj (CLK x10⁶) medidos necesarios para decodificar completamente cada secuencia. También se muestran las tasas de imágenes por segundo (FPS) resultantes.

Configuración		AI				LD				RA			
Secuencia		BD	BM	PS	RH	BD	BM	PS	RH	BD	BM	PS	RH
QP 27	CLKs	365	381	471	359	195	172	234	258	197	187	238	254
	FPS	1,6	1,6	1,3	1,7	3,1	3,5	2,5	2,3	3,0	3,2	2,5	2,3
QP 32	CLKs	307	341	420	330	166	148	197	214	173	167	204	219
	FPS	1,9	1,7	1,4	1,8	3,6	4,0	3,0	2,8	3,4	3,6	2,9	2,7

Tabla 56. Rendimiento en CLKs y FPS del descodificador HM v9.0 sobre un DSP.

También se han caracterizado cada uno de los bloques funcionales que integran el descodificador para cada configuración utilizada. En la Figura 107 se muestran los datos medios en % de la carga computacional utilizada en el procesamiento de cada bloque.

Figura 107. Carga computacional por FUs para distintas configuraciones.

Como se puede observar, en las secuencias con configuración AI el bloque más pesado es el de la predicción intra (IP), mientras que en las configuraciones LD y RA el más pesado es la predicción inter. Se observa también que la carga de la ITQ es mayor en secuencias AI que LD o RA, esto se debe a que las secuencias se codifican con una tasa binaria más elevada para configuraciones AI que LD y RA. El % consumido por las etapas de filtrado (SAO y DF), así como por los bloques incluidos en otros (OT) se mantiene estable para todas las configuraciones.

ANEXO V – Código relativo a la gestión de interrupciones con el procesador TMS320C6678

Todo el código que se muestra a continuación se ha insertado en el archivo scheduler.c de la biblioteca *runtime*.

```
#include <ti/csl/src/intc/csl_intc.h>
#include <ti/csl/src/intc/csl_intcAux.h>
#include <ti/csl/csl_ipcAux.h>

CSL_IntcObj          ipcIntcObj;
CSL_IntcEventHandlerRecord EventRecord;
CSL_IntcParam        vectId;
CSL_IntcHandle        ipcIntcHandle;
CSL_IntcGlobalEnableState state;
CSL_IntcContext context;
...
...
...
static void ipc_interrupt (void *arg)
{    //Función asociada a la interrupción / borrado del evento
    CSL_intcEventClear((CSL_IntcEventId)arg);
}

static Int32 intc_init (void)
{    //Se inicializa el módulo INTC
    context.eventhandlerRecord = NULL;
    context.numEvtEntries      = 0;
    if (CSL_intcInit(&context) != CSL_SOK)
        return -1;
    //Se habilitan las interrupciones
    if (CSL_intcGlobalNmiEnable() != CSL_SOK)
        return -1;

    if (CSL_intcGlobalEnable(&state) != CSL_SOK) {
        printf ("Error: GEM-INTC global enable failed\n");
    }

    return 0;
}

void *scheduler_routine(void *data) {
    ...
    while (1) {
        ... //Se ejecutan los actores en el planificador
        if(my_actor != NULL){
            ...
        }else{ //Si no hay actores planificados se entra en modo idle

            if(intc_iniciado == 0){
                if (intc_init() < 0){
                    printf ("Error: Falló la inicialización de INTC\n");
                }
                intc_iniciado = 1;
            }
        }
    }
}
```

```
vectId = CSL_INTC_VECTID_5;
ipcIntcHandle = CSL_intcOpen(&ipcIntcObj, 91, &vectId, NULL);

EventRecord.handler = (CSL_IntcEventHandler)&ipc_interrupt;
EventRecord.arg = (void *)ipcIntcHandle;
CSL_intcPlugEventHandler(ipcIntcHandle, &EventRecord);

CSL_intcHwControl(ipcIntcHandle, CSL_INTC_CMD_EVTENABLE, NULL);

    CSL_intcGlobalNmiEnable();

    REG(PDCCMD_REG) |= PDCCMD_IDLE;
}

printf("IDLE starts on core: %d\n",MultiProc_self());
//Se entra en modo idle
asm(" idle");
printf("IDLE Ends on core: %d \n", MultiProc_self());
}
}
...
}
...
}
...
}
```

ANEXO VI: Summary of the dissertation

Objective of this summary

The objective of this section is to provide a summary, in the English language, of the thesis dissertation titled “Methodological contributions for the design of latest generation video decoders over multi-DSP platforms” carried out by the Ph.D. candidate Miguel Chavarrías Lapastora in the Universidad Politécnica de Madrid.

This summary has been written with the aim to obtain the “international Ph.D. mention” according to the R.D 99/2011 and the R.D. 1393/2007.

The references are available in chapter “BIBLIOGRAFÍA”.

1. Introduction and objectives

1.1. Introduction

The constant evolution of the video coding tools (codecs) and standards, as well as the improvements introduced in this field, makes shorter the developing time of new codec versions. The video coding and decoding applications demand high-performance platforms with high computing power features in order to achieve real-time decoding rates. Nowadays, there are different technologies used to implement this kind of multimedia applications, they could be classified into three groups: based on General Purpose Processors (GPPs), based on specific hardware architectures and based on Digital Signal Processors (DSPs).

