DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
SISTEMAS DE TELECOMUNICACIÓN

CONTRIBUCIONES EN SISTEMAS DE ADQUISICIÓN
DE DATOS INTELIGENTES PARA ENTORNOS DE
FUSIÓN POR CONFINAMIENTO MAGNÉTICO

TESIS DOCTORAL

Julián Nieto Valhondo

*Master Universitario en Ingeniería de Sistemas y Servicios Accesibles para la Sociedad
de la Información*

DIRECTOR

Guillermo de Arcas Castro

Doctor Ing. de Telecomunicación por la Universidad Politécnica de Madrid

2016

TRIBUNAL

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día de de 2016

Presidente:

Vocales:

Secretario:

Suplentes:

Realizado el acto de lectura y defensa de la Tesis Doctoral el de 2016 en la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la Universidad Politécnica de Madrid.

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

Agradecimientos

ÍNDICE

RESUMEN.....	viii
ABSTRACT	x
1 INTRODUCCIÓN.....	12
1.1 Justificación y motivación de la tesis.....	13
1.2 Objetivos	18
1.3 Metodología de investigación	20
1.4 Estructura de la memoria	23
2 CONTEXTUALIZACIÓN Y ESTADO DEL ARTE.....	25
2.1 Sistemas de adquisición y procesado en dispositivos de fusión.....	26
2.1.1 Introducción	26
2.1.2 Sistemas basados en tecnología PXI/PXIe.....	27
2.1.3 Sistemas basados en tecnología ATCA	29
2.1.4 Sistemas basados en tecnología MTCA.....	30
2.1.5 Sistemas ad-hoc	31
2.2 Sistemas de adquisición de datos y procesado en ITER.....	34
2.2.1 Arquitectura Hardware.....	34
2.2.2 Arquitectura Software	37
2.2.3 EPICS.....	38
2.2.3.1 Necesidades de procesado en ITER.....	44
2.3 GPUS (GRAPHICS PROCESSING UNIT).....	46
2.3.1 Introducción.	46
2.3.2 Arquitectura Hardware.....	47
2.3.3 Modelo de programación en lenguaje C-CUDA.	51
2.3.4 Programación de una GPU en otros lenguajes.	54
2.3.5 Familias de GPUS NVIDIA.....	56
3 EXPERIMENTACIÓN: ANÁLISIS DE ALTERNATIVAS.....	59

3.1	Gestión y manejo de GPUS desde herramientas de alto nivel utilizadas en iDAQS.	60
3.1.1	Introducción.	60
3.1.2	Descripción de la propuesta.	62
3.1.3	Evaluación de la solución.	69
3.1.4	Conclusiones.	71
3.2	Integración de GPUs en entornos basados en EPICS.	73
3.2.1	Introducción	73
3.2.2	Caso de Estudio 1: Cálculo de la temperatura de ion (Ti) en la cámara de intercambio (JET).	75
3.2.3	Ensayos realizados.	78
3.2.4	Caso de Estudio 2: Cálculo de los parámetros de equilibrio del plasma (TCV).	81
3.2.5	Ensayos realizados.	83
3.2.6	Conclusiones	84
4	Descripción del modelo y metodo propuesto.	86
4.1	Introducción. Objetivos	87
4.2	Descripción hardware del sistema.	90
4.3	Descripción software del sistema.	92
4.4	Tecnología NI-FlexRIO.	93
4.5	Sistema de adquisición de datos basado en FlexRIO	94
4.6	Tecnología Nvidia GPU.	98
4.7	Modelo Inicial: Sistema de adquisición de datos y procesado continuo utilizando una GPU	100
4.8	Modelo mejorado basado en el uso de RDMA.	103
4.9	Tecnologia RDMA-GPUDirect de NVIDIA.	105
4.9.1	Modificaciones llevadas a cabo en el Software de la tarjeta FlexRIO	106
4.10	Arquitectura software del modelo mejorado.	108

4.11	Evaluación del rendimiento.....	110
4.12	Metodología propuesta	114
5	EVALUACIÓN DEL MODELO Y CONCLUSIONES.....	117
5.1	Evaluación Del Modelo	118
5.1.1	Nominal Device Support.....	118
5.1.2	Descripción de la implementación.....	122
5.2	Conclusiones	130
5.3	Aportaciones	131
5.4	Producción científica y transferencia de tecnología.....	132
5.4.1	Proyectos financiados en convocatorias competitivas.....	132
5.4.2	Artículos en revistas indexadas.....	133
5.4.3	Congresos internacionales.	134
5.4.4	Propiedad Intelectual y Transferencia de Resultados.....	136
5.5	Líneas Futuras.....	137
6	Referencias.....	139
	ANEXO A: Acrónimos	149

ÍNDICE DE FIGURAS

Figura 1: Ciclo clásico de operación en dispositivos de fusión de pulso corto reproducido de (Vega, Crémy et al. 1999).....	14
Figura 2: Arquitectura de un FPSC con distintas tecnologías e interconexión con el PCS y otras redes	16
Figura 3: Previsión del número de cámaras necesarias para distintos diagnósticos de imágenes en ITER. Figura extraída de (Simrock, Patil et al. 2013).....	17
Figura 4: Metodología de investigación utilizada	20
Figura 5: Diagnósticos del dispositivo JET	26
Figura 6: Arquitectura ITMS utilizada en el dispositivo TJ-II y en el dispositivo JET. Figura extraída de (Barrera, Ruiz et al. 2005)	28
Figura 7: Diagrama de la tarjeta DGP (Digitizer-Generator-Processor) con tecnología ATCA para el dispositivo COMPASS	29
Figura 8: Arquitectura del Sistema de adquisición de imágenes basado en MTCA.....	31
Figura 9: Sistema de adquisición con elementos COTS en el dispositivo ASDEX Upgrade	32
Figura 10: Sistema de adquisición ad-hoc para el diagnóstico de microondas en el dispositivo MAST	33
Figura 11: Arquitectura Física del Sistema de Planta en ITER.....	35
Figura 12: Posibles configuraciones para un Fast Plan System Controller	36
Figura 13: Arquitectura de EPICS	39
Figura 14: Sistema de planta gobernado por EPICS.....	40
Figura 15: Arquitectura Software de un proceso IOC.....	41
Figura 16: Registro en la base de datos del IOC.....	41
Figura 17: Arquitectura software de DPD	43
Figura 18: Elemento funcional en DPD	44
Figura 19: Diagrama interno de conexión de una GPU en un Sistema con CPU Intel.....	48
Figura 20: Arquitectura interna de una GPU Tesla	49
Figura 21: Composición interna de un multiprocesador CUDA.....	50
Figura 22: Arquitectura software en CUDA	52
Figura 23: Comparativa de proceso en CPU y en GPU.....	53

Figura 24: Flujo de operaciones para procesar en GPU	54
Figura 25: Evolución de las distintas arquitecturas NVIDIA GPU	57
Figura 26: Arquitectura hardware del Sistema de adquisición utilizado en el reflectómetro KG8B de JET	61
Figura 27: Diagrama software del sistema de adquisición de datos con frecuencia de muestreo auto-adaptativa	62
Figura 28: Fases necesarias en la ejecución de aplicaciones en GPU desde LabVIEW	64
Figura 29: Etapas del algoritmo NOCOFE.....	65
Figura 30: Código GPU invocado desde un hilo de ejecución de la CPU	67
Figura 31: Aplicaciones EPICS para procesado en GPU de los dispositivos JET y TCV	74
Figura 32: Fases del algoritmo y secuencia de muestras entrada/salida	76
Figura 33: Aplicación en DPD TiCameraFIT	77
Figura 34: Aplicación DPD para cálculo de parámetros de equilibrio	82
Figura 35: Tiempos de ejecución de las tres arquitecturas para diferentes tamaños de bloque.....	83
Figura 36: Desviación estándar medida en ms para diferentes tamaños de bloque	84
Figura 37: Propuesta de mejora de movimiento de datos	88
Figura 38: Capas software en un Sistema basado en Linux	90
Figura 39: Hardware del sistema propuesto	92
Figura 40: Tarjeta NI FlexRIO con módulo adaptador.....	93
Figura 41: Chasis PXI-Express NI-1062Q	94
Figura 42: Pantalla de configuración de una FIFO-DMA	96
Figura 43: Sección del código <code><include>.h</code> obtenido con <i>C API Generator</i>	98
Figura 44: Diagrama de flujo aplicación FPGA-GPU	101
Figura 45: Definición de una estructura FIFO-DMA.....	104
Figura 46: Tabla de descriptores “ <i>chunky links</i> ”	105
Figura 47: Arquitectura software que controla el uso de una GPU desde un “tercer” dispositivo	109
Figura 48: Ancho de banda para distintos tamaños de bloque.....	111
Figura 49: Porcentaje en la mejora del rendimiento GPU-CPU	112
Figura 50: Desviación estándar para distintos tamaños de bloque.....	113

Figura 51: Metodología de diseño	116
Figura 52: Arquitectura Software de un proceso IOC	119
Figura 53: Jerarquía de clases en NDS-EPICS	120
Figura 54: Autómata de estados para la clase <i>device</i>	121
Figura 55: Plataforma hardware utilizada	123
Figura 56: Fast controller (fc18-3) en ITER con chasis PXIe y ATCA	124
Figura 57: Cámaras EOSens 3CL con interface Cameralink	124
Figura 59: Arquitectura software/hardware de la aplicación NDS obtenida ...	127
Figura 60: OPI Panel – Interface de usuario para gobernar la aplicación	128

INDICE DE TABLAS

Tabla 1: Características principales de las tres generaciones de GPU utilizadas	58
Tabla 2: Tiempos de ejecución en las etapas del algoritmo NOCOFE.....	69
Tabla 3: Influencia del tamaño del bloque de datos procesado.....	70
Tabla 4: Comparación de los tiempos de ejecución CPU-GPU.....	71
Tabla 5: Mejora de rendimiento total entre los dos sistemas	71
Tabla 6: Tiempos de ejecución en GPU y en CPU (8 cores).....	79
Tabla 7: Tiempos de ejecución en GPU y en CPU (1 core)	80
Tabla 8: Mejora de rendimiento obtenido con GPU	81

RESUMEN

Los sistemas de adquisición de datos utilizados en los diagnósticos de los dispositivos de fusión termonuclear se enfrentan a importantes retos planteados en los dispositivos de pulso largo. Incluso en los dispositivos de pulso corto, en los que se analizan los datos después de la descarga, existen aún una gran cantidad de datos sin analizar, lo cual supone que queda una gran cantidad de conocimiento por descubrir dentro de las bases de datos existentes. En la última década, la comunidad de fusión ha realizado un gran esfuerzo para mejorar los métodos de análisis off-line para mejorar este problema, pero no se ha conseguido resolver completamente, debido a que algunos de estos métodos han de resolverse en tiempo real. Este paradigma lleva a establecer que los dispositivos de pulso largo deberán incluir dispositivos de adquisición de datos con capacidades de procesamiento local, capaces de ejecutar avanzados algoritmos de análisis.

Los trabajos de investigación realizados en esta tesis tienen como objetivo determinar si es posible incrementar la capacidad local de procesamiento en tiempo real de dichos sistemas mediante el uso de GPUs. Para ello durante el transcurso del periodo de experimentación realizado se han evaluado distintas propuestas a través de casos de uso reales elaborados para algunos de los dispositivos de fusión más representativos como ITER, JET y TCV.

Las conclusiones y experiencias obtenidas en dicha fase han permitido proponer un modelo y una metodología de desarrollo para incluir esta tecnología en los sistemas de adquisición para diagnósticos de distinta naturaleza. El modelo define no sólo la arquitectura hardware óptima para realizar dicha integración, sino también la incorporación de este nuevo recurso de procesamiento en los Sistemas de Control de Supervisión y Adquisición de Datos (SCADA) utilizados en la comunidad de fusión (EPICS), proporcionando una solución completa. La propuesta se complementa con la definición de una metodología que resuelve las debilidades detectadas, y permite trazar un camino de integración de la solución en los estándares hardware y software existentes.

La evaluación final se ha realizado mediante el desarrollo de un caso de uso representativo de los diagnósticos que necesitan adquisición y procesado de imágenes en el contexto del dispositivo internacional ITER, y ha sido testada con éxito en sus instalaciones.

La solución propuesta en este trabajo ha sido incluida por la ITER IO en su catálogo de soluciones estándar para el desarrollo de sus futuros diagnósticos. Por otra parte, como resultado y fruto de la investigación de esta tesis, cabe

destacar el acuerdo llevado a cabo con la empresa National Instruments en términos de transferencia tecnológica, lo que va a permitir la actualización de los sistemas de adquisición utilizados en los dispositivos de fusión.

ABSTRACT

Data acquisition systems used in the diagnostics of thermonuclear fusion devices face important challenges due to the change in the data acquisition paradigm needed for long pulse operation. Even in shot pulse devices, where data is mainly analyzed after the discharge has finished, there is still a large amount of data that has not been analyzed, therefore producing a lot of buried knowledge that still lies undiscovered in the data bases holding the vast amount of data that has been generated. There has been a strong effort in the fusion community in the last decade to improve the offline analysis methods to overcome this problem, but it has proved to be insufficient unless some of these mechanisms can be run in real time. In long pulse devices this new paradigm, where data acquisition devices include local processing capabilities to be able to run advanced data analysis algorithms, will be a must.

The research works done in this thesis aim to determining whether it is possible to increase local capacity for real-time processing of such systems by using GPUs. For that, during the experimentation period, various proposals have been evaluated through use cases developed for several of the most representative fusion devices, ITER, JET and TCV.

Conclusions and experiences obtained have allowed to propose a model, and a development methodology, to include this technology in systems for diagnostics of different nature. The model defines not only the optimal hardware architecture for achieving this integration, but also the incorporation of this new processing resource in one of the Systems of Supervision Control and Data Acquisition (SCADA) systems more relevant at the moment in the fusion community (EPICS), providing a complete solution.

The final evaluation has been performed through a use case developed for a generic diagnostic requiring image acquisition and processing for the international ITER device, and has been successfully tested in their premises.

The solution proposed in this thesis has been included by the ITER IO in his catalog of standard solutions for the development of their future diagnostics. This has been possible thanks to the technologic transfer agreement signed with

National Instruments which has permitted us to modify and update one of their core software products targeted for the acquisition systems used in these devices.

.

Capítulo 1

1 INTRODUCCIÓN

1.1 Justificación y motivación de la tesis

Los trabajos de investigación de esta tesis doctoral se circunscriben en el marco y contexto de los sistemas de adquisición y procesamiento de información utilizados en los dispositivos de fusión termonuclear. En este tipo de dispositivos se busca extraer el mayor conocimiento de las propiedades físicas del plasma, a través de múltiples sistemas, conocidos como diagnósticos. Generalmente los diagnósticos se componen de dos partes, la sensorización y el sistema de adquisición de datos. La sensorización tiene como misión obtener una señal eléctrica proporcional a la magnitud física que se desea analizar, lo que generalmente se realiza mediante métodos indirectos debido a las altas temperaturas existentes en el plasma. Esta es la parte que diferencia fundamentalmente unos diagnósticos de otros, llegando incluso a categorizarlos en función del principio en el que se basan. Por ejemplo se habla de diagnósticos magnéticos, ópticos, de bolometría, espectrometría, reflectometría, microondas, etc. En general, a la salida de esta primera etapa de sensorización todos los diagnósticos proporcionan una señal eléctrica que debe ser tratada mediante un sistema de adquisición de datos.

Hasta hace relativamente poco tiempo la mayoría de los datos adquiridos pasaban directamente a un sistema de almacenamiento para ser procesados una vez finalizase el experimento en curso, o descarga. Sólo una pequeña parte se utilizaban para controlar la máquina en tiempo real, existiendo una clara diferenciación en todos los niveles entre el Sistema de Diagnósticos y el de Control. La duración de las descargas era muy corta, entre cientos de milisegundos y algunos segundos, por lo que a este tipo de dispositivos se les denomina dispositivos de pulso corto. La Figura 1, extraída de (Vega, Crémy et al. 1999), muestra el esquema habitual de operación de este tipo de dispositivos. . El ciclo consta de una primera fase de configuración de los sistemas e instrumentos utilizados en la adquisición de datos. A continuación se procede a la descarga, a la vez que se realizan las medidas de los canales involucrados. Los datos adquiridos son transferidos a los equipos de procesamiento, que tienen como objetivo la compresión de los mismos, para conseguir un almacenamiento óptimo. En una fase posterior, se realizan tareas de análisis con el fin de obtener finalmente los datos elaborados. No obstante, y de forma paralela a este proceso, se puede realizar un preanálisis que permita modificar la configuración inicial para descargas sucesivas.

Figura 1: Ciclo clásico de operación en dispositivos de fusión de pulso corto reproducido de (Vega, Crémy et al. 1999)

Sin embargo este paradigma de análisis presenta importantes carencias y disminuye la eficiencia en la operación de la máquina. Por ejemplo, actualmente se estima que cada descarga del dispositivo de fusión europeo JET puede generar hasta 10 Gbytes de datos para una duración del pulso (*plateau*) entorno a los 35 segundos (Murari, Vega et al. 2007). Esta forma de operar el dispositivo desde el año 1983 ha contribuido a que tan solo se hayan procesado el 10% de los datos adquiridos en todo este tiempo, lo que ha provocado que se pierda mucha información y por tanto valioso conocimiento que queda oculto en las base de datos.

Los problemas mencionados se agravan aún más por la evolución hacia lo que se denomina dispositivos de pulso largo, donde los tiempos de descarga pasan de segundos a decenas de minutos. En este tipo de dispositivos el análisis de los datos ha de realizarse de forma simultánea al proceso de adquisición, por lo que la metodología expuesta previamente carece de validez. Un ejemplo es la duración de los pulsos previstos en ITER (Aymar 1997), que dará lugar a descargas de 30 minutos, y en los que serán necesarios utilizar altas capacidades de procesamiento en los sistemas de captura de datos. En estas condiciones se prevé que exista la captura de señales con velocidades de muestreo del orden de GS/s y captura de imágenes o video de miles de frames por segundo. Además el reto pasa por realizar cálculos en tiempo real que permitan realizar previsiones sobre fenómenos indeseados en el plasma, que lo desestabilicen, como son las disrupciones. Para prever este tipo de fenómenos anómalos existen varios

trabajos (Ruiz, Vega et al. 2010), que avalan la necesidad de anticipación a este tipo de fenómenos, cuestión que solo puede ser solucionada mediante sistemas en tiempo real. Otro ejemplo de las restricciones impuestas a los algoritmos que deben ejecutarse en tiempo real, puede tomarse de aquellos que se encargan de reconstruir la situación de equilibrio del plasma mediante flujo magnético (Giannone, Cerna et al. 2010), para lo cual es necesario evaluar la información proveniente de cientos de sensores y entregar resultados en periodos entre 1 ms y 6 ms.

Por tanto, en este contexto los sistemas de adquisición de datos deben alejarse del rol pasivo que han desempeñado en los sistemas de pulso corto en los que únicamente se limitaban a adquirir, digitalizar y transmitir las señales para su posterior almacenamiento. Este es el rol que por ejemplo deberá satisfacer el Plasma Control System (PCS) en ITER (Di Maio 2015), que en una de sus funciones se encargará de controlar la evolución de los parámetros del plasma en todas las fases de la descarga. Para poder satisfacer el número de variables que han de controlarse en cada momento, de lo cual no existe parangón hasta el momento, es necesario dotar a los sistemas de adquisición de capacidades de procesamiento local utilizando elementos hardware como FPGAs, DSPs, CPUs multicore y/o GPUs (Winter, Makijarvi et al. 2012). Estos requisitos de tiempo real deben ser tenidos en cuenta a la hora de especificar las necesidades software y hardware de los sistemas de adquisición de los diagnósticos. Por ejemplo, para el diseño de la arquitectura de los sistemas encargados de gobernar las tareas de adquisición, supervisión y control en ITER se realizaron distintas propuestas (Gonçalves, Sousa et al. 2012, Ruiz, Vega et al. 2012), que han servido como punto de partida para lo que dentro de la arquitectura del sistema se denomina como Fast Plant System Controller (FPSC). Un diagrama interno de los elementos que aloja, así como la conexión con las redes existentes puede verse en la Figura 2.

Figura 2: Arquitectura de un FPSC con distintas tecnologías e interconexión con el PCS y otras redes

Estos controladores (FPSC), responsables de los procesos de adquisición y procesamiento de la información, deberán utilizar distintas tecnologías como ATCA, MTCA o PXIe, así como tareas de preprocesamiento sobre FPGAs o GPUs, para poder cumplir con los exigentes requisitos impuestos por las condiciones de contorno de este tipo de dispositivos (tiempo real, sincronización, almacenamiento masivo, manejo de elevados anchos de banda, integración con sistemas de control y adquisición de datos (SCADA), etc). También se encargan de interactuar con las distintas redes existentes de intercambio de información utilizando protocolos de sincronización IEEE1588 (Std IEEE 1588 , Sanz, Ruiz et al. 2014, IEEE Std 1588 2008), para poder archivar o intercambiar información con otros dispositivos de la arquitectura. Desde el punto de vista software, y con el fin de disponer de herramientas, que garanticen el determinismo de las aplicaciones que conforman los diagnósticos, es necesario disponer de extensiones de tiempo real de los sistemas operativos utilizados, siendo necesario también que cada elemento hardware disponga del software real-time-compatible.

De las tecnologías hardware mencionadas, el uso de GPUs para procesamiento de imágenes en tiempo real es una tendencia en crecimiento, debido a su gran capacidad de cálculo masivo de datos en paralelo. El número de diagnósticos en ITER que están enfocados al uso de imágenes es elevado tal y como puede observarse en la Figura 3, ya que existe la previsión de al menos 300 cámaras (Simrock, Patil et al. 2013) en las que será necesario realizar algoritmos de compresión, extracción de características o reconocimiento de patrones (Martin, Moncada et al. 2011).

Figura 3: Previsión del número de cámaras necesarias para distintos diagnósticos de imágenes en ITER.
Figura extraída de (Simrock, Patil et al. 2013)

No obstante también se prevé la necesidad de que sean usadas en otros diagnósticos, como magnéticos, bolometría, microondas, etc (X N Yue and B J Xiao and Z P Luo and, Y. Guo 2013) en los que se obtendrán miles de señales digitalizadas con altas tasas de muestreo (hasta 1GS/s). Por estas razones es una solución muy recomendable a valorar, teniendo en cuenta además que puede ser incluida en un sistema de tiempo real, dado su prometedor grado de determinismo en la ejecución de aplicaciones.

No obstante integrar tecnologías como GPUs o FPGAs en los sistemas de adquisición de datos tradicionales no es un camino directo, tanto a nivel técnico como desde el punto de vista de los requisitos impuesto por el contexto de este

tipo de aplicaciones (integración con los sistemas SCADA, sincronización, herramientas de gestión, etc.). Además se debe tener en cuenta las características del usuario al que deben dar servicio estas tecnologías, normalmente un científico que no tiene porqué ser un experto en el manejo a bajo nivel de dichas tecnologías. Por ello es necesario definir modelos que resuelvan los problemas de integrar estas tecnologías en los sistemas de adquisición, así como proporcionar a la comunidad de fusión metodologías para desarrollar las aplicaciones y proponer un catálogo de soluciones que resuelvan la demanda de procesamiento de señales de distinta naturaleza (analógicas, digitales, imágenes, video, etc.) en tiempo real para el catálogo de diagnósticos y sistemas de control que lo necesiten.

1.2 Objetivos

El objetivo general de esta tesis doctoral consiste en proponer un modelo y una metodología, que permitan incrementar la capacidad de procesamiento en tiempo real de los dispositivos de adquisición de datos inteligentes utilizados en los dispositivos de fusión. Para poder materializarlo a continuación se detallan los objetivos específicos en los que apoyar y fundamentar la propuesta general:

1. Analizar el estado del arte para determinar los retos que se deben abordar en el contexto de esta investigación, y determinar los requisitos que deben cumplir las tecnologías y metodologías que se propongan para darles solución. Concretamente, se trata de identificar las necesidades de uso de GPUs en los sistemas de adquisición y procesamiento de los dispositivos de fusión, así como analizar las características de las arquitecturas hardware utilizadas tradicionalmente en este tipo de sistemas para determinar los requisitos de integración. El uso extendido de GPUs en la comunidad científica, incluyendo la mayoría de experimentos de investigación a nivel mundial, demuestra la madurez de este tipo de tecnologías. Sin embargo, en la mayoría de las ocasiones se usan como una herramienta de paralelización de algoritmos para optimizar o simular el procesamiento masivo de información de manera offline (Neufeld, Vilasis-Cardona 2012, Chiariello, Formisano et al. 2013a). En ese sentido, esta investigación plantea un reto importante a la hora de integrar estas

tecnologías en un entorno con elevados requisitos de funcionamiento, tanto en cuanto a su operación en condiciones de tiempo real, como en cuanto a su integración con los sistemas SCADA mediante metodologías de alto nivel.

2. Determinar las alternativas tecnológicas comerciales existentes, para elegir la tecnología de GPUs más adecuada para su integración en este tipo de sistemas bajo las condiciones definidas en el apartado anterior. Será necesario disponer de criterios de selección que ayuden a tomar una decisión apropiada. Entre dichos criterios caben destacar la capacidad de integración con el hardware existente, la disponibilidad de lenguajes/herramientas software que faciliten su uso, la posibilidad de escalabilidad de esta arquitectura a futuras soluciones, procurando disponer de soluciones basadas en componentes COTS. También es necesario valorar el predominio tecnológico de los fabricantes que comercializan GPUs, así como las expectativas de desarrollo y mantenimiento de esta tecnología.
3. Analizar diferentes opciones a la hora de integrar este tipo de tecnologías, planteando las ventajas e inconvenientes de cada una. Para ello se realizarán diferentes casos de uso basados en escenarios reales procedentes de diferentes dispositivos de fusión existentes. Se evaluará la idoneidad de cada una de las alternativas respecto a los requisitos definidos y se identificarán las debilidades que deben ser mejoradas de cara a la propuesta de un modelo y una metodología final. También se identificarán aquellas tareas que requieren un mayor esfuerzo para obtener una adecuada integración.
4. Proponer, en base a la experimentación realizada, un modelo de integración de estas tecnologías, así como una metodología de desarrollo, que permita estandarizar el despliegue de sistemas de adquisición y procesamiento de alta capacidad para los diferentes diagnósticos de los dispositivos de fusión. La aplicación se dirigirá principalmente a diagnósticos de procesamiento de imágenes dada la gran demanda de procesamiento en tiempo real en distintas áreas de control del plasma. No obstante esta aportación podrá extrapolarse a otros campos del ámbito científico, en el que se trabaje con señales de otra naturaleza.

5. Validar la metodología y evaluar el rendimiento del modelo propuesto a través de un caso de uso real de un dispositivo de fusión existente.

1.3 Metodología de investigación

La Figura 4 expone la metodología utilizada en el desarrollo de la investigación llevada a cabo en esta tesis doctoral.

Figura 4: Metodología de investigación utilizada

La hipótesis de partida de la tesis es la siguiente:

Es posible incrementar la capacidad local de procesamiento en tiempo real de los sistemas de adquisición de datos de los dispositivos de fusión termonuclear mediante el uso de GPUs de forma que se puedan afrontar los retos que se plantean de cara a su operación en pulso largo.

Para evaluar dicha hipótesis se definen los cinco objetivos descritos en el apartado anterior, lo que hace necesario comenzar realizando un estudio del marco de referencia en el que se ubica la investigación. Estos objetivos de partida, pueden verse modificados ligeramente, a partir del estudio del estado del arte que permite situar la dimensión de la propuesta frente a lo establecido o existente en la materia. En este caso este estudio implica conocer la problemática asociada a los sistemas de adquisición de datos y procesamiento de entornos de fusión, y determinar la evolución de las soluciones adoptadas en los experimentos realizados, así como analizar las necesidades de futuro, propuestas por los científicos involucrados. Estas consideraciones permiten determinar las condiciones en las que deberá realizarse la fase de experimentación de la tesis.

