UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación

PROYECTO FIN DE GRADO

DISEÑO DE UN BANCO DE PRUEBAS PARA UN SISTEMA MULTIPROCESADOR HETEROGÉNEO ARM/RISC-V

IVÁN VÍTORES VILLALBA

Grado en Ingeniería Electrónica de Comunicaciones
Abril 2025

PROYECTO FIN DE GRADO

TÍTULO: Diseño de un banco de pruebas para un sistema multiprocesador heterogéneo ARM/RISC-V

AUTOR/A: Iván Vítores Villalba

TITULACIÓN: Grado en Ingeniería Electrónica de comunicaciones

TUTOR/A: Pedro José Lobo Perea

DEPARTAMENTO: Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Vicente Lorenzo García

TUTOR/A: Pedro José Lobo Perea

SECRETARIO/A: Ángel Manuel Groba González

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Resumen

El proyecto titulado “Diseño de un banco de pruebas para un sistema multiprocesador heterogéneo ARM/RISC-V” viene impulsado por la situación global actual donde vivimos en un mundo cada vez más digitalizado, la capacidad para procesar y analizar grandes volúmenes de datos se ha convertido en un desafío crítico. Las tecnologías emergentes, como el IoT (Internet of Things) o las novedosas AI (Artificial Intelligence) generan cantidades masivas de datos que requieren soluciones eficientes y rápidas para su tratamiento.

Para lograr estos objetivos, se han adoptado arquitecturas multiprocesador heterogéneas, que combinan procesadores de propósito general y procesadores especializados los cuales permiten manejar grandes volúmenes de datos de manera eficiente. Dentro de este campo una solución destacada es PULP (Parallel Ultra-Low-Power), una arquitectura desarrollada por ETH Zurich y la Universidad de Bolonia, basada en la ISA (Instruction Set Architecture) RISC-V y enfocada en hardware y software libre. Inicialmente orientada al IoT, PULP también es adecuada para aplicaciones de alto rendimiento gracias a su flexibilidad. Dentro del ecosistema PULP, la plataforma HERO (Heterogeneous Research Platform) ha surgido como una solución orientada específicamente a sistemas de altas prestaciones, combinando procesadores y aceleradores especializados e incluyendo herramientas de programación basadas en el estándar OpenMP, facilitando el desarrollo y la optimizando aplicaciones paralelas complejas.

Este proyecto se ha llevado a cabo en GDEM (Grupo de Diseño Electrónico y Microelectrónico) que forma parte del CITSEM (Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad), centrándose en prototipar el sistema HERO en la tarjeta de desarrollo ZCU102, la cual cuenta con un chip XCZU9EG de la familia Zynq UltraScale+ de Xilinx, que forma parte de la arquitectura MPSoC. Posteriormente se han diseñado un conjunto de pruebas software que ejecutadas nos han servido para averiguar la relación en el intercambio de datos entre el procesador host y la lógica programable que contiene el chip antes mencionado.

Tras la realización de este proyecto, que incluye la implementación de la plataforma HERO en la tarjeta de desarrollo, así como el diseño y la ejecución de programas específicos para medir tanto el rendimiento como los tiempos de ejecución, se han llevado a cabo estudios exhaustivos que han permitido caracterizar el procesamiento y el intercambio de datos entre las arquitecturas ARM del host y el acelerador RISC-V. Los resultados obtenidos han revelado tanto las capacidades como las limitaciones de la comunicación entre ambas arquitecturas, proporcionando una base sólida para el diseño de soluciones futuras basadas en plataformas heterogéneas. Además, estos resultados han aportado valiosos conocimientos sobre las mejores prácticas en programación e implementación de técnicas de paralelización, con el fin de mejorar el rendimiento en aplicaciones intensivas.

Abstract

The project titled "Design of a Test Bench for an ARM/RISC-V Heterogeneous Multiprocessor System" is driven by the current global situation, where we live in an increasingly digitalized world, and the capacity to process and analyze large volumes of data has become a critical challenge. Emerging technologies, such as IoT (Internet of Things) and new AI (Artificial Intelligence) innovations, generate massive amounts of data that require efficient and fast solutions for processing.

To achieve these objectives, heterogeneous multiprocessor architectures have been adopted, combining general-purpose processors and specialized processors that allow handling large volumes of data efficiently. Within this field, one notable solution is PULP (Parallel Ultra-Low-Power), an architecture developed by ETH Zurich and the University of Bologna, based on the RISC-V Instruction Set Architecture (ISA) and focused on open hardware and software. Initially oriented toward IoT, PULP is also suitable for high-performance applications due to its flexibility. Within the PULP ecosystem, the HERO (Heterogeneous Research Platform) has emerged as a solution specifically designed for high-performance systems, combining processors and specialized accelerators and including programming tools based on the OpenMP standard, facilitating development and optimizing complex parallel applications.

This project has been carried out at GDEM (Electronic and Microelectronic Design Group), part of CITSEM (Research Center for Software Technologies and Multimedia Systems for Sustainability), focusing on prototyping the HERO system on the ZCU102 development board, which features the XCZU9EG chip from Xilinx's Zynq UltraScale+ family, part of the MPSoC architecture. Subsequently, a set of software tests were designed and executed, allowing us to determine the relationship in the data exchange between the host processor and the programmable logic contained in the after mentioned chip.

After completing this project, which includes the implementation of the HERO platform on the development board, as well as the design and execution of specific programs to measure both performance and execution times, exhaustive studies have been carried out to characterize the processing and data exchange between the ARM architectures of the host and the RISC-V accelerator. The results obtained have revealed both the capabilities and limitations of communication between these two architectures, providing a solid foundation for designing future solutions based on heterogeneous platforms. Furthermore, these results have provided valuable insights into best practices in programming and implementing parallelization techniques, aiming to improve performance in intensive applications.

Índice de figuras

Índice de tablas

Índice de gráficas

Índice de scripts

Lista de acrónimos

AI	Artificial Intelligence
API	Application Programming Interface
BRAM	Block RAM
BW	Bandwidth
CISC	Complex Instruction Set Computing
CITSEM	Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad
CSV	Comma-Separated Values
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
GeMM	General Matrix Multiply
GDEM	Global Digital Elevation Model
GPIO	General-Purpose Input/Output
HDMI	High-Definition Multimedia Interface
HERO	Heterogeneous Embedded Research Platform
I2C	Inter-Integrated Circuit
IoT	Internet of Things
IP	Intellectual Property
ISA	Instruction Set Architecture
LLVM	Low-Level Virtual Machine
MPSoC	Multiprocessor System on Chip
NFS	Network File System
ODS	Objetivos de Desarrollo Sostenible
OpenMP	Open Multi-Processing
PL	Programmable Logic
PS	Processing System
PULP	Parallel Ultra Low Power
RISC	Reduced Instruction Set Computing
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Índice de contenidos

1. Introducción

1.1 Marco y motivación del proyecto

El Grupo de Diseño Electrónico y Microelectrónico (GDEM), adscrito al Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM), ha desarrollado durante años diversas líneas de investigación centradas en el diseño y optimización de sistemas de procesamiento eficiente de señal, con un énfasis particular en imagen y vídeo. Estas investigaciones han encontrado aplicaciones en ámbitos críticos como el procesamiento de las imágenes médicas hiperespectrales o la generación de mapas de profundidad para visión 3D. Para abordar estos desafíos, se han adoptado arquitecturas multiprocesador heterogéneas que combinan procesadores de propósito general con procesadores especializados como DSPs, GPUs o arquitecturas emergentes, permitiendo así el manejo eficiente de grandes volúmenes de datos.

Un elemento clave en el procesamiento de imágenes es la manipulación de grandes matrices de datos, fundamentales en tareas como filtrado, convolución y transformaciones espaciales. Con la evolución de los algoritmos de optimización matricial y la implementación de técnicas de paralelización, se ha conseguido un incremento en la eficiencia computacional, permitiendo la realización de operaciones complejas en tiempo real. Estas mejoras han sido impulsadas gracias a arquitecturas especializadas que optimizan la gestión de memoria, la comunicación entre núcleos y el acceso a datos en caché, maximizando el rendimiento de los sistemas.

Este Proyecto Fin de Grado se inscribe en una línea de investigación orientada al estudio de la transferencia de datos en arquitecturas heterogéneas, enfocándose en la comunicación entre procesadores y aceleradores en sistemas embebidos. En particular, se centra en el diseño y la implementación de pruebas software con el objetivo principal de analizar la transferencia de datos entre el procesador ARM y la lógica programable que incluye el chip XCZU9EG montado en la tarjeta de desarrollo ZCU102, evaluando posibles problemas de embotellamiento y explorando estrategias para optimizar la comunicación. Además, el estudio contribuye a comprender el impacto de la eficiencia en la interconexión de memoria compartida, el ancho de banda y la latencia en sistemas multiprocesador heterogéneos.

Los resultados obtenidos permiten no solo mejorar el diseño de futuras soluciones basadas en este tipo de plataformas, sino también aportar conocimientos sobre las mejores prácticas en la programación y optimización de estos entornos. El análisis del rendimiento en la comunicación entre el procesador ARM y el acelerador RISC-V proporciona información valiosa sobre los desafíos y oportunidades en la implementación de arquitecturas heterogéneas para aplicaciones de alto rendimiento y procesamiento intensivo.

1.2 Objetivos técnicos y académicos

Los objetivos de este proyecto fin de carrera son, desde el punto de vista técnico:

- Prototipar la plataforma HERO en la FPGA para evaluar el rendimiento en sistemas multiprocesador heterogéneos basados en ARM y RISC-V.
- Diseñar y ejecutar pruebas de software para analizar la transferencia de datos entre el procesador ARM y la lógica programable de la FPGA, identificando posibles cuellos de botella en la comunicación.
- Optimizar la configuración de clústeres y núcleos para maximizar la eficiencia del sistema y minimizar la latencia en el intercambio de información.
- Implementar y evaluar la paralelización de algoritmos, utilizando OpenMP como herramienta de programación para la distribución eficiente de tareas.
- Realizar un análisis comparativo del rendimiento computacional en diferentes configuraciones de carga de trabajo y procesamiento paralelo.
- Documentar los hallazgos y proponer mejoras para futuras implementaciones.

Desde el punto de vista académico, el proyecto permite adquirir las siguientes competencias y habilidades:

- Adquirir experiencia en el proceso de implementación y optimización de sistemas multiprocesador heterogéneos.
- Desarrollar habilidades en el uso de herramientas avanzadas como PetaLinux, OpenMP y LLVM para la programación y depuración de sistemas embebidos.
- Adquirir conocimientos en arquitecturas de computadoras y sistemas operativos como Linux dentro de un entorno práctico.
- Mejorar la capacidad de análisis y resolución de problemas en la implementación de soluciones para la transferencia eficiente de datos entre diferentes arquitecturas.
- Desarrollar competencias en documentación técnica y presentación de resultados, preparando un informe detallado con conclusiones sobre el desempeño del sistema.

1.3 Estructura del resto de la memoria

El presente documento se organiza en varios capítulos, cada uno de los cuales aborda una parte fundamental del desarrollo del proyecto. A continuación, se detalla el contenido de cada sección para proporcionar una visión clara de la estructura de la memoria:

- **Marco tecnológico:** Se presentan las tecnologías utilizadas en el proyecto. Se describe en detalle la plataforma HERO, se analizan los fundamentos de arquitecturas heterogéneas, la ISA RISC-V y su relevancia en sistemas embebidos. También se explican herramientas clave como OpenMP y LLVM.

- **Especificaciones y restricciones de diseño:** Se establecen los requisitos técnicos que deben cumplirse para garantizar el correcto funcionamiento del sistema. Además, se detallan las limitaciones encontradas durante el desarrollo del proyecto.

- **Descripción de la solución propuesta:** Se describe la implementación del proyecto, incluyendo la configuración del entorno, la integración del software y los pasos para instalar y configurar HERO en la FPGA del chip XCZU9EG que incluye la tarjeta de desarrollo ZCU102, desde la descarga del código hasta la ejecución de pruebas.

- **Resultados:** Se presentan las pruebas realizadas para evaluar el rendimiento de la solución propuesta mediante la ejecución de programas, midiendo el impacto del número de hilos en la ejecución de tareas. Además, se interpretan los resultados obtenidos mediante gráficos y análisis comparativos.

- **Presupuesto:** Este apartado proporciona una estimación detallada del coste asociado al desarrollo del proyecto. Se presentan tablas con los recursos utilizados.

- **Impacto del proyecto:** Se analiza el impacto que tendría la implementación de la plataforma HERO en distintos ámbitos. Se estudian implicaciones sociales, económicas y tecnológicas. Además, se examina el impacto ambiental de la solución propuesta.

- **Conclusiones y trabajos futuros:** Se realiza un resumen de los principales logros alcanzados en el proyecto, evaluando la solución desarrollada. Se proponen futuras investigaciones para ampliar la aplicación de la plataforma HERO en otros entornos.

- **Referencias y bibliografías:** Se incluye una recopilación de todas las fuentes bibliográficas que fueron utilizadas durante el desarrollo del proyecto, proporcionando información para futuras investigaciones en el área.

- **Anexos**: En esta sección se recopila información complementaria que no se incluye en los capítulos principales son útiles para la comprensión del proyecto, en concreto fragmentos de código relevantes.

2. Marco tecnológico

En este capítulo se presenta una visión general de las tecnologías y herramientas utilizadas en el desarrollo del proyecto, se describen tanto las tecnologías hardware como software que han sido esenciales para la implementación, pruebas y evaluación.