Since 1996, the *Grupo de Diseño Electrónico y Microelectrónico* (GDEM) of the *Universidad Politécnica de Madrid* (UPM) is researching on digital video coding implementations over specific hardware architectures. In this sense, two Ph.D. works were done [San98] [Gar04]. However, since 2004, different companies started to design high-performance DSP chips, thus permitting to implement video applications over these architectures. Moreover, these processors include a different kind of devices, as video input/output connectors, digital-to-analogic converters or LAN access, allowing to implement full systems on a single chip (System-on-Chip or SoC).

The SoC-based implementations allow reducing the development times compared to the specific hardware architectures. This is possible thanks to the easiness of introducing modifications in the software running on the processor. In the other side, the price and the power consumption of the DSPs are typically higher than the specific hardware solutions.

The Ph.D. work presented in [Pes11] was done with the aim to provide a new methodology for the optimization of the performance of video decoders for different coding standards (MPEG-2, MPEG-4, H.264, and H.264/SVC) over different DSP platforms. That methodology is designed for mono-core DSP processors. Currently, it is not suitable to achieve a real-time performance due to the higher spatial and temporal resolutions of the video streams supported by the new coding standards like the High Efficiency Video Coding (HEVC) [ISO15] [SOHW12]. Both issues require a higher power computational load during the decoding process.

To tackle these problems, it is necessary to have more than one core per processor, thus the computational load can be split into more processing units. In this sense, since 2012, some DSP manufacturers released different devices which integrate more than one DSP core in the same processor (Multi-DSP). These products provide a higher computational capacity as well as a reduced power consumption, making them suitable to be used for battery-dependent devices.

Several previous works have already studied the partitioning processes of the computational load of an algorithm running on multicore platforms with complex Operating Systems (OSs) like Windows or Linux. For instance, in [YSJ⁺14] and [PSB14], different methodologies about the automatic partitioning of the computational load are presented. Although, these works are only focused on GPPs.

Nevertheless, the OSs like Windows or Linux are not available for Multi-DSP platforms, but they can incorporate Real-Time Operating Systems (RTOS) [SYS] typically developed by the chip manufacturer. In these cases, the designer should accomplish the partition of the computational load by hand, considering the disadvantages of this process.

Recently, GDEM has concentrated part of its research effort in specific hardware architectures for video coding in the last generation multicore processors. In this context, this thesis work has been involved in the H2B2VS [H2B] and MR-UHDTV [MRU] projects.

1.2. Work methodology

Following, the fulfilled working methodology is presented. It has been divided into four main parts:

- 1) Study about the state of the art on the implementation of algorithms over multicore platforms, as well as the procedures to balance the computational load over more than one core. This study has not been limited to video decoders and Multi-DSP platforms.
- 2) Migration, adaptation and optimization of an HEVC video decoder over a Multi-DSP platform.

- 3) Summarize a design methodology, regarding the experience obtained through the second part. This methodology should increase the flexibility of the designing and developing processes.
- 4) Development of a design tool, it will include all the required steps when applying the methodology, proposed in the third step, in an automatic working flow.

1.3. Structure of the thesis report

The following index relates the subsections of this brief report with the contents of the complete thesis dissertation (written in the Spanish language). The complete thesis document is divided into 5 main parts, which contains 13 chapters and the appendixes.

The first part is dedicated to the introduction, and it contains the first chapter. Here, the reference framework of the GDEM research lines, the objectives of the Ph.D. work and the structure of the document are outlined.

The second part (section 2 of this document) contains three chapters which summarize: chapter 2, the most recent video coding techniques; chapter 3, the state of the art in the implementation technologies for video coding applications, and chapter 4, the state of art of methodologies for the multicore applications design.

The third part (section 3 of this document) deals with the different technologies used during this researching work and contains four chapters. Chapter 5 summarize the OpenHEVC [HRD14b] decoder, mainly developed in the IETR group (INSA of Rennes, France) [INSA]. Chapter 6 presents the most relevant concepts about the Open Multi-Processing (OpenMP) Application Programming Interface (API) [OMP], used for the parallelization of source code. Chapter 7 presents the features of the different platforms used to validate the research work. Finally, the Open RVC-Cal Compiler (Orcc) [ORC] is presented in chapter 8.

Both, second and third parts, are briefly introduced in this document as they deal with the state of the art and the different technologies used in the research work.

The fourth part of the dissertation presents a new methodology for the design of video decoders over Multi-DSP platforms in chapter 9. The proposed methodology has been verified over different platforms and tested with different HEVC video decoder implementations. Thus, chapter 10 presents the workbench, the details of the HEVC video decoder implementations and the obtained results. Both chapters are outlined in this document in sections 4 and 5, respectively.

Finally (section 6 of this document), the fifth part of the document outlines the results of the Ph.D. work (chapter 11), the published works as well as other achievements (chapter 12), and the future research lines (chapter 13).

The dissertation ends with the list of references, the acronyms and the appendixes. In this document, only the related references have been endorsed.

2. State of the art of the research work

2.1. Digital video coding

2.1.1. Basis of the digital video coding

Since the beginning of the digital video coding in the 80's, a wide range of techniques have been implemented in order to reduce the bit rate of the multimedia transmissions. Two international organizations lead the video coding standardization since then, the International Telecommunication Union (ITU) [ITU] and the International Organization for Standardization (ISO) [ISO]. The summary of coding techniques has been periodically improved generating new coding standards.