En la fase de experimentación se examinan las distintas alternativas tecnológicas disponibles a través del análisis de varios casos de uso. En cada uno de ellos se ensayan diferentes modelos y se evalúan distintas metodologías de integración. Los casos de uso analizados en esta investigación permiten evaluar si los objetivos propuestos alcanzan las necesidades detectadas. Los casos de uso a los que se ha tenido acceso en el desarrollo de esta tesis, se han basado en experimentos prácticos, que se circunscriben a situaciones reales en varios dispositivos de fusión en Europa (JET, TCV o ITER). Este esfuerzo ha sido posible, gracias al soporte de los Proyectos Nacionales de I+D: ENE2009-10280, ENE2012-38970-C04-04, y a los proyectos suscritos con la ITER IO: Software Support Bundle for FlexRIO/NVIDIA GPU (IO/CT/14/4300001117) y EPICS Device Support Module for NI RIO boards (IO/CT/15/4300001145).

La línea experimental seguida en cada uno de los casos de uso responde a los siguientes puntos:

1. Contextualización. Permite identificar las carencias o las posibilidades de mejora de la situación de partida. En este punto se abordará el

- estudio del algoritmo de proceso y se analizará su adaptación a tecnologías de alto rendimiento computacional, en concreto las GPUs.
2. Experimentación. Se utilizan datos reales provenientes de distintos diagnósticos de fusión que han sido obtenidos mediante sistemas de adquisición de datos que generalmente operan en modo off-line. Se realizan diversos ensayos que permitan valorar si la propuesta de mejora, aportada por las distintas arquitecturas tecnológicas y/o metodologías de integración, supone un valor añadido en cuanto a capacidad de extracción de información y el rendimiento de la solución.
 3. Análisis de resultados. Se analizan diferentes variables en relación con el rendimiento de la solución y la efectividad de la metodología de desarrollo, tales como el *throughput* global del sistema, la latencia, o el comportamiento determinista de los algoritmos. Estas variables, particularizadas para cada parte del algoritmo, permiten determinar las fortalezas y debilidades de cada una de las arquitecturas analizadas.
 4. Conclusiones. Para cada caso de uso analizado se extraen argumentos que permitan elaborar el modelo final propuesto, como sistema de adquisición de datos inteligente para entornos de fusión. Debe valorarse la capacidad de integración de la nueva propuesta en los sistemas de diagnóstico actuales y la inversión necesaria para una satisfactoria implantación. Otro factor relevante es la validación para sistemas en tiempo real, aunque este extremo depende de las restricciones temporales que imponga el diagnóstico, así como la naturaleza de las señales o imágenes procesadas.

Con la experiencia adquirida en estos casos de uso, se presenta un modelo, así como una metodología de desarrollo para cumplir con el objetivo general de partida de la tesis. Se aprovecha el conocimiento adquirido a través de cada uno de los casos de uso analizados y las propuestas de mejora que se derivan de las debilidades detectadas en cada uno. El objetivo que se persigue es obtener una metodología útil para la comunidad de fusión, y por tanto conseguir su inclusión en los protocolos de desarrollo de algunos de los dispositivos de referencia. Concretamente esta investigación ha tenido un enfoque muy marcado hacia ITER, por ser este el dispositivo de referencia en la actualidad en cuanto a los objetivos marcados en esta tesis.

Finalmente, se evalúan y analizan los resultados obtenidos en el modelo de integración, y su metodología, con el fin de las conclusiones del trabajo realizado, y definir las futuras líneas de investigación.

1.4 Estructura de la memoria

La memoria de esta tesis doctoral está dividida en 5 capítulos a los que se añaden la bibliografía y una tabla de acrónimos utilizados durante su desarrollo. Además de lo propiamente expuesto en este capítulo (capítulo 1) se presenta:

Capítulo 2. Se realiza un estudio del contexto de aplicación y estado del arte de las tecnologías involucradas. Se hace un recorrido por las distintas tecnologías utilizadas en los sistemas de adquisición de datos en los principales dispositivos de fusión, prestando especial atención a ITER, responsable de establecer los distintos estándares para el futuro. A continuación se describen las GPUs (Graphics Processing Unit), explicando y justificando porqué su arquitectura se puede aprovechar como dispositivo de procesamiento masivo de datos. También se exponen las alternativas existentes en este sector, así como las tendencias de futuro.

Capítulo 3. Se analizan distintos casos de usos evaluados a lo largo del desarrollo de este trabajo. Utilizando la metodología de investigación descrita, se describen distintos ensayos que permiten adquirir el conocimiento necesario para adoptar el uso de GPUs como elementos de proceso en sistemas de adquisición de datos inteligentes. Destacar que son ensayos realizados con algoritmos y datos reales, provenientes de dispositivos de fusión europeos.

Capítulo 4. En este capítulo se caracteriza la metodología que permite obtener un sistema de adquisición basado en FPGAs con altas capacidades de proceso utilizando GPUs, a partir de la experiencia aportada por los casos de uso del capítulo 3. En la misma se dan las pautas necesarias que hacen posible la integración de GPUs en diagnósticos de fusión y se resuelven las debilidades encontradas en experiencias anteriores. En concreto se destaca, el esfuerzo realizado para poder realizar transferencias de datos directas entre una familia de tarjetas de adquisición basadas en FPGAs y las GPUs para optimizar el rendimiento del sistema y maximizar su versatilidad. En la propuesta se utilizan los estándares hardware definidos por ITER en su catálogo, así como el software, EPICS, estándar que permite obtener sistemas de control distribuidos en tiempo real y está ampliamente adoptado por la comunidad científica.

Capítulo 5. En este capítulo se realiza la evaluación del modelo, mediante el desarrollo de una aplicación de adquisición de imágenes y procesado en GPU

que utiliza la metodología propuesta. Esta aplicación ha sido testada de forma satisfactoria en las instalaciones de ITER, y para ello se han utilizado componentes de su catálogo para los fast controllers y su integración en EPICS ha sido posible mediante NDS.

Se exponen las conclusiones finales de la tesis, las aportaciones realizadas, la producción científica y la transferencia tecnológica realizada, y por último se plantean las futuras líneas de investigación.

Capítulo 2

2 CONTEXTUALIZACIÓN Y ESTADO DEL ARTE

2.1 Sistemas de adquisición y procesado en dispositivos de fusión

2.1.1 Introducción

El objetivo principal de los sistemas de adquisición de datos utilizados en los dispositivos de fusión consiste en obtener el mayor conocimiento de las propiedades físicas del plasma durante una descarga. Con tal fin se hace uso de una gran cantidad de elementos de medida que permiten obtener información indirectamente a través de diferentes sensores e instrumentos situados en las cercanías del dispositivo, formando lo que comúnmente se denominan el conjunto de diagnósticos del dispositivo. En la Figura 5 se muestra a modo de ejemplo el conjunto de diagnósticos del dispositivo europeo JET, formado por más de 100 diagnósticos que generan más de 8 GB de datos por cada descarga.

Figura 5: Diagnósticos del dispositivo JET

Las arquitecturas hardware que permiten adquirir esta información han ido evolucionando a la par que lo ha hecho la industria electrónica y de la computación. Dicha evolución ha estado marcada por el incremento en la capacidad de dichos sistemas para poder dar soporte a diagnósticos para descargas de mayor duración, a un mayor número de canales, a mayores velocidades de muestreo, así como una mayor resolución de las muestras. No obstante los mayores avances en los sistemas de adquisición se han producido al

dotarlos de ciertas capacidades de inteligencia, al incorporar pre-procesado en el propio sistema utilizando hardware reconfigurable, mediante el uso de FPGAs.

Las primeras plataformas hardware utilizaban instrumentación CAMAC (Schmidt 1998, Manduchi, Luchetta et al. 1999, Vega, Crémy et al. 1999), tecnología muy cara y con pocas alternativas para en el mercado a la hora de implementar nuevos diagnósticos. Más tarde y con la finalidad de añadir al sistema de capacidades de procesamiento local, se adoptó el estándar VME (Manduchi, Hemming et al. 1996), para posteriormente incorporar VXI (Crémy, Vega et al. 1999). La aparición de sistemas basados en PCI (Llobet, Duval 2002) y CompactPCI (Luchetta, Manduchi et al. 2003), supuso una mejora en la relación coste por canal, además de un incremento de la capacidad de procesamiento distribuido, lo cual redundaba en una reducción del uso del ancho de banda de red y de la carga de CPU.

Las soluciones aportadas, en sistemas de adquisición de datos, en los últimos años en los distintos dispositivos de fusión en Europa, han permitido confluir en la definición de los sistemas que se han especificado para ITER. Como ejemplos significativos en esta definición, a continuación se analizan estas alternativas, mencionando los dispositivos que las han usado, y apuntando a qué tipo de aplicaciones pueden ajustarse mejor.

2.1.2 Sistemas basados en tecnología PXI/PXIe

Como ejemplo de un sistema de adquisición de datos basado en arquitectura PXI/PXIe puede tenerse la plataforma ITMS (Intelligent Test and Measurement System) (Barrera, Ruiz et al. 2005, Ruiz, Barrera et al. 2004) que fue aplicada en el TJ-II, para diagnósticos de bolometría y participación remota, y en JET para ser usado en el reflectómetro de correlación KG8B (de Arcas, López et al. 2008). El sistema, desarrollado por el grupo de investigación I2A2-UPM junto con el Grupo de Adquisición de Datos del CIEMAT, utiliza componentes hardware comerciales PXI y CompactPCI, configurando un sistema de adquisición de datos distribuido con capacidades multiproceso. Cuenta con un chasis PXI que incluye una CPU embebida (SCPU) encargada de configurar los parámetros de adquisición y distribuir la información adquirida. En el mismo chasis coexisten varias tarjetas de proceso (PCPU), que permiten incrementar la capacidad de proceso del sistema, y varias tarjetas de adquisición en función del diagnóstico a realizar. La Figura 6 refleja la arquitectura del sistema descrito.

Figura 6: Arquitectura ITMS utilizada en el dispositivo TJ-II y en el dispositivo JET. Figura extraída de (Barrera, Ruiz et al. 2005)

Utilizando esta plataforma se han alcanzado resultados muy satisfactorios, que permiten atribuir el carácter de sistema de adquisición de datos inteligente (iDAQ), destacando entre las soluciones aportadas a la comunidad de fusión una aplicación que permite adaptar la frecuencia de muestreo en función de la información útil de la señal analizada (de Arcas, López et al. 2008). Otra aplicación desarrollada bajo la misma arquitectura permite predecir las disrupciones en el plasma utilizando reconocimiento de patrones en tiempo real (Ruiz, Vega et al. 2011). Para ello se utiliza la arquitectura referenciada anteriormente, a la que se añadió una tarjeta FPGA para incrementar las capacidades de proceso existentes en la SCPU. El uso de soluciones basadas en el estándar PXI/PXIe permite obtener sistemas escalables en los que se obtengan una mayor capacidad de proceso, o un mayor número de canales, utilizando módulos o tarjetas comerciales según se demanden en nuevos diagnósticos. No obstante, en las soluciones aportadas se echa en falta la posibilidad de comunicación entre distintas tarjetas basadas en FPGA, para distribuir adecuadamente la capacidad de proceso, así como una comunicación directa con otros elementos de proceso, como pueden ser GPUs, sin utilizar el controlador principal del sistema (CPU).

2.1.3 Sistemas basados en tecnología ATCA

Como ejemplo de un sistema de adquisición de datos basado en tecnología ATCA, puede encontrarse el diseñado para el tokamak COMPASS, ubicado en Praga, dependiente del IPP.CR (Valcárcel, Neto et al. 2009). Esta arquitectura está concebida para sistemas que necesiten alta disponibilidad, redundancia y escalabilidad para futuras ampliaciones. Está definido entorno a una placa base y un rack en el que se conectan más tarjetas que comparten un backplane con enlaces serie de velocidades de gigabit. En esta tecnología además de la redundancia, esta implementado el hot-plug, lo cual facilita su mantenimiento, además de que puede utilizarse de manera continua. La conexión entre la placa base del módulo de control y los posibles racks del sistema se obtiene mediante una conexión x1 PCIe de 500 MB/s. Cada una de las tarjetas contiene distintos conversores ADC con 18 bits de resolución y hasta 2 MS/s. El esquema de cada uno de los módulos diseñados puede verse en la Figura 7.

Figura 7: Diagrama de la tarjeta DGP (Digitizer-Generator-Processor) con tecnología ATCA para el dispositivo COMPASS

Además de incorporar conversores ADC, destaca la existencia de una FPGA Xilinx Virtex-4 que permite controlar las entradas y salidas del módulo, así como realizar pre-procesado en tiempo real, antes de enviar la información adquirida al sistema de almacenamiento. De este sistema destaca su flexibilidad para implementar distintos tipos de algoritmos, que pueden ser cambiados entre

descargas reprogramando la FPGA, así como su capacidad de incrementar sus prestaciones si se requiere más potencia de cálculo.

Otro de los sistemas que utilizan tecnología ATCA es el diseñado para los diagnósticos de espectrometría de rayos-gamma en JET(Pereira, Fernandes et al. 2010). Integra un controlador basado en un procesador x86 que gobierna 5 tarjetas. Cada tarjeta de adquisición dispone de 8 canales de adquisición de 14 bits de resolución con una frecuencia máxima de muestreo de 400 MS/s. También cuenta con una FPGA que realiza tareas de pre-procesado, enfocadas a detectar eventos. Estos eventos pueden depender por ejemplo de una comparación con un umbral, o bien por superarse una diferencia preestablecida entre muestras. Mediante estos eventos, el sistema es capaz de adquirir la información de interés en proximidades del mismo, lo cual le confieren cierta capacidad de inteligencia. Concluir que en esta tecnología destaca la alta disponibilidad ofrecida, al tratarse de sistemas que permiten redundancia de equipos (dos tarjetas con la misma funcionalidad), aunque no tiene un catálogo extenso de soluciones y alternativas, al estar en plena expansión.

2.1.4 Sistemas basados en tecnología MTCA

Puede considerarse como una de las arquitecturas más recientes, puesto que fue introducida por la asociación PICMG en 2006, y está evolucionando como un estándar utilizado en los más importantes centros de investigación internacionales (CERN, DESY, ITER, CMS LHC). Los sistemas basados en MTCA comparten la arquitectura interna con los sistemas basados en ATCA, aunque son de dimensiones más reducidas y más económicos.

Un sistema de adquisición de imágenes, utilizando esta arquitectura, ha sido desarrollado como una primera implementación para distintos diagnósticos que se llevaran a cabo en ITER (Makowski, Mielczarek et al. 2014). Dicho sistema trabaja con cámaras de alta resolución que utilizan el interface CameraLink, permitiendo trabajar con distintos *framegrabber* al mismo tiempo. El diagrama de la arquitectura propuesta puede verse en la Figura 8.

Figura 8: Arquitectura del Sistema de adquisición de imágenes basado en MTCA.

El sistema permite la adquisición y procesamiento de las imágenes provenientes de distintas cámaras a través de un chasis MTCA, a través de módulos framegrabber con capacidades moderadas de procesamiento. Dichas tarjetas que se alojan en el chasis han sido construidas utilizando el estándar Advanced Mezzanine Card (AMC), utilizando como módulo base una tarjeta comercial TEWS TAMC641, la cual incorpora una FPGA Virtex-5, para permitir funciones de procesamiento. Dicho chasis está conectado a una CPU externa, en la que se puede realizar procesamiento paralelo basado en GPU. La transmisión de información entre el chasis y la CPU se realiza utilizando un enlace PCIexpress 2.0 de 4 líneas, lo cual permite un teórico ancho de banda de 16 Gb/s. Esto representa que cada sistema de este tipo, solo pueda gobernar dos cámaras, que utilicen la mayor capacidad del estándar CameraLink, al mismo tiempo. Además estas tasas de transferencia son las existentes hasta llegar a la memoria principal del sistema CPU. Si se pretende realizar procesamiento paralelo en la GPU, estas tasas de transferencia se verán reducidas, debido a la necesidad de realizar movimientos de datos adicionales.

2.1.5 Sistemas ad-hoc

Existen otras experiencias en las que se han utilizado soluciones ad-hoc para realizar sistemas de adquisición de datos, con resultados satisfactorios. Cabe destacar la obtenida para el diagnóstico de reflectometría del plasma en ASDEX Upgrade, ubicado en Alemania (Santos, Zilker et al. 2011). Dicho sistema se basa en el uso de una tarjeta que incorpora una FPGA y otras dos que albergan conversores ADC de 12 bits y 100 MS/s, todas ellas alojadas en un sistema biprocesador Xeon, que utiliza LinuxRT (Linux OpenSuse 11.2 con kernel

2.6.31.12-rt21) como sistema operativo. En estas condiciones el sistema es capaz de enviar al sistema de procesamiento un flujo de información, agregado de todos los canales utilizados de hasta 1 GB/s. En este caso la FPGA solo se utiliza para controlar la adquisición y el movimiento de la información con el bus PCIeexpress, aunque podría realizar también tareas de preprocesado, extremo que llevaría a rediseñar el hardware alojado en la FPGA mediante lenguaje VHDL. Por ello los autores recomiendan escribir los algoritmos de procesamiento en lenguaje C/C++, más cómodos a la hora de realizar tareas de depuración. El diagrama de bloques correspondiente a dicha solución puede observarse en la Figura 9.

Figura 9: Sistema de adquisición con elementos COTS en el dispositivo ASDEX Upgrade

Otra muestra de un sistema construido a medida fue el diseñado para el diagnóstico de imagen de microondas en el dispositivo MAST (Huang, Vann et al. 2012), localizado en el Culham Centre for Fusion Energy (Oxfordshire, UK). Destaca la utilización únicamente de una FPGA Virtex 6 y dos tarjetas FMC108(FPGA Mezzanine Card) con 16 canales ADC de 14 bits capaces de muestrear a 250 MS/s (mostrado en la Figura 10). Una de las ventajas que aporta es la independencia de un sistema host o PC para capturar la información, al ser

la propia FPGA la que aloja un procesador embebido que ejecuta un sistema operativo Embedded Linux (distribuci. Una modificación en el driver del mismo, que mejora las transferencias DMA por Ethernet, permite realizar transferencias hasta 83 MB/s, lo cual representa un buen resultado frente a los teóricos 125 MB/s del estándar Gigabit Ethernet utilizado.

Figura 10: Sistema de adquisición ad-hoc para el diagnóstico de microondas en el dispositivo MAST

No obstante, en un diseño a medida como el expuesto, se encuentran debilidades como la imposibilidad de comunicación con el bus PCIeexpress, ampliamente extendido, para poder acceder a otros sistemas de procesamiento o archivado, lo cual constriñe el ámbito de aplicación de este desarrollo. Otro hándicap de estos sistemas es el esfuerzo necesario para modificar e incorporar nuevas funcionalidades y algoritmos a la FPGA, cuestión que pasa por realizar dicho trabajo en lenguajes de bajo nivel como el VHDL. Por último se debe tener en cuenta los problemas de mantenimiento y escalabilidad de este tipo de soluciones fuertemente vinculadas a las capacidades técnicas del personal disponible en cada momento en los dispositivos y la evolución de las tecnologías sobre las que se sustentan, lo cual viene agravado por el hecho de que este tipo de soluciones deben garantizar tiempos de vida elevados.

2.2 Sistemas de adquisición de datos y procesado en ITER

Las experiencias obtenidas en sistemas de adquisición y procesado, en los dispositivos de fusión por confinamiento magnético, han servido como base para establecer los requisitos y características que se han de cumplir en ITER. Este proyecto internacional, de dimensiones no conocidas hasta el momento, es en el que se encuadra la caracterización del modelo y metodología de esta tesis, por lo que a continuación se describe el sistema de instrumentación y control adoptado en su especificación. Con el fin de contextualizar y situar los sistemas de adquisición y procesado, dentro de la jerarquía de un proyecto de tal magnitud, a continuación se describe brevemente su organización relativa a su instrumentación, control y aspectos software.

2.2.1 Arquitectura Hardware

En la arquitectura hardware que ITER ha definido para cada uno de los sistemas que la integran, existen más de doscientos elementos denominados Plan System Instrumentation and Control (PS-I&C). Cada uno de estos subsistemas dará soporte al control de los campos magnéticos para confinar el plasma o para controlar el sistema de refrigeración del agua. Todos estos elementos conforman el sistema de control, comunicación y acceso a datos, denominado Control, Data Access and Communication Control system (CODAC), de acuerdo a la Figura 11 (Wallander, Abadie et al. 2010). Cada uno de ellos está constituido por distintos elementos, pertenecientes al catálogo de opciones que el propio proyecto ha estandarizado y normalizado. Los elementos que lo integran son: sensores y actuadores, Fast Controllers (FC-I&C), Slow Controllers (SC-I&C), Plant System Host (PSH-I&C), Plan Interlock System (PIS-I&C) y el Safety Controller.

Figura 11: Arquitectura Física del Sistema de Planta en ITER

Para poder conectar todos los subsistemas existentes se han definido distintas redes de área local, que a su vez conectan con otros elementos de categoría superior en la jerarquía como son el CODAC-HPC, los sistemas centrales de bloqueo y seguridad CODAC Central Interlock System y Central Safety System (CODAC-CIS & CODAC-CSS), los servidores de servicios y aplicaciones CODAC, y los terminales de monitorización. Como elemento de sincronización entre los distintos subsistemas se ha optado por la adopción del protocolo IEEE 1588-2008, el cual permite una exactitud de ns en la sincronización entre dispositivos conectados a una misma red Ethernet.

De todos los subsistemas referidos anteriormente, perteneciente al Plan System-I&C, uno de los elementos que incorpora los sistemas de adquisición y procesamiento de datos es el Fast Controller I&C. Son dispositivos cuya arquitectura está basada en procesadores Intel x86 y bus PCI/PCIe. Estos dispositivos presentarán un formato industrial 4U PICMG 1.3 (PICMG CONSORTIUM), en los que se alojará un backplane con varios slots PCIe, una tarjeta CPU biprocesador con Intel Xeon Quad Core, cuatro interfaces de red 10 Gbit y un sistema de alimentación redundante. Según las necesidades para diagnósticos, algunos de estos elementos pueden incorporar una o varias GPUs para procesamiento de datos. Además estarán interconectados mediante un interfaz de conversión de bus, a un chasis con formato PXIe (Ruiz, Vega et al. 2012), ATCA o MTCA (Gonçalves,

Sousa et al. 2012). Estas tres posibles configuraciones se muestran en la Figura 12. La principal característica de los FC-I&C radica en su capacidad de adquirir datos a velocidades desde 100S/s hasta GS/s, con posibilidad de incluir marcas de tiempo.

Figura 12: Posibles configuraciones para un Fast Plan System Controller

ITER está definiendo un catálogo de soluciones estándar (Makijarvi 2014) que permitan implementar los diferentes sistemas requeridos en cada uno de los diagnósticos. El catálogo presenta diversas soluciones hardware para diferentes tecnologías y fabricantes, con el objetivo de estandarizar y facilitar la obtención de diferentes soluciones garantizando una serie de requisitos definidos por esta organización. Por ejemplo, además de las funcionalidades en cuanto a adquisición de datos se definen otros requisitos como la capacidad de realizar control en tiempo real de manera local, gobernar actuadores con alta precisión o intercambiar información con otros elementos del sistema. Para ello, las redes a las que podrá conectarse cada uno de estos controladores son las siguientes:

- PON: mediante esta red se realizarán tareas de mantenimiento frente a los servidores de software CCS. Además será la interfaz que comunique los distintos procesos de control sobre el hardware (EPICS IOC).
- TCN: es la red que permite la sincronización de los Fast Controllers con el resto de los componentes del sistema.
- DAN: permite almacenar los datos adquiridos en servidores de almacenamiento, además de permitir recuperarlos posteriormente para un análisis off-line de los mismos.
- SDN: se utiliza para la comunicación con el CODAC-HPC. Es una red de baja latencia y máxima disponibilidad, ya que será la responsable de cerrar el lazo de control del plasma.

Otro elemento que cobra especial importancia en el PS-I&C son los Interlock Controllers, que supervisan las medidas de seguridad para asegurar el sistema y evitar la pérdida del control del plasma. Si se diera una situación que ponga en peligro la integridad del sistema, se transmitiría una orden al resto de sistemas de las otras plantas para parar el sistema. El uso de controladores lógico programables (PLCs) no puede satisfacer las restricciones impuestas a la detección de estos eventos, entre 2 μ s y 100 μ s, y respetar una alta disponibilidad e integridad del sistema. Por ello se ha optado por la utilización de hardware embebido en FPGAs, eligiendo para ello sistemas CompactRIO del fabricante National Instruments.

2.2.2 Arquitectura Software

Dado que se trata de un proyecto internacional que involucra la actividad de científicos e ingenieros de cada una de las agencias domésticas nacionales integrantes, se hace necesario un conjunto de herramientas software que sean compatibles con los productos existentes en su catálogo hardware. Para ello se han basado en una distribución basada en Linux Red Hat Enterprise (RED HAT ENTERPRISE 2015), que ha nombrado como CCS (Codac Core System) y que se adapta de forma automática al sistema en el que se instale de acuerdo a su funcionalidad dentro de la arquitectura global. Esta estrategia además permite una fácil automatización del proceso de actualización de todo el software existente en todos los sistemas y se asegura un correcto control de versiones en todos los paquetes. En esta personalización se ha adoptado e incluido las siguientes herramientas:

- SDD: (Self Description Data configuration model) Está pensada para construir y configurar unidades software de forma automática. A partir de éstas se puede obtener los servicios y procesos necesarios para los PS-I&C (Abadie, Di Maio et al. 2012).

- EPICS: Es un conjunto de aplicaciones y herramientas que permiten obtener sistemas de control distribuidos. Es de código abierto y es ampliamente utilizado en muchos proyectos de carácter experimental. Utiliza un modelo de red cliente/servidor para interconectar todos los elementos del sistema (EPICS 2015b). En el siguiente epígrafe se realizará una descripción de los elementos que lo integran.
- CSS: (Control System Studio) Es un conjunto de paquetes software basadas en Eclipse, para poder crear interfaces de usuario que gestionen todo el sistema de control construido en EPICS.

2.2.3 EPICS.

En este apartado se va a hacer una breve descripción de este sistema de instrumentación y control, dado que se ha utilizado en los casos de uso realizados en la fase de experimentación y en la fase de evaluación de la metodología propuesta. Se explicarán los conceptos más relevantes de su arquitectura software y sus mecanismos de comunicación y funcionamiento.

Como se ha referido previamente, el software que lo constituye es de código abierto (EPICS 2015a) y es utilizado en grandes proyectos de física experimental a lo largo del mundo, como los aceleradores de partículas del PSI (Paul Scherrer Institute) (PSI 2015) en Suiza, o el sincrotrón perteneciente al proyecto Diamond (Diamond Light Source 2015) en Reino Unido. Aunque las primeras versiones sólo estaban soportadas por el sistema operativo VxWorks, en la actualidad existen versiones para Linux, OSX, Windows, Solaris o RTEMS.

Arquitectura de EPICS

Se puede definir como una arquitectura distribuida para construir sistemas de instrumentación y control escalables, basada en la estrategia cliente/servidor, utilizando los protocolos TCP/IP y UDP en una red de área local. Los elementos que integran el papel de servidores son los denominados IOC (Input/Output Controller), son los responsables de la comunicación con los dispositivos de adquisición, actuación e instrumentación de los sistemas de planta, utilizando diferentes interfaces de comunicación. Estos controladores de entradas/salidas son procesos software responsables de que todos los clientes tengan acceso al control y la información de un dispositivo hardware gobernado. Los clientes o

suscriptores utilizan un protocolo especial, definido como CA (Channel Access), para poder interactuar con el servidor. Los mensajes de comunicación están establecidos entorno a lo que los servidores publican como PVs (process variables), las cuales permiten controlar o monitorizar el estado del IOC. Todos los elementos mencionados, pertenecientes a la arquitectura, pueden observarse en la Figura 13.