2.1 RISC-V/ISA

A continuación, se exploran los conceptos clave relacionados con las ISAs, con un enfoque particular en la arquitectura RISC-V, ya que desempeñan un papel fundamental en el diseño de procesadores al facilitar la interacción entre el hardware y el software

2.1.1 ISA (Instruction Set Architecture)

Como ARM define en su página web [6], la ISA es parte del modelo abstracto de una computadora que define cómo el software controla la CPU, actúa como una interfaz entre el hardware y el software , especificando tanto lo que el procesador es capaz de hacer como la forma en que lo hace.

Esta interfaz establece las operaciones que el hardware puede realizar y cómo el software debe estructurarse para aprovechar esas operaciones. Es el único medio a través del cual un usuario puede interactuar con el hardware, porque es la parte de la máquina que está visible para los programadores.

La ISA define los tipos de datos admitidos, los registros, cómo el hardware gestiona la memoria principal, las características clave, qué instrucciones puede ejecutar un microprocesador y el modelo de entrada/salida de múltiples implementaciones de ISA. La ISA puede ampliarse mediante la incorporación de nuevas instrucciones, capacidades adicionales o el soporte para direcciones y valores de datos más grandes.

Las ISAs se clasifican principalmente en dos categorías:

- **ISA cerrada**: Son arquitecturas propietarias desarrolladas y controladas por empresas específicas, como ARM. Estas arquitecturas ofrecen confiabilidad y una compatibilidad bien establecida, ya que las empresas propietarias mantienen un control estricto sobre su desarrollo y evolución. Sin embargo, esta propiedad exclusiva limita la capacidad de personalización y adaptación por parte de terceros, debido a restricciones de licencia y acceso.
- **ISA abierta**: Son arquitecturas cuyo diseño y especificaciones están disponibles públicamente, permitiendo que cualquier entidad las utilice, modifique o implemente sin restricciones significativas. Esto promueve la innovación y la adaptación de necesidades específicas al eliminar las limitaciones impuestas por licencias propietarias. Esta apertura facilita la colaboración comunitaria y la transparencia en el desarrollo de hardware, permitiendo a diversas organizaciones contribuir y beneficiarse de mejoras compartidas, un ejemplo destacado es RISC-V.

2.1.2 RISC-V (Reduced Instruction Set Computing)

En los años 70, los procesadores estaban dominados por arquitecturas CISC (Complex Instruction Set Computing), que contaban con conjuntos de instrucciones complejos. Estas arquitecturas requerían grandes cantidades de transistores, lo que implicaba un mayor consumo de recursos.

En la década de los 80, David Patterson, profesor de la Universidad de Berkeley, California, lideró un equipo de investigadores que desarrolló la arquitectura RISC-I. Este diseño demostró que un conjunto reducido de instrucciones no solo simplificaba el hardware, sino que también podía mejorar el rendimiento general al reducir la cantidad de transistores necesarios. Esta innovación marcó el inicio del enfoque RISC, que resultó ser más rápido y eficiente en comparación con las arquitecturas CISC de la época.

Décadas después, en 2010, Krste Asanović, también académico de la Universidad de Berkeley, California, retomó los principios fundamentales de RISC para desarrollar RISC-V. Este nuevo proyecto tenía un enfoque abierto, permitiendo que la arquitectura fuera utilizada, modificada y ampliada por la comunidad sin las restricciones de licencias propietarias.

Hoy en día RISC-V es una ISA abierta y gratuita basada en los principios de la computación de conjunto reducido de instrucciones (RISC), está disponible públicamente, lo que permite a cualquier persona diseñar, fabricar y vender chips y software basados en esta arquitectura.

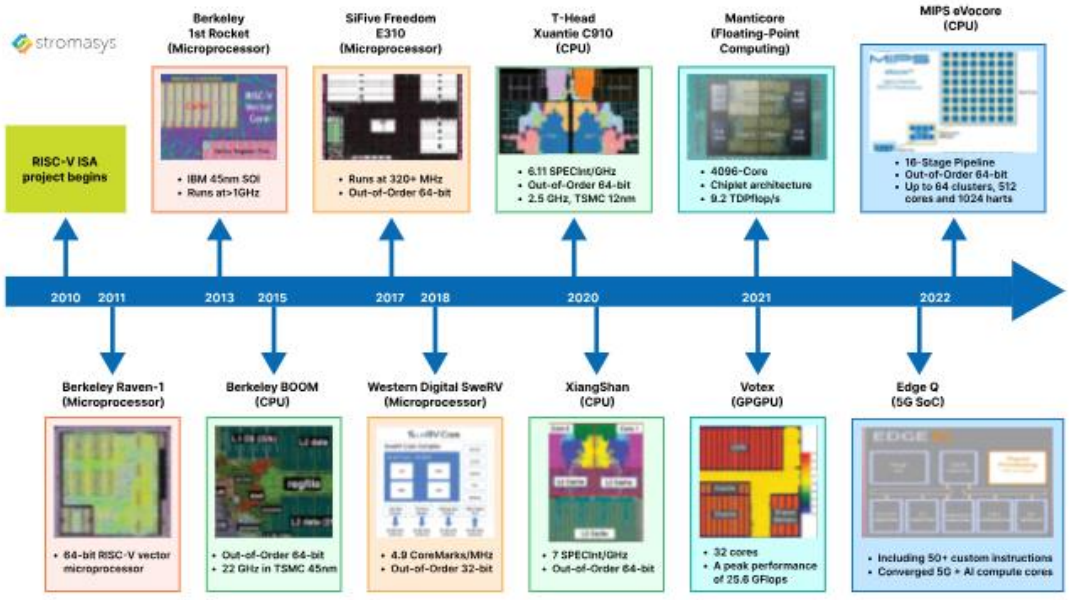


Figura 1. Evolución de procesadores RISC-V

La figura 1 ilustra los hitos en la evolución de procesadores basados en RISC-V, desde el inicio del proyecto en 2010, ha habido una notable progresión en términos de desempeño, aplicaciones y adopción comercial. Cada etapa destaca contribuciones significativas de equipos académicos y empresas que han adoptado esta arquitectura abierta, lo que ha permitido avanzar en innovación y personalización dentro de la industria de semiconductores.

Una de las principales características de RISC-V es su modularidad y extensibilidad. Está diseñada de manera modular, con un conjunto base de instrucciones que puede ser ampliado mediante extensiones opcionales. Esta flexibilidad permite a los diseñadores adaptar la ISA a necesidades específicas, optimizando el rendimiento y la eficiencia para aplicaciones particulares.

Otro aspecto destacado de RISC-V es su simplicidad y eficiencia. La arquitectura se centra en mantener un conjunto reducido de instrucciones simples, lo que facilita la implementación de procesadores eficientes y de alto rendimiento. Además, esta simplicidad contribuye a una menor complejidad en el diseño del hardware y a una mayor facilidad para la verificación y el mantenimiento.

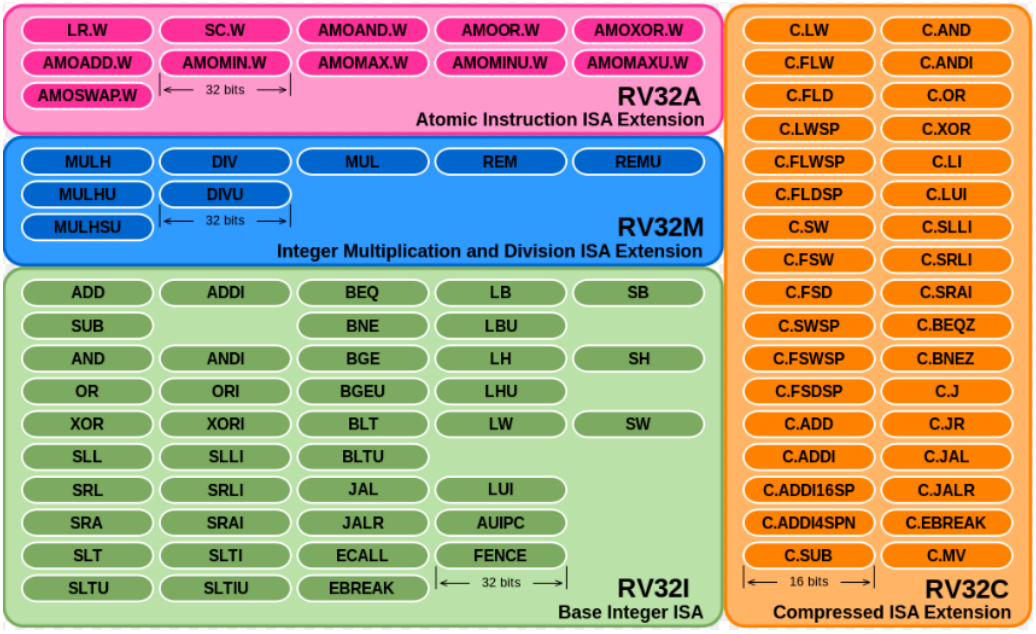


Figura 2. Instrucciones RV32IMAC de RISC-V

En la figura 2 podemos observar un conjunto de instrucciones del estándar RISC-V, específicamente la variante RV32IMAC, esta variante combina el conjunto de instrucciones básicas para operaciones enteras (zona verde), extensiones opcionales como instrucciones atómicas (zona rosa), instrucciones de multiplicación y división (zona azul) o instrucciones comprimidas de 16 bits (zona naranja). Con esto se pretenden demostrar propiedades de RISC-V como modularidad, eficiencia, escalabilidad, compatibilidad y portabilidad.

Su capacidad para ser extendida y adaptada a necesidades específicas la hace particularmente útil en sistemas embebidos e IoT, donde permite desarrollar soluciones personalizadas y altamente eficientes en consumo energético, esta característica de personalización es un claro ejemplo de su modularidad. Además, en el ámbito de la computación de alto rendimiento, RISC-V se adapta perfectamente a tareas de procesamiento intensivo, mostrando su escalabilidad para gestionar diferentes niveles de complejidad. Su naturaleza abierta y su capacidad de ser implementada en diversas plataformas resaltan la compatibilidad y accesibilidad de RISC-V, lo que la convierte en una excelente opción para la investigación y la educación, facilitando el aprendizaje y la innovación en el diseño de arquitecturas de procesadores.

2.2 PULP y HERO

A continuación, se presenta una revisión sobre la historia y evolución de la plataforma PULP [3] y el surgimiento posterior de la plataforma HERO [4]. A lo largo de esta revisión, se explorarán los aspectos fundamentales que definen a ambas plataformas, subrayando especialmente los elementos técnicos que hacen de HERO una herramienta clave en la implementación de la solución propuesta.

2.2.1 PULP (Parallel Ultra Low Power)

La Plataforma PULP nació de una colaboración entre el Laboratorio de Sistemas Integrados de ETH Zürich y el grupo de Sistemas Embebidos de Alta Eficiencia Energética de la Universidad de Bolonia en 2013. Su propósito inicial era explorar y desarrollar arquitecturas de computación eficientes utilizando la ISA abierta RISC-V y un diseño de hardware colaborativo y de código abierto, originalmente se enfocó en dispositivos de bajo consumo, pero PULP ha ampliado su alcance para incluir sistemas de alto rendimiento.

Con el paso de los años, PULP ha evolucionado hasta convertirse en uno de los proyectos de código abierto más destacados a nivel mundial, en gran parte gracias a la adopción generalizada de sus IPs (Propiedad Intelectual) de código abierto en la industria.

El objetivo era desarrollar una plataforma de investigación y desarrollo de hardware y software abierta y escalable, con el fin de superar los límites de la eficiencia energética dentro de un consumo de solo unos pocos milivatios, además de satisfacer las demandas computacionales de aplicaciones IoT que requieren un procesamiento flexible de flujos de datos generados por múltiples sensores.

PULP incluye un sistema de microcontrolador de última generación y una plataforma multinúcleo capaz de alcanzar una eficiencia energética y un rendimiento altamente ajustable. En comparación con las unidades de microcontrolador de un solo núcleo, su arquitectura programable de bajo consumo paralela permite satisfacer los requisitos computacionales de aplicaciones IoT sin exceder el consumo típico de milivatios en sistemas miniaturizados y alimentados por baterías.

Desde sus inicios, el proyecto ha adoptado un enfoque de código abierto. Hasta ahora, se han lanzado implementaciones eficientes de 32 y 64 bits basadas en la arquitectura RISC-V. Además, PULP está diseñado para soportar varias interfaces de programación de aplicaciones como OpenMP, OpenCL y OpenVX, que facilitan la portabilidad ágil de aplicaciones, su desarrollo, optimización de rendimiento y depuración

La plataforma de investigación PULP es un proyecto de hardware colaborativo y de código abierto disponible en GitHub. Todo su código actualizado, junto con ejemplos de aplicación, está accesible para que cualquier desarrollador pueda utilizarlo y contribuir al proyecto.

2.2.2 HERO (Heterogeneous Research Platform)

La plataforma de investigación HERO es una extensión o evolución de la plataforma PULP, diseñada para combinar un procesador anfitrión junto a un acelerador multinúcleo RISC-V. Utiliza la tecnología y los principios de PULP para ofrecer una solución centrada en sistemas heterogéneos, combinando las capacidades de procesamiento de los sistemas embebidos, facilitando investigaciones en computación de alto rendimiento y eficiencia energética.