Any video stream can be compressed (introducing losses or not) to reduce the payload taking advantage of three existing redundancies: spatial redundancy (quantification processing), temporal redundancy (movement prediction) and perception redundancy (profiting the characteristics of the human view).

2.1.2. High-Efficiency Video Coding

The last digital video coding standard is HEVC/H.265 [ISO15] [SOHW12] under the rule ISO/IEC 23008-2. It has been jointly developed by the ITU and the Moving Pictures Experts Group (MPEG) [MPEG] through the Joint Collaborative Team on Video Coding (JCT-VC) [JCT].

The HEVC standard was developed with the aim of both, reducing the transmission bit rate with the new high-resolution images (up to 3840x2160 pixels or UHD) and ease the parallel processing of the algorithms implementing the decoder. HEVC does not introduce great differences looking to its predecessor, the AVC/H.264 standard [ISO03]. Quite the opposite, it performs a set of improvements over the different modules of the codec to achieve up to a 50% bit rate reduction.

Other referencies about HEVC related-works can be found in: [BBSF12] [OSS⁺12] [ITU15].

2.1.3. MPEG Reconfigurable Video Coding

The MPEG Reconfigurable Video Coding (RVC) standard emerges with the aim to provide a solution to the classic and monolithic development of video coding codecs. The MPEG RVC framework has been standardized under the ISO/IEC 23001-4 [ISO09] and ISO/IEC 23002-4 [ISO10].

RVC permits to design of video decoders from a high-level specification based on data-flow graphs called Abstract Decoder Models (ADMs). The main benefits of

RVC are the reusability of the already written code, the modularity of the specifications, a coarse-grained parallelism and an abstract programming language called RVC-CAL Actor Language (CAL) [CAL].

The functionality of the applications is expressed in entities called actors, in which the code implementing the algorithms is encapsulated. The actors are interconnected through FIFO communication channels.

2.2. Implementation technologies

A detailed study about the state of the implementations of video decoders published in the related scientific journals has been done. This study has been focused on the three platforms used in this thesis (high-performance GPPs, embedded systems based on ARM processors and Multi-DSPs).

2.2.1. High-performance GPPs

Some powerful processing systems are integrated by some high-performance GPP cores. These systems typically support OSs like Linux or Windows. Actually, two processors manufacturers have taken the market, Intel [INT] and AMD [AMD]. These kind of platforms are very useful during the developing of new software tools and applications, as it is easy to work with a set of programming tools available on the same platform. These platforms achieve the best decoding rates; but it is worth noting that they require powerful energy sources and heat dissipation systems, making them not suitable for embedded systems.

The Intel and AMD processors have SIMD instruction sets, like SSE [SSE] and 3DNow [3DN] respectively, which can be used to optimize the performance of any application. In [JRS⁺14] and [JYS⁺16], related work with this kind of optimizations and RVC-CAL specifications are detailed.

2.2.2. GPPs for embedded systems

Nowadays, the embedded systems are being used in a wide range of applications. Most of the multimedia portable devices are integrated into this kind of systems. In this sense, the ARM Holding [ARM] leads the market, and its products are being used by the most important electronics manufacturers as Samsung, Apple or Sony Ericsson. This is mainly due to the benefits offered by the inner architecture, in terms of low power consumption and heat dissipation.

Some relevant works over HEVC decoders using these systems are the following: in [NBPM⁺14] researchers of the INSA Rennes presents an implementation of the OpenHEVC decoder (see section 3.1), achieving real-time decoding with sequences with a resolution of 1280x720pixels using four cores. In a similar sense other researchers present works where a real-time decoding rate is achieved using two ARM Cortex A-9 cores with the same images resolution, [DSY⁺14] and [BLRP14].

2.2.3. Multi-DSP platforms

Since its beginning, the DSPs technology has been related to the multimedia processing. But these platforms have been typically used as an accelerator device inside high computational systems ruled by a GPP processor, which dispatches some data to be processed by the accelerator (a DSP in this case).

Thanks to its specific architecture, the DSP platforms presents some advantages in front of the GPP. The DSP architecture allows the parallel processing of multiple data with a single instruction. They also present some advantages at the pipeline processing level, as well as a reduced power consumption.

In this sense, in [PGJS13] members of the GDEM present the implementation of the reference HEVC code (HM v5.0) over a TI DSP platform, using one single core. Also, in [PCG⁺17] an implementation of the OpenHEVC decoder over a Multi-DSP platform is presented.

2.3. Methodologies for multicore programming

The programming of multicore platforms presents a challenge from the point of view of the differences between the features of the multicore systems, and the final use and optimization of the applications. During the last decade, the appearance of new multicore systems, with reduced power consumption and suitable for portable devices, have changed the multimedia market. The reduction of developing times, while achieving high performances, by using programming methodologies is the holy grail.

In this work a revision on the state of art of methodologies for the multicore platforms programming has been done, some of the most relevant works are the following: HEAP project [HEAP], Alma project [ALMA] or PREESM [PDH⁺14], among others. Instead of these solutions, which are not suitable for the developing of complex video decoders, a methodology based on a Dataflow Process Networks (DPN) [LP95] Model of Computation (MoC) is used for the RVC-based implementations.