Figura 13: Arquitectura de EPICS

Los clientes pueden interactuar mediante órdenes en líneas de comandos, o bien mediante un interface gráfico o paneles denominados como OPIs (Operator Interfaces). También es posible obtener clientes, que utilicen el protocolo channel access, en otros lenguajes de programación como LabView (disponible desde la versión 2011), C/C++, Java o Python. Dado que todos los elementos conviven en una infraestructura de red, es posible que los clientes se alojen en múltiples ubicaciones, e incluso en la propia máquina que resida el servidor. La Figura 14 muestra un posible escenario, en el que se gobierna una planta industrial, en el que conviven todos los elementos descritos.

Figura 14: Sistema de planta gobernado por EPICS

IOCs en EPICS

Estos procesos pueden ejecutarse en una misma máquina o en cualquiera de las pertenecientes a la misma red, y se encargan de interactuar con los dispositivos hardware o bien ejecutar procesos software de control, gestión o monitorización. La estructura interna de un proceso IOC, se muestra en la Figura 15.

Figura 15: Arquitectura Software de un proceso IOC

El mecanismo de funcionamiento de todos los subprocessos internos depende de la base de datos interna, en la que se encuentran los registros asociados a las PVs (process variables) controladas por el IOC. Estos registros tienen una estructura predeterminada, para la mayoría de los tipos de variables que pueden existir en un sistema de control (analógicas, digitales, imágenes, etc), aunque existen campos que el usuario puede personalizar. Los campos de uso frecuente en cada registro son los que determinan por ejemplo el tipo de dato a utilizar, las unidades de la medida realizada, el mecanismo de actuación sobre el dispositivo y la función que debe invocar el módulo Device Support sobre el hardware controlado.

```
record(ai, "Device:Vmax")
{
    field(DESC, "Tensión máxima")
    field(DTYP, "asynFloat32")
    field(SCAN, ".5 second")
    field(INP, "asyn(Device,0)Voltage")
    field(EGU, "V")
}
```

Figura 16: Registro en la base de datos del IOC

La Figura 16 muestra el contenido de un registro para una *process variable* asociada a una variable analógica que permite leer la tensión máxima de dispositivo 0, cada 0,5 segundos. Cualquier cliente que se comuniquen con el IOC que guarde en su base de datos esta información, tendrá acceso al valor en voltios que se lea del dispositivo.

Para gobernar el acceso a los dispositivos hardware, EPICS dispone de los módulos de Device Support y asynDriver (Figura 15). Para ello estos módulos interactúan con las librerías (C-API) específicas del dispositivo controlado. El interface creado por estos módulos permite definir y registrar funciones específicas para leer o escribir información del dispositivo. Estas funciones se ejecutan cuando una PV es procesada. El módulo asynDriver simplifica el acceso asíncrono a distintas interfaces estándar de comunicación (serie, Ethernet, Gpib, etc), estableciendo un camino directo para conectar las PVs con funciones básicas de manejo del dispositivo.

DPD (Data Process and Distribution).

En aquellas situaciones en las que, por razones de eficiencia, a la hora de adquirir y procesar información, el IOC debe residir en la misma máquina física, EPICS tiene una importante limitación. Para poder mejorar estas carencias mencionadas existe una propuesta construida para EPICS denominada DPD (Data Process and Distribution) (Castro, Vega et al. 2012) (Ruiz, Vega et al. 2012), encargada de optimizar las transferencias entre procesos del sistema, alojado en una misma máquina física. El sistema propuesto se puede observar en la Figura 17 en la que pueden existir varias unidades de adquisición y varias unidades de procesado, utilizados por distintos dispositivos, así como posibilidad de monitorización y configuración. También cabe destacar que a pesar de que puedan existir varios módulos gestionados, en el sistema se minimiza el uso de interrupciones y la carga computacional de la CPU.

Figura 17: Arquitectura software de DPD

Cada uno de los elementos funcionales tienen la misma arquitectura, permitiendo leer información de sus entradas, procesar datos internamente, generar nuevas señales o bien encaminar datos hacia el enlace de salida (Figura 18). Esta propuesta es además compatible y fácilmente integrable con el modelo asynDriver que permite una capa de abstracción añadida en el acceso al dispositivo. Los parámetros que permiten configurar la ruta de datos pueden utilizarse en tiempo de configuración o bien en tiempo de ejecución a través de distintas process variables (PVs), gobernando cada uno de los módulos existentes, concebidos como hilos dependientes del IOC. Cada elemento funcional aloja parámetros que permiten obtener distintas medidas de throughput, así como información acerca del nivel de ocupación de los buffers de memoria utilizados en los enlaces. Dichos buffers de memoria admiten posibilidad multinivel (memoria o disco), lo que significa una alta capacidad de los enlaces, cuando las necesidades lo requieran.

Figura 18: Elemento funcional en DPD

2.2.3.1 Necesidades de procesamiento en ITER.

Conocidos los estándares definidos por ITER, se hace necesario extraer las necesidades de procesamiento especificadas en la definición de los diagnósticos del *Plan Control System* (PCS) (Winter, Makijarvi et al. 2012). Para satisfacer dichas necesidades es necesario disponer de sistemas de adquisición de datos con procesamiento local, que utilicen máquinas dedicadas con distintas tecnologías. Las especificaciones de rendimiento y la naturaleza de cada uno de los diagnósticos llevarán a seleccionar una de las posibles arquitecturas de proceso existentes.

De la definición establecida para los *Fast Plant System Controller* se puede adivinar que dichas plataformas son: CPU's multicore, FPGAs y GPUs. Existen experiencias previas que aportan soluciones de procesamiento para sistemas de adquisición de datos basados en arquitecturas multicore (Giannone, Schneider et al. 2009), y otros también basados en plataformas PXI escalables (Barrera, Ruiz et al. 2005), que permiten realizar procesamiento en CPU para sistemas distribuidos.

Los diagnósticos con restricciones temporales más severas (incluyendo sistemas de adquisición con altas tasas de muestreo con la mayor resolución posible) se prestarán a ser gobernados por una alternativa basada en FPGAs. Además la normalización del uso de este tipo de arquitecturas para la comunidad de fusión (Sanz, Ruiz et al. 2013), hace que el esfuerzo de integración de esta tecnología sea cada vez menor. Otra ventaja añadida es la experiencia acumulada de su utilización para distintos dispositivos, y para distintas arquitecturas hardware (Carvalho, Batista et al. 2010).

La plataforma hardware más apropiada, en aquellos diagnósticos en los que existan un elevado número de canales a procesar, o bien exista la necesidad de trabajar con imágenes, son las GPUs. Aunque existen trabajos previos que permiten avalar la afirmación anterior ((X N Yue and B J Xiao and Z P Luo and, Y. Guo 2013), se hacen necesarios los estudios de experimentación y análisis planteados en el capítulo 3 de esta tesis. los cuales permitirán establecer su validez de utilización y su capacidad de integración en los sistemas de adquisición y procesado propuestos por ITER.

2.3 GPUS (GRAPHICS PROCESSING UNIT).

2.3.1 Introducción.

El aprovechamiento de las GPUs en el ámbito científico para incrementar la capacidad de cálculo es un hecho que puede encontrarse en la mayoría de las ciencias en investigación, como por ejemplo la medicina, la biología o la astronomía. Además los centros de alta computación (HPC) de referencia mundial (Titan - Cray XK7 en el Oak Ridge National Laboratory de Tennessee, USA), incorporan en su estructura clusters de proceso constituidos por este tipo de elementos. El interés despertado en la comunidad de fusión ha permitido que se hayan realizado varios trabajos orientados a validar el uso de esta tecnología en este entorno ((X N Yue and B J Xiao and Z P Luo and, Y. Guo 2013), (Chiariello, Formisano et al. 2013b). Siguiendo esta línea de interés, ITER ha seleccionado el uso de GPUs como elemento de cálculo y proceso para algunos de los diagnósticos que más carga computacional necesitan. Entre ellos se encuentran los relacionados con imágenes, realizando compresión de las mismas en tiempo real, o aquellos relacionados con sistemas de adquisición de datos de múltiples canales, como por ejemplo los sensores magnéticos.

De las opciones existentes para esta tecnología, ITER ha seleccionado al fabricante NVIDIA, y la familia profesional Quadro basada en la generación Kepler. Aunque existen otros fabricantes, como Intel o AMD ATI, que también cuentan con tecnologías de procesamiento en GPU, es NVIDIA la que más apoya y lidera con predominancia este sector. Muestra de ello es la cantidad de trabajos científicos existentes, su utilización en experimentos internacionales de referencia, así como la existencia de un congreso científico internacional anual, GPU Technology Conference (NVIDIA 2015), en el que se exponen los últimos avances del fabricante y trabajos en los que se exponen los beneficios de uso de las GPUs.

Al igual que en el ámbito de la CPU, el incremento de las prestaciones ha ido acompañado con el incremento de la frecuencia de reloj, en el ámbito de las GPUs el incremento de prestaciones se ha conseguido aumentando el número de cores o procesadores disponibles en cada nueva generación. Desde los 128 cores disponibles en la arquitectura inicial NVIDIA G80 en el año 2006, se ha pasado a los 3072 que incorpora cualquier tarjeta basada en la arquitectura actual NVIDIA

Maxwell. Este aumento permite que con cada nueva arquitectura se incremente la capacidad de cálculo, que en computación suele medirse en xFLOPS¹. La mejora constante en este ámbito también conlleva la actualización de este producto a nuevos estándares de comunicación, como es el uso de PCIexpress 3.0 en las últimas tarjetas, lo cual permite obtener tasas de transferencia para acceder a la memoria de la GPU, cercanas a 1 GB/s por cada una de las líneas utilizadas de dicho interface (*PCIe lane/link*).

Antes de exponer el estado actual de esta tecnología, y con el fin de justificar la capacidad de proceso de propósito general, a continuación se describe la arquitectura hardware de las GPUs, así como se exponen las distintas formas de obtener aplicaciones que realicen procesamiento paralelo.

2.3.2 Arquitectura Hardware.

Una GPU no puede trabajar como un elemento aislado, sino que necesita convivir en una arquitectura con otros elementos como una CPU, la memoria principal del sistema, controladores y concentradores de buses de comunicaciones e incluso otros procesadores gráficos. Aunque en la actualidad la tendencia es integrar la GPU en la propia CPU, cuando se necesitan procesadores de gráficos o de cálculo de altas prestaciones es necesario recurrir a tarjetas dedicadas que se comunican con el sistema principal utilizando el bus PCIexpress. La distribución conceptual de interconexión de todos los elementos en un sistema basado en arquitectura Intel monoprocesador puede observarse en la Figura 19.

¹ FLOPS: número de operaciones en coma flotante por segundo, x:M(10⁶),G(10⁹),T(10¹²)

Figura 19: Diagrama interno de conexión de una GPU en un Sistema con CPU Intel

Esta distribución es dependiente de una arquitectura en constante evolución de tal forma que el bus PCIexpress puede obtenerse bien a través de concentradores (*switches*) dependientes del propio controlador (*Iohub*), o incluso puede existir un número limitado de líneas del bus gobernadas por el propio procesador. En cualquiera de las disposiciones posibles la GPU está conectada en un slot PCIexpress utilizando el mayor número de líneas posible.

Un sistema híbrido con el expuesto en la Figura 19, dispone de más de un espacio de memoria. De hecho la GPU tiene su propio espacio interno de memoria, independiente del espacio gobernado por la CPU. Este hecho implica que exista un constante movimiento de datos a través del bus PCIexpress, en el que además pueden coexistir otros dispositivos.

Arquitectura interna de procesadores.

NVIDIA propone en el año 2006, una solución hardware y software a todos aquellos que quieran utilizar una GPU para computación de propósito general. El término acuñado para esta nueva tecnología lo denomina CUDA (Computer Unified Device Architecture), en la que se propone una arquitectura hardware con una serie de herramientas software para obtener aplicaciones de computación paralela. Hasta ese momento si un científico quería ejecutar algoritmos de cálculo, era necesario basarse en el API gráfico, por lo que era una tarea harto compleja. Por otra parte también propone estandarizar una

arquitectura hardware que sea vista por el programador como un conjunto de procesadores que pueden trabajar en paralelo con múltiples datos. Otra ventaja adicional es que en este modelo, cualquier procesador tiene acceso a toda la memoria y además se evitan cuellos de botella existentes en arquitecturas precedentes.

NVIDIA TESLA es la primera arquitectura que cumple estas premisas. Su diagrama interno puede verse en la Figura 20. La organización interna de computación está construida entorno a lo que se denominan multiprocesadores o streaming multiprocessors. En esta familia existen 8 elementos de este tipo y cuenta con distintos elementos de control para la ejecución de múltiples hilos, al margen de disponer de memoria compartida para cada todos los procesadores que contiene.

Figura 20: Arquitectura interna de una GPU Tesla

Cada multiprocesador contiene a su vez otros 8 procesadores o streaming processors, que son la unidad básica de ejecución, también denominado como

cuda core. Tienen la capacidad de realizar operaciones aritméticas de enteros y punto flotante bien para gráficos, bien para cálculos de propósito general. Este procesador será el responsable de la ejecución de las funciones kernel (funciones nativas de código GPU) que se ejecutarán concurrentemente a lo largo de múltiples hilos con datos diferentes. En la Figura 21 puede observarse la disposición interna de cada uno de los multiprocesadores de la familia TESLA.

Figura 21: Composición interna de un multiprocesador CUDA.

Modelo interno de memoria.

Además del uso como elemento de cálculo paralelo, el objetivo inicial de las GPUs es su utilización para aplicaciones gráficas. Dichas aplicaciones requieren un uso intensivo de datos con una alta demanda de tráfico de información. En las mismas es necesario procesar muchas instrucciones para determinar las características de representación de cada pixel y objetos existentes en cada escena de una imagen. Teniendo en cuenta este uso intensivo, el subsistema de memoria de una GPU se construye: con un bus lo más ancho posible para maximizar las transferencias, y con la mayor velocidad de reloj que permita una tasa de transferencia elevada, además de establecer una jerarquía que maximice su uso. Este diseño es aprovechado por las aplicaciones de cálculo paralelo, que también demandan un uso intensivo de datos.

Los distintos espacios de memoria que propone el modelo CUDA se describen brevemente a continuación.

- Memoria Local y Global. Estos dos tipos de memoria están localizados en memoria DRAM, fuera de cualquier SM. Cualquier hilo en ejecución tiene acceso a memoria global, por lo que es necesario establece técnicas que preserven la integridad de los datos contenidos en cada posición. Además cada grupo hilos en ejecución tiene asignado un bloque de memoria, lo cual implica que se penalizará el acceso a la memoria asignada a otro bloque. Esta es un ejemplo de las optimizaciones del acceso a la memoria cuando quieren obtener aplicaciones eficientes para GPUs. Por el contrario la memoria local es únicamente visible para cada uno de los hilos en ejecución, y por tanto el tamaño máximo utilizado vendrá determinado por el número máximo de hilos en ejecución.
- Memoria constante y memoria de texturas. Ambas son de solo lectura, localizadas también en memoria DRAM y están pensada para contener valores escalares y grandes arrays de solo lectura, datos muy utilizados en operaciones gráficas.
- Memoria compartida y memoria caché. Las memorias descritas anteriormente están ubicadas fuera de cualquier procesador, por lo que su acceso requiere de más tiempo que las memorias pertenecientes a cada procesador. Sin embargo esta memoria reside en SDRAM y es visible por cada grupo de SM. Tiene un ancho de banda mucho más alto que las anteriores. Por último la memoria caché es la más próxima a los procesadores, se utiliza para cálculos intensivos de operaciones en pocos ciclos de reloj y es utilizada directamente por el hardware.

2.3.3 Modelo de programación en lenguaje C-CUDA.

La obtención de aplicaciones, bajo el estándar CUDA, se basa en una estructura de capas que en su nivel más bajo se apoya en un driver, un runtime, y una serie de librerías (por ejemplo cuBLAS, cuFFT, etc) que implementan funciones de un extenso uso en computación de propósito general. Como se puede observar en la Figura 22, el usuario puede construir aplicaciones interactuando con cualquiera de estas capas utilizando distintas funciones provistas en el System Development Kit (NVIDIA-SDK 2015). La aplicación se ejecutará en la CPU, y utilizará

funciones específicas, bien de las librerías, del runtime o del kernel, cuando pretenda acceder a recursos propios de la GPU. El usuario también puede crear funciones de proceso nativas (*kernel*), y este caso su ejecución se realizará en los núcleos de los procesadores seleccionados.

Desde el punto de vista del programador la GPU puede verse como un conjunto de M multiprocesadores que contienen N cores, de tal forma que es posible la ejecución simultanea de MxN instancias de una única función kernel que opera con datos diferentes realizando la misma tarea.

Figura 22: Arquitectura software en CUDA

El compilador de lenguaje C/C++ que incorpora CUDA para obtener aplicaciones que se ejecutan en una GPU, es NVCC. Realmente es un compilador que se basa en el apoyo de compiladores externos como gcc, e incorpora distintas extensiones que permiten compilar código nativo para la GPU o funciones kernel. Estas funciones se pueden escribir en lenguaje ensamblador ligado a la arquitectura de las GPU, denominado PTX, aunque es más efectivo su desarrollo en un lenguaje de alto nivel como C.

La aplicación obtenida, que se ejecutará en un hilo de la CPU, se basa en el uso del runtime de CUDA para poder interactuar con los recursos internos de la GPU. De esta forma, por ejemplo, se puede realizar el manejo de la memoria interna, o la carga y ejecución de las funciones nativas de la GPU.

Ejemplo básico de ejecución.

La idea básica del incremento de rendimiento que se puede tener, cuando se ejecuta un determinado proceso en una CPU o en una GPU, puede verse en la Figura 23. Para sumar dos vectores de 512 componentes, en la CPU se ejecutarían sumas en serie, dando como tiempo total de cómputo la suma de todas las operaciones con cada componente. En una GPU dado que se puede ejecutar simultáneamente múltiples hilos con cada dato, el tiempo total del cómputo responde al tiempo utilizado para una sola operación.

Figura 23: Comparativa de proceso en CPU y en GPU

Realizar una aplicación para GPUs implica conocer la arquitectura anteriormente descrita y tener en cuenta que es un sistema que debe considerarse como un elemento de cooperación o esclavo de la CPU. Por ello será necesaria la existencia de un hilo en la CPU que secuencie las operaciones que a continuación se describen de manera simplificada.

Figura 24: Flujo de operaciones para procesar en GPU

1. Desde la memoria de la CPU se envía la información a procesar a la memoria global de la GPU. Para ellos se utilizan transferencias por DMA utilizando el bus que comunica la memoria de ambos sistemas, PCIexpress.
2. Desde la CPU se lanza la ejecución paralela del código (en funciones *kernel*) por parte de la GPU.
3. Con el fin de aprovechar todos los recursos disponibles, se ejecutarán tantos hilos como procesadores disponga la arquitectura.
4. Los datos procesados son de nuevo copiados, esta vez desde memoria de la GPU a la CPU.

Como puede observarse, en un esquema de este tipo, es fundamental minimizar el trasiego de datos entre CPU y GPU, ya que existen dos copias de datos que pueden penalizar el rendimiento. Para resolver esta situación en las últimas versiones de GPUs se ha implementado la copia asíncrona de datos, que permite mover información a la vez que se está procesando.

•

2.3.4 Programación de una GPU en otros lenguajes.

Para obtener aplicaciones que saquen provecho a las capacidades computacionales de una GPU, existe un amplio abanico de lenguajes de programación, unos de alto nivel con cierta capacidad de abstracción en el uso

del dispositivo, y otros de bajo nivel mucho más dependientes de la arquitectura hardware, lo cual permite maximizar el rendimiento y control de las aplicaciones obtenidas. A continuación se describen brevemente las opciones más relevantes, disponibles por el desarrollador de aplicaciones, teniendo en cuenta que la mayoría son soluciones comerciales de pago, a diferencia de las herramientas suministradas por NVIDIA que son de libre distribución.

- Matlab. Desarrollado por el fabricante MathWorks(Parallel Computing Toolbox 2015), permite utilizar un toolbox para realizar funciones de álgebra lineal con vectores y matrices. También permite la realización de funciones de filtrado, FFT o procesamiento de imágenes que se ejecutan directamente en la GPU. Es posible invocar directamente funciones *kernel* de código nativo CUDA.
- LabView. Mediante el uso del Analysis Toolkit(LabVIEW GPU Analysis Toolkit, NI 2015), es posible comunicarse, interactuar y ejecutar código en una GPU. Las funciones provistas en este kit de desarrollo añaden la capacidad de utilizar las librerías de álgebra lineal (BLAS) o transformadas rápidas de Fourier (FFT), así como funciones *kernel* compiladas con las herramientas de NVIDIA mediante el uso de librerías de enlace dinámico (DLL) en MS Windows.
- OpenAcc. Es un lenguaje de programación basado en definiciones y directivas para poder convertir código C, C++ o Fortran a código paralelo para GPUs. Este estándar creado por varios fabricantes como NVIDIA y PGI (OpenACC Compiler 2015), incorpora un compilador que traslada código que se ejecuta en CPU a código paralelo de aquellas zonas que más peso computacional tienen, y que el desarrollador selecciona mediante directivas.
- OpenCL. Lenguaje creado por Apple en 2008, con un modelo bastante similar de ejecución a CUDA. La ventaja de este lenguaje es la posibilidad de ejecutarse en otras plataformas hardware como GPUs del fabricante AMD, CPUs Intel y AMD, FPGAs de Altera o procesadores ARM.

El conjunto de herramientas que propone NVIDIA, de forma gratuita, es el CUDA System Development Kit (NVIDIA-SDK 2015), y está disponible para los sistemas operativos Windows, Linux y Mac-OSX. Además de añadir un nuevo compilador para lenguaje C-C++ para código paralelizable, añade diferentes herramientas de depuración y desarrollo, y cuenta con distintas librerías

optimizadas para procesamiento de imágenes, video, álgebra lineal o procesamiento matemático. Existe una constante actualización de este Kit, para incorporar mejoras, o bien para adaptarse a las nuevas funcionalidades que aportan cada nueva generación de GPUs que el fabricante lanza al mercado.

2.3.5 Familias de GPUS NVIDIA

Casi cualquier dispositivo electrónico en la actualidad incorpora una GPU: teléfonos móviles, tabletas, ordenadores, televisores inteligentes, etc. NVIDIA, fabricante dominante de este mercado, dispone de varias familias de dispositivo en función del segmento al que van destinados. Las tarjetas orientadas a ser utilizadas en ordenadores de escritorio o portátiles, reciben la denominación GeForce, y está optimizadas para procesar gráficos. Aunque son tarjetas pertenecientes al mercado de consumo, permiten realizar procesamiento de cómputo general. En este segmento del mercado el precio de cada tarjeta difiere en función de sus prestaciones para proporcionar mejores reproducciones de imágenes y videos, situándose en unos precios que pueden oscilar desde decenas de euros hasta los mil, que puede llegar a costar la mejor tarjeta de estas series en la actualidad, GTX Titan X.

Existen otro grupo de tarjetas orientadas al uso profesional, denominadas Quadro y Tesla. La primera tiene altas capacidades para procesar gráficos y utilizar varios monitores simultáneos con altas resoluciones, la segunda únicamente se utiliza para procesar. En este caso el precio a pagar por estas series arranca en los mil euros y puede llegar hasta los cuatro mil para el caso de las tarjetas más potentes de la familia, en concreto el modelo k80, perteneciente a la arquitectura Kepler. Además la serie Tesla, permite su uso en sistemas multi-gpu o grid, para ser utilizadas en sistemas HPC(Mare Nostrum SBC Barcelona, 2015).

Por último, el fabricante está apostando por procesadores móviles que incorporan una GPU, orientados a sistemas embebidos. Conforman la familia Tegra y disponen de un potente procesador ARM y una GPU de la plataforma Kepler, para la que se pueden desarrollar aplicaciones CUDA(mCOM10-K1 General Electric, 2015). Los precios contenidos que tiene esta plataforma, en torno a los cien euros, hace que sea una opción muy atractiva para desarrollos de sistemas a medida autónomos, con altas capacidades de procesamiento.

Desde la aparición de la arquitectura Tesla hasta el presente, finales de 2015, se han ido sucediendo distintas arquitecturas que han ido aportando más funcionalidades y tecnologías que mejoran el rendimiento del primer modelo CUDA. En la Figura 25 extraída de la conferencia que ofreció el CEO de la compañía NVIDIA, Jen Hsun Huang, en la GPU Tech Conference de 2015, se puede observar la evolución que ha tenido esta tecnología, así como la evolución hacia nuevos productos.

Figura 25: Evolución de las distintas arquitecturas NVIDIA GPU

En la Figura 25, se puede destacar que además de incrementar la capacidad de proceso, incluyendo más cores en cada nueva generación, la ratio DP GFLOPS/Watt (operaciones de doble precisión por vatio) ha ido incrementándose, lo cual implica un rendimiento más eficiente, redundando en un menor consumo energético. En la Tabla 2, se pueden ver las características más relevantes de las tres generaciones de GPUs utilizadas a lo largo de esta tesis.

	Modelo	CUDA cores	GPU clock	GFLOPS(SP)
Tesla	C1060	240	1300	933
Fermi	GTX580	512	1150	1288
Kepler	K20	2496	706	3520

Tabla 1: Características principales de las tres generaciones de GPU utilizadas

Tomando como punto de partido la familia Tesla, primera en adoptar el estándar CUDA, la aparición de la familia Fermi propuso, además de un incremento de 32 cores por SM (Streaming Multiprocesor), un nuevo sheduler de mayor jerarquía (GigaThread) que mejora notablemente la gestión y distribución entre todos los procesadores disponibles. De la generación más reciente, Kepler, cabe destacar la utilización del bus PCIexpress 3.0, la posibilidad de lanzar funciones *kernel* desde la propia GPU (Dynamic Parallelism) o la capacidad de realizar transferencias DMA entre otros dispositivos y la memoria de la GPU (GPUDirect RDMA).

El uso de tres arquitecturas (Tesla, Fermi y Kepler), a lo largo del desarrollo de esta tesis, da una idea de la constante actualización de esta tecnología. No obstante se ha mantenido el modelo propuesto por CUDA, lo cual permite establecer que los resultados y conclusiones obtenidas en la fase de experimentación, abordados en el próximo capítulo, son fácilmente extrapolables a nuevas arquitecturas.

Las mejoras introducidas por los fabricantes también hacen que sea necesario actualizar las tecnologías GPUs, utilizadas por la comunidad de fusión. Este hecho se refleja en los requerimientos que ITER propone como sistemas de procesado en sus *fast controllers*. Si en un primer momento de la especificación, en el catálogo figuraba el uso de la arquitectura Fermi, en la actualidad se refleja la necesidad de mayor capacidad de procesado utilizando la arquitectura Kepler. Esta adopción posibilita además la propuesta y metodología que se realizará en el capítulo 4 de esta tesis.

Capítulo 3

3 EXPERIMENTACIÓN: ANÁLISIS DE ALTERNATIVAS

3.1 Gestión y manejo de GPUS desde herramientas de alto nivel utilizadas en iDAQS.

3.1.1 Introducción.

Las herramientas y lenguajes de programación utilizados por los científicos en algunas de las fases que permiten desarrollar y validar los modelos y las arquitecturas de los diagnósticos, son de alto nivel (Manduchi, De Marchi et al. 2013). Se utilizan herramientas como LabVIEW o Matlab, que permiten desarrollar aplicaciones con un esfuerzo menor que el que requiere el uso de lenguajes tradicionales como C, C++ o Fortran.

En este caso de uso, uno de los objetivos propuestos, ha sido explorar y evaluar vías de integración de procesamiento en GPUs, en este tipo de herramientas. En concreto se ha elegido LabVIEW bajo sistema operativo MS Windows, que tiene un amplio uso en sistemas de adquisición de datos (Giannone, Eich et al. 2011). Otro de los objetivos perseguido ha sido valorar si se consigue mejorar el rendimiento de la ejecución de aquellos procesos que son más costosos computacionalmente. Para poder realizar esta valoración ha sido necesario partir de un algoritmo conocido desarrollado íntegramente en LabVIEW.