Aunque HERO ha sido desarrollada para varias tarjetas de desarrollo la implementación más madura es la ZCU102 de Xilinx, la cual estamos utilizando en la realización de este proyecto. Actualmente HERO cuenta con núcleos RISC-V de 32 bits en el acelerador y núcleos ARMv8 de 64 bits como CPU host. Permite compartir datos entre el host y el acelerador mediante una interfaz de programación heterogénea basada en OpenMP 4.5 y un compilador heterogéneo basado en LLVM, de los cuales se hablará en secciones posteriores.

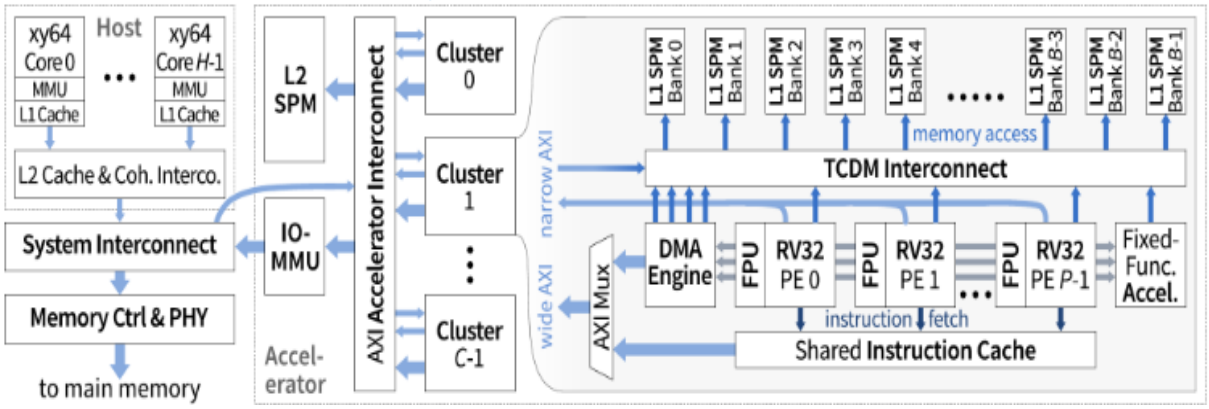


Figura 3. Arquitectura HERO

En la figura 3, extraída del documento "readme" en GitHub [5], se muestra la arquitectura hardware de HERO. En la esquina superior izquierda se encuentra el procesador host, actuando como la CPU de propósito general. Debajo de este, se observa el sistema de interconexión, que facilita la comunicación entre el host y el acelerador, permitiendo también el intercambio eficiente de datos entre la memoria principal y la capa física. A la derecha de la imagen se observa el acelerador de lógica programable, compuesto por múltiples clusters, cada uno de los cuales está formado por varios núcleos, diseñados para tareas específicas. Esta arquitectura permite una integración efectiva entre el host y el acelerador, optimizando el procesamiento de datos en aplicaciones heterogéneas.

El proyecto HERO incluye una configuración predeterminada para la lógica programable del chip XCZU9EG que incluye la tarjeta de desarrollo ZCU102, que permite sintetizar un clúster compuesto por ocho núcleos, el código fuente de HERO está disponible en GitHub, con aportaciones y verificaciones continuas por parte de múltiples desarrolladores.

2.3 Tarjeta de desarrollo ZCU102 de Xilinx

Las tarjetas de desarrollo son herramientas esenciales para el diseño y prototipado de sistemas electrónicos avanzados. Estas plataformas integran SoC (System on Chip) o MPSoC (Multiprocessor System on Chip), que incluyen procesadores, memoria, interfaces de comunicación y lógica programable. Gracias a estos chips, las tarjetas de desarrollo permiten experimentar, validar y optimizar tanto hardware como software en un solo entorno antes de la producción.

Gracias a su versatilidad, estas tarjetas facilitan el prototipado rápido y la evaluación de tecnologías específicas, como procesadores o arquitecturas de lógica programable, permitiendo explorar sus capacidades y características en un entorno controlado sin necesidad de fabricar hardware personalizado. En el desarrollo de software embebido, además, permiten a los ingenieros escribir, probar y depurar código en tiempo real, interactuando directamente con el hardware.

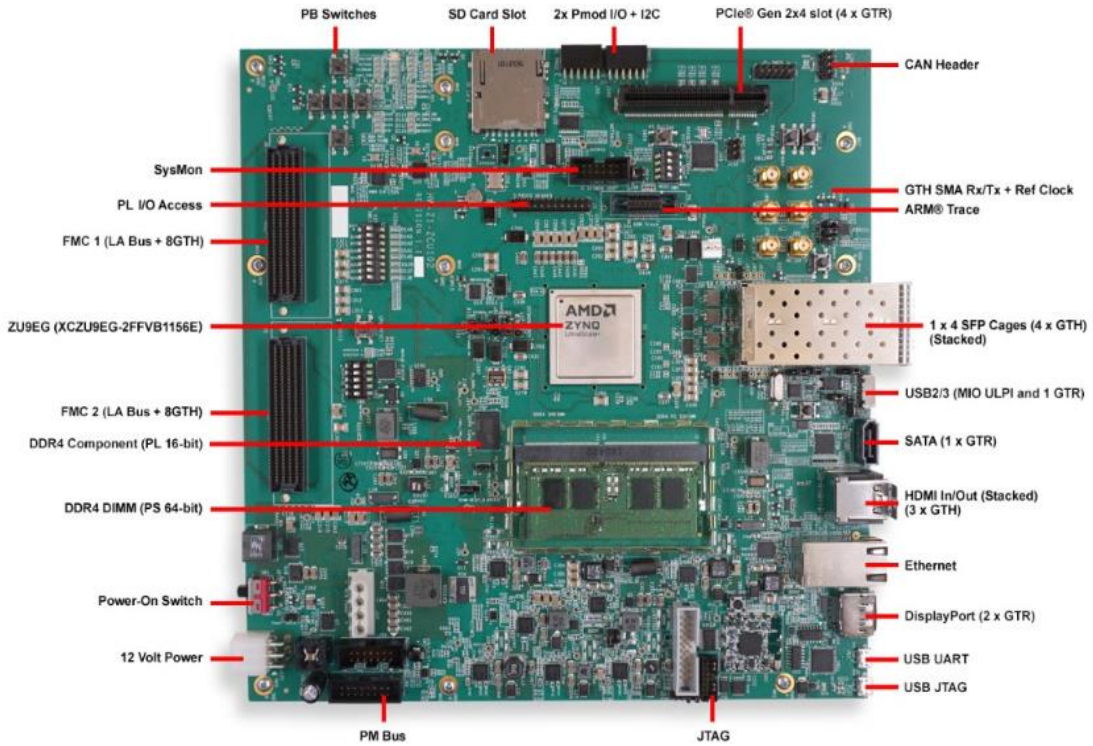


Figura 4. Tarjeta de desarrollo ZCU102 Xilinx

En la figura 4 obtenida de la guía de usuario de ZCU102 [1], se puede observar la tarjeta de desarrollo utilizada durante la realización de este proyecto. La tarjeta de desarrollo ZCU102 es una plataforma versátil diseñada por Xilinx que permite el desarrollo y la evaluación de soluciones en una variedad de aplicaciones.

La tarjeta de desarrollo ZCU102 utiliza el chip XCZU9EG, un dispositivo de la familia Zynq UltraScale+ de Xilinx, que forma parte de la arquitectura MPSoC. Este chip combina la flexibilidad de la lógica programable con la potencia de procesamiento de múltiples núcleos ARM, lo que permite ejecutar tareas complejas tanto en el dominio de procesamiento general como en el de lógica programable.

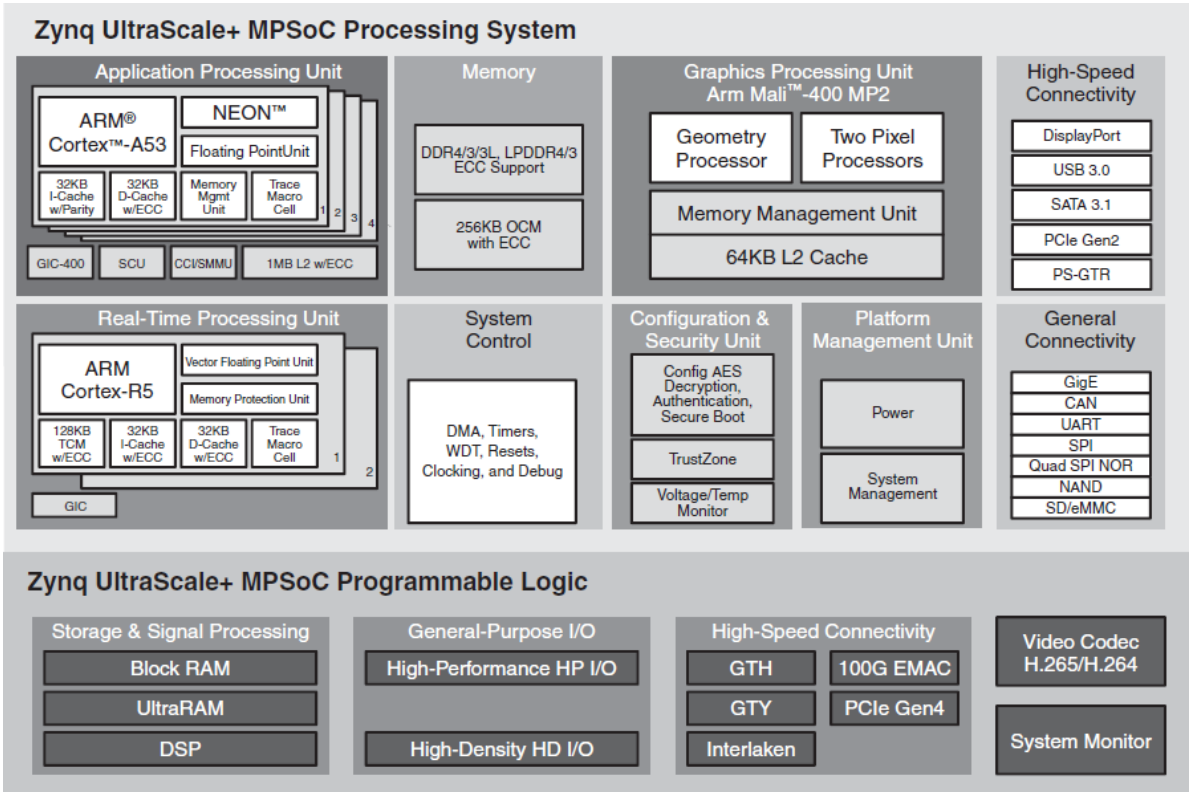


Figura 5. Diagrama de bloques Zynq UltraScale+

En la figura 5 obtenida de la guía de usuario Zynq UltraScale+ [2], se puede observar el diagrama de bloques de alto nivel de la arquitectura donde se realiza una separación entre la lógica programable y el sistema de procesamiento.

2.3.1 Procesadores ARM

El chip XCZU9EG de la tarjeta de desarrollo ZCU102 integra varios procesadores ARM diseñados para satisfacer diferentes necesidades. A continuación, se presenta un resumen de las características de los procesadores incluidos en este chip:

La unidad de procesamiento de aplicaciones está compuesta por cuatro núcleos Cortex-A53, basados en la arquitectura ARMv8 de 64 bits. Este procesador está diseñado para manejar el sistema operativo y aplicaciones de propósito general, ofreciendo un alto rendimiento con una buena eficiencia energética.

La unidad de procesamiento en tiempo real contiene dos núcleos Cortex-R5, basados en la arquitectura ARMv7 de 32 bits. Este procesador es ideal para aplicaciones en tiempo real y tareas críticas, ya que ofrece procesamiento determinista con tiempos de respuesta predecibles, lo que lo hace adecuado para el control embebido.

Además, el chip XCZU9EG de la tarjeta de desarrollo ZCU102 incorpora un procesador gráfico Mali-400 MP2 con dos núcleos. Este procesador está optimizado para el procesamiento de gráficos y multimedia, y cuenta con una memoria caché L2 de 64 KB, lo que permite manejar aplicaciones gráficas de forma eficiente.

2.3.2 Lógica programable

El chip XCZU9EG integra un bloque de lógica programable (PL). Este componente ofrece capacidades avanzadas para el diseño y la implementación de hardware personalizado, permitiendo a los desarrolladores crear soluciones específicas para aplicaciones que requieren un alto rendimiento.

El chip XCZU9EG de la tarjeta de desarrollo ZCU102 cuenta con 599550 celdas lógicas en su bloque de lógica programable (PL), lo que proporciona una gran flexibilidad para diseñar circuitos personalizados y sistemas embebidos complejos. Además, incorpora 32.1 Mb de memoria de bloque (BRAM), un tipo de memoria interna distribuida dentro de la FPGA que es utilizada para el almacenamiento temporal de datos y operaciones de alta velocidad, como buffers de datos. También incluye 2,520 bloques DSP, unidades especializadas diseñadas para realizar cálculos matemáticos intensivos, como multiplicaciones, sumas acumulativas y operaciones en paralelo, esenciales en aplicaciones como el procesamiento de señales o imágenes.

Una de las principales ventajas de la lógica programable es su capacidad para trabajar de manera conjunta con el sistema de procesamiento (PS), esto permite descargar tareas específicas a la lógica programable, como algoritmos de procesamiento de video o inteligencia artificial, maximizando la eficiencia del sistema y liberando recursos del procesador principal.

2.3.3 Interfaces de conectividad

La tarjeta de desarrollo ZCU102 integra varias interfaces de conectividad, de la cuales vamos a realizar un posible resumen:

Las interfaces de alta velocidad son clave para conectar dispositivos de almacenamiento o acelerar el procesamiento de datos, incluyendo PCIe Gen2, USB 3.0/2.0, y SATA 3.1.