The DPN-based MoC permits a high expressivity level of the algorithms, increasing the functionality of the applications. On the contrary, as the system is no deterministic, the prediction of the application' behavior before the execution is not possible.

The already mentioned MPEG RVC standard and its implementations are mainly based on DPN MoCs. The internal execution of an actor is managed by an action scheduler, where an internal Finite State Machine (FSM) determines the next action (function) to be executed, regarding the data-tokens received through the FIFOs. But, there is also an actors scheduler, where the actors are sequentially called to execution. These scheduling routines may be associated with different execution threads in order to parallelize the execution of the actors. More details about this topic can be found in [YCWR11].

3. Concerning technologies

3.1. OpenHEVC

OpenHEVC is an HEVC decoder written in C language and developed as an open-source project with research purposes inside the Libav [LAV]. It also includes some recent extensions of the HEVC standard, as SHEVC [ISO12] and MV-HEVC [TWCH⁺14]. The code is optimized, by using SIMD instructions, to be compiled over GPPs, like Intel and ARM platforms with the SSE [SSE] repertory. OpenHEVC is being used in projects like 4EVER [4EV], H2B2VS [H2B] or VLC [VLC].

The source code of OpenHEVC is available on *GitHub* [OHEVC], and it is easy to compile and test the decoder following the instructions included.

3.2. OpenMP

The OpenMP API [OMP] [OMPb] [Ope3] is a framework suitable for a wide-range of platforms which permits the parallel execution of C, C++ or Fortran code through a set of directives, library functions and environment variables. These OpenMP elements may be added to an already written code in order to control the behavior of the execution over a multi-core platform. Besides, the synchronization or concurrency problems are controlled automatically by OpenMP.

OpenMP is based on the fork/join execution model (see Figure 1) by means of which a master thread starts the execution until a parallel region arrives. Then, the parallel region is simultaneously executed by a team of threads as far as the join point. Thus, OpenMP allows the serial execution of tasks (top of Figure 1) into parallel regions (bottom of Figure 1).

Figure 1. Fork/join execution model.

OpenMP is integrated by three kinds of constructions. Following, the most relevant are outlined:

- *Directives*: the directives specify the behavior of the parallel regions. They are all preceded by a `#pragma` sentence. There are the following three main directives available: *for*, *sections*, *parallel* and *single*. The first one parallelizes the execution of a *for* loop, the second one permits to specify a set of different code sections to be simultaneously executed. Finally, the *single* directive allows the execution of a code by only one core within a parallel region.
The functionality of the directives may be complemented thanks to a wide range of clauses. There are clauses to specify the use of variables (shared, private...), the synchronization of the execution and the scheduling of the loops' execution.
- *Library functions*: OpenMP permits to control the behavior of the execution at run-time. Through a set of routines, some features like set/get the number of threads may be satisfied.
- *Environment variables*: the features to control the execution of the code that can be expressed with library functions are also configurable as environment variables.

Finally, OpenMP presents some advantages and disadvantages. On one side, it is easy to parallelize a serial code with wide granularity levels, it is supported by any platform compliant with the API with shared memory, and it is not necessary to do any modification to the serial code. On the other side, it could be difficult to debug the parallel code and the compiler must support OpenMP.

3.3. Development platforms

Throughout this research work, different platforms have been used in order to verify both, the proper functioning of the video decoders and the presented methodology, as well as to obtain the results and to compare them with the different architectures and other implementations. Three different platforms have been chosen: a high-performance GPP, an embedded system based on a multicore ARM and a Multi-DSP. While the Multi-DSP platform has been the focus according to the goals of this thesis, the others have been used to validate the design methodology with OpenMP and verify the results in terms of speedup while using more than one core.

The chosen Multi-DSP platform is the TMDSEVM6678 [EVM6678] manufactured by Texas Instruments [TI]. The main device of this platform is the TMS320C6678 [C6678] processor, placed together with other devices (extern flash and RAM memories, switches, LEDs...) and interfaces (JTAG, UART, Ethernet...) to ease design and test processes.

The working environment with the TI platforms is supported by the Code Composer Studio IDE [CCS]. The basic features of this tool should be complemented with some extra packages, provided by TI, depending on the design needs. In this case, it has been necessary to install the Multi-Core Software Development Kit (MCSDK) [MCS]. CCS also gives support to the RTOS of the DSP, as well as the platform configuration files.

Taking into account the most recent progress on the development of heterogeneous platforms some of the last milestones of this thesis have been tested using the EVMK2H [K2H] platform. It is also manufactured by TI and includes the 66AK2H12 SoC [66A] which contains a quad-core ARM together with the TMS320C6678 processor. This platform contains some internal shared memory, regarding the TMDSEVM6678, which allows us to develop some additional tests of the implementations.

The GPPs chosen platforms are a PC with an Intel core i7 [I7] processor and the Odroid U3 [ODR] with the Exynos 4412 processor. The last one contains a quad-core ARM Cortex A-15. Table 1 summarizes the main features of all these platforms.

Platform / Processor	Number of cores	Freq. (GHz)	Mem. L1 (KB)	Mem. L2 (KB)	Other memory	Extern memory
TMDSEVM6678 / TMS320C6678	8	0.80~1.20	32	512	4 MB (shared)	DDR3 @ 667MHz
EVMK2H / TCI6636	8 DSPs 4 ARMs	1.00~1.20 1.70	32 32	1024 1024	6 MB (shared)	DDR3 @ 800MHz
PC / Intel Core i7	4	3.07	32	256	8 MB (L3 cache)	DDR3 @ 1600 MHz
Odroid U3 / Exynos 4412	4	1.76	32	1024	-	DDR2 @ 880MHz

Table 1. Summary of features of the used platforms.