En la ejecución de varios proyectos del Plan Nacional de I+D+i, varios profesores investigadores del Grupo de Investigación en Instrumentación y Acústica Aplicada (I2A2) de la Universidad Politécnica de Madrid, junto a investigadores del Grupo de Adquisición de Datos del Laboratorio Nacional de Fusión por Confinamiento Magnético (CIEMAT), diseñaron una plataforma hardware/software de sistemas de inteligentes de instrumentación. Esta plataforma (ITMS) está orientada a la obtención de sistemas de adquisición y procesamiento inteligente de datos, aplicados a diagnósticos empleados en entornos de fusión termonuclear por confinamiento magnético.

En concreto uno de los sistemas diseñados se utilizó en el reflectómetro de correlación KG8B en JET (de Arcas, López et al. 2008). El sistema de adquisición de datos se diseñó con la finalidad de reducir la cantidad de datos almacenados tras una descarga, procedentes de dicho reflectómetro. Sobre una plataforma hardware COTS basada en PXI esquematizada en la Figura 26, se realizaron

distintos ensayos encaminados a obtener un método que permite adaptar la frecuencia de muestreo en función del ancho de banda de la señal analizada.

Figura 26: Arquitectura hardware del Sistema de adquisición utilizado en el reflectómetro KG8B de JET

Dicho sistema cuenta con una tarjeta de adquisición NI PXI-6115 que adquiere cuatro señales analógicas con una frecuencia fija de muestreo máxima de 10 MS/s. También cuenta con un ordenador embebido, NI PXI-8105, responsable de gobernar la adquisición y realizar tareas de procesamiento local. Por último el sistema dispone de un sistema de almacenamiento basado en una configuración RAID0 utilizado para almacenar los datos adquiridos.

La arquitectura software del sistema descrito es mostrada en la Figura 27. Permite adquirir información a la máxima frecuencia de muestreo posible y transferirla a buffers de memoria, en los que se aplica un factor de diezmado, optimizado de acuerdo al ancho de banda de cada bloque. La aplicación, que se ejecuta en el controlador embebido, ha sido desarrollada en su totalidad utilizando LabView 8.2.1.

Figura 27: Diagrama software del sistema de adquisición de datos con frecuencia de muestreo auto-adaptativa

Se evaluaron distintas alternativas para poder estimar qué información de la adquirida era relevante. Estas alternativas dieron lugar a distintos algoritmos que permitían conocer el ancho de banda óptimo, en el que se encontraba la información relevante que se debía almacenar. Uno de ellos, NOCOFE (*NOise CORner Frequency Estimation*), es el que se eligió como ejemplo, para este caso de uso, y evaluar los objetivos previamente mencionados.

3.1.2 Descripción de la propuesta.

Para determinar si los objetivos propuestos podían cubrirse, se desarrollaron dos aplicaciones. Una cuya ejecución se realiza únicamente en la CPU y otra que tiene parte de su ejecución en GPU. En ambos casos la información proviene de ficheros que almacenan datos previamente adquiridos de una descarga en el plasma, conformando una plataforma susceptible de realizar múltiples ensayos.

En el entorno de desarrollo seleccionado no existe de manera nativa, la ejecución de código en GPUs, por ello es necesario el uso de un conjunto de herramientas añadidas que constituyen el LabVIEW GPU Analysis Toolkit (LabVIEW GPU Analysis Toolkit, NI 2015). El conjunto de funciones que aporta este kit, constituyen una capa software o *wrapper*, que actúa como interface de forma que desde la aplicación que se ejecuta en la CPU se puedan derivar cálculos complejos hacia la ejecución de código en paralelo en la GPU.

El Toolkit incluye diversas funciones que permiten gestionar la comunicación con una GPU agrupadas en las siguientes categorías: creación y destrucción de manejadores de contexto, reserva y liberación de memoria, y funciones de movimientos de datos entre CPU y GPU. Además de lanzar la ejecución de funciones nativas *kernel*, existe la posibilidad de utilizar librerías optimizadas, pertenecientes a CUDA, como pueden ser: cuFFT (funciones para trabajar con transformadas rápidas de Fourier) y cuBLAS (funciones de operaciones de álgebra lineal).

Para construir una aplicación en LabVIEW que acceda a una GPU mediante funciones básicas de CUDA se ha de completar el siguiente ciclo de ejecución:

1. Crear un contexto de ejecución o manejador de un dispositivo CUDA existente en el sistema.
2. Determinar y reservar los bloques necesarios de memoria en la CPU y en la GPU, inicializando dichas zonas.
3. Transferir la información desde la memoria del host hasta la memoria interna de la GPU.
4. Invocar a las funciones de proceso que operen con los datos previamente almacenados en memoria global de la GPU. Estas funciones pueden pertenecer a la paleta de funciones de procesado que utilizan funciones de librerías CUDA, o bien funciones personalizadas de acuerdo a la naturaleza del proceso.
5. Mover la información resultante del proceso desde memoria de la GPU a memoria de la CPU o host.
6. Liberar los recursos reservados previamente y cerrar el contexto seleccionado en la GPU.

En la Figura 28 puede observarse un diagrama que resume el ciclo descrito anteriormente, cuando se realiza en la GPU el procesado de una FFT de múltiples canales.

Figura 28: Fases necesarias en la ejecución de aplicaciones en GPU desde LabVIEW

En dicho esquema todas las operaciones entre CPU y GPU se realizan síncronamente, es decir no se inicia una etapa hasta que no ha concluido la anterior. La única excepción es la invocación a las funciones embebidas en la librería personalizada que se ejecuta en la GPU. En este caso el proceso de la CPU lanza la ejecución de forma asíncrona, por lo que no espera a que este código termine de ejecutarse para pasar a la siguiente acción programada.

Esta condición permitiría que el procesamiento pudiera ser cooperativo entre ambos sistemas, utilizando técnicas que balacearan dicha carga computacional, pero esta posibilidad no ha sido tomada en cuenta en esta aproximación. En cualquier caso la función de movimiento de datos desde la GPU a la CPU, permite que la información fluya de manera síncrona y secuencialmente, ya que hace de barrera y espera a que la información esté completamente procesada.

Un factor que podría penalizar el rendimiento en este esquema es la gran cantidad de información que ha de fluir entre la memoria del host y la memoria de un dispositivo CUDA. En este caso, en el que se utilizó una tarjeta NVIDIA Tesla C1060 que utiliza 16 líneas del interface PCIexpress 2.0, y dado los tamaños de bloque que se utilizaron no representó ningún problema.

De la fase 4, descrita anteriormente se desprende que, para poder derivar el cálculo hacia la GPU y sacarle el máximo provecho, es necesario dedicar los esfuerzos de programación a funciones y librerías que implementen el algoritmo elegido. Para poder comunicarse con una librería personalizada, LabVIEW

utiliza el estándar de encapsulamiento DLL (Dynamic Link Library), cuando se trabaja bajo sistemas operativos Windows. El medio para obtener dicha librería es utilizar como lenguaje de programación C o C++, utilizando el compilador y herramientas suministradas en el SDK de CUDA. En este nivel de programación es posible acceder a todas las capacidades de computación de la GPU, mediante la adecuada gestión de sus recursos (procesadores y sistema de memoria) invocando funciones de código nativo *kernel* personalizadas, e incluso enlazando librerías optimizadas ya existentes, que permitan realizar por ejemplo cálculos de álgebra lineal o procesamiento de imágenes.

Figura 29: Etapas del algoritmo NOCOFE

La Figura 29 se presenta un diagrama de las etapas de las que consta el algoritmo NOCOFE para la estimación del ancho de banda de una señal que contiene información relevante. A continuación se describen las particularidades y estrategias llevadas a cabo en la consecución de cada una de ellas.

1. Powerspectrum. Permite obtener toda la información de la energía de la señal a analizar utilizando la transformada rápida de Fourier. Se podría haber optado por implementar esta función matemática utilizando funciones personalizadas *kernel*, pero dado que se ofrece de manera nativa en el SDK de NVIDIA se han elegido las funciones de la librería cuFFT que están optimizadas para arquitecturas CUDA. Una vez obtenida la transformada, se obtiene la potencia el espectro de la señal de acuerdo a la ecuación (1).

$$S_{xx} = \frac{F\{x\}^* F^*\{x\}}{n^2} \quad (1)$$

En la que $F\{x\}$ es la FFT de la señal y $F^*\{x\}$ su conjugada, siendo n el número de muestras de la señal x . Esta operación debe realizarse para cada uno de los valores de salida, de los componentes del espectro entre 0 y π , obtenidos al realizar la FFT. Para realizar esta operación se ha desarrollado una función personalizada *kernel*, que aprovecha al máximo todas las capacidades de computación de la GPU. De esta forma se distribuyen sendas operaciones por todos los procesadores disponibles, en las que se computan las operaciones básicas señaladas en la expresión anterior, para cada una de las muestras obtenidas.

2. Normaliza-DB Scale. En esta etapa el objetivo es obtener un espectro de potencia normalizado. Como es de esperar no existen funciones de librería que permitan realizar estos cálculos, por lo que ha sido necesario también crear una función *kernel* que dé soporte a estas operaciones. En la parte inferior de la Figura 30 se muestra el código correspondiente a esta función. En la parte superior de la misma figura puede observarse el código que se ejecuta en el hilo de la CPU y que va secuenciado todas las fases del algoritmo.

Figura 30: Código GPU invocado desde un hilo de ejecución de la CPU

En el hilo de la CPU, parte superior, se realizan llamadas a las funciones que se ejecutan en la GPU, mediante la extensión que añade el compilador nvcc del SDK de NVIDIA:

`normalizespectrum <<<Blocks, ThreadsPerBlocks>>>`

en donde Blocks representa el número de multiprocesadores que ejecutan la función y ThreadsPerBlock indica el número de hilos que se lanzan en cada multiprocesador. Estas variables son dependientes de la arquitectura de la GPU utilizada y deben ser personalizados, para cada una de ellas, con el fin de obtener el máximo aprovechamiento de todos los recursos disponibles.

3. Filter. El objetivo de esta etapa es obtener un suavizado de la señal obtenida tras la normalización. De esta forma es posible realizar en la siguiente etapa búsquedas más coherentes y consistentes cuando se analizan espectros similares. La elección del tipo de filtro elegido condicionará el resultado final en la mejora del rendimiento global del

algoritmo, en función de la posibilidad de paralelización de su estructura. Se realizaron distintos ensayos, en primer lugar con un filtro IIR (filtro de respuesta infinita al impulso), desechando rápidamente esta opción dado que para cada valor en la señal de salida, es necesario acumular varios valores previos así como otras salidas. Por esta razón se eligió un filtro FIR (filtro de respuesta finita al impulso) de 30 coeficientes, cuyo algoritmo permite un alto grado de paralelización ya que sus salidas solo dependen de ciertos coeficientes y de los valores de entrada.

4. Search. En esta etapa se calcula, para todo el ancho de banda de la señal, el punto en el que el resultado de la señal filtrada coincide con el nivel de ruido, utilizando dos pasos. Primero se calcula el nivel de ruido como la media más el 20% de la desviación estándar de todo el espectro de potencia. Después se obtiene la frecuencia “corner” que alcanza dicho valor, buscando en el espectro de potencia filtrado. Aunque el primer paso si es paralelizable dividiendo en secciones los cálculos, el segundo paso, no es posible realizar una función *kernel* optimizada, debido a la linealidad de la operación de búsqueda.
5. Decimate. La última etapa es la reducción de los valores de las señales de entrada de acuerdo al ancho de banda calculado. Se han desarrollado funciones *kernel* al igual que en pasos precedentes.

3.1.3 Evaluación de la solución.

Conseguida la integración perseguida mediante la metodología expuesta anteriormente, es necesario valorar y evaluar si aporta una mejora en el rendimiento del sistema de partida. Para ello es necesario realizar medidas pormenorizadas de los tiempos de ejecución de todas las etapas del algoritmo, en cada uno de los dos escenarios utilizados, incluyendo la dependencia de éstos con la magnitud de los datos procesados.

Para poder medir dichos tiempos es necesario introducir trazas a lo largo de la ejecución del algoritmo. Existen dos posibilidades basadas en el uso de timers: funciones de librerías estándar de C que utilizan la base de tiempo del reloj de la CPU, o bien funciones del API de CUDA que utilizan como base de tiempos el reloj de la GPU. Se ha optado por la segunda posibilidad, ya que ofrece una mayor resolución en la medida, y según consta en la documentación de NVIDIA es de 0,5 μ s.

La primera evaluación que se ha realizado es determinar el tiempo de ejecución de cada etapa, escalando las medidas a diferentes tamaños del bloque de datos procesado. La Tabla 2 muestra dichos resultados, y en la misma se observa que la operación “Search” es en la que más tiempo se invierte, dado el bajo grado de paralelización asociado a dicho proceso.

Tamaño bloque datos /canal	Power Spectrum(ms)	NormalizedB Scale(ms)	Filter(ms)	Search(ms)	Decimate(ms)
16kS	0.936	0.615	0.110	1.988	0.31
32kS	0.945	0.677	0.176	3.438	0.62
64kS	0.958	0.776	0.320	6.227	1.20
128kS	1.196	0.974	0.620	11.715	2.39

Tabla 2: Tiempos de ejecución en las etapas del algoritmo NOCOFE.

Es necesario hacer notar que puede darse el caso que al intentar optimizar cierto algoritmo para su ejecución en paralelo en una GPU, puedan obtenerse resultados peores que en la misma implementación utilizando librerías nativas. Como se ha visto un factor determinante, es la naturaleza del algoritmo, y otro

muy relevante es el tamaño del bloque de los datos procesados. Con el fin de evaluar este factor y determinar si al incrementar el tamaño del bloque de datos se incrementa el tiempo de proceso en la misma relación, se ha elaborado la Tabla 3, en la que se detalla el incremento del tiempo de ejecución cuando se multiplica por dos el tamaño de los datos procesados. De hecho es una medida del grado de paralelización de cada fase y su independencia del tamaño de los datos procesados, siempre que no se sobrepase la capacidad de los hilos que puede ejecutar la arquitectura en un determinado momento. Es necesario destacar que la primera etapa, en la que se realiza la FFT, es la que menos dependencia tiene del tamaño del bloque, dado su alto grado de optimización, resultante al utilizar las librerías cuFFT de NVIDIA.

Tamaño bloque datos /canal	Power Spectrum	Normalize dB Scale	Filter	Search	Decimate
16 kS	-	-	-	-	-
32kS	1%	10%	60%	73%	99%
64kS	1%	15%	82%	81%	94%
128kS	25%	26%	94%	88%	99%

Tabla 3: Influencia del tamaño del bloque de datos procesado

Con el fin de determinar el grado de mejora, respecto a las condiciones iniciales de todo el proceso, se ha realizado una comparativa entre los tiempos de ejecución utilizando solo LabVIEW (CPU) y los tiempos obtenidos en esta nueva propuesta que tiene parte de su ejecución en GPU. La Tabla 4 muestra dichos resultados para distintos tamaños de bloque utilizados para cada uno de los canales adquiridos. Conviene señalar que los tiempos de ejecución en GPU, incluyen el movimiento de datos, hacia o desde la CPU, cuestión que puede penalizar el rendimiento global del sistema. Si esta cuestión representara un problema significativo podría resolverse, bien utilizando interfaces entre memorias más rápidos, bien desarrollando técnicas que solapen la transferencia de información con el procesado.

Tamaño bloque datos /canal	LabVIEW tiempos ejec. (ms)	GPU tiempos ejec.(ms)	Mejora rendimiento
16 kS	6	4	33.3%
32kS	14	6	57.1%
64kS	26	10	61.5%
128kS	51	18	64.7%

Tabla 4: Comparación de los tiempos de ejecución CPU-GPU

Para el sistema descrito en la introducción, en el que se aplica este algoritmo, el tiempo total de proceso para cada uno de los bloques, determina la frecuencia máxima de muestreo. Este parámetro determina el ancho de banda de las señales a analizar y el número de canales que pueden evaluarse al mismo tiempo. La Tabla 5 muestra la mejora conseguida, reflejada en un notable incremento en la máxima frecuencia de muestreo que se puede utilizar con la solución propuesta que utiliza GPU frente a la implementación de partida que se ejecuta utilizando solo LabView en CPU.

Tamaño bloque datos /canal	LabVIEW (ms)	Transf. datos + procesado (ms)	Mejora rendimiento
16 kS	2.73	4.09	50.0%
32kS	2.34	5.46	133.3%
64kS	2.52	6.55	160.0%
128kS	2.57	7.28	183.3%

Tabla 5: Mejora de rendimiento total entre los dos sistemas

3.1.4 Conclusiones.

Teniendo en cuenta que el sistema de partida, ITMS, es una plataforma de por sí ya optimizada frente a sistemas tradicionales de adquisición de datos, y de acuerdo a los buenos resultados obtenidos en la propuesta descrita con anterioridad, es posible aseverar las siguientes conclusiones:

- Se han alcanzado un camino que permite incluir procesado en GPUs en lenguajes de alto nivel, como LabView, ampliamente utilizado para sistemas de adquisición de datos. De esta forma se abre la posibilidad de obtener sistemas heterogéneos que tengan parte de su ciclo de ejecución en CPU y parte en GPU, en función de la naturaleza del algoritmo y de los datos a procesar.
- Es recomendable, siempre que sea posible, la utilización de librerías optimizadas de proceso frente al desarrollo propio de funciones de bajo nivel o funciones *kernel*. Las razones son por una parte que implica un conocimiento profundo de la arquitectura CUDA por parte del desarrollador, y por otra parte es necesario sopesar el esfuerzo del desarrollo, cuando puede existir una solución comercial que soporta una funcionalidad similar. Podría darse el caso, si no se dominan profundamente las técnicas de programación paralela, de obtener procesos poco eficientes que penalicen el comportamiento final del algoritmo.
- Gracias a la optimización de uno de los procesos en el algoritmo, se ha constatado, el buen comportamiento de la arquitectura CUDA, en términos de rendimiento, cuando se ejecutan múltiples instancias de un proceso con valores independientes. En caso contrario, con interdependencias entre los datos, otras arquitecturas como CPUs multicore, podrán tener un mejor rendimiento.
- Dado la mejora obtenida en los tiempos de ejecución del algoritmo en la GPU, frente a los tiempos de ejecución utilizando en CPU con lenguajes de alto nivel, esta propuesta sirve para aplicaciones que demanden un uso intensivo de procesado, en las que es necesario ajustarse a tiempos estrictos de proceso. El tiempo de ejecución en CPU puede verse afectado por otros procesos en ejecución, lo cual obliga a optar por extensiones en tiempo real para el sistema operativo. Por tanto esta mejora, para el sistema de partida, capacita a la solución para poder utilizar una frecuencia de muestreo mayor, que puede redundar en una mayor resolución o bien en la posibilidad de utilizar un mayor número de canales al mismo tiempo.
- Aunque para el algoritmo analizado, el tamaño de los bloques de datos utilizados, no tienen un impacto considerable, en los tiempos completos de ejecución, es necesario llamar la atención a la influencia que tiene el movimiento de datos entre CPU y GPU. Siempre que sea posible se han

de minimizar estas transferencias, y asegurarse que la información permanece en la memoria de la GPU hasta que esté completamente procesada.

3.2 Integración de GPUs en entornos basados en EPICS.

3.2.1 Introducción

En los laboratorios de investigación, en los que existen gran cantidad de instrumentos y dispositivos a controlar, se hace imprescindible el uso de sistemas de instrumentación y control distribuidos. Frente a soluciones SCADA comerciales, existen opciones de código abierto, como es el caso de EPICS o TANGO. ITER, el proyecto de mayor envergadura en entornos de fusión, ha optado por utilizar EPICS frente a otras soluciones, dado su carácter de plataforma de código abierto, ya que supone una garantía de soporte frente a soluciones comerciales dependientes del mercado (Wallander, Abadie et al. 2010). Además esta selección ha estado basada en experiencias previas, como el KSTAR (Kim, Baek et al. 2010), que han servido como banco de pruebas para el diseño de toda la arquitectura de ITER.

Conocido este escenario, en estos dos casos de uso se plantea el reto de incluir servicios de procesado en GPU, en sistemas de instrumentación y control, dada la incipiente necesidad de aumentar las capacidades de procesado para futuros diagnósticos. Estas nuevas capacidades permitirán evolucionar hacia nuevas funcionalidades necesarias en los dispositivos de fusión de pulso largo, como son reconocimiento de patrones, detección de eventos o compresión de datos. Para ellos se analizarán las capacidades de integración, así como se descubrirán las posibilidades de interconexión y comunicación con otros servicios existentes en el sistema.

Otro de los objetivos perseguidos en estos dos ensayos ha sido poner frente a frente a dos arquitecturas hardware, CPU-multicore y GPU, ejecutando algoritmos representativos en entornos de fusión, buscando además una respuesta a la incógnita que representa la utilización de GPUs en sistemas de adquisición de datos y procesado en tiempo real. En el primero de ellos, el cometido principal es detectar la posición y la amplitud de un espectro de Gaussianas, utilizando el método de Levenberg-Marquardt (Moré 1978). Este

procedimiento es utilizado en JET para calcular la temperatura de ión (Ti) en la cámara de carga de intercambio (Heesterman, Sall et al. 2003). En el segundo se ha utilizado como algoritmo, el cálculo de parámetros fundamentales de equilibrio en el plasma a partir de datos pertenecientes a los sensores magnéticos del tokamak alojado en el dispositivo TCV (Moret, Buhlmann et al. 1998).

Las propuestas analizadas han sido integradas en EPICS, utilizando el framework DPD, descrito en el capítulo 2 de esta tesis. Además de esta forma se mejora y simplifica la distribución de datos en EPICS. Los algoritmos analizados completan la integración en los sistemas de instrumentación y control utilizados en la comunidad de fusión, de sistemas de adquisición y procesado en tiempo real. La Figura 31 esquematiza los dos casos de uso que se analizan a continuación.

Figura 31: Aplicaciones EPICS para procesado en GPU de los dispositivos JET y TCV

3.2.2 Caso de Estudio 1: Cálculo de la temperatura de ion (T_i) en la cámara de intercambio (JET).

El método de Levenberg-Marquardt es un proceso iterativo de convergencia, que permite obtener el mínimo de una función que es suma de los cuadrados de otras funciones no lineales. Se utiliza en un análisis espectral de las imágenes, registradas por la cámara de intercambio en JET, para obtener distintos parámetros del plasma. Los parámetros a extraer son la temperatura de ion y la rotación del plasma. En cada uno de los frames analizados cada uno de dichos parámetros son estimados por medio de funciones Gaussianas. El algoritmo, computacionalmente intensivo, parte de un conjunto de coeficientes que apuntan a una posible localización de las Gaussianas. La aplicación desarrollada parte de estas señales, a las que se les añade ruido y un offset, con el fin de partir de una situación lo más próxima a la realidad. El objetivo es obtener la posición y la amplitud exacta, considerando un mínimo error, utilizando también como parámetros el número máximo de iteraciones permitido.

Con el fin de buscar una solución predecible, y conocer su comportamiento, se ha partido de dicho algoritmo desarrollado en Matlab para realizar un seguimiento de la información en cada una de las fases del mismo. Esto permite evaluar la capacidad de convergencia del algoritmo, que se ha estimado en 6 u 8 iteraciones en función del ruido introducido. De esta forma también es posible estimar la complejidad de las operaciones matemáticas a realizar.

En la Figura 32 se muestra cada una de las fases llevadas a cabo en el algoritmo y en la parte derecha puede observarse una secuencia de datos para los coeficientes provistos y para la información calculada a la salida del procesado. Una vez completado todo el proceso el algoritmo encuentra una versión acotada de la secuencia de datos de entrada, además de los coeficientes que determinan su posición y su amplitud.

Figura 32: Fases del algoritmo y secuencia de muestras entrada/salida

Con el fin de cubrir los objetivos propuestos en la introducción del capítulo, se ha elaborado una aplicación que, basada en DPD, se esquematiza de forma simplificada la Figura 33. En la misma se han construido distintos elementos funcionales, que permiten simular el comportamiento que tendría un sistema de adquisición de datos y procesado. La comunicación con el sistema se realiza en base a órdenes, invocadas desde la línea de comandos, que permiten cambiar el estado de cada uno de los módulos, o bien permiten establecer los valores que deben adoptar las diferentes PVs (Process Variables) compartidas. La solución cuenta con dos módulos responsables del procesado, optimizados para cubrir el algoritmo en cada una de las dos arquitecturas hardware evaluadas.

Figura 33: Aplicación en DPD TiCameraFIT

La funcionalidad adoptada por cada uno de estos elementos funcionales, es la que a continuación se describe:

TiC Data Generator. Permite obtener la señal de entrada, mostrada en la Figura 32, añadiendo ruido a un set predeterminado de Gaussianas. Los datos se suministran como entrada y obviamente simulan una situación real de adquisición de información. Las entradas de información al sistema están parametrizadas mediante el uso de PVs, por lo que es posible elegir el tamaño de bloque, permitiendo una evaluación posterior en distintos escenarios.

EPICS Waveform Monitoring. Utilizando waveforms process variables (WPV) este módulo es capaz de visualizar las señales procedentes del resto de módulos a través de sus “input block links”.

GPU Tic Data Fit. Representa el módulo que da soporte al algoritmo especificado en la Figura 32, ejecutándose en una GPU. La ejecución de todo el proceso se ha optimizado utilizando librería provistas por NVIDIA (CUBLAS v3.2) y por el fabricante EMPhotonics (CULA Tools Premium Library R11). No obstante existen etapas del mismo que han tenido que optimizarse para su ejecución nativa en funciones kernel de la GPU. Tal es el caso de la fase en la que se computa la matriz Jacobiana, ya que no existen realizaciones comerciales para tal caso. Además se han intentado optimizar todas las transferencias de datos entre

memoria de la CPU y de la GPU, intentando que toda la información que se procesa resida en memoria de la GPU.

CPU Tic Data Fit. Este módulo incluye la implementación del algoritmo para CPUs-multicore. Para poder obtener el máximo rendimiento a este tipo de arquitectura, se han utilizado las librerías Intel's Math Kernel Library (Intel MKL) v10.3 en su versión de 64 bits para plataformas Linux.

Adicionalmente a estos procesos se han concebido distintas process variables (PVs) que ayudan a monitorizar el sistema y permiten conocer el estado del mismo:

- Throughput (bytes/s) de la señal generada por el módulo TiC Data Generator.
- Tiempo de ejecución e iteraciones, en cada uno de los dos módulos de proceso.
- Porcentaje de ocupación de los buffers utilizados para conectar cada uno de los módulos, así como la tasa de transferencia obtenida en los mismos.

3.2.3 Ensayos realizados.

Se ha realizado diferentes test de rendimiento de ambos módulos de proceso, para distintas situaciones que a continuación se describen. Dichos test se ha parametrizado con el fin de conocer el impacto de tamaño del bloque de información utilizado y poder estimar los diferentes tiempos de ejecución obtenidos en cada caso. Como ocurrió en los ensayos realizados en el anterior capítulo, se ha decidido utilizar los timers internos de la GPU, con una precisión de 0,5 microsegundos, dado que la medida que ofrecen las primitivas de temporización Linux no garantiza que los tiempos medidos ofrezcan la precisión pretendida.

En la Tabla 6 se pueden observar los tiempos de ejecución en ambos módulos, para distintos tamaños de bloque. En este caso es necesario llamar la atención de que la ejecución en la GPU no es mucho más rápida que en la CPU cuando ésta utiliza todos sus cores disponibles, en concreto 8 cores al tratarse de una plataforma bi-procesador. Obviamente son resultados lógicos cuando se enfrenta un sistema bi-procesador de altas prestaciones, Intel Dual Xeon X5500 QuadCore, frente a una GPU del segmento de consumo de gama media-baja, NVIDIA GTX580. Aun

siendo así, el rendimiento obtenido en la GPU mejora el entregado por la CPU, sobre todo a medida que el tamaño del bloque aumenta, ya que el objetivo de esta arquitectura (SIMD) es poder realizar miles de operaciones atómicas en funciones nativas kernel con muestras o valores diferentes.