Las conexiones de propósito general complementan el diseño al facilitar el control e integración de periféricos adicionales. Entre estas se incluyen Gigabit Ethernet para aplicaciones de red de alta velocidad, UART, SPI, Quad-SPI para comunicaciones serie y paralela en sistemas embebidos, I2C para la configuración de periféricos y sensores, y puertos GPIO para la interacción directa con dispositivos externos.

También cabe destacar que cuenta con capacidad de vídeo mediante conexión HDMI que permite la entrada y salida de video con soporte para resoluciones 4K, conexiones MIPI CSI-2 y DisplayPort, todas ellas hacen que sea adecuada para aplicaciones multimedia.

Además, ofrece opciones de almacenamiento como SD/eMMC, utilizable como dispositivo de arranque y almacenamiento adicional, y NAND Flash, que proporciona almacenamiento no volátil para datos persistentes.

2.4 OpenMP (Open Multi-Processing)

OpenMP es una API (Application Programming Interface) ampliamente utilizada para programar aplicaciones, compatible con los lenguajes C, C++ y Fortran, su objetivo principal es facilitar el uso de múltiples hilos de ejecución para mejorar el rendimiento en aplicaciones que requieren un uso intensivo de recursos computacionales.

Basada en el modelo Fork-Join, la programación en OpenMP permite dividir una tarea pesada en hilos más ligeros (fork), que operan de manera concurrente y luego sincroniza los resultados (join). Este modelo de programación paralela utiliza memoria compartida, permitiendo que varios hilos accedan a las mismas variables. Cada hilo tiene un identificador único que facilita la coordinación y evita intercambios explícitos de datos entre procesadores, lo que simplifica la programación y mejora el rendimiento en tareas intensivas.

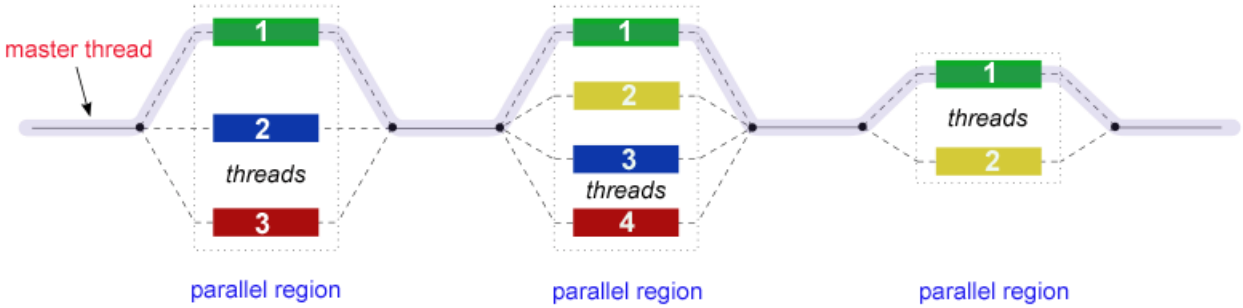


Figura 6. Regiones paralelas OpenMP

La figura 6 ilustra el modelo Fork-Join, este modelo comienza con un único proceso, el hilo maestro, al alcanzar una región paralela, el hilo maestro crea un equipo de hilos, que ejecutan de manera simultánea las instrucciones especificadas, una vez completadas, los hilos se sincronizan y terminan, regresando al hilo maestro para continuar la ejecución secuencial.

Esta API consta de tres elementos principales: las directivas de compilador, que permiten controlar de forma explícita la creación y gestión de hilos; las funciones de biblioteca, que facilitan la sincronización, el manejo de datos y el control de la ejecución paralela; y las variables de entorno, que permiten ajustar parámetros en tiempo de ejecución, como el número de hilos o el tipo de paralelismo.

Podemos resumir que OpenMP es una herramienta clave en la computación de alto rendimiento que simplifica tareas como la paralelización de bucles, sincronización y distribución de tareas. Su facilidad de uso, compatibilidad con compiladores estándar y características avanzadas, como la paralelización anidada y el soporte para arquitecturas heterogéneas, la hacen ideal para áreas como simulaciones científicas y análisis de datos a gran escala, aprovechando al máximo los recursos de hardware modernos.

2.5 LLVM (Low Level Virtual Machine)

LLVM es un conjunto de herramientas diseñadas de manera modular, lo que significa que sus componentes pueden ser combinados y utilizados de forma flexible según las necesidades del usuario. Esta estructura permite a los desarrolladores crear compiladores personalizados o experimentar sin necesidad de desarrollar todo el sistema desde cero.

La modularidad y la flexibilidad de LLVM también lo hacen muy atractivo para la investigación, ya que permite probar y ajustar diferentes optimizaciones, técnicas de análisis y generación de código sin tener que crear un nuevo compilador completo.

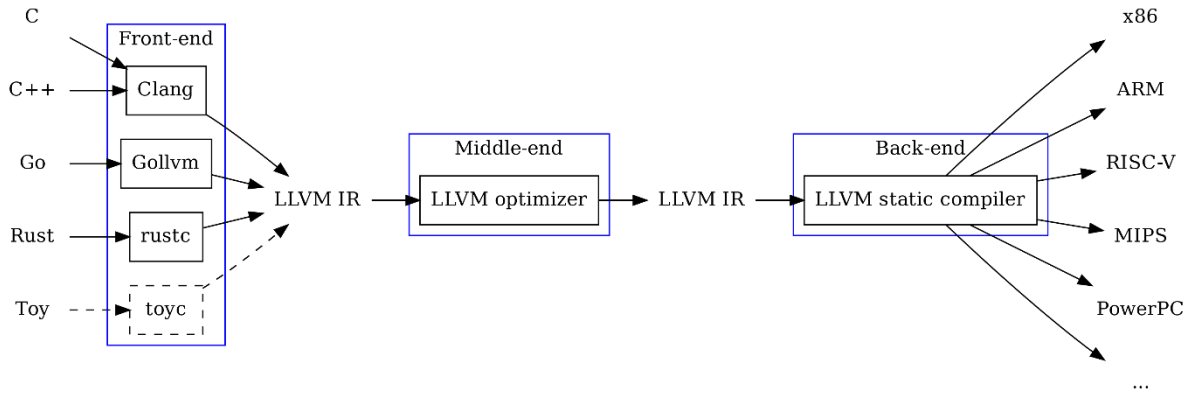


Figura 7. Herramientas compilación LLVM

A continuación, se va a realizar una explicación de la figura 7, que muestra el proceso de compilación mediante el uso de herramientas LLVM:

1. **Análisis léxico y sintáctico:** En LLVM, la interpretación del código fuente se realiza mediante Clang, el front-end del compilador, que verifica la sintaxis del código en C o C++, detecta errores y lo convierte en una representación intermedia (LLVM IR).

2. **Generación de código intermedio**: El código fuente se convierte en LLVM IR, una representación independiente de la arquitectura que facilita la optimización y generación de código para diferentes plataformas.

3. **Optimización de código:** En esta etapa denominada middle-end, LLVM IR Optimizer mejora la eficiencia del código eliminando redundancias, optimizando registros y memoria, y mejorando el rendimiento general del programa.

4. **Generación de código para el sistema objetivo**: El back-end de LLVM, a través de LLVM Code Generator, convierte la representación intermedia en instrucciones de bajo nivel específicas para la arquitectura de destino, generando código ensamblador o un binario ejecutable.

5. **Enlace y creación del ejecutable:** Finalmente, se integran las bibliotecas necesarias y se genera el archivo ejecutable compatible con la arquitectura de destino.

2.6 Compilador Cruzado

Un compilador cruzado es una herramienta que permite compilar código en un sistema (host) para que pueda ejecutarse en otro sistema diferente (target).

Los compiladores cruzados son esenciales en el desarrollo de software para sistemas embebidos y otras plataformas con recursos limitados. Sus principales ventaja incluyen:

- **Portabilidad del software**: Facilitan la creación de programas que pueden ejecutarse en distintos sistemas operativos y arquitecturas de hardware.
- **Mayor eficiencia en el desarrollo**: Permiten trabajar en un entorno estable y familiar, reduciendo tiempos y esfuerzo en comparación con la compilación directa en el dispositivo de destino.
- **Soporte para sistemas embebidos**: Son fundamentales para programar dispositivos como microcontroladores, sistemas IoT y otros equipos con capacidades de procesamiento restringidas.

La plataforma HERO utiliza la cadena de herramientas heterogénea de LLVM, que, a través de Clang y OpenMP, permite la compilación cruzada y la generación de aplicaciones, tal como se indica en el documento README [5].

A continuación, basado en los conocimientos previos explicados en este capítulo, se explica el proceso de compilación cruzada utilizado en este proyecto:

El proceso comienza con Clang, que traduce el código en C a una representación intermedia independiente de la arquitectura de destino. En esta fase, se generan etiquetas que indican las arquitecturas específicas en las que se ejecutará el código. Posteriormente, herramientas de LLVM optimizan y adaptan esta representación para cada procesador dentro de la plataforma, garantizando una ejecución eficiente tanto en el host ARM como en los aceleradores RISC-V.

Gracias a la API OpenMP, la cual es soportada por Clang, el código puede dividirse estratégicamente, las tareas generales se ejecutan en el procesador host ARM, mientras que las operaciones intensivas en cómputo se delegan a los aceleradores RISC-V. Una vez aplicadas estas directivas, LLVM genera código optimizado para cada arquitectura y gestiona la distribución eficiente de los hilos de ejecución.

Este enfoque no solo permite una compilación cruzada efectiva, sino que también maximiza el rendimiento del hardware mediante la computación heterogénea.

3. Especificaciones y restricciones de diseño

A continuación, se detallarán las especificaciones planteadas para el diseño de este proyecto:

- **Rendimiento:** El sistema debe maximizar el rendimiento computacional mediante una configuración óptima de los clústeres, garantizando así una alta eficiencia en el uso de los recursos.

- **Gestión eficiente de datos:** Se debe garantizar que el sistema sea capaz de procesar grandes volúmenes de datos con un mínimo de latencia, optimizando la comunicación entre la memoria y los componentes de procesamiento.

- **Compatibilidad**: Las pruebas implementadas mediante software deben ser compatibles con la configuración específica del procesador ARM Cortex-A53 como host y los clústeres basados en RISC-V. Además, se requiere que el sistema operativo sea Linux, dado su soporte para herramientas de código abierto como OpenMP.

- **Consumo energético**: El sistema trabajara con el conjunto de instrucciones RISC-V y utilizara lógica programable garantizando así un equilibrio entre rendimiento y eficiencia energética.

- **Escalabilidad:** Se debe considerar la posibilidad de escalar el sistema para adaptarse a diferentes volúmenes de procesamiento de datos y tareas en el futuro, permitiendo la integración de nuevos componentes y configuraciones.

Por otro lado, vamos a revisar las restricciones a tener en cuenta durante el desarrollo del proyecto:

- **Recursos computacionales limitados**: El diseño debe considerar las restricciones del chip XCZU9EG de la tarjeta de desarrollo ZCU102, incluyendo las capacidades del procesador ARM y la FPGA, como la memoria disponible, el número de núcleos, las unidades lógicas y las diferencias en velocidades de procesamiento.

- **Herramientas software**: Se deberá utilizar y adaptarse a las herramientas y versiones específicas recomendadas dentro del proyecto HERO, esto incluye el uso de un entorno Linux compatible, como Ubuntu 18.04.

- **Plataforma en investigación:** La plataforma HERO es un proyecto en investigación y desarrollo continuo, lo que podría requerir pequeñas modificaciones o ajustes para su completa adaptación a las necesidades de este proyecto.

4. Descripción del trabajo realizado

4.1 Prototipado de HERO

En este apartado se van a detallar los pasos especificados en el documento "readme" del proyecto HERO [5], en los que hay realizar pequeñas modificaciones o se añaden advertencias en su ejecución. Para los pasos detallados en este apartado se han generado pequeños scripts que automatizan cada la ejecución de cada subapartado.

4.1.1 Adecuación del entorno y comienzo del proyecto

Para comenzar a trabajar con la plataforma HERO, es necesario preparar el entorno de desarrollo adecuadamente. El script 1, presentado a continuación, automatiza la descarga del código fuente, la creación de los directorios necesarios y la declaración de variables.

```
#!/bin/bash

cd /data/local/hero
git clone --recursive https://github.com/pulp-platform/hero.git
cd hero
mkdir logs
export LOG_ROOT=/data/local/hero/hero/logs
mkdir install
export HERO_INSTALL=/data/local/hero/hero/install
make prem-unset 2>&1 | tee $LOG_ROOT/log.make.prem-unset.txt
```

Script 1. Adecuación y comienzo del proyecto

El proceso de configuración del entorno comienza accediendo al directorio base del proyecto y clonando el repositorio de HERO junto con sus submódulos, asegurando así la descarga completa de todos los componentes necesarios.

A continuación, dentro del directorio descargado, se crea el directorio "logs" donde incluiremos ficheros de texto que guarden toda la información en procesos de compilación, posteriormente se define la variable de entorno "LOG_ROOT", esta variable de entorno contiene la ruta hacia la carpeta "logs", de esta forma nos ahorramos tener que escribir la ruta siempre que compilemos.

Luego, se crea el directorio "install" que contendrá todas las herramientas y bibliotecas necesarias en HERO, posteriormente se define la variable "HERO_INSTALL", esta variable de entorno contiene la ruta hacia la carpeta "install", algunos ficheros de HERO necesitan esta variable de entorno definida para poder ser ejecutados.