3.4. Open RVC-Cal Compiler – working environment

The Open RVC-CAL Compiler is a tool developed since 2010 in the INSA of Rennes [Wip10] for the automatic generation of source code from RVC-CAL specifications and data-flow descriptions of video decoders. Orcc can generate source code in different programming languages, like C, Java or LLVM, among others, thanks to a set of backends included in the compiler.

The compiling process performs three steps: in the first one, or front-end, an Intermediate Representation⁷⁶ (IR) of the code of the actors is performed, in the second one, the middle-end, the information about the communication channels between the actors is included. Finally, on the back-end, the generation of the source code in one of the available programming languages is performed. The generation is accomplished by using templates in Xtend language [XTN], which could be modified in order to adapt the code for different platforms.

Once the code has been generated it should be compiled for the target platform. But, due to the amount of generated files and the existing dependencies with native resources (as threads and synchronization elements), it is desirable to use a pre-compiling tool like CMake [CMA] to generate the makefiles.

⁷⁶ An Intermediate Representation is a low-level representation of the structure and the syntax of a dataflow-based representation.

Figure 2 summarizes the working flow with Orcc. Here, the original working flow with threads-based solutions is described in the center of the figure, and the new workflow based on OpenMP (maintaining the same working methodology) in the left. Also, Orcc generates an XML-format file (*.xcf) that may be used to map the execution of the actors among cores (or mapping) without recompiling the decoder. More details about this topic can be found in [CPG⁺16b]

Figure 2. Working flow diagram with Orcc.

4. Design of digital video decoders for multicore DSPs

The working method that has allowed the development of the design methodology for multi-core chip-based video decoders proposed in the thesis is based on the experimental implementation of HEVC video decoders from RVC-CAL descriptions, using as target technology multicore chips. Although, the initial approach was to perform a single implementation based on a Multi-DSP chip, but during the development of the thesis have been arising different versions of HEVC-RVC decoders, whose use has greatly heightened the development process of the design methodology. Also, along the thesis has seen the convenience, and in some cases, the opportunity, to use other multicore chips based on DSP or GPP technologies.

Finally, the experimental part of the work has been more dilated than expected, as well as more fruitful. Table 2 summarizes the different RVC-CAL versions of the implemented HEVC decoders, as well as the platforms that have been used during its development.

Platform Video codec	TMDSEVM6678	ARM / Odroid U3	PC / I7	EVMK2H
HEVC-RVC REF	✓	✓	✓	
HEVC-RVC YUV	✓	✓	✓	
HEVC-RVC FB3			✓	✓

Table 2. Platforms and decoder implementations performed during the design of the methodology.

The following study cases have been taken into account during the experimental work, all of them have been included in the design methodology:

- ✓ Automatic generation of code: the automatic generation of source code using Orcc and RVC-CAL specifications is well solved, but not when the target platform is a DSP. The C source code (Orcc libraries and actors' code) has been modified in order to be compliant with the TI DSPs compilers, using only one core.
- ✓ Parallel execution of code over multicore platforms: as the source code generated by Orcc uses typical execution threads (threads or pthreads libraries) to manage the parallel execution of actors, they should be replaced by OpenMP parallel regions, in order to be compliant with the DSP platform. In the same sense, all synchronization elements have been replaced by the respective OpenMP ones.
- ✓ Data-flow management and cache miss: in the first place, the management of the data transmissions over a dataflow-based implementation should be carried out in the most efficiency way. The possible bottlenecks along the graph or an excessive use of synchronization barriers will dramatically reduce the global performance.

Besides, the DSP platforms do not include automatic cache coherence protocols, in face of GPPs, where the data transfers between the memories are guaranteed by the OS or by hardware, like using a Snoopy Control Unit (SCU). In this sense, an optimized software solution has been proposed at the level of FIFO read/write operations, while both, code and FIFOs are placed into internal shared memory.

- ✓ Actors' mapping: it is worth nothing that the procedures to balance the computational load among a number of cores over multicore platforms become an unaffordable task when the number of cores or the number of actors (or both) increase. In this sense, automatic routines to carry out mapping should be implemented.

The automatic mapping is performed as follows: firstly, a profiling over one single core is performed, this profile will register information about the computational load of each actor, the communication volume across the FIFO channels and the efficiency of the actor's scheduler. Then, the decoding process is stopped while the mapping is calculated taken into account the profile. When the new mapping has been loaded the decoding resumes. It is possible to recalculate a mapping every 'n' frames.

There are different strategies to follow at the time to calculate a new mapping. Each strategy will promote the enhancement of the following critical factors: the load balancing among cores, the reduction of the communication volume, or the reduction of the number of communications between different cores.

- ✓ Power consumption control: as much as the number of cores increase as much the power consumption of the system. The power consumption of a platform is crucial on portable devices, battery dependent. In this work, an algorithm to balance the performance (in decoded FPS) and the number of active cores is implemented.
- ✓ Code optimization: it is not an objective of this theses the optimization of the code in order to exploit the capabilities of the platforms. But, it has been included the possibility to use optimized modules instead of some parts of the Orcc generated code. In this sense, some functions of the OpenHEVC code have been used, as it presents a better performance, in terms of FPS than the achieved with RVC-based implementations.