Un factor muy relevante obtenido en la elaboración de estos ensayos es el determinismo en la ejecución de ambos módulos, sobre todo cuando se trata de un aspecto a tener en cuenta en aplicaciones de control en tiempo real. Este factor reflejado en la desviación estándar puede observarse en la Tabla 6 cuando se realiza un ciclo de 100 ejecuciones del mismo algoritmo por cada uno de los bloques. Cuando se dedican todos los cores disponibles en la CPU, el determinismo de este bloque puede verse comprometido en la medida que otros procesos del sistema puede interferir el comportamiento y rendimiento de la ejecución en el bloque, incluyendo tareas propias del sistema operativo en estas consideraciones.

Tamaño Bloque	GPU		CPU	
	Tiempo (ms)	StdDev (%)	Tiempo (ms)	StdDev (%)
256	2,9	5	2,7	1
512	2,8	1	4,8	7
1024	3,6	0	8,7	5
2048	5,2	0	16,1	39
4096	16,4	0	28,5	0
8192	42,8	0	55,3	3
16384	85,4	0	107,5	1
32768	168,6	0	210,0	3
65536	334,8	0	425,0	0

Tabla 6: Tiempos de ejecución en GPU y en CPU (8 cores)

Para poder disponer de un determinismo aceptable en la ejecución del módulo en la CPU, es necesario reducir el número de unidades de computación, cores, cuestión que penaliza el tiempo de ejecución del bloque. Esta apreciación puede observarse en la Tabla 7 en la que se ha evaluado el rendimiento cuando se utiliza un único core de la CPU. Al evaluar esta misma característica para el módulo que reside en la GPU, puede observarse que su desviación estándar es muy baja. Este comportamiento que puede considerarse constante para el tamaño de bloque, únicamente podría verse afectado por el movimiento de datos entre CPU y GPU. No obstante considerando que el porcentaje de tiempo invertido en esta tarea no

es relevante, su influencia es menor, de hecho puede llegar a ser nula cuando el tamaño de bloque utilizado es mayor. Por tanto este alto determinismo capacita a sistemas basados en GPU para ser utilizados en sistemas de adquisición y procesado en tiempo real, aun cuando no se utilizan extensiones de sistemas operativos para tal fin. Cuando se integre este servicio, en un sistema de adquisición y procesado en tiempo real, la máxima frecuencia de trabajo del sistema responderá al tiempo de ciclo, el que se han de contemplar: el tiempo de proceso, el movimiento de datos y un tiempo de guarda, que modele los factores no determinísticos del sistema. Para los tiempos mostrados en las tablas anteriores se puede concluir que estos tiempos se cumplen holgadamente y por tanto la solución cumple la propuesta de tiempo real.

Tamaño Bloque	GPU		CPU	
	Tiempo (ms)	StdDev (%)	Tiempo (ms)	StdDev (%)
256	2,9	5	2,7	1
512	2,8	1	5,1	1
1024	3,6	0	9,2	0
2048	5,2	0	15,9	0
4096	16,4	0	29,1	0
8192	42,8	0	55,4	0
16384	85,4	0	107,9	0
32768	168,6	0	216,0	0
65536	334,8	0	445,7	0

Tabla 7: Tiempos de ejecución en GPU y en CPU (1 core)

Por último se ha relacionado la mejora de rendimiento entre las dos situaciones descritas previamente. En la Tabla 8 se refleja dicha comparativa, en la que queda de manifiesto que es necesario trabajar con tamaños de bloques de información suficientemente grandes para poder sacar todo el potencial de proceso a una GPU. De hecho si los datos que se procesan, tienen un tamaño pequeño, es preferible su proceso en CPU, debido sobre todo a que el tiempo necesario para lanzar procesos kernel se mantiene constante, independientemente del número de muestras procesadas.

Tamaño Bloque	GPU vs CPU-8t	GPU vs CPU-1t
256	-3,85%	-6,36%
512	40,94%	44,02%
1024	58,14%	60,49%
2048	67,54%	67,26%
4096	42,49%	43,57%
8192	22,45%	22,70%
16384	20,56%	20,86%
32768	20,07%	22,27%
65536	21,41%	24,90%

Tabla 8: Mejora de rendimiento obtenido con GPU

3.2.4 Caso de Estudio 2: Cálculo de los parámetros de equilibrio del plasma (TCV).

El objetivo del algoritmo implementado es obtener parámetros críticos del plasma, en el tokamak alojado en el TCV, que determinan su situación de equilibrio. Dichos parámetros, como inductancia interna del mismo o el desplazamiento (shift) Shafranov, son calculados a partir de los datos obtenidos de los 80 sensores magnéticos que rodean el perímetro toroidal de dicho dispositivo. De ellos, existe un conjunto de 38 pequeñas bobinas dispuestas tangencialmente al perímetro de la vasija que contiene el plasma, que adquiere la componente de campo poloidal. Otro conjunto de 38 “flux loops²”, situados también en el perímetro de manera concéntrica con la corriente en el plasma, permiten obtener la componente poloidal perpendicular al perímetro de la vasija de vacío. Por último, se utilizan otros 4 sensores flux loops encargados de medir otras componentes provenientes del plasma como diamagnetismo, componentes de campo toroidal o las corrientes de Eddy, utilizando el desarrollo llevado a cabo en (Romero, Coda et al. 2012, Shafranov 1971). Con el fin de realizar dichos cálculos de manera eficiente, se han realizado distintas operaciones que permiten discretizar los pasos más complejos del algoritmo, como es la determinación de la integral de Shafranov.

² Lazo que rodea el plasma, en cuyos extremos se mide una corriente proporcional al campo magnético existente en el plasma

Figura 34: Aplicación DPD para cálculo de parámetros de equilibrio

En la Figura 34 se muestra un esquema de la aplicación, que utilizando DPD, permite computar el algoritmo en los distintos escenarios que se describirán a continuación. La versión de DPD utilizada (Castro, Romero et al. 2014), mejora su comportamiento gracias a mecanismos que balancean la sincronización entre hilos, optimizan el bloqueo de los buffers utilizados entre los enlaces, o permiten la posibilidad de la utilización de múltiples entradas de datos, desde varias fuentes.

Se han construido tres módulos de proceso distintos para evaluar su comportamiento:

- CPU-multicore. Para el desarrollo del módulo correspondiente a la CPU se ha recurrido al uso de la librería GSL (GNU Scientific Library) que permite realizar operaciones complejas con matrices y vectores, así como funciones y subrutinas de álgebra lineal
- GPU. El módulo correspondiente a la GPU, ha sido optimizado mediante funciones kernel, compiladas mediante la herramientas provistas en el SDK de CUDA, y aprovecha las rutinas optimizadas existentes en la librería cuBLAS.
- Motor de procesamiento MATLAB. El módulo de proceso MATLAB, ejecutado también en la CPU-multicore, permite conectar y manejar el motor de esta aplicación, no solo para enviar o recibir datos, si no para ejecutar también código sin necesidad de realizar ningún tipo de conversión.

La plataforma hardware utilizada consta de una estación de trabajo basada en Intel Core i7 3770K (4 cores físicos/8 cores lógicos) con 16 GB DDR3 RAM, PCIe

3.0, que aloja una GPU Asus GTX580 con 512 CUDA cores. Todo el sistema utiliza RedHat Enterprise 6, EPICS (v3.14.12.2) y asynDriver (4.19)

3.2.5 Ensayos realizados.

Con el fin de evaluar la capacidad de la solución propuesta se han construido distintos escenarios que ayudan a extraer distintos parámetros de interés. Los datos utilizados pertenecen a valores adquiridos en TCV, y se leen desde varios ficheros, simulando el comportamiento que se produciría al adquirir señales desde distintos sensores. El sistema se ha parametrizado, con la finalidad de simular diferentes tamaños de bloque. Los tamaños de bloque evaluados van desde 928 bytes hasta 92.800.00 bytes, correspondientes a los factores de muestreo aplicados a cada señal de entrada (1-100.000).

La Figura 35 presenta los tiempos de proceso de cada uno de los módulos implementados para cada arquitectura. Resaltar que el mejor comportamiento responde al módulo que aprovecha la capacidad de cálculo de la GPU, manteniendo prácticamente el tiempo de ejecución independientemente del tamaño del bloque de datos utilizado. Para los otros dos módulos el tiempo de ejecución es proporcional al tamaño del bloque utilizado, observándose mejores resultados en MATLAB que en el que se ejecuta en la CPU. Esto hace pensar que el rendimiento de las librerías utilizadas, GSL, de código abierto no está tan optimizadas como las que se utilizaron en el caso de uso anterior, Intel MKL, que aunque de pago obtienen el máximo rendimiento a las arquitecturas de los procesadores Intel.

Figura 35: Tiempos de ejecución de las tres arquitecturas para diferentes tamaños de bloque.

La desviación estándar mostrada en la ejecución de cada uno de los módulos también ha sido evaluada y es mostrada en la Figura 36, medida en ms. Como en los trabajos anteriores, el módulo correspondiente a la GPU presenta valores despreciables para cualquier tamaño de bloque. No ocurre lo mismo para el módulo basado en el motor de MATLAB que presenta una dependencia lineal con el tamaño de bloque. Este factor invalida el uso de este módulo en sistemas de tiempo real, aunque puede ser una solución interesante para sistemas con menos restricciones, dado el rápido ciclo de desarrollo necesario para implementar algoritmos en dicha arquitectura.

Figura 36: Desviación estándar medida en ms para diferentes tamaños de bloque

3.2.6 Conclusiones

En el ensayo realizado, se han cubierto los objetivos propuestos, que han permitido obtener las siguientes conclusiones. Estas afirmaciones ayudan a tomar decisiones de futuro, en la comunidad de fusión, a la hora de contemplar la introducción de procesamiento en GPU en los sistemas de control e instrumentación, para aquellos diagnósticos que requieran una alta demanda de cálculo computacional.

- La propuesta planteada supone un primer paso en la integración de procesamiento paralelo en GPU, modularizado como servicio, para poder

ser usado en diagnósticos de planta en ITER. Ello se consigue utilizando el middleware EPICS adoptado por la organización como software para controlar aplicaciones en tiempo real de forma distribuida.

- Como medio de integración de procesado en GPUs, en el sistema de control e instrumentación distribuido EPICS, se ha maximizado el uso del modelo DPD. Esta aproximación solventa la carencia detectada cuando se han de transferir grandes cantidades de información entre distintos procesos y podría ser tomada en cuenta para la red de almacenamiento en ITER (DAN).
- Se ha concluido que existe un mayor determinismo del algoritmo procesado en la GPU que en una plataforma multicore-CPU. Esto es debido a la independencia del procesado, en un sistema considerado como coprocesador, que solo se dedica a estas tareas. Dado que la CPU es responsable de otros procesos, es fácil entender que simplemente la gestión que el sistema operativo hace de cada uno de los cores, pueda penalizar la ejecución de tareas que requieran una alta demanda computacional. Este comportamiento determinista en la GPU, capacita esta arquitectura para ser incluida en sistemas de adquisición y procesado en tiempo real, teniendo en cuenta además la mejora de rendimiento obtenida en la ejecución del algoritmo utilizado.
- Los diagnósticos susceptibles de utilizar procesado en GPU, serán aquellos que utilicen algoritmos en los que se procese el tamaño del bloque suficientemente grande, para obtener la máxima ocupación de los recursos de proceso. Serán aquellos en los que se manejen un gran número de canales, o bien en los que se puedan trabajar con imágenes.

Capítulo 4

4 DESCRIPCIÓN DEL MODELO Y METODO PROPUESTO.

4.1 Introducción. Objetivos

En los modelos expuestos anteriormente se ha introducido el uso de GPUs como sistemas de procesamiento masivo en bloques, que permiten acelerar cálculos y estimaciones, siempre con el objetivo de maximizar la información obtenida sobre datos previamente adquiridos y almacenados.

Sin embargo, el objetivo final que se persigue es poder disponer de aplicaciones que utilizando GPUs permitan desplegar soluciones de procesamiento de datos en tiempo real, tales como algoritmos de reducción de datos, de compresión, de tratamiento de imágenes, o la extracción de características para métodos de reconocimiento de patrones. Estas herramientas, combinadas con los sistemas DAQ basados en FPGA, se consideran esenciales para solucionar los retos que afrontan los nuevos dispositivos de fusión desde el punto de vista de la implementación de sus sistemas de diagnóstico y control.

En el modelo que se propone a continuación el procesamiento se realizará con información adquirida en tiempo real y es abierto con la finalidad de poder utilizar imágenes o señales provenientes de sensores de distinta naturaleza. La metodología propuesta permitirá en el próximo capítulo evaluar un caso de uso que permite adquirir imágenes mediante el protocolo CAMERAlink (BASLER, COGNEX, et al. 2015) al cual se le aplica un primer nivel de pre-procesado en una tarjeta de adquisición basada en FPGA's, y que posteriormente se complementa en un segundo nivel de mayor complejidad ejecutado en la GPU.

La propuesta supone el uso de la tecnología GPUDirect-RDMA de NVIDIA para sus dispositivos compatibles CUDA. Aunque existen soluciones de drivers de terceros no propietarios (Rath, Bialek et al. 2012), esta aportación supone la primera que aprovecha estas prestaciones para entornos de fusión. Existen desarrollos comerciales que también usan esta tecnología (Dustin, GE GpuTechConference 2015, Infiniband, Mellanox 2013) , aunque son sistemas a medida que no utilizan hardware COTS, como el propuesto.

La propuesta que a continuación se desarrolla, describe una metodología para su aprovechamiento en entornos de fusión, utilizando los modelos y estándares adoptados actualmente en la comunidad. Dado que es en ITER donde se está llevando a cabo la estandarización de nuevos modelos de sistemas de adquisición

y procesado, así como de los sistemas de control e instrumentación, se particularizará la propuesta para las definiciones establecidas en dicho proyecto. Entre los objetivos perseguidos está el optimizar el flujo de datos del proceso, de forma que se eviten copias o movimientos de datos innecesarios. El punto crítico es la transferencia de datos directa entre el sistema de adquisición de datos basado en FPGA y el sistema de procesado (GPU). Este paso de datos directo evita tener que pasar por la memoria principal del host, lo que redundaría en una reducción del tiempo de ciclo, a la vez que proporciona una solución más determinista y con una latencia menor. Esta mejora, que se justificará adecuadamente mediante los ensayos correspondientes, hace al modelo menos dependiente de otras tareas computacionales que se estén desarrollando en el host. La idea básica de la mejora que se pretende obtener se ve reflejada en la Figura 37, aunque a lo largo del capítulo se desarrollará el modelo del sistema propuesto incluyendo un estudio de las tecnologías involucradas.

Figura 37: Propuesta de mejora de movimiento de datos

En un enfoque tradicional de un sistema de adquisición y procesado en GPU se deben completar los pasos, marcados desde el 1 al 5, que pueden observarse en la parte izquierda de la Figura 37:

1. Adquisición de datos y movimiento de los mismos hacia memoria principal del sistema en CPU.
2. Movimiento de los datos adquiridos desde la memoria de la CPU a la memoria de la GPU.

3. Proceso paralelo en GPU.
4. Movimiento de los resultados desde la memoria de la GPU a la memoria de la CPU.
5. Los resultados, opcionalmente, pueden ser utilizados para generar una salida o bien para disparar otros procesos.

La propuesta recoge la eliminación de dos copias de datos, dotando al sistema de adquisición de un acceso directo a la memoria de la GPU, como puede observarse en la parte derecha la misma figura. Los pasos en este caso, marcados desde el 1 al 3, son:

1. Adquisición de datos y movimiento de los mismos hacia memoria principal del sistema en GPU.
2. Proceso paralelo en GPU.
3. Movimiento de los resultados hacia el propio sistema DAQ o incluso a la memoria principal del sistema para otros procesos.

Una cuestión trascendental para poder acometer este reto es poder contar con el apoyo de empresas que ofrezcan la disponibilidad de sus productos y el código fuente del software que los maneja de manera abierta. En concreto cuando se trabaja en sistemas Linux es necesario disponer tanto del código correspondiente a la librería de usuario, como el código que permite construir del driver kernel que controla el dispositivo hardware. El modelo de una aplicación de usuario que hace uso de esta arquitectura software, para el caso correspondiente a un sistema DAQ con dispositivos NI FlexRIO, puede verse de forma simplificada en la Figura 38. En la misma se han remarcado los módulos que es necesario personalizar.

Figura 38: Capas software en un Sistema basado en Linux

Con esta posibilidad se puede acceder al código fuente del driver que maneja el dispositivo de adquisición a bajo nivel, pudiendo introducir cambios en el mismo, para tener acceso directo a la memoria de la GPU sin pasar por la memoria principal del sistema. En este caso, y tras consultar con diversos fabricantes, se ha colaborado estrechamente con la empresa National Instruments, proveedor del catálogo de la ITER IO (Makijarvi 2015), para poder alcanzar los objetivos expuestos. Dicha colaboración se ha sustentado en un acuerdo de propiedad intelectual, en el que se reconocen los derechos de autoría al equipo de esta investigación. En la misma se ceden los derechos de explotación a la empresa de las modificaciones realizadas sobre el código del driver de las tarjetas de adquisición de datos basadas en FPGA FlexRIO.

4.2 Descripción hardware del sistema.

El sistema hardware utilizado para implementar el modelo propuesto se basa en gran medida en elementos incluidos en la especificación de los FAST PLAN SYSTEM CONTROLLER (FPSC) de la ITER IO (Makijarvi 2015). Los elementos hardware esenciales para el modelo son:

- Workstation, basado en tecnología Intel, que integra los siguientes elementos:
 - Intel Dual Xeon
 - Memoria DDR3 Ram 24 Gbytes
 - Disco duro de 500 GB
 - Tarjeta PCI-Express a MXI-Express NI-8372 que permite una conexión de 4 enlaces PCI-Express 1.0 a 250 Mbytes/s.
 - GPU Nvidia Tesla K20x
- Chasis PXIe-1062Q que aloja distintos sistemas de control y adquisición, en concreto en este trabajo se ha utilizado:
 - Tarjeta MXI-Express a PCI-Express NI-8370 que permite la conexión del chasis con la Workstation.
 - Tarjeta FlexRIO NI 7966R.

En la Figura 39 se muestra un esquema de los elementos principales y cómo se realiza su interconexión a nivel del bus PCI-Express. La figura muestra un esquema simplificado ya que en realidad la tarjeta FlexRIO 7966R incluye la conexión de un módulo adaptador NI-1483 que permite gestionar el interface Cameralink que transporta imágenes en dicho estándar. Por otra parte, para poder desarrollar casos de uso de extremo a extremo, a partir de datos reales almacenados previamente, se ha optado por generar las imágenes con un emulador hardware alojado en un ordenador personal que permite generar el flujo de información como si de una cámara real se tratara.

Figura 39: Hardware del sistema propuesto

En el esquema de la Figura 39, la conexión entre el sistema CPU con la tarjeta de adquisición de datos NI-FlexRIO se obtiene mediante un enlace MXI express para PXI-Express que utiliza 4 líneas PCIe v1.0a, lo que supone la capacidad de obtener una tasa de 1 GB/s en ambas direcciones. De hecho, el uso de esta arquitectura permite, que desde el punto de vista del programador, todos los dispositivos PCIe estén bajo el mismo IOHUB y por tanto aparecerán como si estuvieran dentro del mismo equipo. De esta forma, como se justificará más adelante, se podrán realizar transacciones entre los dispositivos PCI-Express utilizando técnicas DMA, con una intervención mínima de la CPU.

4.3 Descripción software del sistema.

Para alcanzar el objetivo perseguido en la introducción, se han utilizado distintas herramientas de edición, compilación y depuración bajo entorno Linux. Para el desarrollo de las modificaciones realizadas, se ha utilizado como sistema operativo, la distribución personalizada Codac Core System 5.0 de ITER, basada en Red Hat Enterprise Linux. En la misma ya están integradas todas las herramientas que permiten construir todos los elementos que constituyen la unidad software que gobierna y gestiona la tarjeta de adquisición basada en FPGA, previamente referenciada.

Para poder desarrollar el software que gobierna la GPU se ha utilizado el NVIDIA SDK Toolkit versión 6.5. En el mismo se encuentran todas las herramientas que permiten construir aplicaciones para GPU, distintas librerías de cálculo, y el driver kernel Linux (v3.40.29) tanto en formato binario, como en código fuente. Este código fuente, al igual que el de la FPGA, es de vital importancia para las modificaciones planteadas y posteriormente se referenciará de nuevo.

4.4 Tecnología NI-FlexRIO.

Como elemento de adquisición de datos ITER ha seleccionado e incluido en su catálogo para los FC-I&C dispositivos basados en tecnología FlexRIO. Son tarjetas de adquisición basadas en FPGA (Figura 40) lo cual hace que su hardware sea reconfigurable, además de que su funcionalidad pueda cambiar ya que incorporan recursos en E/S intercambiables a través de un conector que permite incorporar distintos tipos de módulos adaptadores (FAM).

Figura 40: Tarjeta NI FlexRIO con módulo adaptador

En el catálogo de National Instruments existen distintos módulos adaptadores, en función de la naturaleza de las señales a adquirir: señales analógicas, señales digitales, imágenes, etc (NATIONAL INSTRUMENTS, FAM 2015). También existe la posibilidad de utilizar módulos adaptadores de otros fabricantes o construir uno a medida utilizando el kit de desarrollo correspondiente. Esta solución robusta permite configurar sistemas de adquisición de datos flexibles, que pueden ser controlados por un host, o bien pueden funcionar de manera autónoma.

Los modelos que se podrán utilizar en la propuesta del modelo de adquisición de imágenes son aquellas que disponen de bus PXIe: PXIe-7961R, PXIe-7962R

PXIE-7963R, PXIE-7965R, PXIE-7966R. Todas incorporan un modelo similar de FPGA (Virtex 5 SX50T y Virtex 5 SX95T) y permiten transferencias de hasta 860 MB/s entre la memoria interna y la memoria del host que la controle mediante comunicación DMA. La razón por la que se utilizan esta serie de tarjetas frente a otra serie de la misma tecnología como pueden ser la PXI-795XR, está en el tipo de controlador interno DMA (MITE), menos documentado y de uso cada vez más limitado en el hardware del fabricante.

Para poder alojar cualquiera de las tarjetas mencionadas se requiere de un chasis PXIe, que al margen de ser una solución robusta y flexible para construir sistemas de instrumentación y control en entornos industriales, permite la conexión a un controlador externo mediante un bus PCIe-PXIe. En la Figura 41 se muestra un chasis 1062Q que permite establecer una comunicación con un host de 4 líneas PCIe obteniendo un teórico ancho de banda de 1GB/s.

Figura 41: Chasis PXI-Express NI-1062Q

No obstante existen en el catálogo de NI, chasis que permiten incrementar el ancho de banda disponible ya que utiliza un enlace de 8 líneas PCIexpress bajo la norma 2.0 (NI-1085). Esto supone una máxima transferencia de 4GB/s.

4.5 Sistema de adquisición de datos basado en FlexRIO

Para poder completar un sistema de adquisición de datos basado en tecnología FlexRIO son necesarias dos piezas software: una descripción hardware del comportamiento del sistema basado en FPGA y una aplicación que controla dicho comportamiento y que se ejecutará en el host. Si para obtener una aplicación, que utilice un sistema basado en FPGA, se utilizan habitualmente lenguajes HDL, para obtener una aplicación que utilice un dispositivo FlexRIO, es necesario el uso del entorno gráfico LabVIEW para FPGA. El control de la

configuración hardware adoptada por el dispositivo será llevado a cabo por el usuario mediante una aplicación obtenida en LabVIEW para Windows. No obstante y dado el continuo incremento de utilización de estos dispositivos, National Instruments, provee de herramientas y software que permiten su control bajo otros sistemas operativos. En concreto, para Linux, NI da soporte a esta tecnología mediante el uso del driver NIRIO. De este driver existen dos versiones, una cerrada (sin código fuente) y otra abierta (con código fuente de uso abierto). La versión abierta de este driver es a la que en posteriores epígrafes se hará referencia y ha sido incluida en el acuerdo de colaboración con la empresa National Instruments.

A continuación se describen las etapas necesarias para poder obtener una aplicación que gobierne o controle un sistema de adquisición de datos que utiliza un sistema FlexRIO basado en FPGA.

Diseño, modelado del sistema

Será necesario partir de unas especificaciones funcionales a las que deberá dar soporte el sistema. Ello llevará a decidir qué modelo de tarjeta FlexRIO y módulo adaptador se ciñen a las funcionalidades necesarias. Será necesario evaluar parámetros como: E/S digitales, canales analógicos, parámetros del convertidor A/D y D/A o bien interface/norma para sistemas de adquisición de imágenes.

Los recursos que van a permitir el control del hardware modelado en la FPGA van a ser registros de E/S y canales DMA. Haciendo uso de ellos será posible que una aplicación se comunice con la tarjeta FlexRIO y pueda controlar su funcionalidad. En algunas configuraciones es posible no depender de un host, de tal forma que la tarjeta FlexRIO y el módulo adaptador puede considerarse como un sistema autónomo, ya que esta tecnología incorpora memoria flash interna. Utilizando este modo es posible reconfigurar el hardware en el encendido, mediante la carga del bitfile almacenado. En el modelo aquí propuesto, la configuración del hardware se realizará de forma dinámica y por tanto el bitfile que describe el comportamiento de la FPGA se cargará desde la aplicación de usuario.

Para poder obtener el código que describe el comportamiento hardware de la FPGA es necesario utilizar LabVIEW para FPGA. Presenta como ventaja, frente

a lenguajes de programación VHDL, que es un lenguaje gráfico que acelera la obtención del bitfile mediante la utilización de elementos u objetos funcionales disponibles en distintas paletas del entorno. Es necesario tener en cuenta que lo que se construye en este entorno es el comportamiento hardware, por lo que existirán limitaciones a la hora de incluir elementos que si se puede utilizar en LabVIEW para Windows. En la construcción del proyecto será necesario definir elementos como canales DMA necesarios, FIFOs o memorias. Para ello el entorno dota de distintos asistentes, como el mostrado en la Figura 42 que permite establecer las propiedades correspondientes a una FIFO-DMA.

Figura 42: Pantalla de configuración de una FIFO-DMA

Test y simulación

Como paso previo a la realización de la aplicación que controlará la tarjeta FlexRIO, es necesario desarrollar una aplicación que chequee las funcionalidades que se quieren obtener de la FPGA. Para ello se utilizará el entorno LabVIEW para Windows, en el que se puede obtener además realizar un test funcional que permite realizar mejoras iterativas hasta cumplir las especificaciones marcadas. Además y dado que se trata de una simulación no existirán tiempos de espera para obtener el bitfile que se programará en la FPGA. Por el contrario no es posible realizar pruebas de rendimiento o medidas de tiempo, tareas que obligatoriamente deben ser supervisadas y controladas por el diseñador

Generación del bitfile y mapa de recursos

Una vez comprobadas las funcionalidades del sistema, es necesario la compilación del diseño mediante LabVIEW para FPGA, con el fin de que no se viole ninguna regla de diseño, y en un paso posterior poder traducirlo a lenguaje VHDL. Este código VHDL será internamente compilado por las herramientas de síntesis de Xilinx (fabricante de las FPGAs que incorporan las tarjetas FlexRIO) para conseguir finalmente el bitfile para la configuración de la FPGA.

La obtención del bitfile es un proceso en el que es necesario esperar entre varios minutos o varias horas, en función de la complejidad del sistema diseñado, o en función de los recursos lógicos disponible en la FPGA. Por tanto antes de comenzar esta fase, es altamente recomendable, superar con éxito la fase anteriormente mencionada de simulación funcional. Cualquier disfunción, implica una modificación del código y un proceso de compilación de todo el sistema.

Aplicación software de usuario.