Por último, se desactiva la compatibilidad con el modelo PREM (Predictable Execution Model), aunque por defecto suele venir desactivado debemos asegurarnos.

Mediante la ejecución de este script se garantiza que el entorno de HERO esté correctamente preparado antes de continuar con la instalación y compilación del software.

El uso del comando “make” permite automatizar el proceso de compilación y gestión de dependencias entre los distintos componentes.

Siempre que realicemos la ejecución del comando “make” vamos a seguir un patrón definido, por ello se realiza una pequeña explicación de lo que estamos haciendo, ayudándonos de la siguiente imagen:

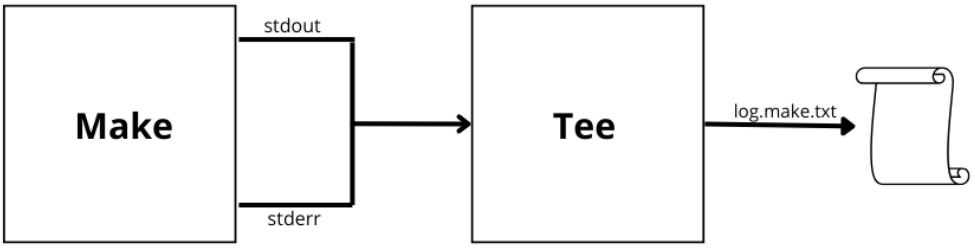


Figura 8. Explicación redirección de la salida de comandos a un archivo de texto

Al ejecutar el comando “make prem-unset”, se generan dos tipos de salidas:

- La salida estándar (stdout), que muestra la información normal o esperada durante la ejecución.
- La salida de error (stderr), que contiene los mensajes de errores o advertencias generados.

El operador “2>&1”, redirige la salida de error a la salida estándar, permitiendo que ambos tipos de información sean visibles en el mismo flujo.

Posteriormente, el operador “| “conecta este flujo combinado a la entrada estándar (stdin) del comando “tee”, el cual lee los datos de su entrada, los muestra en pantalla y, al mismo tiempo, guarda esta información en un archivo de texto ubicado en la ruta definida por LOG_ROOT, con el nombre log.make.prem-unset.txt.

Este enfoque resulta especialmente útil para ejecutar procesos que generan una gran cantidad de información, ya que permite revisar detenidamente los detalles de la ejecución, incluidos los errores, en el archivo generado, evitando que se pierdan detalles importantes en el terminal.

4.1.2 Generación y configuración de toolchains y SDKs

En este apartado se describe el proceso de generación y configuración de las toolchains y SDKs necesarios para la plataforma HERO. Estas herramientas permiten la compilación y ejecución de aplicaciones heterogéneas, asegurando la compatibilidad entre el host y el acelerador.

El script 2, presentado a continuación, automatiza la instalación de las toolchains y SDKs requeridos, garantizando un entorno de trabajo funcional.

```
#!/bin/bash

cd /data/local/hero/hero
make tc-pulp 2>&1 | tee $LOG_ROOT/log.make.tc-pulp.txt
make tc-har-olinux 2>&1 | tee $LOG_ROOT/log.make.tc-har-olinux.txt
make sdk-pulp 2>&1 | tee $LOG_ROOT/log.make.sdk-pulp.txt
make sdk-har 2>&1 | tee $LOG_ROOT/log.make.sdk-har.txt
make tc-llvm 2>&1 | tee $LOG_ROOT/log.make.tc-llvm.txt
```

Script 2. Generación y configuración toolchains y SDKs

En primer lugar, se compila la toolchain para el acelerador PULP, la cual incluye compiladores, enlazadores etc.… Proporcionando las herramientas necesarias para el desarrollo en el procesador RISC-V de 32 bits. Posteriormente se realiza la compilación del toolchain de Linux para el procesador principal ARMv8 Cortex-A53 , permitiendo la ejecución de software en la CPU principal.

Posteriormente, configura los entornos de desarrollo (SDKs) para cada arquitectura, tanto para el acelerador PULP, donde se incluyen bibliotecas y entornos para su desarrollo, como para el host ARMv8, asegurando compatibilidad con la arquitectura del sistema.

Finalmente, se compila la toolchain basada en LLVM12, la cual está diseñada específicamente para crear aplicaciones heterogéneas que pueden ejecutarse en la plataforma HERO, esto incluye soporte para Clang y OpenMP, facilitando la creación de software optimizado para la plataforma.

Todas las toolchains y SDKs se instalan en ruta que anteriormente fue declarada en la variable “HERO_INSTALL”.

Como se ha explicado anteriormente, durante la ejecución de cada proceso se redirige su salida a un archivo de registro dentro de una ruta predefinida por “LOG_ROOT”, permitiendo un seguimiento detallado y la identificación de posibles errores en la instalación y configuración del entorno.

4.1.3 Síntesis del hardware de HERO en la tarjeta ZCU102 con Vivado

Para trabajar con herramientas de diseño FPGA, es necesario configurar el entorno y definir las variables que permitan ejecutar el software de síntesis e implementación. En este apartado, se presenta el script 3, el cual automatiza la configuración de Vivado y la generación del bitstream que permite la configuración de la FPGA presente en el chip XCZU9EG de la tarjeta de desarrollo ZCU102.

Antes de ejecutar estos comandos, es fundamental asegurarse de que la licencia de Vivado está activa en el sistema, ya que de lo contrario se producirá un error en los pasos siguientes.

```
#!/bin/bash

cd /data/local/hero/hero/hardware/fpga
source /opt/Xilinx/Vivado/2019.2/settings64.sh
make VIVADO=vivado hero_exilzcu102 2>&1|tee $LOG_ROOT/log.make.hero_exilzcu102.txt
```

Script 3. Síntesis del hardware con Vivado

El proceso empieza ejecutando el script “settings64.sh” proporcionado por Xilinx, este configura las variables de entorno necesarias para utilizar Vivado en su versión 2019.2.

Tras la configuración del entorno, se inicia el proceso de compilación del diseño para la FPGA, este proceso incluye la compilación, la síntesis, la implementación y la generación de un bitstream o fichero binario.

4.1.4 Configuración de variables para Petalinux

PetaLinux es una herramienta de Xilinx integrada con Vivado para la generación y personalización del software de sistemas embebidos basados en Linux. Para su correcta ejecución, es necesario definir ciertas variables de entorno y configurar los archivos necesarios. En este apartado, se presenta el script 4 que automatiza dicho proceso.

```
#!/bin/bash

cd /data/local/hero/hero
echo"BR2_HERO_BITSTREAM=data/local/hero/hero/hardware/fpga/hero_exilzcu102/
hero_exilzcu102.runs/impl_1/hero_exilzcu102_wrapper.bit" >> local.cfg
source /opt/Xilinx/Petalinux/2019.2/settings.sh
```

Script 4. Configuración de variables Petalinux

En principio, se comienza definiendo la variable “BR2_HERO_BITSTREAM”, la cual almacena la ruta donde se encuentra el bitstream. La variable se añade al archivo “local.cfg”, asegurando que la configuración del bitstream esté disponible para el sistema de compilación de HERO en futuras ejecuciones.

A continuación, se ejecuta el script “settings.sh”, que configura automáticamente las variables de entorno necesarias para utilizar las herramientas de PetaLinux, permitiendo la compilación y personalización del sistema operativo embebido que se ejecutará en el host ARMv8.

4.1.5 Creación y compilación de la imagen de sistema

En este apartado se lleva a cabo la compilación y generación de la imagen utilizando Buildroot, una herramienta que facilita la creación de sistemas de archivos para sistemas embebidos. Buildroot compila el sistema operativo, el kernel de Linux y todas las dependencias necesarias, generando una imagen del sistema operativo que se puede cargar en la tarjeta SD para arrancar el sistema. Además, se utiliza PetaLinux para generar los archivos de arranque requerido, configurado específicamente para el procesador ARM del chip XCZU9EG presente en la tarjeta de desarrollo ZCU102. El script 5 automatiza la compilación y generación de la imagen:

```
#!/bin/bash

cd /data/local/hero/hero
unset LD_LIBRARY_PATH
NO_IIS=1 make br-har-exilzcu102 2>&1 | tee $LOG_ROOT/log.make.br-har-exilzcu102.txt
```

Script 5. Creación y compilación de la imagen del sistema

Se comienza ejecutando el comando que elimina cualquier valor previamente definido para la variable de entorno “LD_LIBRARY_PATH”, esto evitará posibles conflictos.

A continuación, se ejecuta el comando que inicia la compilación de la imagen de Buildroot para la arquitectura ARMv8, la cual, gracias a la herramienta Petalinux genera un sistema Linux embebido.

Durante este paso, surgieron inconvenientes relacionados con la versión del sistema operativo. El proyecto se está desarrollando en un ordenador con Ubuntu 20.04, mientras que PetaLinux 2019.2 requiere Ubuntu 18.04 como sistema compatible. Además, se presentó otro problema al ejecutar este comando, ya que intenta clonar un repositorio de QEMU desde Git, pero la ubicación original del repositorio ha cambiado, lo que genera un error al no poder acceder al recurso esperado. Por lo tanto, se debe ejecutar el script 5 hasta que surjan los errores, esto es necesario porque se necesita que descargue ciertos paquetes para la solución propuesta, una vez se obtiene el error se ejecuta el siguiente script:

```
#!/bin/bash

cd /data/local/hero/hero
patch -p0 -d petalinux/zcu102 < /mnt/hero/src/hero.patches/qemu-xilinx-native.patch
patch -p0 -d petalinux/zcu102 < /mnt/hero/src/hero.patches/qemu-native.patch
```

Script 6. Parches Qemu

El script 6 ejecuta unos parches que modifican algunos paquetes de QEMU, los cuales se descargaron anteriormente. Además, incluyen la corrección de la URL del repositorio de Git, solucionando el problema causado por el cambio de ubicación del repositorio.

A continuación, se vuelve a ejecutar la última instrucción del script 5, y de forma exitosa generará la imagen del sistema.

4.1.6 Copia de las imágenes en la tarjeta SD

En este apartado se lleva a cabo la copia de la imagen de sistema, previamente generada, a la tarjeta SD. Esta tarjeta SD funcionará como memoria ROM en la tarjeta de desarrollo ZCU102, permitiendo la correcta inicialización y ejecución del sistema embebido Linux.

```
ivan.vitores@gdem18:/$ df -h
S.ficheros     Tamaño Usados  Disp Uso% Montado en
udev             16G     0   16G   0% /dev
tmpfs           3,2G   2,0M 3,2G   1% /run
/dev/sda5       457G   414G  20G  96% /
tmpfs            16G     0   16G   0% /dev/shm
tmpfs           5,0M   4,0K 5,0M   1% /run/lock
tmpfs            16G     0   16G   0% /sys/fs/cgroup
/dev/loop0      128K   128K     0 100% /snap/bare/5
/dev/loop3       74M    74M     0 100% /snap/core22/1663
/dev/loop1       56M    56M     0 100% /snap/core18/2829
/dev/loop2       64M    64M     0 100% /snap/core20/2379
/dev/loop4       64M    64M     0 100% /snap/core20/2434
/dev/loop6      360M   360M     0 100% /snap/emacs/2504
/dev/loop5      219M   219M     0 100% /snap/gnome-3-34-1804/90
/dev/loop10     350M   350M     0 100% /snap/gnome-3-38-2004/143
/dev/loop8       65M    65M     0 100% /snap/gtk-common-themes/1514
/dev/loop9      330M   330M     0 100% /snap/emacs/2307
/dev/loop7      219M   219M     0 100% /snap/gnome-3-34-1804/93
/dev/loop11      56M    56M     0 100% /snap/core18/2846
/dev/loop14     350M   350M     0 100% /snap/gnome-3-38-2004/140
/dev/loop12     497M   497M     0 100% /snap/gnome-42-2204/141
/dev/loop20      13M    13M     0 100% /snap/snap-store/1113
/dev/loop19      45M    45M     0 100% /snap/snapd/23258
/dev/loop18      92M    92M     0 100% /snap/gtk-common-themes/1535
/dev/loop13      74M    74M     0 100% /snap/core22/1722
/dev/loop15     506M   506M     0 100% /snap/gnome-42-2204/176
/dev/loop16      13M    13M     0 100% /snap/snap-store/1216
/dev/sda1       511M   4,4M  507M   1% /boot/efi
tmpfs           3,2G   104K  3,2G   1% /run/user/2128
/dev/loop21      45M    45M     0 100% /snap/snapd/23545
/dev/sdb2        52M    20M   28M  42% /media/ivan.vitores/rootfs
/dev/sdb1       256M   178M   79M  70% /media/ivan.vitores/50C9-A1AF
```

Figura 9. Salida del comando df -h

A continuación, se debe introducir la tarjeta SD en el ordenador, si fuese necesario mediante el uso de un adaptador. En la figura 9, se puede observar que el comando "df -h" nos muestra información sobre el espacio de almacenamiento disponible, al introducir la tarjeta SD podemos observar la ruta asociada a ella, la cual será necesaria en los pasos siguientes.

```
#!/bin/bash

cd /data/local/hero/output
sudo dd if=har-exilzcu102-base-sdcard.img of=/dev/sdb bs=4k
```

Script 7. Copia imagen del sistema en SD

El script 7 permite realizar la copia del fichero "har-exilzcu102-base-sdcard.img", el cual contiene la imagen de sistema, en la tarjeta SD cuya ruta como se observa en la figura 9 es "/dev/sdb". La copia del fichero se realiza en bloques de cuatro kilobytes.

```
ivan.vitores@gdem18:/$ sudo parted /dev/sdb
[sudo] contraseña para ivan.vitores:
GNU Parted 3.3
Usando /dev/sdb
Bienvenido(a) a GNU Parted! Escriba 'help' para ver una lista de órdenes.
(parted) print
Modelo: Generic- Multiple Reader (scsi)
Disco /dev/sdb: 63,9GB
Tamaño de sector (lógico/físico): 512B/512B
Tabla de particiones: msdos
Banderas de disco:

Número  Inicio  Fin    Tamaño  Tipo     Sistema de archivos  Banderas
 1      512B    268MB  268MB   primary  fat16                arranque, lba
 2      268MB   331MB  62,9MB  primary  ext4

(parted)
```

Figura 10. Revisar archivos SD

En la figura 10, se observa que mediante el comando "parted /dev/sdb" y la opción "print" se ha realizado correctamente la copia de las particiones necesarias en la tarjeta SD.