To do that the OpenHEVC source code has been migrated to the same DSP environment.

- ✓ The configuration of the target platform: finally, the configuration of the resources of the DSP platform, like memories sizes and configuration, is performed. This process is done taking into account the previous steps and satisfying its requirements.

Finally, the set of solutions provided defines a set of steps, a methodology, which may be applied to the design of video decoders from RVC-CAL descriptions for different platforms and multicore chips. This set of steps has been formalized in a new Orcc Multi-DSP backend (DSP backend), providing a tool to generate new implementations of video decoders automatically. Nevertheless, the new backend is easy to modify, this will allow to update the functionality and to improve it.

5. Implementation of HEVC decoders

In this section, the most relevant contributions of the thesis are presented, as well as the results of the video decoders implementations.

5.1. The HEVC decoders

Three different HEVC-RVC-based implementations have been used to verify the proposed design methodology. The complexity of each implementation is different, as well as the granularity in terms of computational load of the actors that integrate the decoders. Following, the main features of each implementation is outlined:

- HEVC-RVC Reference: it is the first RVC-based implementation of an HEVC decoder, it presents a low parallelization degree due to the unbalanced computational load of the inter prediction actors. Figure 3 (a) shows the top of the hierarchy of the video decoder implementation. It is integrated by 33 actors, 21 only for the Inverse Transformation processing, and 72 FIFOs.
- HEVC-RVC YUV: the second RVC-based version of an HEVC decoder splits the processing of the images by its luminance and chrominance components. This permits to increase the parallelization degree of the algorithms. Figure 3 (b) shows the top of the hierarchy of the video decoder implementation. It is integrated by 24 actors⁷⁷ and 70 FIFOs.

Figure 3. Top of the hierarchy of two HEVC-RVC decoders.

- HEVC-RVC FBn: these are the last versions of an RVC-based HEVC decoder. In these implementations the decoding of some frames (up to 4) is processed in parallel, this is possible by instantiating several instances of YUV decoders plus a parser to dispatch the NALs to decode (see Figure 4). Thus, these implementations permit to achieve higher speedup factors by increasing the memory requirements of the decoder. In our experiments the FB3 version has been used, it is integrated by 72 actors and 212 FIFOs.

⁷⁷ Even though the number of actors of the YUV-based implementation is lower than the Reference one the parallelism is higher in the first one as the functionality of the actors is managed in a most efficiency way. Also, the Reference version use several actors only for the inverse transformation with a reduced parallelization degree.

Figure 4. Representation of the top of the hierarchy of the HEVC-RVC FB3 decoder.

5.2. Test bench

Table 3 summarizes the features of the sequences used in the test bench.

Configuration	Name (acronym)	Number of frames	QP	Bitrate (kbps)	PSNR (dB)
Low Delay (LD)	Basketball Drill (BD)	500	27	1705	37,0
			32	822	34,1
	BQMall (BM)	600	27	1856	37,2
			32	910	34,3
	PartyScene (PS)	500	27	3391	34,5
			32	1493	31,0
	RaceHorses (RH)	300	27	2225	36,2
			32	987	33,1

Table 3. List of sequences.

5.3. Results and verification of the design methodology

In this section, some results about the verification of the presented design methodology are presented. The amount of tests and results performed along the thesis is huge, and out of the scope of this summary. In this sense, the most of the results are available in the published works (attached with this document) following outlined:

- The results of the tests with HEVC-RVC reference decoder implementation are presented in [CPG⁺15b]. In this paper, also a detailed profile of the implementation using the Multi-DSP platform is carried out.
- The results with the HEVC-RVC YUV implementation are available in [CPG⁺16b]. Here, a comparison of the performance of the implementation obtained using the GPPs is presented. This comparison demonstrates that the performance of the new OpenMP-based backend and the legacy C backend are quite similar. Also, the results of the same decoder implementation using the Multi-DSP platform are presented. In this sense, the speedup achieved using the Multi-DSP is compared with the obtained over the GPPs.

However, some results have not been already published, in particular, those relative to the global test of the methodology (including dynamic mapping of actors and using optimized modules).

The design methodology has been tested using four different platforms by implementing three RVC-HEVC based decoders (see Table 2 in section 4). But, with the aim to keep this summary as much concise as possible, only the most relevant results are given. Thus, only results obtained with the TMS320C6678 processor are presented, considering that the DSP platform is the main target of this thesis.

a) Dynamic reconfiguration and optimizations

Figure 5 presents the performance of the HEVC-RVC YUV implementation using 1, 2, 4, 6 and 7 cores. In this test, the MKEC [METIS] strategy has been chosen as an algorithm to perform an automatic and dynamic assignment of actors among cores. The remapping is calculated and set each 50 decoded frames. The figure shows the average performance, in FPS, when the 8 sequences of the test bench are decoded (see Table 3). For each number of cores, the speedup factor is also presented. To verify the use of optimized modules over RVC-CAL specifications, some extra optimizations using SIMD instructions and OpenHEVC have been carried out, mainly in the Inter Y prediction. Thus, the results are shown in two groups: using optimizations (OPT) and without optimizations (N_OPT).