Como se ha mencionado anteriormente, en el modelo propuesto se va a construir una aplicación de usuario encargada de controlar la configuración hardware de la FPGA. Dicha aplicación puede realizarse en distintos lenguajes de programación bajo sistemas operativos Linux o Windows. Dado que las herramientas adoptadas mayoritariamente en los entornos de fusión, tienden a ser estandarizadas bajo sistemas operativos Linux, la aplicación será desarrollada en lenguaje C bajo este sistema.

Partiendo del bitfile obtenido en la etapa de compilación, es necesario utilizar la herramienta C API Generator, que examina dicho fichero y obtiene un mapa de todos los recursos de la FPGA a los que se puede acceder. Esta lista de recursos permite construir un fichero <include.h>, en el que se encuentran las referencias necesarias para que puedan ser accedidos desde una aplicación C. En la Figura 43 puede observarse una sección de dicho fichero en el que se obtienen las referencias a distintos recursos del hardware implementado en la FPGA.

```
#ifndef __NiFpga_TESTCOUNTMFCV5_h__
#define __NiFpga_TESTCOUNTMFCV5_h__
typedef enum
{
    NiFpga_TESTCOUNTMFCV5_IndicatorU32_ExpectedIOModuleID = 0x70,
    NiFpga_TESTCOUNTMFCV5_IndicatorU32_Fref = 0x74,
    NiFpga_TESTCOUNTMFCV5_IndicatorU32_InsertedIOModuleID = 0x6C,
} NiFpga_TESTCOUNTMFCV5_IndicatorU32;
typedef enum
{
    NiFpga_TESTCOUNTMFCV5_ControlBool_GroupEnable0 = 0x2A,
    NiFpga_TESTCOUNTMFCV5_ControlBool_auxDO2 = 0x12,
} NiFpga_TESTCOUNTMFCV5_ControlBool;
.
#endif
```

Figura 43: Sección del código `<include>.h` obtenido con *C API Generator*

Este fichero que define los recursos accesibles del hardware configurado en la FPGA, junto al bitfile, permitirán gobernar el comportamiento de la tarjeta de adquisición. Además, en la aplicación será necesario el uso de las librerías que dan acceso a un set de funciones básicas que permiten: cargar y lanzar la ejecución de un bitfile determinado, interactuar con los recursos mapeados o bien realizar lecturas y escrituras de las estructuras FIFO DMA.

4.6 Tecnología Nvidia GPU.

Dado que ya se ha introducido en capítulos previos el uso de esta tecnología en sistemas de adquisición y procesamiento de datos inteligentes, aquí solo se exponen las pautas necesarias para construir una aplicación que utilice un algoritmo conocido. Para ello se utilizarán las herramientas pertenecientes al SDK CUDA de NVIDIA, y conviene abordar dicho desarrollo de manera aislada al resto del sistema, en una primera aproximación. De esta forma es posible optimizar su rendimiento antes de integrar el procesamiento en un sistema que consta de otros módulos software con los que ha de interactuar.

Flujo de trabajo para obtener una aplicación para GPU

A la hora de realizar una aplicación de procesamiento de información que saque el máximo provecho a los recursos hardware que tiene una GPU es de vital importancia seguir las pautas que a continuación se detallan. Lógicamente se ha de concretar el algoritmo a resolver y el código que lo soporta, así como la arquitectura GPU en la que se va a implantar. Se puede partir desde cero, aunque es conveniente, comenzar el desarrollo a partir de un algoritmo escrito en

cualquier lenguaje de programación convencional (C-C++, FORTRAN, MATLAB) o incluso pseudocódigo. Las etapas a cubrir pueden resumirse en:

- Análisis y evaluación del código inicial. Es necesario localizar aquellas secciones que tienen un alto consumo computacional. Pensando en que se va contar con una arquitectura que puede ejecutar muchas operaciones en paralelo, es recomendable seleccionar operaciones básicas con datos que no tienen ninguna dependencia entre sí. Utilizando herramientas de profiling, pueden aislarse aquellas partes del código que son susceptibles de paralelizar.
- Paralelizar. Para obtener la solución paralelizada del código existen distintas posibilidades. La que requiere más esfuerzo por parte del desarrollador es el desarrollo de funciones de código nativo que dan soporte al código paralelo, denominadas funciones *kernel*. Otra posibilidad es utilizar librerías especializadas en la resolución de problemas concretos, incorporadas en el propio SDK de NVIDIA, si bien existen en el mercado un extenso catálogo de empresas de software que proporcionan librerías optimizadas para proceso paralelo en GPUs. Una última posibilidad es el uso de compiladores que haciendo uso de directivas especializadas se encargan de trasladar secciones de código lineal a código paralelo. (OpenACC, Trust)
- Optimizar. Antes de dar por válido las etapas anteriores, será necesario realizar un análisis de rendimiento de la solución obtenida. Dentro de este análisis, al margen de determinar si el nuevo código es más rápido que el de partida, se han de contemplar cuestiones como: el movimiento de datos entre la memoria de la CPU y la memoria de la GPU, aprovechamiento de todos los *cores* disponibles o evaluar una correcta utilización de la memoria caché de los multiprocesadores. En esta fase conviene utilizar herramientas de profiling que ayudan a localizar las debilidades que pueden ser mejoradas en la solución final.

Consideraciones en la integración de procesado GPU en un sistema de adquisición de datos.

Como ha quedado de manifiesto en el capítulo 3, una GPU es un candidato perfecto como unidad de proceso de cómputo general. Sin embargo hace falta incidir en las siguientes consideraciones:

Tiempo real. El propietario de la tecnología, NVIDIA, no provee de kernel de dispositivos para sistemas operativos en tiempo real. Esta cuestión obliga a que cualquier requerimiento temporal sea diseñado apoyándose en técnicas de análisis que proporciona el SDK de NVIDIA.

Movimiento de datos. Aunque los buses que permiten mover información entre CPU y GPU siguen mejorando, es recomendable minimizar estas transferencias. Hasta no acabar el algoritmo correspondiente en GPU, no se debe devolver ningún resultado hasta la CPU.

Actualización de hardware/software. Dado que es un mercado en continua evolución, resulta normal encontrarse con tarjetas GPU de nueva generación cada año. La tendencia es obtener más capacidad de proceso, aumentando el número de multiprocesadores, y conseguir un menor consumo. Además en cada cambio NVIDIA introduce nuevas tecnologías que dan lugar a que se renueven los drivers y los SDK existentes, con el fin de dar soporte a estas nuevas funcionalidades. No obstante, hasta la fecha, siempre ha existido un camino fácil y directo para migrar aplicaciones, desarrolladas para una arquitectura, hacia otras más evolucionadas.

4.7 Modelo Inicial: Sistema de adquisición de datos y procesado continuo utilizando una GPU

Una vez descritas las plataformas hardware para la adquisición y procesado, así como el método individualizado para trabajar con cada uno de ellas, a continuación se propone la arquitectura software del sistema completo. Es necesario partir de la asunción que previamente se han obtenido en una primera fase:

- Bitfile que define el comportamiento hardware de la FPGA, así como los ficheros asociados a los recursos disponibles.
- Código paralelizado que se ejecuta en la GPU.

En el esquema propuesto será la CPU del sistema la que se encargue de controlar y configurar los dos procesos. Por ello será necesario construir una aplicación que

se ejecute en la CPU para gobernar las dos arquitecturas hardware y que ha de contar con las siguientes fases en el diagrama de flujo mostradas en la Figura 44.

Figura 44: Diagrama de flujo aplicación FPGA-GPU

A continuación se desarrolla una descripción de cada una de las subtarefas:

- Inicialización de recursos. En esta fase se incluye la comprobación que existen los elementos hardware necesarios para realizar las siguientes fases. Por una parte en la CPU, se reservarán todos los buffers necesarios para poder manejar la información procedente de la FPGA, así como los buffers que se van a utilizar en la GPU, y que ésta utilizará para intercambiar información con la CPU. La tarjeta FPGA se configurará mediante su bitfile y mediante el fichero que lista todos sus recursos.
- Adquisición de la información. Los parámetros necesarios para la adquisición dependerán de la naturaleza del fenómeno físico a diagnosticar. En el caso particular de una aplicación que adquiriera imágenes los parámetros típicos a establecer son por ejemplo: el tamaño del frame capturado, la profundidad en bits utilizada para cada pixel o la tasa del reloj. Para poder controlar estos parámetros se utilizarán los controles obtenidos desde el bitfile que configura el hardware de la FPGA. El método que mejor rendimiento ofrece para el movimiento de

información de información entre la memoria interna de la FPGA y la memoria de la CPU, es utilizar estructuras FIFO DMA.

- Movimiento de datos desde la CPU a la GPU. Una vez los datos han llegado a la memoria de la CPU, será necesario llevar dicha información a la memoria global de la GPU. Se utilizarán los buffers previamente reservados y la tasa de transferencia en este movimiento de información dependerá de los parámetros de la conexión PCI-Express de la tarjeta GPU (norma y número de líneas utilizadas).
- Procesado en la GPU. Mediante una llamada a la función de procesado, el runtime del driver, se encargará de lanzar el código de dicho procesado en la GPU. Recordar que dicho algoritmo se debe haber obtenido en una fase previa, en la que además se haya optimizado su rendimiento de acuerdo a la arquitectura de la GPU utilizada (nucleos, hilos, contextos).
- Movimiento de datos desde la GPU a la CPU. Es el proceso inverso de la etapa 3 y la cantidad de información a llevar de nuevo a la memoria de la CPU dependerá del resultado del algoritmo de procesado. Una vez completada esta etapa, la aplicación estaría preparada para una nueva adquisición y volver de nuevo al paso 2 del esquema mostrado en la Figura 44.
- Liberación de los recursos. Con el fin de dejar a todo el sistema preparado para nuevas tareas, será necesario liberar los recursos y los dispositivos asociados a la adquisición y el procesado.

Como puede observarse el elemento encargado de supervisar cada una de las acciones a realizar por el sistema es la CPU, puede considerarse como el elemento *maestro*. Tanto la GPU, como la FPGA siempre van a estar gobernadas por instrucciones que reciban desde el hilo de ejecución la CPU, considerándose de esta manera como elementos *esclavos*.

Es necesario remarcar que todo este proceso está condicionado por la necesidad de consultar continuamente el estado de la FPGA (pooling) para determinar el estado de las transferencias entre este dispositivo y la GPU. Sería deseable que dicho proceso pudiese gestionarse mediante interrupciones, pero el driver que controla la tarjeta no soporta dicha funcionalidad. Caso de que existiera este mecanismo, el modelo sería distinto, e incluso la CPU quedaría aún más relegada a tareas de supervisión, control y ejecución de eventos asociados a los fines de transmisión de información, o finalización de los procesos ejecutados en la GPU.

4.8 Modelo mejorado basado en el uso de RDMA.

Partiendo de la propuesta desarrollada en el apartado anterior, y teniendo en cuenta que se pretende obtener una mejora en el tiempo del bucle adquisición-movimiento de datos-procesado, se ha realizado una optimización en una de estas fases. Además, esta mejora surge para dar respuesta a una de las debilidades detectadas en la fase de experimentación desarrollada en esta tesis. Para ello es necesario, en primer lugar conocer en profundidad cada uno de los elementos software involucrados y las alternativas tecnológicas existentes para afrontar esta innovación en sistemas de adquisición y procesado.

Para llevar a cabo esta mejora es necesario modificar y personalizar los elementos de la arquitectura software que provee el fabricante National Instruments descritos en el apartado 4.3. Lógicamente para que esto pueda realizarse es necesario disponer de todo el código fuente, tanto del driver (módulo *kernel*) del dispositivo físico, como de la librería que permite la interacción desde la zona de usuario (*C API functions*). Las consideraciones que se realizan a continuación emanan de la información suministrada por el fabricante a nivel de usuario y del profundo análisis llevado a cabo del código fuente existente.

El elemento principal de almacenamiento utilizado, desde el sistema de adquisición, para el movimiento de datos entre la FPGA y la CPU es una FIFO, estructura de memoria *First Input First Output*, y sirve como elemento que permite realizar transacciones DMA. Esta estructura de memoria está compuesta por dos partes, una que reside en la propia memoria de la FPGA y otra parte que reside en la memoria del *Host*. Entre estas dos partes se establece un canal de transmisión por DMA que es gobernado por el controlador DMA alojado en la tarjeta FPGA y para el caso de la familia de tarjetas utilizada (FlexRIO), recibe el nombre *CHINCH controller*. En la Figura 45 pueden verse los componentes de esta estructura FIFO.

Figura 45: Definición de una estructura FIFO-DMA

En la parte izquierda de la Figura 45 se hace referencia a la zona de memoria de esta estructura que reside en la FPGA y cuyas características (tamaño, tipos de datos) se han diseñado en el momento de la generación del bitfile que describe su comportamiento hardware. La parte derecha describe la zona de la estructura que reside en memoria del host, a la que tendrá acceso el usuario. Las transferencias entre ambas memorias las gobernará el controlador DMA alojado en la tarjeta FlexRIO. Por tanto para poder establecer un canal de comunicación directo entre la tarjeta de adquisición y otra memoria que no sea la memoria principal del sistema, será necesario reprogramar el controlador de DMA, indicándole las nuevas direcciones de memoria con las que realizar las transferencias de datos.

En el driver Linux del dispositivo existen mecanismos que permiten proporcionar al controlador, alojado en la tarjeta de adquisición, una lista de direcciones físicas de memoria que el kernel de Linux proporciona mediante llamadas al sistema de reserva de memoria. Esta tabla de segmentos de memoria, que el propietario denomina "*chunky links*", sirven para que el controlador pueda tener acceso a la memoria del host. Las páginas de memoria que el gestor de memoria del sistema reserva tienen por defecto en la mayoría de los casos un tamaño de 4K y no tienen por qué ser consecutivas. Además según se utilicen, será necesario mantener el espacio libre susceptible de escritura o lectura, tarea de la que se encarga también el propio controlador. En la Figura 46 se puede observar una tabla de los descriptores que utiliza el controlador DMA para gobernar las transferencias. Estas tablas son de vital importancia, dado que sus

entradas deben ser construidas con direcciones físicas pertenecientes al espacio de la memoria de la GPU

Figura 46: Tabla de descriptores “*chunky links*”

4.9 Tecnologia RDMA-GPUDirect de NVIDIA.

La introducción de la familia Kepler por parte del fabricante NVIDIA marcó la aparición de la tecnología GPUDirect-RDMA que permite una capacidad para las tarjetas de las familias profesionales Tesla o Quadro, que es trascendental en este modelo. Esta tecnología representa la posibilidad de realizar accesos directos a la memoria interna global de la GPU mediante DMA. En este modelo implica que es posible realizar un paso de datos directo entre FPGA y GPU, sin pasar por la memoria principal del sistema HOST CPU, siempre que se respeten determinadas restricciones impuestas por el fabricante y que en sucesivos epígrafes se dará cumplida cuenta.

Esta tecnología, presente únicamente en tarjetas profesionales de NVIDIA, desde CUDA SDK 5.5 da la posibilidad de realizar transferencias DMA entre la memoria de dispositivos alojados en el mismo y la memoria de la GPU.

Para ello ambos dispositivos utilizan el bus PCIexpress siempre y cuando cumplan una serie de restricciones:

- Ambos dispositivos deben compartir el mismo IOHUB.
- En el camino de transferencia entre ambos dispositivos solo pueden existir switches PCIexpress.
- No se puede utilizar en sistemas multiprocesador cuando cada dispositivo está conectado al IOHUB de cada procesador.

- Caso de que el sistema cuente con una unidad IOMMU, ha de desactivarse.
- Esta tecnología solo está disponible para sistemas Linux de 64 bits.

4.9.1 Modificaciones llevadas a cabo en el Software de la tarjeta FlexRIO

Una aplicación de adquisición de datos basada en FPGA necesita de varios elementos software o niveles que interactúan entre sí, permitiendo al usuario controlar los parámetros de dicho proceso y decidir el cometido de la información obtenida o generada. Para que desde zona de usuario se puedan realizar aplicaciones que adquieran información y dicha información sea transferida directamente a la memoria de la GPU, ha sido necesario realizar las modificaciones que a continuación se describen funcionalmente.

Espacio KERNEL.

Dado que es el nivel software más cercano al dispositivo, en el módulo kernel (también denominado como driver kernel), se encuentran las funciones básicas de control del hardware que interactúa con el resto del sistema y que es gobernado por el sistema operativo Linux. Debido a que en esta propuesta se trata de controlar funciones básicas de dos dispositivos: la FPGA y la GPU, es necesario que los módulos que gobiernan cada elemento estén cargados para poder completar la cooperación descrita anteriormente. Ello conllevará a que entre ambos existan dependencias, ya que desde el módulo de la tarjeta de adquisición se van a realizar llamadas a funciones básicas del hardware de la GPU, en concreto todas las relacionadas con el manejo de memoria de acceso DMA. Por ello es necesario disponer del código fuente de módulo kernel que NVIDIA proporciona para sus tarjetas, de esta forma es posible realizar la compilación que obtiene los símbolos usados en las dependencias existentes entre ambos drivers. Aunque los cambios llevados a cabo en el driver kernel de la tarjeta FlexRIO han sido numerosos, a continuación se describen los más relevantes. Además la incorporación de estos cambios ha tenido repercusión en el comportamiento de las funciones de manejo de movimiento de datos de DMA hacia o desde memoria principal del sistema CPU.

```
nirio_gpu_fifo_set_buffer(struct nirio_fifo *fifo,void *gpu_buff_ptr,size_t
num_bytes)
```

En esta función se crea la parte de memoria de la estructura FIFO que reside fuera de la FPGA. Habitualmente para reservar bloques que se solicitan al gestor de Linux, se utiliza la función `get_user_pages()`, pero en este caso dichas funciones deben ser sustituidas por las que permiten acceder la memoria de la GPU. En concreto la función `nvidia_get_user_pages()`, se ocupará de exponer las direcciones físicas del buffer correspondiente de la GPU.

Los parámetros que utiliza esta función llegan desde zona usuario (desde la función de librería que se describirá posteriormente), utilizando procedimientos IOCTL, y que son: dirección de memoria virtual en zona de usuario, y tamaño del bloque de memoria a utilizar. Antes de utilizar esta dirección de memoria virtual de GPU, es necesario comprobar que está correctamente alineada a espacios de 64KB, de no ser así es necesario aplicar patrones que fueren esta situación. Con esto dos parámetros se mapean tantas páginas de memoria como sean necesarias, teniendo en cuenta que las páginas de memoria física que ofrece la GPU son de 64 KB y los bloques de memoria que necesita la tabla de descriptores son de 4 KB. Por tanto cada página de memoria de GPU da lugar a 16 páginas de memoria, que servirán para construir las listas que permiten el enlace DMA.

Tanto en esta función, como en las que se derivan de ésta, también se han sustituido las funciones básicas del API de bajo nivel Linux DMA que permiten gestionar las direcciones físicas para el controlador. En vez de utilizar instrucciones que manejan listas *scatter-gather*, se han creado listas enlazadas, con el fin de no crear conflictos con la IOMMU del sistema. Esta última modificación es aplicable a la gestión DMA para los dos tipos de FIFOS (memoria CPU o GPU).

```
nirio_gpu_free_callback(void *data)
```

Cuando se ha terminado de utilizar una zona de memoria en GPU, es necesario realizar la liberación de dichos recursos. Para ello el driver de NVIDIA incorpora funciones del tipo `nvidia_put_user_pages()` o `nvidia_free_user_pages()` que devuelven al gestor de memoria de la GPU el buffer previamente reservado. Estas funciones se deben ejecutar cuando el usuario desea liberar dichos buffers. No obstante es obligatorio definir esta función *callback*, que se ejecutará

automáticamente cuando ha existido un error al gestionar la memoria reservada o bien si el usuario realiza una liberación del buffer en la aplicación antes de liberar el espacio de memoria en zona kernel.

Espacio de USUARIO.

La librería de zona de usuario, que permite controlar y gestionar los recursos de la tarjeta FlexRIO también ha sido modificada. Se han agregado nuevas funciones y otras relacionadas con los cambios descritos en el punto anterior han sido modificadas. En concreto la función que más relevancia cobra es la que permite establecer una FIFO DMA con espacio en memoria GPU. Su prototipo es el siguiente:

```
NiFpgaEx_ConfigureFifoGpu(session, fifo0, u64elementsReq, gpu_buffer)).
```

Es la función que se va a comunicar con las nuevas funciones creadas en zona kernel que permiten el nuevo camino para la transferencia de datos. Antes de utilizar esta función es necesario haber reservado memoria bajo el parámetro **gpu_buffer**, utilizando funciones del API de CUDA. El parámetro **fifo0** relaciona esta parte de memoria con la memoria existente en la tarjeta FlexRIO, y que debe haberse obtenido del fichero de recursos generado a partir del bitfile que configura la tarjeta de adquisición.

```
NiFpga_AcquireFifoReadElementsU64(session, fifo0, buffer, u64elementsReq, timeout, &elementsAcquired, &elementsRemaining))
```

En esta función se dispara el movimiento de datos entre la parte de la memoria FIFO de la tarjeta FPGA y la memoria de dicha estructura que reside en GPU. Es una función que conserva el prototipo existente en la librería original, aunque ha sido necesario modificarla internamente para dar soporte al nuevo manejo de memoria. De hecho la función chequea el tipo de fifo con la que se está trabajando y devuelve el puntero adecuado en el que se encuentran los datos adquiridos.

4.10 Arquitectura software del modelo mejorado.

En la Figura 47 se muestra cual sería el comportamiento de una aplicación de usuario que se beneficiara del nuevo driver del dispositivo obtenido. Resume las

etapas seguidas en una aplicación de adquisición y procesado, que evite copias innecesarias en la memoria principal del sistema CPU.

Figura 47: Arquitectura software que controla el uso de una GPU desde un "tercer" dispositivo

Los pasos necesarios para que el dispositivo pueda acceder a la memoria de la GPU son los siguientes:

1. Desde zona de usuario, se reserva memoria en la GPU. Esta memoria que solo puede ser manejada por procesos que se ejecuten en la propia GPU, debe gestionarse utilizando funciones propias del API de CUDA. Dicho bloque de memoria será el que se utilizará para hacer DMA.
2. La dirección virtual del comienzo de dicho bloque será suministrado al módulo kernel del dispositivo, para que utilizando funciones especiales de movimiento de datos *peer-to-peer* se pueda pedir y reservar dicho bloque al driver de la GPU. Además se asegura que esa memoria se protege para que ningún otro proceso pueda acceder a él.
3. El driver kernel de la GPU responderá con las direcciones físicas reservadas, en bloques de 64KB, para que el controlador DMA del dispositivo pueda manejarlas y realizar movimientos de datos. En la actualidad el máximo tamaño de memoria de GPU soportado es de 256 MB, y viene determinado por el valor de cada una de las "ventanas de

memoria" (PCI BAR) que un dispositivo puede mostrar al sistema operativo.

4. Desde la aplicación de usuario que gobierna los dos dispositivos se arrancará una transacción DMA.
5. Una vez los datos están en la memoria de la GPU, pueden ser procesados mediante funciones nativas de procesamiento paralelo o mediante el uso de librerías optimizadas.

4.11 Evaluación del rendimiento.

Las modificaciones llevadas a cabo en los apartados anteriores han sido exitosas en cuanto a su funcionalidad. A continuación se va a valorar la mejora de rendimiento cuando se utiliza el método tradicional de movimiento de datos frente a la nueva propuesta llevada a cabo.

Para realizar esta evaluación se han construido distintos tests que permiten medir distintos parámetros de rendimiento como son: el *ancho de banda* y la desviación estándar, utilizando las funciones básicas descritas anteriormente para distintos tamaños de bloque.

Con el fin de medir el tiempo de transferencia de un paquete de datos a través del bus PCIexpress se puede utilizar los siguientes mecanismos:

- Uso de Timers en el Host o en la propia GPU.
- Uso de alguna señal hardware en la tarjeta FPGA, que mediante su activación permita su lectura y medida en un osciloscopio o analizador lógico.
- Utilización de algún protocolo de marca de tiempos como es el caso del PTP-1588.
- Uso de Timers y contadores dentro de la propia FPGA que determinen el comienzo y la finalización de una transferencia.

En la fase de experimentación de esta tesis, se utilizó la primera opción para los casos de uso estudiados. En este caso, al contar con recursos hardware más potentes y precisos en la propia tarjeta de adquisición, la opción elegida ha sido el uso de recursos de la propia FPGA por ser además más independiente a la hora de realizar las medidas. La base de tiempos para los contadores existentes en la

tarjeta FPGA es el reloj base, que tal como indica el propio fabricante tienen una deriva casi despreciable. En cuanto al tamaño de estructura FIFO-DMA utilizado, se ha seleccionado un buffer de 8192 elementos de 64 bits en la memoria de la FPGA y un buffer 10 veces mayor en la memoria del host (CPU o GPU) del mismo tipo de elementos.

En el primer test, el ciclo de adquisición propuesto en el apartado 4.5 se ha simplificado de tal forma que no existe la fase de adquisición y la de procesado. Para ello los datos son generados internamente dentro de la FPGA, ya que solo interesa medir tasas de transferencias y su desviación estándar. La aplicación de usuario que gobierna el test lanza el movimiento continuo de datos interactuando sobre los elementos de control que controlan el hardware modelado en el bitfile de la FPGA.

Figura 48: Ancho de banda para distintos tamaños de bloque

En la Figura 48 se muestra la comparativa de tasa de transferencia o ancho de banda medido en MB/s para distintos tamaños de bloque utilizados. Es necesario reseñar que la gráfica que corresponde a la etiqueta “CPU” responde al método tradicional de movimiento de datos, en el que los datos son copiados previamente a la memoria de la CPU y después a la GPU. Estas dos copias de datos dependen de la tasa de transferencia que se puede obtener entre los movimientos DMA FPGA-CPU y CPU-GPU. El enlace DMA FPGA-CPU se

realiza utilizando 4 líneas PCI-Express 1.1, arrojando una teórica tasa de transferencia de 1GB/s. El enlace DMA entre CPU y GPU, para el hardware utilizado, es de 16 líneas PCIexpress 2.0 que proporciona una teórica tasa de transferencia 8GB/s. La máxima tasa de transferencia obtenida ronda los 550 MB/s y se obtiene cuando para tamaños de bloque mayores a 512KiB. Al margen de la doble copia, es necesario remarcar que nunca se alcanzan las velocidades teóricas de cada una de las normas PCI-Express, llegando en el mejor de los casos a un 80% del ancho de banda especificado.

La gráfica de etiqueta “GPU” corresponde al movimiento de datos utilizando RDMA desde la memoria de la FPGA a la memoria de la GPU. Los valores obtenidos entre 700 y 870 MB/s avalan una mejora notable para el nuevo método, llegando al máximo real del bus cuando el tamaño del bloque utilizado superior a 256 KiB.

El porcentaje de la mejora obtenida en la tasa de transferencia puede observarse en la Figura 49, en función también del tamaño de bloque utilizado para las transferencias. La mejora obtenida satura entorno al 60% para tamaños superiores a 512KiB. A partir de ese valor comienza a ser despreciable, la influencia en los tiempos medidos de factores como puede ser el acceso a los registros de control en la FPGA, que gobiernan el comportamiento del hardware, o el acceso a las funciones de la librería utilizada.

Figura 49: Porcentaje en la mejora del rendimiento GPU-CPU

La Figura 50 muestra la desviación estándar, expresada en %, para cada una de las dos realizaciones, en las que cabe destacar un mayor determinismo cuando se utiliza un solo movimiento de datos hacia la GPU.

Figura 50: Desviación estándar para distintos tamaños de bloque

4.12 Metodología propuesta

Conocidas las mejoras obtenidas gracias a la comunicación directa de datos entre la memoria de la tarjeta de adquisición y la memoria de la GPU, a continuación se enumeran las pautas que se deben seguir para construir una aplicación que aproveche esta solución. Para obtener una aplicación que dé soporte a un diagnóstico en fusión, es necesario contar con equipo humano en el que existen funciones concretas asignadas a distintos grupos organizativos de acuerdo a su nivel de especialización. Al menos será necesario disponer de expertos no solo a nivel científico, que se encarguen de especificar el diagnóstico, también será necesario contar con distintos grupos de ingenieros cualificados capaces de diseñar aplicaciones para sistemas de adquisición basados en tarjetas que usan FPGA, o bien capaces de elaborar algoritmos optimizados para computación paralela en GPU.