4.2 Arranque y configuración de la tarjeta de desarrollo ZCU102

En este apartado se describen los pasos para conectar y configurar la tarjeta de desarrollo ZCU102, incluyendo la conexión de periféricos, la configuración del sistema de arranque y el establecimiento de comunicación con el ordenador mediante una terminal serie. Además, se explica el inicio de sesión en el sistema y la configuración de una conexión NFS para compartir archivos entre el ordenador y la tarjeta.

4.2.1 Conexionado inicial

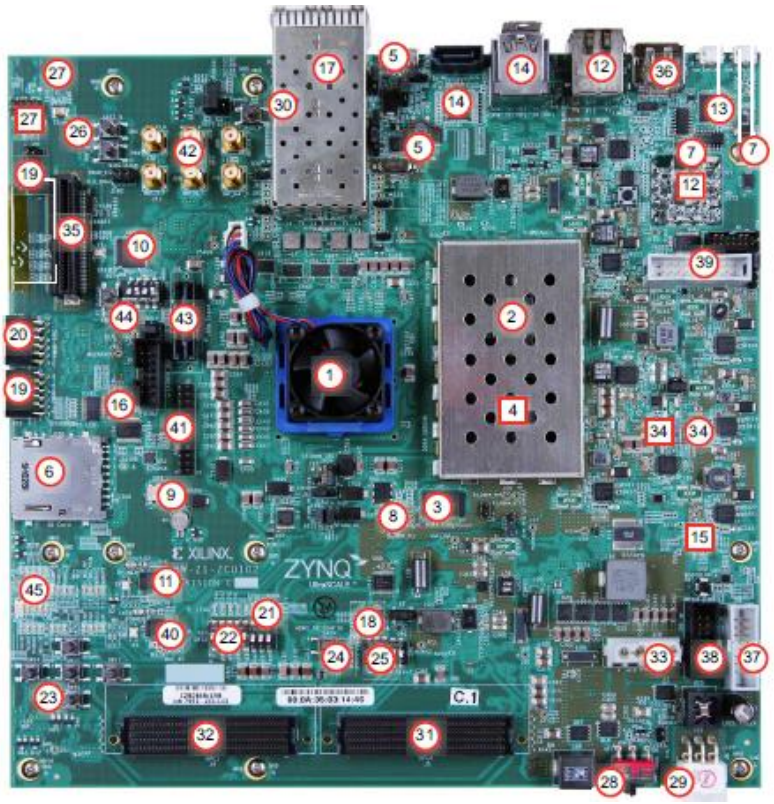


Figura 11. Enumeración componente ZCU102

La figura 11, obtenida de la guía de usuario [1], muestra la tarjeta con la numeración de sus componentes y se utiliza para referenciar las conexiones necesarias. Primero, se conecta la alimentación en el conector 29 y se inserta la tarjeta SD en el conector 6. Luego, se establece la conexión USB micro-B para UART en el conector 13, además de conectar un cable Ethernet al conector 12.

Boot Mode	Mode Pins [3:0]	Mode SW6 [4:1]
JTAG	0000	on, on, on, on
QSPI32	0010(1)	on, on, off, on
SD	1110	off, off, off, on

Figura 1. Configuración de arranque

Según la figura 12, obtenida de la guía de usuario [1], el tipo de arranque depende de la configuración del switch 6,el cual corresponde al número 44 de la figura 11. Dado que el arranque se realiza desde la tarjeta SD, se debe seleccionar la última opción de la figura. Finalmente, la tarjeta se enciende a través del conector 28 de la figura 11.

4.2.2 Configuración inicial

Para trabajar con la tarjeta de desarrollo ZCU102, es necesario utilizar una interfaz gráfica que permita visualizar la terminal en la pantalla del ordenador. En este proyecto se ha optado por GTKTerm, una aplicación de terminal serie diseñada para comunicarse con dispositivos a través de puertos serie. Esta herramienta permite interactuar con sistemas embebidos, microcontroladores y otros dispositivos que utilizan la comunicación serie UART.

Figura 13. Terminal GTKTerm

En la figura 13 se muestra la pantalla inicial de la terminal GTKTerm. Para configurarla, se selecciona la opción "Configuration" en la barra de menú superior y, a continuación, se hace clic en "Port", esto abrirá la ventana de configuración del puerto serie.

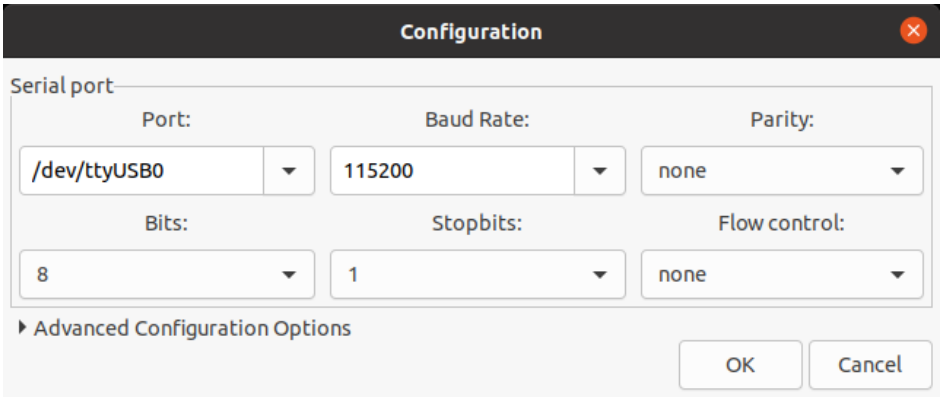


Figura 14. Ventana de configuración GTKTerm

En la figura 14 se muestra la ventana de configuración con los parámetros óptimos para la comunicación serie con la tarjeta de desarrollo. El puerto utilizado en este caso es "USB0", aunque este puede variar en cada conexión. Por ello, es necesario verificarlo cada vez que se encienda la tarjeta de desarrollo.

Tras configurar la terminal GTKTerm para la comunicación serie mediante el protocolo UART y conectar todos los elementos en la tarjeta de desarrollo ZCU102, se puede encender. Si todo el proceso se ha realizado correctamente, se deberá observar la siguiente salida en pantalla:

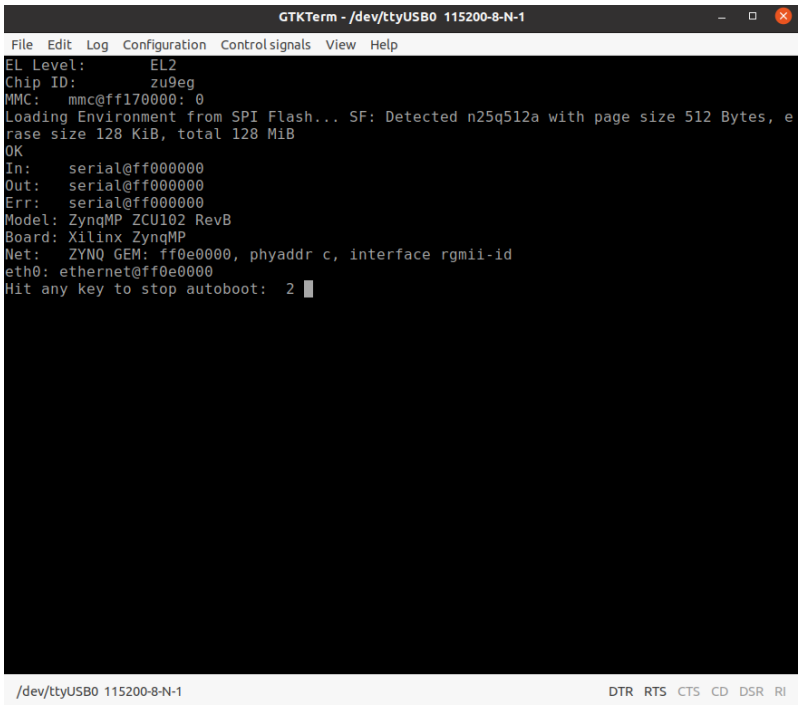


Figura 15. Autoboot GTKTerm

En la Figura 15, se observa el sistema de arranque "U-Boot**"**, al ejecutarse busca automáticamente la imagen del sistema operativo. En este proyecto, dicha imagen está almacenada en la tarjeta SD.

La primera vez que se inicia el sistema, es necesario realizar una configuración inicial que especifica la región de memoria donde deben ser almacenados ciertos datos evitando así el desbordamiento. El documento "readme" del proyecto HERO [5] menciona esta configuración y hace referencia al "issue #107". Sin embargo, debido a cambios en los repositorios de GitHub, esta nota ya no está disponible. Este problema fue resuelto en la comunidad de PULP [7]. Para realizar la configuración inicial del sistema de arranque, se debe interrumpir el proceso pulsando cualquier tecla antes de que finalice la cuenta regresiva mostrada en la figura 34 e introducir los siguientes comandos:

```
Setenv initrd_high 0x10000000
Saveenv
boot
```

Una vez introducidos los comandos, el sistema se configurará y arrancará. Como se ha mencionado, esta operación solo debe realizarse la primera vez que iniciamos el sistema.

En las siguientes ocasiones en que se encienda la tarjeta de desarrollo ZCU102, la configuración ya estará guardada. Por lo tanto, al finalizar la cuenta regresiva del sistema de arranque "U-Boot" se cargará automáticamente la imagen almacenada en la tarjeta SD.

4.2.3 Inicio y cierre de sesión

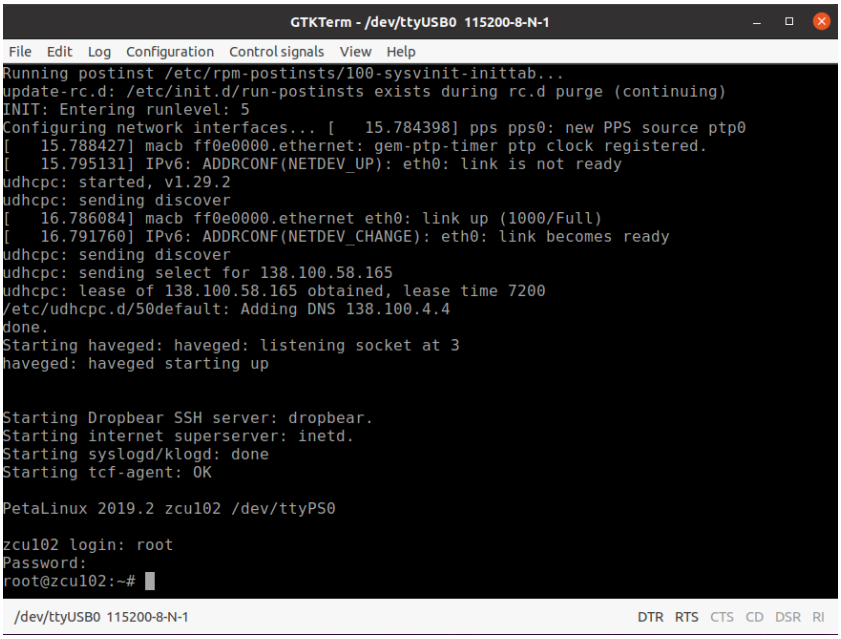


Figura 16. Inicio de sesión

En la figura 16 se observa que, después de mostrar toda la información relevante sobre la configuración del sistema, se solicita usuario y contraseña, ambos se muestran a continuación:

```
Login: root
Password: root
```

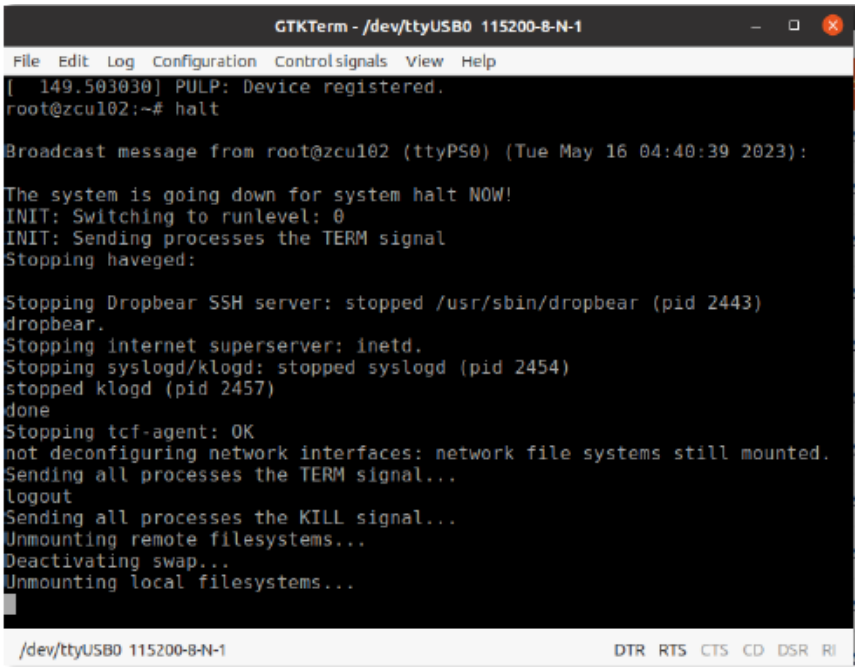


Figura 17. Cierre de sesión

En la figura 17 se observa como el comando “halt” detiene los procesos y desmonta el sistema de archivos, esto permite detener los procesos activos en el procesador ARM de la tarjeta de desarrollo ZCU102 de forma segura para realizar el apagado de la tarjeta.