Figure 5. Average performance with the HEVC-RVC YUV implementation using the Multi-DSP platform.

As it can be seen, an average speedup of 3 is achieved when 7 cores are used, regarding the decoding rate with 1 core. Also, an additional average improvement of 14% is achieved when using optimizations. The results also demonstrate that the

maximum speedup is almost achieved with 7 cores as the slope of the gain tends to level. In this sense, no results about 8 cores are given considering that one core is performing the remapping routine⁷⁸. Even more, the same curve and values of the speedup factor have been observed over the GPPs platforms⁷⁹, more details are given in [CPG⁺16].

It is worth noting that the limit of the speed up is mainly due to the high data dependencies between the actors. In this sense, some tests with All Intra-configuration sequences have been done, as the data dependencies are lower than with Low Delay configuration. These tests show speedup factors close to 5 over the RVC-HEVC YUV implementation and to 6 over the RVC-HEVC FB3 implementation when using 7 cores.

b) Power consumption

Finally, power consumption reduction techniques have been evaluated. This topic is already under development and no results have been published yet. In this case, the power consumption of the TMS320C6678 has been profiled, when the number of cores and the frequency are modified. The combination of both factors gives us a set of consumption scenarios, showing the most efficient ones. Figure 6 shows the total power needed to decode the BQMall QP 32 sequence with different configuration scenarios.

Figure 6. The power consumption of decoding the BQMall QP 32 sequence with different configurations of a number of cores and frequency.

⁷⁸ This issue have been recently solved, and the related results will be included in the thesis dissertation.

⁷⁹ The speedup factor has been verified when using up to 4 cores with the GPP platforms, as no GPP with more cores were available.

As it can be seen, the most power efficiency situation happens at decoding with 6 cores at 1,2 GHz. The increase in the power consumption with 7 cores is due to the reduced increase of the performance between use 6 or 7 cores.

These consumption scenarios provide us a tool to dynamically adapt the configuration of the resources (number of cores and frequency). In this regard, some tests have been carried out by decoding longer sequences with different features (complexity of the sequence, QP factor and resolution). To do this, some of the sequences listed in Table 3 have been concatenated. Thus, by setting a fix decoding frame rate, the decoder is allowed to move along the described power consumption scenarios, obtaining power savings as it is able to dynamically adapt at runtime the needed resources.

6. Contributions, published results and future research topics

6.1. Contributions

6.1.1. Design Methodology

The DSPs have been involved with the implementation of multimedia applications since their inception, and their use in various technological platforms is now fully extended.

In section 2 it was presented an analysis of the state of the art in the use of various technologies for the implementation of digital video decoders. In this sense, several works carried out by different research groups, and published in the scientific literature, have been analyzed. However, no reference has been found about a methodology comprehending the entire design process over Multi-DSP platforms. Likewise, the only references found on methodological procedures that allow the development of this kind of applications on different platforms are those related to the work done with the RVC standard, mainly by the IETR and GDEM research groups.

The design methodology proposed in this dissertation is the main contribution of this doctoral thesis. This methodology includes not only the design of video decoders for DSP platforms but also allows different architectures thanks to the integration of OpenMP.

Finally, it is worth noting that the developed methodology is integrated within the Orcc toolkit, fully compliant with the RVC standard. It provides an integrated work structure, on which both, updates and improvements, can be carried out in a simple way.

6.1.2. Results of the implementations

The results of the implementations permit to demonstrate the following milestones:

- The speedup factor achieved over the Multi-DSP platform is very similar to the obtained with GPPs, keeping in mind that the memory issues have been solved as a generic software solution for data flow based applications.
- For all tests and implementations, the speedup achieved is close to the theoretical maximum, calculated with the Amdahl's Law taking into account the computational load of each actor on the decoder implementation and the high number of communications.
- Both, speedup factor and decoding rate (in FPS) performance of the implementations carried out with the presented methodology for (using OpenMP) platforms, are similar to those obtained for GPPs using pthreads to parallelize the decoder execution.

6.1.3. Power consumption reduction

The power consumption is one of the most relevant issues to be considered when design applications for portable devices. In this sense, a methodological proposal to implement power-saving algorithms has been included in the design cycle.

The introduced methods permit to dynamically modify the resources of the platform (number of cores and frequency) used to decode a sequence. Thus, allowing to use only the necessary resources to achieve a targeted frame rate.

6.2. Published works and other merits

6.2.1. Published results

Following, the papers published in indexed JCR journals are listed:

- Fernando Pescador, Miguel Chavarrías, Matías J. Garrido, Eduardo Juárez, César Sanz. "Complexity analysis of an HEVC decoder based on a digital signal processor". *IEEE Transactions on Consumer Electronics*, vol. 59, no. 2, pp. 391-399. May. 2013. [PCG⁺13].

In this work, a complexity analysis of an HEVC video decoder is published. It has been implemented by using the reference code HM v9.0 over a DSP platform.

- Miguel Chavarrías, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, Mickaël Raulet. "A DSP-Based HEVC decoder implementation using an actor language dataflow model". *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 839-847. Nov. 2013. [CPG⁺13].

In this paper, the first implementation of an HEVC-RVC based decoder over a DSP platform is presented. Also, a detailed study of the computational load of the actors is presented. *This paper received the Chester Shall award* (see section 6.2.2).