Los pasos necesarios, mostrado en la Figura 51, se detallan a continuación:

1. El punto de partida es la caracterización y especificación del diagnóstico a realizar por parte del equipo de científicos experto de la comunidad de fusión. De acuerdo a la experiencia acumulada, en algunos casos no se partirá desde cero, si no de resultados previos obtenidos en dispositivos de similares características.
2. Con la información aportada del diagnóstico, el equipo de ingenieros son responsables en primer lugar de especificar las características de las señales eléctricas que han de ser evaluadas o bien los parámetros de las imágenes susceptibles a analizar. En este caso las características del fenómeno físico determinará el tipo de módulo adaptador, descritos anteriormente, que permite a la tarjeta de adquisición NI FlexRIO capturar señales o imágenes. En el caso del sistema de procesado, la GPU, la selección se limita a utilizar alguna de las existentes en las gamas profesionales del fabricante NVIDIA, que se corresponden con las pertenecientes a la familia Tesla o Quadro. En esta fase también se deben identificar las necesidades que permitan establecer mecanismos de control y monitorización por parte del usuario.
3. El siguiente paso a completar será el desarrollo del algoritmo que da soporte al diagnóstico. En primer lugar es necesario diseñar el

hardware del dispositivo FlexRIO, de acuerdo al ciclo de trabajo descrito en el apartado 4.3. De este proceso recordar que se obtiene un fichero de descripción hardware (bitfile), y un fichero de descripción de recursos que permite su uso desde aplicaciones de usuario, (include.h). En segundo lugar el grupo de ingenieros expertos en computación paralela, serán encargados de elaborar las funciones de procesado en GPU. Los datos a procesar serán obtenidos de las estructuras FIFO, que comunican por DMA, la memoria de la tarjeta de adquisición y la memoria de la GPU. En esta fase es necesario el uso de distintas herramientas software, bajo distintos sistemas operativos. Para el caso de la tarjeta de adquisición es necesario utilizar LabView para FPGA en Windows y en el caso del código que se ejecuta en la GPU, se hará uso del SDK de NVIDIA.

4. Por último, con los recursos elaborados hasta el momento, se puede construir una aplicación de usuario que dé soporte al diagnóstico. En esta aplicación es necesario utilizar la librería de usuario y el driver modificado de la familia de tarjetas NI FlexRIO. El usuario deberá utilizar el conjunto de herramientas que permitan la integración de la aplicación en los sistemas de control e instrumentación distribuidos utilizados en entornos de fusión, de acuerdo a la selección del framework elegido. Existen distintas opciones para ello: DPD (analizada en el capítulo anterior), EPICS utilizando el modelo basado en control de dispositivos Asyndriver o bien EPICS utilizando el modelo basado en la capa de abstracción NDS (Nominal Device Support).

Con los pasos descritos se puede construir un sistema de adquisición de datos y procesado de altas prestaciones, en el que además será posible balancear la carga de proceso entre las dos arquitecturas existentes (FPGA y GPU). Es una solución con un alto grado de reconfiguración dada la posibilidad de actualizar con un esfuerzo menor el código que da soporte al algoritmo correspondiente al diagnóstico en estudio. La forma más rápida y directa sería sustituir el código paralelo que se ejecuta en la GPU, por una nueva versión u otro completamente distinto.

Figura 51: Metodología de diseño

Capítulo 5

5 EVALUACIÓN DEL MODELO Y CONCLUSIONES

5.1 Evaluación Del Modelo

Como medio para poder evaluar la metodología descrita en el capítulo anterior se ha construido una aplicación típica que puede dar soporte a distintos diagnósticos en los entornos de fusión. Se trata de una aplicación de adquisición de imágenes que utiliza una tarjeta FlexRIO con el módulo correspondiente al interface Camera Link y procesa las imágenes capturadas en una GPU, aprovechando el camino directo de movimiento de datos, entre las memorias de ambos dispositivos.

Es necesario hacer constar que esta aplicación ha sido testeada en un entorno real, en concreto en las instalaciones de ITER en Cadarache (Francia). En la evaluación llevada a cabo se han utilizado los elementos necesarios incluidos en el catálogo hardware para los Fast Controllers. Como framework para el desarrollo de la aplicación se ha elegido EPICS basado en NDS (Nominal Device Support), por lo que se hace necesario realizar una breve descripción del mismo.

5.1.1 Nominal Device Support

Como ya se adelantó en capítulo 2 el sistema de control distribuido EPICS, basa su ejecución en torno a procesos denominados IOCs. Estos procesos pueden ejecutarse en una misma máquina o en cualquiera de las pertenecientes a la misma red, y se encargan de interactuar con los dispositivos hardware o bien ejecutar procesos software de control, gestión o monitorización. La estructura interna de un IOC de EPICS, se muestra de nuevo en la Figura 52.

Figura 52: Arquitectura Software de un proceso IOC

Todas las comunicaciones que tienen lugar entre los records de EPICS y el driver kernel del dispositivo para interactuar con él, deben realizarse utilizando un módulo software denominado EPICS Device Support. Existen dos mecanismos de implementar este EPICS device support. El primero se basa en establecer el enlace entre el record y la librería que gestiona el dispositivo hardware mediante una sintaxis definida por el usuario siendo este el encargado de conocer cómo se debe gestionar el flujo de comunicación entre EPICS y el dispositivo hardware. Este mecanismo presenta el inconveniente de impedir la reutilización y la homogenización de los interfaces entre los records y los dispositivos hardware.

El segundo método se basa en el uso del módulo asynDriver. Este módulo simplifica parte del desarrollo de los EPICS device support al estandarizar la sintaxis de los records y definir el flujo de comunicación entre este y el dispositivo hardware. Es importante destacar que ninguna de las dos aproximaciones estandariza la denominación de las PVs ni define el comportamiento básico del driver en cuanto a su autómata de estados, interacción con él, etc. La complejidad

de desarrollar y mantener estos EPICS device support ha llevado a ITER a definir un modelo de implementación que se denomina Nominal Device Support (Isaev, Claesson et al. 2013). Este estándar se basa en el uso de un conjunto de clases en C++ que encapsulan la creación del driver mediante la capa asynDriver y simplifican el desarrollo de las interfaces de comunicación entre los records y el dispositivo.

El núcleo básico de clases que proporciona NDS ha sido concebido para soportar dispositivos de adquisición de datos, procesamiento y de temporización. Existen varias clases que conforman la caracterización de un dispositivo que son:

- Device class: define el comportamiento a nivel de dispositivo.
- Channel group class: incorpora el comportamiento de un conjunto de varios canales o grupo de canales.
- Channel class: define el comportamiento de un solo canal. La clase canal está implementada por defecto para señales analógica, digitales y para imágenes.

En la Figura 53 puede observarse la jerarquía existente entre todas las clases que conforman NDS.

Figura 53: Jerarquía de clases en NDS-EPICS

Para definir el comportamiento del driver NDS incluye una máquina de estados y varias PVs que permiten provocar transiciones y conocer el estatus del driver software. Esta máquina de estados también define el comportamiento del IOC ante determinadas condiciones como errores o respuestas no esperadas. En la Figura 54 puede observarse un ejemplo del autómata que da soporte en concreto a la clase dispositivo.

La gran ventaja de utilizar NDS, es partir de una serie de plantillas (*templates*) en las que ya existen predefinidos una clase de objeto y el autómata asociado a cada uno de ellos. El usuario ha de completar y definir aquellos objetos que necesite, así como definir el comportamiento atribuible a cada uno de sus estados.

Figura 54: Autómata de estados para la clase *device*

5.1.2 Descripción de la implementación.

Para desarrollar la aplicación se ha aplicado la metodología descrita en el capítulo anterior, de esta forma es posible obtener un caso de uso que posibilite validar la propuesta. El ámbito de uso de la solución podrían ser todos aquellos diagnósticos que necesiten adquisición de imágenes y procesamiento paralelo con GPU. A continuación se enumeran los pasos seguidos, comentando las particularidades correspondientes a cada fase.

1. Caracterización y especificación del diagnóstico.

Dado el carácter genérico de la solución, no ha sido necesario tomar ningún caso concreto de diagnóstico. Como se ha señalado, son susceptibles todos aquellos diagnósticos que tengan como naturaleza, el procesamiento de imágenes. Una alternativa para caracterizar el diagnóstico, sería utilizar datos simulados, caso de contar con imágenes capturadas o grabadas previamente.

2. Selección hardware.

Esta elección está basada en la naturaleza del diagnóstico a realizar y condiciona el desarrollo de las siguientes etapas. Dado que este caso se trata de imágenes, es necesario utilizar un módulo adaptador para la familia NI FlexRIO, que permita capturar imágenes. El dispositivo NI 1483 permite obtener imágenes utilizando el interface Camera Link, uno de los estándares adoptados en la comunidad de fusión para cámaras de altos requisitos de resolución y tasas de captura. La tarjeta de adquisición basada en FPGA, según el modelo propuesto, debe pertenecer a la familia FlexRIO, y entre las opciones disponibles (8 modelos diferentes) se seleccionará aquella que satisfaga la cantidad de memoria necesaria, canales DMA o capacidad de la FPGA incorporada. La selección de la tarjeta GPU, está delimitada a las familias profesionales de NVIDIA, Tesla o Quadro, El modelo concreto determinará los recursos computacionales disponibles para procesamiento paralelo, tanto a nivel de multiprocesadores como de memoria interna de la tarjeta.

Los componentes utilizados, incluidos en el catálogo de ITER como estándares para sus fast controllers, son la tarjeta NI FlexRIO 7966R alojada en un chasis NI 1065, a la que se conecta el módulo NI 1483 para acceder al

interface Camera Link. Como GPU, se ha elegido una NVIDIA Quadro k4200, incluida también en el catálogo. El diagrama de la arquitectura hardware completa del sistema se muestra en la Figura 55.

Figura 55: Plataforma hardware utilizada

La Figura 56 muestra un fast controller perteneciente a las instalaciones de ITER, en las que se testeó la aplicación, en el que se observa la CPU PICMG en la parte inferior, el chasis PXIe NI-1065 en la parte central del rack y una chasis ATCA del fabricante ELMA en la parte superior del mismo.

Figura 56: Fast controller (fc18-3) en ITER con chasis PXIe y ATCA

En la Figura 57 pueden observarse dos cámaras que utilizan el interface CameraLink del fabricante Mikroton, modelo EOSens 3CL. Estas cámaras pueden capturar hasta 600 imágenes por segundo, alcanzando la máxima transferencia posible de la norma, 850 MB/s.

Figura 57: Cámaras EOSens 3CL con interface Cameralink

3. Diseño del comportamiento del hardware y desarrollo del algoritmo.

La funcionalidad del hardware configurado en la FPGA vendrá determinado por los requisitos derivados del tipo de diagnóstico (tipo de señales, etc), las necesidades de procesamiento y las capacidades de la arquitectura. En este caso la FPGA se dedicará a controlar el módulo adaptador que permite la

captura de imágenes. Es necesario por tanto determinar el modelo concreto de cámara encargada de capturar las imágenes, ya que sus especificaciones permitirán determinar el modo de trabajo perteneciente a la norma, y por tanto el diseño a realizar en el hardware de la FPGA.

En el estándar Camera Link será necesario conocer por ejemplo el número de píxeles que entrega la cámara por ciclo de reloj (*pixelClock*), o el número de bits utilizados por píxel. Una configuración 25 MHz Tap 10x16, consiste en que la cámara genera datos utilizando un reloj de 25 MHz, y en cada ciclo transfiere 10 píxeles de 16 bits cada uno. Estos y otros parámetros de la norma deben ser gobernados por el hardware diseñado, a través de controles e indicadores. En el diseño generado debe existir al menos una estructura DMA FIFO, encargada de recibir los bits correspondientes a los píxeles generados por la cámara y transferirlos a la memoria de la GPU. Por último, en esta aplicación se han tomado como ejemplo, dos algoritmos en la GPU: uno encargado de invertir todos los píxeles de la imagen y otro encargado de realizar un filtro paso bajo a la imagen. Dichos algoritmos han sido desarrollados utilizando las herramientas del SDK CUDA de NVIDIA, para poder obtener el máximo rendimiento a la arquitectura seleccionada (Quadro K4200) con 1344 *cuda*cores y 4GB de memoria GDDR5.

4. Construcción de la aplicación e integración en EPICS usando NDS.

En la implementación ha sido necesario la definición de una clase device, que define un comportamiento básico en base a tres estados: *init*, *on* y *off*. Las funcionalidades recogidas en cada estado se describen a continuación:

Estado *init*: inicializa la FPGA cargando su bitfile y configura todos los registros y controles para que se exista una comunicación correcta entre la tarjeta FPGA y el módulo adaptador CameraLINK. Además en este estado la GPU es consultada para determinar si tiene las capacidades requeridas de acceso RDMA. Una vez comprobado este requisito es inicializada y se reservan los recursos necesarios para definir la parte de la estructura FIFO-DMA en su memoria.

Estado *on*: En este estado se dispara la adquisición de imágenes con la configuración del estándar Camera Link adecuada. Cada imagen capturada

se lleva directamente a memoria de la GPU. En este estado es posible elegir el tipo de procesado que va a realizar la GPU con las imágenes obtenidas. Además se pone la clase channel en el estado de procesando.

Estado Channel processing: Para esta realización solo se ha utilizado un canal dentro de la clase channel. Es en esta clase en la que se dispara la recepción de las imágenes desde la GPU y la publicación mediante las correspondientes PVs de EPICS.

Estado Off: Se detiene la adquisición si ésta estaba arrancada. El canal/canales y el grupo/grupos de canales se desactivan.

La arquitectura hardware/software que resume el comportamiento de la aplicación se ilustra en la Figura 58. En la parte inferior residen los dispositivos hardware a controlar (GPU y FlexRIO con módulo Cameralink). Dichos dispositivos se controlan mediante sus device drivers, cargados en zona kernel. Para poder acceder desde la aplicación NDS a la configuración de la adquisición o al procesado, es necesario realizar llamadas a las funciones de las librerías, que residen en zona de usuario. Todo el software utilizado funciona bajo el sistema operativo Linux, en concreto se ha utilizado la distribución Codac Core System 5.1, versión de Red Hat Enterprise que ITER ha personalizado para la arquitectura software que controla todos sus sistemas. En el epígrafe 2.1.7 de esta tesis se hace una breve descripción de dicha distribución.

Figura 58: Arquitectura software/hardware de la aplicación NDS obtenida

Con el fin de no vincular la aplicación a una cámara concreta, se ha utilizado un simulador que permite generar imágenes de acuerdo a la norma Camera Link, seleccionando una configuración determinada. Este simulador reside en una máquina diferente y ha de ser conectado al módulo NI 1483 utilizando los buses que define la norma.

La forma de interactuar con el IOC que gobierna toda la aplicación es mediante el acceso a las PVs que permiten gobernar los estados del autómata anteriormente descrito. Sin embargo para poder gestionar de una manera más intuitiva el control de la aplicación, se ha construido un interface de usuario que permite establecer el estado en el que se encuentra el autómata. Se ha utilizado la herramienta Control System Studio (CSS), incorporada también en la distribución Linux mencionada anteriormente. Básicamente sirve para construir

paneles (denominados *OPI panels*) con una serie de indicadores y controles que permiten una conexión con las PVs que instancia el proceso IOC.

La Figura 59 muestra dicho interface de usuario. En el mismo existen controles que permiten transitar entre los estados del autómata, iniciar la adquisición o seleccionar el tipo de procesamiento a realizar por la GPU. También se muestra la imagen original y la resultante tras aplicar el algoritmo de procesamiento elegido.

Figura 59: OPI Panel – Interface de usuario para gobernar la aplicación

Tal y como se ha comentado, para el desarrollo de la aplicación se ha utilizado la metodología descrita en el epígrafe 4.8 del capítulo anterior. El resultado ha sido satisfactorio, y además ha permitido añadir un último detalle importante al ciclo de desarrollo utilizado. Dado que las herramientas que se han utilizado para obtener la aplicación EPICS-NDS (integradas en el SDD de ITER), están gestionadas por una cadena de desarrollo que no permite integrar el acceso a las herramientas utilizadas del SDK de NVIDIA, ha sido necesario construir un mecanismo software que posibilite un fácil acceso a la programación de la GPU. Se ha desarrollado un *wrapper* en forma de librería en la que se han empaquetado todas las funciones, tanto las de manejo y liberación de recursos, como las propias funciones nativas *kernel* de procesamiento paralelo. Con esta idea además de solventar el problema antes descrito, se añade la posibilidad de modificar o añadir los algoritmos de procesamiento que el usuario determine.

5.2 Conclusiones

Durante el periodo de investigación que ha dado lugar a esta tesis doctoral, se han elaborado distintas contribuciones que han servido como hilo conductor para la obtención de la metodología final propuesta. El método y las técnicas elaboradas permiten afrontar y resolver algunos de los retos planteados en los sistemas de adquisición de datos para los diagnósticos de los dispositivos de fusión termonuclear por confinamiento magnético. Dada la evolución hacia dispositivos de pulso largo en los que existe una alta generación de datos a procesar, la comunidad puede beneficiarse de estas experiencias, con el fin de diseñar diagnósticos que cuenten con sistemas de procesamiento de altas prestaciones, que permitan extraer mucho más conocimiento útil a partir de la ingente cantidad de datos generados.

La constatación de la validez del uso de GPUs, como arquitectura que permite cubrir las necesidades mencionadas, permite prever una estandarización para su uso en los principales dispositivos de fusión. La posibilidad de incluir esta arquitectura en sistemas de adquisición y procesamiento en tiempo real, será una cuestión a valorar dado el grado de determinismo encontrado en la ejecución de los algoritmos en los casos de uso analizados. Por otra parte la mejora de rendimiento de estas arquitecturas de proceso paralelo, frente a arquitecturas tradicionales, permite abordar el desarrollo de complejos algoritmos en el futuro de acuerdo a la evolución que exija la comunidad de fusión. El coste para obtener el máximo aprovechamiento a esta tecnología, debe destinarse a la optimización de procesos de cálculo existentes o la creación de otros nuevos. Es necesario hacer notar que el uso de GPUs está encaminado a resolver procesos de cálculo en los que se computan una gran cantidad de datos, es el escenario ideal para obtener su potencial. Por el contrario, en aquellos procesos en los que el tamaño del bloque de datos sea pequeño, conviene considerar otras arquitecturas de procesamiento paralelo, tal como la propia FPGA integrada en el sistema de adquisición o incluso el uso de la propia CPU.

En el modelo obtenido se ha reducido la distancia existente entre el sistema de adquisición y el sistema de procesamiento, utilizando dispositivos comerciales disponibles en el catálogo de ITER, lo cual representa una solución flexible, de cómoda mantenibilidad, cuestión que no es alcanzable en sistemas ad-hoc. El acuerdo de colaboración con la empresa National Instruments, posibilita que en un futuro se puedan incorporar las nuevas funcionalidades desarrolladas a otros productos y puedan ser aprovechadas por cualquier disciplina que requiera los logros aquí obtenidos.

La integración del modelo en los sistemas de control e instrumentación distribuidos ha servido para establecer el procedimiento a seguir para la realización de futuros diagnósticos con altos requerimientos computacionales. La evaluación resuelta con éxito, en las instalaciones de ITER, conlleva a concluir que la solución aportada va a ser adoptada con toda seguridad por la comunidad de fusión en un breve plazo de tiempo.

Por todo lo anterior se puede asegurar que se ha validado la hipótesis planteada al inicio de esta tesis, de forma que...

Si es posible (y se ha logrado) incrementar la capacidad local de procesamiento en tiempo real de los sistemas de adquisición de datos de los dispositivos de fusión nuclear mediante el uso de GPUs de forma que se puedan afrontar los retos que se plantean de cara a su operación en pulso largo.

5.3 Aportaciones

Las investigaciones y metodología presentada en esta tesis, a través de las publicaciones realizadas y la participación de distintos proyectos de investigación competitivos, han permitido realizar las siguientes contribuciones a la comunidad científica dedicada a los entornos de fusión:

- Aportación de un modelo y una metodología de desarrollo para poder integrar GPUs como elementos de procesamiento local en los sistemas de adquisición de datos incluyendo su integración con los Sistemas de Control de Supervisión y Adquisición de Datos (SCADA).
- Participación en la especificación y diseño de los primeros prototipos de los Fast Plan Controller System para ITER basados en tecnología PXIe. Concretamente se han especificado las tarjetas disponibles comercialmente más adecuadas para su inclusión en estos sistemas, determinando los modelos y tecnologías más adecuados.
- Implementación a través de varios casos de uso reales de distintos algoritmos de procesamiento en GPU en varios dispositivos de fusión de referencia a nivel internacional. Además de servir como ejemplo de aplicación de la metodología desarrollada el trabajo realizado permite que la comunidad pueda extrapolar las conclusiones obtenidas a futuros diagnósticos susceptibles de realizar procesamiento paralelo de altas prestaciones.

- Modificación del software (Linux Kernel Driver y Librerías) de las tarjetas de adquisición de datos basadas en FPGA del fabricante National Instruments. Esta modificación permite que se puedan transferir datos directamente entre este tipo de tarjetas y las GPUs sin intervención de la CPU, lo que optimiza el rendimiento de aquellos sistemas en los que convivan ambas tecnologías abriendo la puerta a numerosas aplicaciones dentro y fuera del campo de la fusión termonuclear. La metodología seguida sienta las bases y abre el camino de cómo ampliar esta funcionalidad a otras tarjetas de este y otros fabricantes.
- Realización de la primera implementación completa y funcional de un sistema de adquisición y procesamiento de imágenes con capacidad de proceso en GPUS utilizando los estándares utilizados en ITER. Dicha implementación también supone el primer desarrollo que aprovecha las modificaciones mencionadas previamente.
- Participación en el desarrollo de la librería IRIO, que permite un ciclo de desarrollo directo para la realización de aplicaciones de adquisición de datos y procesamiento bajo EPICS. En concreto se ha aportado toda la parte que da soporte a movimientos DMA entre la familia de tarjeta FlexRIO y las GPU del fabricante NVIDIA.

5.4 Producción científica y transferencia de tecnología.

A continuación se enumeran los resultados aportados por el trabajo de investigación realizado en esta tesis agrupados en diferentes categorías.

5.4.1 Proyectos financiados en convocatorias competitivas.

El desarrollo de la tesis ha sido posible gracias a la financiación captada en convocatorias competitivas a través de los siguientes proyectos.

- Nombre del proyecto: Aplicación de sistemas de adquisición de datos inteligentes en dispositivos de fusión termonuclear.
Entidad/es financiadora/s: Ministerio de Ciencia e Innovación.
Nombre del programa: Plan Nacional de I+D+i (2009-2012).
Código entidad financiadora: ENE2009-10280.
Fecha de inicio-fin: 01/01/2010-31/12/2012.
Cuantía total: 114.950 €.

- Nombre del proyecto: Análisis de datos basados en aprendizaje automático y sistemas inteligentes de adquisición de datos: modelos avanzados para entornos de fusión.
Entidad/es financiadora/s: Ministerio de Economía y Competitividad.
Nombre del programa: Plan Nacional de I+D+i (2012-2015).
Código entidad financiadora: ENE2012-38970-C04-04.
Fecha de inicio-fin: 01/01/2013 - 31/12/2015.
Cuantía total: 106.000 €.

- Nombre del proyecto: Fast Plant System Controller Targeting Data Acquisition
Entidad/es financiadora/s: EURATOM - Comunidad Europea de la Energía Atómica
Nombre del programa: Contrato de Asociación EURATOM-CIEMAT
Cód. según financiadora: FU07-CT-2007-00050
Fecha de inicio-fin: 1/02/2011-1/2/2012
Cuantía total: 58.150 €.

- Nombre del proyecto: Software Support Bundle for FlexRIO/NVIDIA GPU
Entidad/es financiadora/s: ITER IO.
Nombre del programa: ITER International Fusion Project.
Código entidad financiadora: IO/CT/14/4300001117.
Fecha de inicio-fin: 01/01/2015 - 30/11/2015.
Cuantía total: 25.000 €.

- Nombre del proyecto: EPICS Device Support Module for NI RIO boards
Entidad/es financiadora/s: ITER IO.
Nombre del programa: ITER International Fusion Project.
Código entidad financiadora: IO/CT/15/4300001145.
Fecha de inicio-fin: 23/03/2015-31/12/2016.
Cuantía total: 93.000 €.

5.4.2 Artículos en revistas indexadas.

- Autores: J. Nieto, G. de Arcas, J. Vega, M. Ruiz, J.M López, E. Barrera, A. Murari, A. Fonseca
Título: Exploiting Graphic Processing Units Parallelism to Improve Intelligent Data Acquisition System Performance in JET's Correlation Reflectometer
Revista: IEEE TRANSACTIONS ON NUCLEAR SCIENCE
Año: 2011
Indicios de calidad: Revista indexada en el JCR (Q1: 8/35).

- Autores: J. Nieto, G. de Arcas, M. Ruiz, J. Vega, J.M. López, E. Barrera, R. Castro, D. Sanz, N. Utzel, P. Makijarvi, L. Zabeo
Título: A GPU-based real time high performance computing service in a fast plant system controller prototype for ITER
Revista: Fusion Engineering and Design
Año: 2012
Indicios de calidad: Revista indexada en el JCR (Q3: 18/34).
- Autores: M. Ruiz, J. Vega, R. Castro, D. Sanz, J.M. López, G. de Arcas, E. Barrera, J. Nieto, B. Gonçalves, J. Sousa, B. Carvalho, N. Utzel, P. Makijarvi
Título: ITER Fast Plant System Controller prototype based on PXIe platform
Revista: Fusion Engineering and Design
Año: 2012
Indicios de calidad: Revista indexada en el JCR (Q3: 18/34).
- Autores: R. Castro, J.A. Romero, J. Vega, J. Nieto, M. Ruiz, D. Sanz, E. Barrera, G. De Arcas
Título: Soft real-time EPICS extensions for fast control: A case study applied to a TCV equilibrium algorithm
Revista: Fusion Engineering and Design
Año: 2014
Indicios de calidad: Revista indexada en el JCR (Q2: 11/34).
- Autores: J. Nieto, G. de Arcas, M. Ruiz, R. Castro, J. Vega, P. Guillén
Título: A high throughput data acquisition and processing model for applications based on GPUs
Revista: Fusion Engineering and Design
Año: 2015
Indicios de calidad: Revista indexada en el JCR (Q2: 11/34).
- Autores: J. Nieto, D. Sanz, P. Guillén, S. Esquembri, G. de Arcas, M. Ruiz, J. Vega, R. Castro
Título: High Performance Image Acquisition and Processing Architecture for Fast Plant System Controllers based on FPGA and GPU.
Revista: Fusion Engineering and Design
Año: enviado para su revisión en 2015, pendiente de publicación en 2016
Indicios de calidad: Revista indexada en el JCR (Q2: 11/34).