4.2.4 Configuración y establecimiento de conexión NFS

Una conexión NFS (Network File System) es un método que permite acceder a archivos de un sistema remoto como si estuvieran en el disco local de tu máquina, NFS es un protocolo de red que facilita la compartición de archivos entre sistemas en una red.

Para compartir archivos del ordenador con la tarjeta de desarrollo ZCU102 se establecerá una conexión NFS, el ordenador actuará como servidor, mientras que la tarjeta de desarrollo se configura como cliente, permitiendo el acceso remoto a los archivos compartidos. A continuación, se realiza una breve explicación de cómo realizar esta conexión:

```
root@zcu102:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
3: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0a:35:00:22:01 brd ff:ff:ff:ff:ff:ff
    inet 138.100.58.165/24 brd 138.100.58.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20a:35ff:fe00:2201/64 scope link
       valid_lft forever preferred_lft forever
```

Figura 18. Comando ip a en cliente

En la figura 18 se observa la ejecución del comando “ip a” en el terminal de la tarjeta de desarrollo, el cual muestra información detallada sobre las direcciones IP asignadas a las interfaces de red del sistema. En este caso, podemos ver que, mediante la conexión Ethernet, se ha asignado la dirección IP “138.100.58.165” a una de las interfaces de red, lo que permite establecer comunicación con otros dispositivos en la red.

```
lvan.vitores@gdem18:~$ sudo exportfs -v
/opt/Xilinx     gdem18-ubuntu1804.local(ro,wdelay,root_squash,no_subtree_check,sec=sys,ro,secure,root_squash,no_all_squash)
/home/pjlobo/src
                gdem18-ubuntu1804.local(rw,wdelay,root_squash,no_subtree_check,sec=sys,rw,secure,root_squash,no_all_squash)
/mnt/hero       gdem18-ubuntu1804.local(rw,wdelay,root_squash,no_subtree_check,sec=sys,rw,secure,root_squash,no_all_squash)
/mnt/hero       138.100.58.165(rw,wdelay,no_root_squash,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/data/local/hero/hero
                gdem18-ubuntu1804.local(rw,wdelay,root_squash,no_subtree_check,sec=sys,rw,secure,root_squash,no_all_squash)
/data/local/hero/hero
                138.100.58.165(rw,wdelay,no_root_squash,no_subtree_check,sec=sys,rw,secure,no_root_squash,no_all_squash)
/mnt/hero       <world>(ro,wdelay,root_squash,no_subtree_check,sec=sys,ro,secure,root_squash,no_all_squash)
/data/local/hero/hero
                <world>(ro,wdelay,root_squash,no_subtree_check,sec=sys,ro,secure,root_squash,no_all_squash)
```

Figura 19. Comando exportfs -v en servidor

En la figura 19 se muestra la ejecución del comando “exportfs -v” en la terminal del ordenador con privilegios de superusuario (root). Este comando permite revisar las conexiones NFS configuradas en el servidor, visualizar el sistema de carpetas que han sido compartidas y verificar las direcciones IP de los clientes conectados. En caso de que alguna configuración no sea correcta, es necesario modificar el fichero “exports”. Como se puede observar, el sistema de carpetas compartidas con el cliente y su dirección IP están correctamente configurados.

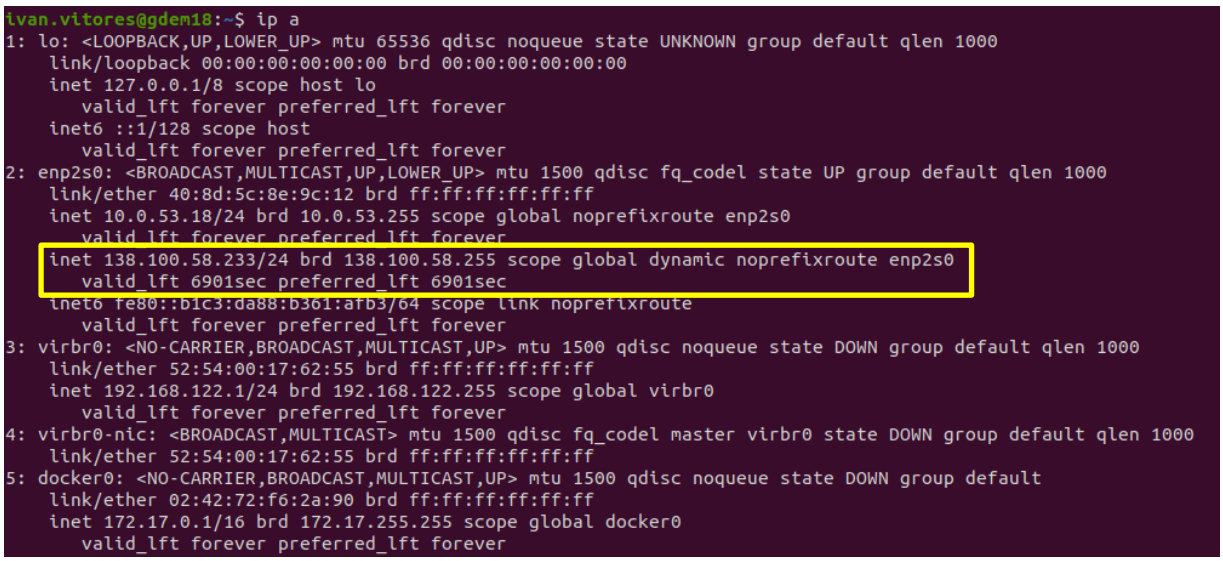


Figura 20. Comando ip a en servidor

Ahora es necesario realizar la operación inversa. En la figura 20 se muestra la ejecución del comando “ip a” en la terminal del ordenador, que proporciona información sobre las direcciones IP de las interfaces de red del sistema. En este caso, podemos ver que, mediante la conexión Ethernet, se le ha asignado la dirección IP “138.100.58.233”, la cual será necesaria para incluir en el fichero de configuración del cliente.

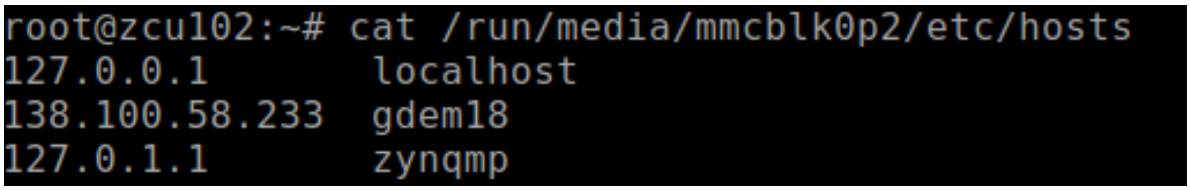


Figura 21. Fichero hosts cliente

En la figura 21 se muestra el contenido del fichero hosts en el terminal de la tarjeta de desarrollo. En este fichero se debe verificar si la dirección IP del servidor está correctamente incluida; si no es así, debe añadirse junto con su nombre correspondiente. En este caso, podemos observar que la dirección IP está correctamente establecida y asignada al nombre gdem18.

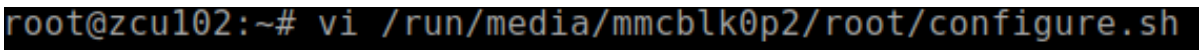


Figura 22. Comando vi en cliente

Como se puede observar en la figura 22, se ejecuta el comando “vi” esto permitirá crear un script en la ruta específica mostrada en la figura, es importante asegurarse de que la ruta coincida exactamente con la indicada. Este script será utilizado cada vez que se encienda la tarjeta de desarrollo para establecer la conexión NFS.

A continuación, se muestra el contenido del script que debe ser introducido en el archivo recién creado y se detalla su funcionalidad:

```
#!/bin/sh

export MMROOT=/run/media/mmcblk0p2
export LD_LIBRARY_PATH=$MMROOT/usr/lib:$MMROOT/lib
insmod $MMROOT/lib/modules/4.19.0/extra/pulp.ko
mkdir -p /data/gdem18/hero/hero
$MMROOT/sbin/mount.nfs -o rw `grep gdem18 $MMROOT/etc/hosts | awk '{print
$1}'`:/data/local/hero/hero /data/gdem18/hero/hero
```

Script 8. Configuración de la terminal

En primer lugar, las dos primeras líneas definen variables de entorno esenciales para el sistema, estableciendo rutas que apuntan a directorios importantes en el sistema de archivos.

A continuación, el script 8 carga el módulo del kernel “pulp.ko”, utilizando el comando “insmod”, lo que permite integrar el soporte necesario para la plataforma PULP. Después, se asegura de que exista el directorio de trabajo “/data/gdem18/hero/hero” en la tarjeta de desarrollo, creándolo si no está presente.

Finalmente, el script 8 monta el directorio remoto "/data/local/hero/hero" desde el servidor en el directorio local “/data/gdem18/hero/hero” de la tarjeta de desarrollo. Esto se realiza utilizando el comando “mount.nfs” con opciones de lectura y escritura, facilitando así el acceso a los archivos a través de la red mediante el sistema de archivos NFS.

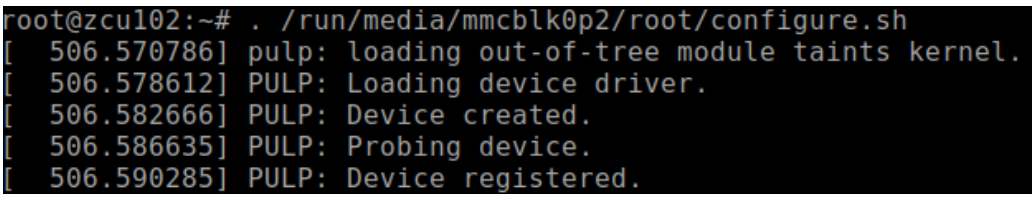


Figura 232. Ejecución script configure.sh

En la figura 23 se puede observar la ejecución del script 8 de configuración en la terminal de la tarjeta de desarrollo, en ella se observa que la configuración se ha realizado correctamente.

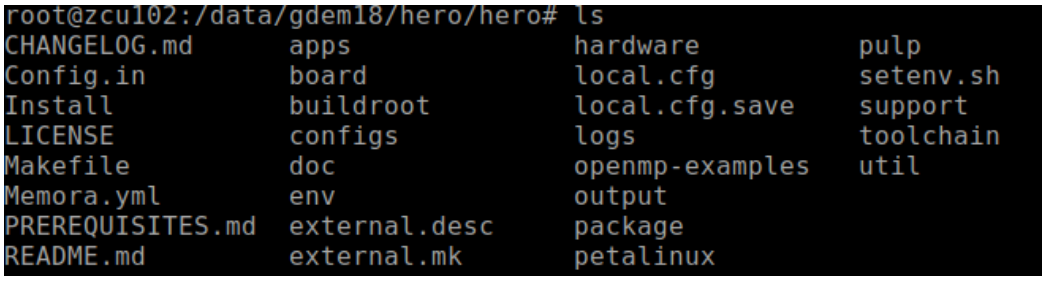


Figura 24. Archivos compartidos servidor-cliente

En la figura 24 se observa el contenido presente en la ruta “/data/gdem18/hero/hero”, el cual coincide con el contenido presente en la ruta “/data/local/hero/hero”, esto confirma que se ha establecido de manera exitosa la conexión NFS, permitiendo así compartir los ficheros entre servidor y cliente.

4.3 Comprobación del funcionamiento

En este apartado se verifica el correcto funcionamiento del entorno de desarrollo configurado en la plataforma HERO. Para ello, se llevará a cabo tanto el proceso de compilación cruzada, que permite ejecutar diferentes partes del código en el procesador ARM o en los aceleradores RISC-V, como la ejecución en la tarjeta de desarrollo, verificando que ambos procesos se realizan correctamente.

4.3.1 Compilación de programas para el sistema HERO

Entre los ficheros descargados de HERO existe un directorio llamado “openmp-examples”, esta carpeta contiene el código de varios programas que nos permitirán realizar la prueba del correcto funcionamiento de todo lo desarrollado hasta este punto del proyecto.

```
ivan.vitores@gdem18:/data/local/hero/hero/openmp-examples/helloworld$ ls
helloworld.c  Makefile
```

Figura 25. Comando ls helloworld

En la figura 25 se puede observar el contenido de la carpeta “helloworld”, en ella se incluye el código del programa “helloworld.c” y el fichero “Makefile” que contiene las reglas para la compilación del código de programación.