- Miguel Chavarrias, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, César Sanz. “A multicore DSP HEVC decoder using an actor based dataflow model and OpenMP”. *IEEE Transactions on Consumer Electronics*, vol. 61, no. 2, pp. 236-244. May. 2015. [CPG⁺15b].

In this work, the first multicore implementation of an HEVC-RVC based decoder is presented. In this sense, results about the speedup achieved using two cores is detailed.

- Miguel Chavarrias, Fernando Pescador, Matías J. Garrido, Alexandre Sánchez, César Sanz. “Design of Multicore HEVC Decoders Using Actor-based Dataflow Models and OpenMP”. *IEEE Transactions on Consumer Electronics*, vol. 62, no. 3, pp. 325-333. Aug. 2016. [CPG⁺16b].

In this paper, the design of a new backend is presented. With this new contribution, it is possible to generate source code directly compilable for Multi-DSP platforms by using OpenMP. An HEVC-RVC implementation is tested with good speedup results while using up to 6 cores.

Following, the communications to international congresses are listed:

- Miguel Chavarrias, Fernando Pescador, Eduardo Juárez, Matías J. Garrido. “An automatic tool for the static distribution of actors in RVC-CAL based multicore designs”. *Conference on Design of Circuits and Integrated Circuits DCIS 2014*, pp. 1-6. 26-28 Nov. 2014. [CPJ⁺14].
- Pablo Rodríguez, Fernando Balseiro, Miguel Chavarrias, Fernando Pescador, Matías J. Garrido. “A DSP-based HEVC decoder implementation using RVC-CAL and native OpenHEVC code”. *International Symposium on Consumer Electronics ISCE 2015*, pp. 1-2. 24-26 Jun. 2015. [RBC⁺15].
- Miguel Chavarrias, Fernando Pescador, Matías J. Garrido, Eduardo Juárez, César Sanz. “A multicore DSP HEVC decoder using an actor-based dataflow model”. *IEEE International Conference on Consumer Electronics ICCE 2015*, pp. 370-371. 9-12 Jan. 2015. [CPG⁺15].
- Miguel Chavarrias, Fernando Pescador, Matías J. Garrido, Maxime Pelcat, Eduardo Juárez. “Design of multicore HEVC decoders using actor-based dataflow models and OpenMP”. *IEEE International Conference on Consumer Electronics ICCE 2016*, pp. 287-288. 7-11 Jan. 2016. [CPG⁺16].
- Fernando Pescador, Miguel Chavarrias, Matías J. Garrido, Javier Malagón, César Sanz. “Real-time HEVC decoding with OpenHEVC and OpenMP”. *IEEE*

International Conference on Consumer Electronics ICCE 2017, pp. 370-371 Jan. 2017. [PCG⁺17].

6.2.2. Other merits achieved

The following list of merits, related to this Ph.D. work, have been achieved:

- The work developed in this thesis has been strongly related to three international researching projects in which the GDEM participates: the HEVC Hybrid BroadBand Video Services (H2B2VS) [H2B] project, the Mixed Reality over Ultra High Definition Television (MR-UHDTV) [MRU] and the Immersive Visual Media Environments (IVME) [IVME].
- Since the last 6 years, there is a strength relationship between the IETR [IETR] laboratory of the INSA of Rennes and the GDEM. In this context, three publications related to this Ph.D. work have been co-written and published: [CPG⁺13], [CPG⁺16] and [CPG⁺16b].
- In the same collaborative context, the Ph.D. candidate did a three months internship in the IETR laboratory during the spring of 2015. This internship was supervised by the Ph.D. Maxime Pelcat.
- During the researching work, 10 Final Degree Projects have been (or are being) done related to this thesis topic in the GDEM laboratory, 8 of this projects have been supervised by the Ph.D. candidate.
- In January 2016, the paper titled “*A DSP-Based HEVC Decoder Implementation Using an Actor Language Dataflow Model*” [CPG⁺13] received the Chester Shall third best paper award from the IEEE Consumer Electronics Society. It was given during the International Conference on Consumer Electronics (ICCE16).

6.3. Future research lines

The main future research lines purposed and based on this thesis work are outlined in the following sections.

6.3.1. Heterogeneous platforms

Nowadays, different chip manufacturers are developing new heterogeneous platforms, those which include different architecture cores. These kinds of platforms offer both, high computational features and the possibility to exploit the benefits of different architectures.

In this sense, the presented methodology may be extended to take advantage of the new heterogeneous platforms. Some related work has been started in the GDEM, working with RVC-based implementations over the K2H [66A] platform, which contains a chip with 4 ARMs and 8 DSP cores. In order to provide the multicore software support over these platforms, two frameworks are being studied: OpenMP v4.0 [OMP4] and OpenCL [OCL].

6.3.2. Future Video Coding

The constant developing of new video codecs makes necessary to keep the design methodologies updated. Currently, the new Future Video Coding is being implemented by different research groups. These new techniques are not yet standardized, but it should be considered to study the different improvements by implementing the corresponding performance tests by using the presented design methodology.

6.3.3. Multi-application extension

Finally, the design methodology has been focused on the development of video decoders. But, as the methodology is based on generic data flow-based implementations it is suitable for a wide range of different applications. In this sense, the benefits of the presented methodology may be verified, and extended, to other working and research fields.