5.4.3 Congresos internacionales.

- *Título del trabajo:* Exploiting Graphic Processing Units Parallelism to Improve Intelligent Data Acquisition System Performance in JET's Correlation Reflectometer

Autores: J. Nieto, G. de Arcas, J. Vega, M. Ruiz, J.M López, E. Barrera, A. Murari, A. Fonseca

Nombre del congreso: 17th Real Time Conference

Ciudad de celebración: Lisboa (Portugal)

Fecha de celebración: 2010

Indicios de calidad: Congreso Internacional con revisión por pares

- *Título del trabajo:* A GPU-based real time high performance computing service in a fast plant system controller prototype for ITER

Autores: J. Nieto, G. de Arcas, M. Ruiz, J. Vega, J.M. López, E. Barrera, R. Castro, D. Sanz, N. Utzel, P. Makijarvi, L. Zabeo

Nombre del congreso: 8th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research

Ciudad de celebración: San Francisco (USA)

Fecha de celebración: 2011

Indicios de calidad: Congreso Internacional con revisión por pares

- *Título del trabajo:* ITER Fast Plant System Controller prototype based on PXIe platform

Autores: M. Ruiz, J. Vega, R. Castro, D. Sanz, J.M. López, G. de Arcas, E. Barrera, J. Nieto, B. Gonçalves, J. Sousa, B. Carvalho, N. Utzel, P. Makijarvi

Nombre del congreso: 8th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research

Ciudad de celebración: San Francisco (USA)

Fecha de celebración: 2011

Indicios de calidad: Congreso Internacional con revisión por pares

- *Título del trabajo:* Soft real-time EPICS extensions for fast control: A case study applied to a TCV equilibrium algorithm

Autores: R. Castro, J.A. Romero, J. Vega, J. Nieto, M. Ruiz, D. Sanz, E. Barrera, G. De Arcas

Nombre del congreso: 9th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research

Ciudad de celebración: Hefei (China)

Fecha de celebración: 2013

Indicios de calidad: Congreso Internacional con revisión por pares

- *Título del trabajo:* A high throughput data acquisition and processing model for applications based on GPUs

Autores: J. Nieto, G. de Arcas, M. Ruiz, R. Castro, J. Vega, P. Guillen

Nombre del congreso: 28th Symposium on Fusion Technology

Ciudad de celebración: San Sebastián (España)

Fecha de celebración: 2014

Indicios de calidad: Congreso Internacional con revisión por pares

- *Título del trabajo:* High Performance Image Acquisition and Processing Architecture for Fast Plant System Controllers based on FPGA and GPU
Autores: J. Nieto, D. Sanz, P. Guillén, G. de Arcas, M. Ruiz, J.Vega, R. Castro
Nombre del congreso: 10th IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research
Ciudad de celebración: Ahmedabad Gujarat, India
Fecha de celebración: 2015
Indicios de calidad: Congreso Internacional con revisión por pares

5.4.4 Propiedad Intelectual y Transferencia de Resultados.

El modelo propuesto y su metodología se han transferido con éxito a la organización ITER IO en forma de un nuevo producto que ha sido incluido en el catálogo de soluciones estándar para los futuros diagnósticos del dispositivo ITER. Siendo este el dispositivo de referencia a nivel internacional actualmente en la comunidad de fusión se puede considerar que el trabajo realizado tiene por tanto carácter de “Desarrollo tecnológico relevante” según se ha recogido en diferentes convocatorias de la Comisión Nacional Evaluadora de la Actividad Investigadora, como por ejemplo en la del año 2008 . Es de esperar por tanto que la tecnología generada en esta investigación permanezca en uso en la comunidad de fusión, extendiéndose posiblemente a otros dispositivos, y seguramente a otros campos como otros experimentos científicos.

El desarrollo de este producto, denominado Software Support Bundle for FlexRIO/NVIDIA GPU, ha requerido la colaboración con la empresa National Instruments para realizar modificaciones sustanciales en el código de su driver para los dispositivos de la familia FlexRIO con objeto de incluir el modelo de interconexión propuesto en esta tesis para su manejo junto con tarjetas GPU de NVIDIA. El trabajo realizado no ha consistido en la realización de un prototipo, sino que se ha participado en el proceso completo de re-diseño y producción del nuevo producto hasta su puesta en servicio.

Este trabajo ha supuesto transferir parte de los resultados de esta investigación a la citada empresa, formalizado a través de un acuerdo de cesión de los derechos

de explotación donde se reconoce expresamente la autoría de los investigadores de la UPM involucrados en este trabajo en la parte del producto final que se ha generado.

5.5 Líneas Futuras

Las aportaciones realizadas en esta tesis, deben servir como punto de inicio para futuras líneas de investigación tanto dentro como fuera de la comunidad de fusión como medio en busca de su ansiado logro, obtener energía de manera eficiente utilizando una fuente inagotable. A continuación se trazan algunos caminos que esbozan retos a cubrir en un futuro y que darán lugar a distintas investigaciones:

- Partiendo de la solución desarrollada que permite un paso directo de datos desde un sistema basado en FPGA hasta la memoria de la GPU, será necesario exportar dicho procedimiento a otros dispositivos utilizados en los sistemas de adquisición y procesamiento en fusión. Dentro de dichos dispositivos cabe la utilización de tarjetas de adquisición con altas tasas de muestreo o capturadoras de imágenes de ultra-alta resolución. Además fijando el punto de atención en ITER, y dado que su catálogo puede considerarse en constante actualización, será necesario aplicar las técnicas aplicadas a otras tecnologías, no solo basadas en PXIe, sino también a arquitecturas microTCA, en plena expansión.
- Adaptar la solución a sistemas operativos con extensiones en tiempo real (Red Hat Enterprise MRG, MARTE) a medida que los fabricantes soporten e incorporen dicha capacidad en sus productos mediante nuevos drivers y librerías de desarrollo.
- Desarrollar un catálogo de algoritmos, en estrecha colaboración con la comunidad de fusión, que permita incrementar el aprovechamiento de la información resultante tras las descargas. Serán los científicos junto a los desarrolladores y programadores los encargados de obtener el máximo rendimiento de arquitecturas de procesamiento paralelo, utilizando llegado el caso estructuras de computación distribuida con múltiples GPUs o bien estructuras HPC.
- Explorar el uso de otras herramientas software que permitan computación en paralelo en distintas arquitecturas hardware (FPGAs, GPUs o DSPs). En la actualidad OpenCL, es un firme candidato a tener en cuenta, dada su disponibilidad para todas arquitecturas hardware mencionadas.

Capítulo 6

6 REFERENCIAS

- ABADIE, L., DI MAIO, F., KLOTZ, W., MAHAJAN, K., STEPANOV, D., UTZEL, N. and WALLANDER, A., 2012. The self-description data configuration model. *Fusion Engineering and Design*, **87**(12), pp. 2213-2217.
- AYMAR, R., 1997. The ITER project. *Plasma Science, IEEE Transactions on*, **25**(6), pp. 1187-1195.
- BARRERA, E., RUIZ, M., LOPEZ, S., MACHON, D. and VEGA, J., 2005. PXI-based architecture for real time data acquisition and distributed dynamical data processing, *Real Time Conference, 2005. 14th IEEE-NPSS 2005*, pp. 4 pp.
- BASLER, COGNEX, CORECO, DALSA, DATA TRANSLATION, DATACUBE, EPIX, EURESYS, FORESIGHT IMAGINING, INTEGRAL TECHNOLOGIES, MATROX, NATIONAL INSTRUMENTS, and PULNIX AMERICA/INTEGRAL TECHNOLOGIES, 2000, 2015-last update. Available: <http://www.imagelabs.com/wp-content/uploads/2010/10/CameraLink5.pdf> [10/1, 2015].
- CARVALHO, B.B., BATISTA, A.J.N., CORREIA, M., NETO, A., FERNANDES, H., GONÇALVES, B. and SOUSA, J., 2010. Reconfigurable ATCA hardware for plasma control and data acquisition. *Fusion Engineering and Design*, **85**(3-4), pp. 298-302.
- CASTRO, R., ROMERO, J.A., VEGA, J., NIETO, J., RUIZ, M., SANZ, D., BARRERA, E. and DE ARCAS, G., 2014. Soft real-time EPICS extensions for fast control: A case study applied to a TCV equilibrium algorithm. *Fusion Engineering and Design*, **89**(5), pp. 638-643.
- CASTRO, R., VEGA, J., RUIZ, M., DE ARCAS, G., BARRERA, E., LÓPEZ, J.M., SANZ, D., GONÇALVES, B., SANTOS, B., UTZEL, N. and MAKIJARVI, P., 2012. NetCDF based data archiving system applied to ITER Fast Plant System Control prototype. *Fusion Engineering and Design*, **87**(12), pp. 2223-2228.
- CHIARIELLO, A.G., FORMISANO, A. and MARTONE, R., 2013a. Fast magnetic field computation in fusion technology using GPU technology. *Fusion Engineering and Design*, **88**(9-10), pp. 1635-1639.
- CHIARIELLO, A.G., FORMISANO, A. and MARTONE, R., 2013b. Fast magnetic field computation in fusion technology using GPU technology. *Fusion Engineering and Design*, **88**(9-10), pp. 1635-1639.

- CRÉMY, C., VEGA, J., SÁNCHEZ, E., DULYA, C.M. and PORTAS, A., 1999. Multiprocessor architecture to handle TJ-II VXI-based digitization channels. *Review of Scientific Instruments*, **70**(1), pp. 513-516.
- DE ARCAS, G., LÓPEZ, J.M., RUIZ, M., BARRERA, E., VEGA, J., MURARI, A., FONSECA, A. and JET-EFDA CONTRIBUTORS, 2008. Self-adaptive sampling rate data acquisition in JET's correlation reflectometer. *Review of Scientific Instruments*, **79**(10), pp. 10F336.
- DI MAIO, F., 2015, 2015-last update, CODAC Core System Overview. Available: <https://user.iter.org/default.aspx?uid=34SDZ5&version=v5.1> [2015, 2015].
- DIAMOND LIGHT SOURCE, 2015-last update. Available: <http://www.diamond.ac.uk/Science/Computing/Controls.html>.
- DUSTIN, F. and GE GPU TECH CONFERENCE, 2015-last update. Available: <http://on-demand.gputechconf.com/gtc/2013/presentations/S3266-GPUDirect-RDMA-Green-Multi-GPU-Architectures.pdf>.
- EPICS, 2015a-last update. Available: <http://www.aps.anl.gov/epics/license/index.php> [2015].
- EPICS, E., 2015, 2015b-last update, Experimental Physics and Industrial Control System. Available: <http://www.aps.anl.gov/epics/>.
- GIANNONE, L., CERNA, M., COLE, R., FITZEK, M., KALLENBACH, A., LÜDDECKE, K., MCCARTHY, P.J., SCARABOSIO, A., SCHNEIDER, W., SIPS, A.C.C., TREUTTERER, W., VRANCIC, A., WENZEL, L., YI, H., BEHLER, K., EICH, T., EIXENBERGER, H., FUCHS, J.C., HAAS, G., LEXA, G., MARQUARDT, M., MLYNEK, A., NEU, G., RAUPP, G., REICH, M., SACHTLEBEN, J., SCHUHBECK, K.H., ZEHETBAUER, T., CONCEZZI, S., DEBELLE, T., MARKER, B., MUNROE, M., PETERSEN, N. and SCHMIDT, D., 2010. Data acquisition and real-time signal processing of plasma diagnostics on ASDEX Upgrade using LabVIEW RT. *Fusion Engineering and Design*, **85**(3-4), pp. 303-307.
- GIANNONE, L., EICH, T., FUCHS, J.C., RAVINDRAN, M., RUAN, Q., WENZEL, L., CERNA, M. and CONCEZZI, S., 2011. Data acquisition and real-time bolometer tomography using LabVIEW RT. *Fusion Engineering and Design*, **86**(6-8), pp. 1129-1132.

- GIANNONE, L., SCHNEIDER, W., MCCARTHY, P.J., SIPS, A.C.C., TREUTTERER, W., BEHLER, K., EICH, T., FUCHS, J.C., HICKS, N., KALLENBACH, A., MARASCHEK, M., MLYNEK, A., NEU, G., PAUTASSO, G., RAUPP, G., REICH, M., SCHUHBECK, K.H., STOBER, J., VOLPE, F., ZEHETBAUER, T., CERNA, M., DEBELLE, T., MARKER, B., MCCASLIN, S., MUNROE, M. and WENZEL, L., 2009. Real time magnetic field and flux measurements for tokamak control using a multi-core PCI Express system. *Fusion Engineering and Design*, **84**(2–6), pp. 825-828.
- GONÇALVES, B., SOUSA, J., CARVALHO, B.B., BATISTA, A., NETO, A., SANTOS, B., DUARTE, A., VALCÁRCEL, D., ALVES, D., CORREIA, M., RODRIGUES, A.P., CARVALHO, P.F., RUIZ, M., VEGA, J., CASTRO, R., LÓPEZ, J.M., UTZEL, N. and MAKIJARVI, P., 2012. ITER fast plant system controller prototype based on ATCA platform. *Fusion Engineering and Design*, **87**(12), pp. 2024-2029.
- HEESTERMAN, P., SALL, I., GIRAUD, C., ZASTROW, K., MEIGS, A., FELTON, R. and JOFFRIN, E., 2003. Real-time analysis of charge-exchange spectroscopy data at JET. *Review of Scientific Instruments*, **74**(3), pp. 1783-1786.
- HUANG, B.K., VANN, R.G.L., FREETHY, S., MYERS, R.M., NAYLOR, G., SHARPLES, R.M. and SHEVCHENKO, V.F., 2012. FPGA-based embedded Linux technology in fusion: The MAST microwave imaging system. *Fusion Engineering and Design*, **87**(12), pp. 2106-2111.
- IEEE STD 1588, 2008. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*.
- INFINIBAND, and MELLANOX, 2013-last update. Available: <http://on-demand.gputechconf.com/gtc/2013/webinar/gtc-express-gpudirect-rdma.pdf>.
- ISAEV, V., CLAEISSON, N., PLESKO, M. and ZAGAR, K., 2013. EPICS DATA ACQUISITION DEVICE SUPPORT. *Proceedings of ICALEPCS2013, San Francisco, CA, USA*, .
- KIM, M., BAEK, S.H., LEE, T.G., LEE, S.I., PARK, M.K. and KWON, M., 2010. The EPICS based plant control system for KSTAR—First operation period implementations and experiences. *Fusion Engineering and Design*, **85**(3–4), pp. 515-519.

- LABVIEW GPU ANALYSIS TOOLKIT, NI, 2015, 2015-last update. Available: <http://sine.ni.com/nips/cds/view/p/lang/es/nid/210829>.
- LLOBET, X. and DUVAL, B.P., 2002. 128 Channel PCI-based data acquisition system for MDSplus. *Fusion Engineering and Design*, **60**(3), pp. 285-289.
- LUCHETTA, A., MANDUCHI, G. and TALIERCIO, C., 2003. mdsplus data acquisition in RFX and its integration in legacy systems. *Fusion Engineering and Design*, **66–68**, pp. 959-963.
- MAKIJARVI, P., 2015-last update, ITER Catalogue of I&C Products. Available: [http://static.iter.org/codac/pcdh7/Folder%202/9-ITER Catalog of I%26C products - Fast Cont 345X28 v2 1.pdf](http://static.iter.org/codac/pcdh7/Folder%202/9-ITER%20Catalog%20of%20I%26C%20products%20-%20Fast%20Cont%20345X28%20v2%201.pdf).
- MAKOWSKI, D., MIELCZAREK, A., PEREK, P., JABLONSKI, G., ORLIKOWSKI, M., NAPIERALSKI, A., MAKIJARVI, P., SIMROCK, S. and MARTIN, V., 2014. High-performance image acquisition and processing system with MTCA.4, *Real Time Conference (RT), 2014 19th IEEE-NPSS 2014*, pp. 1-2.
- MANDUCHI, G., DE MARCHI, E. and MANDELLI, A., 2013. A new LabVIEW interface for MDSplus. *Fusion Engineering and Design*, **88**(6–8), pp. 1196-1199.
- MANDUCHI, G., HEMMING, O.N., LUCHETTA, A., TALIERCIO, C. and VITTURI, S., 1996. The data acquisition system of the RFX tomographic diagnostic. *Nuclear Science, IEEE Transactions on*, **43**(1), pp. 222.
- MANDUCHI, G., LUCHETTA, A., TALIERCIO, C. and VITTURI, S., 1999. The data acquisition system of the RFX experiment. *Fusion Engineering and Design*, **43**(3–4), pp. 275-284.
- MARE NOSTRUM SBC BARCELONA, 2015, 2015-last update. Available: <https://www.bsc.es/marenostrum-support-services/other-hpc-facilities/nvidia-gpu-cluster>.
- MARTIN, V., MONCADA, V. and TRAVERE, J.-., 2011. Challenges of video monitoring for phenomenological diagnostics in Present and Future Tokamaks, *Fusion Engineering (SOFE), 2011 IEEE/NPSS 24th Symposium on 2011*, pp. 1-6.

- MCOM10-K1 GENERAL ELECTRIC, 2015, 2015-last update. Available: <http://www.geautomation.com/products/mcom10-k1-mini-com-express>.
- MORE, J., 1978. The Levenberg-Marquardt algorithm: Implementation and theory. In: G.A. WATSON, ed, Springer Berlin Heidelberg, pp. 105-116.
- MORET, J.-., BUHLMANN, F., FASEL, D., HOFMANN, F. and TONETTI, G., 1998. Magnetic measurements on the TCV Tokamak. *Review of Scientific Instruments*, **69**(6), pp. 2333-2348.
- MURARI, A., VEGA, J., ALONSO, J.A., DE LALUNA, E., FARTHING, J., HIDALGO, C., RATTA, G.A., SVENSSON, J. and VAGLIASINDI, G., 2007. New Techniques and Technologies for Information Retrieval and Knowledge Extraction from Nuclear Fusion Massive Databases, *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on 2007*, pp. 1-6.
- NATIONAL INSTRUMENTS, and FAM, 2015-last update. Available: <http://sine.ni.com/nips/cds/view/p/lang/es/nid/206645>.
- NEUFELD, N. and VILASIS-CARDONA, X., 2012. Many-core processors and GPU opportunities in particle detectors, *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on 2012*, pp. 1-5.
- NVIDIA, N., 2015-last update. Available: <http://www.gputechconf.com/>.
- NVIDIA-SDK, N., 2015-last update. Available: <https://developer.nvidia.com/cuda-downloads>.
- OPENACC COMPILER, P., 2015, 2015-last update. Available: <https://www.pggroup.com/resources/accel.htm>.
- PARALLEL COMPUTING TOOLBOX, M., 2015, 2015-last update. Available: <http://es.mathworks.com/discovery/matlab-gpu.html>.
- PEREIRA, R.C., FERNANDES, A.M., NETO, A., SOUSA, J., VARANDAS, C.A.F., CARDOSO, J., CORREIA, C.M.B.A., TARDOCCHI, M., NOCENTE, M., GORINI, G., KIPTILY, V., SYME, B. and JENNISON, M., 2010. Pulse analysis for gamma-ray diagnostics ATCA sub-systems of JET tokamak, *Real Time Conference (RT), 2010 17th IEEE-NPSS 2010*, pp. 1-7.

PICMG CONSORTIUM, , PCI Industrial Computer Manufacturers Group.
Available: <http://www.picmg.org/> [11/7, 2015].

PSI, 2015-last update. Available: <http://epics.web.psi.ch/>.

RATH, N., BIALEK, J., BYRNE, P.J., DEBONO, B., LEVESQUE, J.P., LI, B., MAUEL, M.E., MAURER, D.A., NAVRATIL, G.A. and SHIRAKI, D., 2012. High-speed, multi-input, multi-output control using GPU processing in the HBT-EP tokamak. *Fusion Engineering and Design*, **87**(12), pp. 1895-1899.

RED HAT ENTERPRISE, L., 2015, 2015-last update. Available: <https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux>.

ROMERO, J.A., CODA, S., FELICI, F., MORET, J., PALEY, J., SEVILLANO, G., GARRIDO, I. and LE, H.B., 2012. Sliding mode control of a tokamak transformer, *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on* 2012, pp. 386-393.

RUIZ, M., BARRERA, E., LÓPEZ, S., MACHÓN, D., VEGA, J. and SÁNCHEZ, E., 2004. Real-time data acquisition and processing platform for fusion experiments. *Fusion Engineering and Design*, **71**(1-4), pp. 135-140.

RUIZ, M., VEGA, J., ARCAS, G., RATTI, G., BARRERA, E., MURARI, A., LOPEZ, J.M. and MELENDEZ, R., 2011. Real Time Plasma Disruptions Detection in JET Implemented With the ITMS Platform Using FPGA Based IDAQ. *Nuclear Science, IEEE Transactions on*, **58**(4), pp. 1576-1581.

RUIZ, M., VEGA, J., CASTRO, R., SANZ, D., LÓPEZ, J.M., DE ARCAS, G., BARRERA, E., NIETO, J., GONÇALVES, B., SOUSA, J., CARVALHO, B., UTZEL, N. and MAKIJARVI, P., 2012. ITER Fast Plant System Controller prototype based on PXIe platform. *Fusion Engineering and Design*, **87**(12), pp. 2030-2035.

RUIZ, M., VEGA, J., RATTI, G., BARRERA, E., MURARI, A., LOPEZ, J.M., ARCAS, G. and MELENDEZ, R., 2010. Real time plasma disruptions detection in JET implemented with the ITMS platform using FPGA based IDAQ, *Real Time Conference (RT), 2010 17th IEEE-NPSS 2010*, pp. 1-4.

SANTOS, J., ZILKER, M., GUIMARAIS, L., TREUTTERER, W., AMADOR, C. and MANSO, M., 2011. COTS-Based High-Data-Throughput

- Acquisition System for a Real-Time Reflectometry Diagnostic. *Nuclear Science, IEEE Transactions on*, **58**(4), pp. 1751-1758.
- SANZ, D., RUIZ, M., CASTRO, R., VEGA, J., LOPEZ, J.M., BARRERA, E., UTZEL, N. and MAKIJARVI, P., 2013. Implementation of Intelligent Data Acquisition Systems for Fusion Experiments Using EPICS and FlexRIO Technology. *Nuclear Science, IEEE Transactions on*, **60**(5), pp. 3446-3453.
- SANZ, D., RUIZ, M., LOPEZ, J.M., CASTRO, R., VEGA, J. and BARRERA, E., 2014. IEEE 1588 clock distribution for FlexRIO devices in PXIe platforms. *Fusion Engineering and Design*, **89**(5), pp. 652-657.
- SCHMIDT, V., 1998. The development of the JET control and data acquisition system. *Nuclear Science, IEEE Transactions on*, **45**(4), pp. 2026-2032.
- SHAFRANOV, V., 1971. Determination of the parameters $\hat{I}^2$ and I_i in a Tokamak for arbitrary shape of plasma pinch cross-section. *Plasma Physics*, **13**(9), pp. 757.
- SIMROCK, S., PATIL, P., MARTIN, V., ABADIE, L., BARNSELY, R., BERTALOT, L., MAKIJARVI, P., MAKOWSKI, D., JOURNEAUX, J., REICHLE, R., STEPANOV, D., VAYAKIS, G., YONEKAWA, I., WALLANDER, A. and WALSH, M., 2013. Diagnostic Use Case Examples for ITER Plant Instrumentation and Control, *14th International Conference on Accelerator & Large Experimental Physics Control Systems 2013*, pp. 1-5.
- STD IEEE 1588, , Std. Available: <http://www.nist.gov/el/isd/ieee/ieee1588.cfm> [12/15/2015, 2015].
- VALCÁRCEL, D.F., NETO, A., SOUSA, J., CARVALHO, B.B., FERNANDES, H., FORTUNATO, J.C., GOUVEIA, A.S., BATISTA, A.J.N., FERNANDES, A.G., CORREIA, M., PEREIRA, T., CARVALHO, I.S., DUARTE, A.S., VARANDAS, C.A.F., HRON, M., JANKY, F. and PÍSAČKA, J., 2009. An ATCA Embedded Data Acquisition and Control System for the Compass tokamak. *Fusion Engineering and Design*, **84**(7-11), pp. 1901-1904.
- VEGA, J., CRÉMY, C., SÁNCHEZ, E. and PORTAS, A., 1999. The TJ-II data acquisition system: an overview. *Fusion Engineering and Design*, **43**(3-4), pp. 309-319.

- WALLANDER, A., ABADIE, L., DAVE, H., DI MAIO, F., GULATI, H.K., HANSALIA, C., JOONEKINDT, D., JOURNEAUX, J., KLOTZ, W., MAHAJAN, K., MAKIJARVI, P., SCIBILE, L., STEPANOV, D., UTZEL, N. and YONEKAWA, I., 2010. ITER instrumentation and control—Status and plans. *Fusion Engineering and Design*, **85**(3–4), pp. 529-534.
- WINTER, A., MAKIJARVI, P., SIMROCK, S., SNIPES, J., WALLANDER, A. and ZABEO, L., 2012. Present status of the ITER real-time Plasma Control System development, *Real Time Conference (RT)*, 2012 18th IEEE-NPSS 2012, pp. 1-6.
- X N YUE AND B J XIAO AND Z P LUO AND Y.GUO, 2013. Fast equilibrium reconstruction for tokamak discharge control based on GPU. *Plasma Physics and Controlled Fusion*, **55**(8), pp. 085016.

Anexos

ANEXO A: Acrónimos

Término	Definición
ADC	Analog to Digital Converter
AMC	Advanced Mezzanine Card
AMD	Advanced Micro Devices
API	Application Programming Interface
ASDEX	Axially Symmetric Divertor Experiment
ATCA	Advanced Telecommunications Computing Architecture
BLAS	Basic Linear Algebra Subprograms
CA	Channel Access
CAMAC	Computer-Automated Measurement and Control, IEEE-583
CCS	CODAC Core System
CEO	Chief Executive Officer
CIEMAT	Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas
CIS-I&C	Central Interlock System Instrumentation & Control
CODAC	Control, Data Access and Communication Control system
CODAC-CIS	CODAC Central Interlock System
CODAC-CSS	CODAC Central Safety System
CODAC-HPC	CODAC High Performance Computing
COMPASS	Tokamak COMPact ASSEMBly
COTS	Commercial Off-The-Shelf
CPCI	Compact PCI

Término	Definición
CPU	Central Process Unit
CSS	Control System Studio
CUDA	Computer Unified Device Architecture
DAN	Data Archiver Network
DAQ	Sistema de Adquisición de Datos
DLL	Dynamic-link library
DMA	Direct Memory Access
DP GFLOPS	Double Precision Giga Flops
DPD	Data Process and Distribution
DSP	Digital Signal Processor
E/S	Entrada/Salida
EPICS	Experimental Physics and Industrial Control System
EURATOM	Comunidad Europea de la Energía Atómica
FAM	FlexRIO Adapter module
FFT	Fast Fourier Transform
FIFO	First Input First Output
FlexRIO	Flexible RIO
FLOPS	Floating-Point operations per second
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HPC	High-performance computing

Término	Definición
IDAQ	Intelligent Acquisition System
IEEE	Institute of Electrical and Electronics Engineers
IOC	Input Output Controller
IOCTL	Input Output Control, Linux system call
IOHUB	Input Output Controller Hub (ICH)
ITER	International Thermonuclear Experiment Reactor
JET	Joint European Torus
KSTAR	Korea Superconducting Tokamak Advanced Research
LAN	Local Area Network
MAST	Mega Amp Spherical Tokamak.
MTCA	Micro Telecommunications Computing Architecture
NDS	Nominal Device Support
NI	National Instruments
OPI	Operator Interfaces
PCI	Peripheral Component Interconnect
PCI BAR	PCI Base Address Register
PCIe	PCI Express
PCS	Plasma Control System
PGI	Portland Group, Inc
PICMG	PCI Industrial Computer Manufacturers Group
PLC	Programmable Logic Controller
PON	Plant Operation Network

Término	Definición
PSH-I&C	Plant System Host
PS-I&C	Plant System Instrumentation & Control
PV	Process Variable
PXI	PCI eXtensions for Instrumentations
PXIe	PXI Express
RAID	Redundant Array of Independent Disks
RAM	Random Address Memory
RDMA	Remote direct memory access
RHLE	Red Hat Linux Enterprise
RIO	Reconfigurable Input Output
RT	Real Time
RTEMS	Real-Time Executive for Multiprocessor Systems
SCADA	Supervisory Control And Data Acquisition
SC-I&C	Slow Controller Instrumentation & Control
SDD	Self-Description Data configuration model
SDN	Synchronous Data bus Network
SIMD	Single instruction, multiple data
TCN	Time Communication Network
TCP/IP	Transmission Control Protocol/Internet Protocol
TCV	Tokamak à configuration variable
UDP	User Datagram Protocol
VHDL	VHSIC Hardware Description Language

Término	Definición
VME	Versa Module Euro-Card, IEEE-1014
VXI	VME eXtensions for Instrumentations, IEEE-1155