Para realizar una correcta compilación del programa, si se ha cerrado la terminal del ordenador, se debe volver a definir la variable “HERO_INSTALL y posteriormente ejecutar el script “exilzcu102.sh” que define variables de entorno necesarias, este es el trabajo del siguiente script:

```
#!/bin/bash

cd /data/local/hero/hero
export HERO_INSTALL=/data/local/hero/hero/install
cd
. /data/local/hero/hero/env/exilzcu102.sh
```

Script 9. Definición de variables para la compilación

```
ivan.vitores@gdem18:/data/local/hero/hero/openmp-examples/helloworld$ make
DEFMK_ROOT=/data/local/hero/hero/openmp-examples/common
MAKEFILE_LIST= Makefile /data/local/hero/hero/openmp-examples/helloworld/../common/default.mk
clang -c -emit-llvm -S -MT helloworld.ll -MMD -MP -MF .deps/helloworld.d --sysroot=/data/local/hero/hero/Install/aarch64-hero-linux-gnu/ -target aarch64-hero-linux-gnu  -fopenmp=libomp -O3 -static -fopen
mp-targets=riscv32-hero-unknown-elf -I/data/local/hero/hero/openmp-examples/common -include hero_64.h helloworld.c
clang-offload-bundler -inputs=helloworld.ll -outputs="helloworld-host.ll,helloworld-dev.ll" -type=ll -targets="host-aarch64-hero-linux-gnu,openmp-riscv32-hero-unknown-elf" -unbundle
hc-omp-pass helloworld-host.ll OmpKernelWrapper "HERCULES-omp-kernel-wrapper" helloworld-host.TMP.1.ll
cp helloworld-host.TMP.1.ll helloworld-host.OMP.ll
cp helloworld-dev.ll helloworld-dev.TMP.1.ll
hc-omp-pass helloworld-dev.TMP.1.ll OmpKernelWrapper "HERCULES-omp-kernel-wrapper" helloworld-dev.TMP.2.ll
hc-omp-pass helloworld-dev.TMP.2.ll OmpHostPointerLegalizer "HERCULES-omp-host-pointer-legalizer" helloworld-dev.TMP.3.ll
cp helloworld-dev.TMP.3.ll helloworld-dev.OMP.ll
clang-offload-bundler -inputs="helloworld-host.OMP.ll,helloworld-dev.OMP.ll" -outputs=helloworld-out.ll -type=ll -targets="host-aarch64-hero-linux-gnu,openmp-riscv32-hero-unknown-elf"
clang  --sysroot=/data/local/hero/hero/Install/aarch64-hero-linux-gnu/ -target aarch64-hero-linux-gnu  -fopenmp=libomp -O3 -static -fopenmp-targets=riscv32-hero-unknown-elf  helloworld-out.ll  -static -l
hero-target -o helloworld
aarch64-hero-linux-gnu-objdump -d helloworld > helloworld.dis
/data/local/hero/hero/openmp-examples/common/extract_device_image.py helloworld helloworld_riscv.elf \
      && riscv32-hero-unknown-elf-objdump -d helloworld_riscv.elf > helloworld.pulp.dis
299344+0 registros leídos
299344+0 registros escritos
299344 bytes (299 kB, 292 KiB) copied, 0,882948 s, 339 kB/s
rm helloworld-host.OMP.ll helloworld-dev.OMP.ll
```

Figura 26. Comando make en helloworld

En la figura 26 se observa la ejecución del comando “make”, esto iniciará la compilación siguiendo las reglas establecidas en el fichero “Makefile” y se generará un ejecutable.

4.3.2 Ejecución de programas en el sistema HERO implementado en la tarjeta ZCU102

El proceso de compilación cruzada utilizado genera ficheros intermedios que permiten inspeccionar cada etapa del proceso, siendo menos opaco para el usuario en comparación con herramientas más automatizadas o entornos cerrados.

```
ivan.vitores@gdem18:/data/local/hero/hero/openmp-examples/helloworld$ ls
helloworld          helloworld-dev.TMP.1.ll  helloworld.dis            helloworld.ll        helloworld_riscv.elf
helloworld.c        helloworld-dev.TMP.2.ll  helloworld-host.ll        helloworld-out.ll    log.make
helloworld-dev.ll   helloworld-dev.TMP.3.ll  helloworld-host.TMP.1.ll  helloworld.pulp.dis  Makefile
```

Figura 27. Resultado compilación ordenador

En la Figura 27 se pueden observar los archivos generados tras la compilación. Entre ellos se incluyen representaciones intermedias del código, versiones desensambladas para su análisis y ficheros específicos para las arquitecturas ARM y RISC-V. Al final del proceso, todos estos archivos conducen a la generación de un ejecutable identificado en color verde con el nombre del programa “helloworld”.

```
root@zcu102:/data/gdem18/hero/hero/openmp-examples/helloworld# ls
Makefile                 helloworld-dev.ll         helloworld.dis
helloworld               helloworld-host.TMP.1.ll  helloworld.ll
helloworld-dev.TMP.1.ll  helloworld-host.ll        helloworld.pulp.dis
helloworld-dev.TMP.2.ll  helloworld-out.ll         helloworld_riscv.elf
helloworld-dev.TMP.3.ll  helloworld.c
```

Figura 28. Resultado compilación tarjeta desarrollo

En la figura 28 también se puede observar el resultado de la compilación en la terminal de la tarjeta de desarrollo, con esto se comprueba el correcto funcionamiento de la conexión NFS.

```
root@zcu102:/data/gdem18/hero/hero/openmp-examples/helloworld# ./helloworld
[ 1753.304324] PULP: Device opened.
Starting program execution.
>>> PRINTING BUFFER OF CORE 0:
Hello World, I am t[ 1753.527447] PULP: Device released.
hread 0 of 0
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 1:
Hello World, I am thread 0 of 1
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 2:
Hello World, I am thread 0 of 2
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 3:
Hello World, I am thread 0 of 3
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 4:
Hello World, I am thread 0 of 4
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 5:
Hello World, I am thread 0 of 5
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 6:
Hello World, I am thread 0 of 6
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 7:
Hello World, I am thread 0 of 7
<<< END OF BUFFER
Done offloading, cycles to execute kernel: 12595!
```

Figura 29. Ejecución helloworld

En la figura 29 se puede observar la ejecución del fichero ejecutable “helloworld” en la tarjeta de desarrollo ZCU102 y el resultado obtenido, cabe resaltar que el proyector HERO por defecto tiene configurado un cluster con ocho núcleos, por eso y debido a la funcionalidad del programa se obtiene ese resultado. Con esto se confirma el correcto funcionamiento de todos los pasos detallados y realizados hasta el momento en el proyecto.

4.4 Programa para la suma de vectores

Para comenzar con el propósito principal de este proyecto se realiza el diseño de un programa que sume vectores, llamado “sumaVectores.c”, este tipo de programa tiene baja complejidad en términos de programación, con accesos secuenciales a memoria y baja carga computacional, esto lo hace ideal para una primera prueba y su paralelización.

4.4.1 Diseño, compilación y ejecución del programa en el ordenador personal

Inicialmente el programa se diseña en el ordenador personal, con sistema operativo Windows, a través de la aplicación Visual Studio Code, permitiendo probar distintas versiones y realizar un testeo exhaustivo. Su objetivo es sumar dos vectores de diez millones de elementos, evaluando el rendimiento con diferentes números de hilos.

Dimensión	Nº Hilos	Tiempo ACC(ms)
10000000	1	62,000036
10000000	2	29,999971
10000000	4	17,000198

Tabla 1. Suma de vectores PC

En la tabla 1 se observa el resultado de la ejecución del programa, al incrementar el número de hilos se reduce el tiempo de ejecución, esto demuestra su correcto funcionamiento.

4.4.2 Compilación en el ordenador y ejecución en la tarjeta de desarrollo

En este apartado se debe modificar el programa añadiendo directivas y funciones de la API OpenMP que permitan dividir el trabajo entre el host ARM y el acelerador RISC-V.

```
ivan.vitores@gdem18:/data/local/hero/hero/openmp-examples/polybench-acc/gemm$ ls
a.out-openmp-riscv32-hero-unknown-elf  gemm.c  gemm.exp  gemm.h  log.make  Makefile
```

Figura 30. Ficheros gemm

En la Figura 30, se muestra la ruta del programa "gemm", el cual realiza la multiplicación de matrices basada en fórmulas de álgebra lineal, este programa realiza la división de tareas entre host y acelerador mediante directivas openMP, por lo tanto, se toma como ejemplo para la modificación del programa que realiza la suma de vectores, además de realizar una copia del fichero “Makefile” en nuestro proyecto, el cual es modificado para que realice la correcta compilación del programa “sumaVectores.c”.

Dimensión	Nº Hilos	Tiempo ACC(ms)
10000000	1	4175,930977
10000000	2	2504,062891
10000000	4	1667,866945

Tabla 2. Suma de vectores ZCU102

En la tabla 2 se observan los resultados de la ejecución del programa, se aprecia la disminución proporcional de tiempo con el aumento del número de hilo, el programa ha sido adaptado con éxito para la división de tareas entre el host ARM y el acelerador RISC-V y su ejecución.

4.5 Programa para la multiplicación de matrices

Para continuar con el estudio se realiza el diseño de un programa que multiplique matrices, existe todo un campo relacionado con el diseño de algoritmos que realicen la multiplicación eficiente de matrices, debido a su gran uso en el mundo de la computación. Este programa llamado "multMatrices.c" se realiza mediante un algoritmo básico. Este tipo de programa posee gran cantidad de accesos a memoria y carga computacional, esto permite realizar un estudio en mayor profundidad sobre las capacidades de la tarjeta de desarrollo.

4.5.1 Diseño, compilación y ejecución del programa en el ordenador personal

Como en el caso anterior inicialmente el programa se diseña en el ordenador personal, con sistema operativo Windows, mediante la aplicación Visual Studio Code, permitiendo realizar varias iteraciones y un testeo exhaustivo de resultados. El objetivo de este programa es multiplicar dos matrices cuadradas evaluando el rendimiento con diferentes números de hilos.

Dimensión	Nº Hilos	Tiempo ACC(ms)
100 x 100	1	16,999960
100 x 100	2	8,000135
100 x 100	4	4,999876

Tabla 3. Multiplicación de matrices PC

En la tabla 3 se aprecian los resultados tras la ejecución del programa, el incremento del número de hilos reduce el tiempo de ejecución, demostrando el correcto funcionamiento.

4.5.2 Compilación en el ordenador y ejecución en la tarjeta de desarrollo

Como se explicó en apartados anteriores, en el directorio "openmp-examples" existe el fichero "gemm" el cual realiza la multiplicación de matrices mediante el mismo algoritmo utilizado en esta prueba, esto facilita la adaptación mediante directivas y funciones de la API OpenMP para la división de trabajo entre el host ARM y el acelerador RISC-V en este programa, por lo tanto, para su compilación y ejecución se sigue el mismo procedimiento que en el programa que realiza la suma de vectores.

Dimensión	Nº Hilos	Tiempo ACC(ms)
100 x 100	1	219,457797
100 x 100	2	219,329937
100 x 100	4	219,209481

Tabla 4. Multiplicación de matrices errónea ZCU102

En la tabla 4 se puede apreciar el resultado de la ejecución del programa que realiza la multiplicación de matrices, como se observa no existe una reducción de tiempo, lo cual demuestra la existencia de algún error en el proceso de adaptación del programa, puesto que el número de operaciones realizadas tanto en el programa que realiza la suma de vectores como en este son del mismo orden, se deberían apreciar tiempos de ejecución similares.

A partir de los resultados obtenidos se decide realizar varias pruebas en el programa que realiza la multiplicación de matrices para intentar averiguar dónde se encuentra el fallo.

En primer lugar, se decide modificar la función que realiza la multiplicación de matrices en el programa con el objetivo de medir el tiempo que tarda el sistema en realizar el arranque y la transferencia de datos entre el procesador host ARM y el acelerador RISC-V, sin la carga computacional de la multiplicación de matrices. Para ello, se eliminan los bucles de multiplicación de matrices y se deja únicamente una operación simple sobre el primer elemento de las matrices, manteniendo las directivas de OpenMP.

Dimensión	Nº Hilos	Tiempo ACC(ms)
100 x 100	1	219,449043
100 x 100	2	219,317198
100 x 100	4	219,189167

Tabla 5. Movimiento de datos ZCU102

Como se puede observar realizando la comparación de las tabla 4 y la tabla 5, el programa no está ejecutando la operación que realiza la multiplicación de matrices, puesto que los tiempos obtenidos son similares, solo está realizando el arranque y movimiento de datos.

En segundo lugar, se decide añadir otra función que realice la multiplicación de matrices en el host ARM para comparar los resultados con los obtenidos en el acelerador RISC-V, para ello se realiza una copia de la función y se eliminan las directivas OpenMP. También se deciden modificar los datos introducidos en las matrices a multiplicar para que sean números primos, este método evita obtener resultados repetidos en la matriz resultante y posibilita una mejor detección de errores. Posteriormente, se añade una función que tras la ejecución total de la multiplicación de matrices en ambos procesadores realice una comparación de los resultados y muestre un aviso si encuentra discrepancias, así se automatiza el proceso de comparación.

Tras la ejecución de la prueba anterior se observa que los resultados obtenidos realizando la multiplicación de matrices en ambos procesadores son totalmente diferentes, mientras el host ARM realiza una multiplicación que arroja resultados correctos el acelerador RISC-V proporciona resultados erróneos, en este momento se decide realizar una tercera prueba modificando el programa para que trate las matrices como vectores.

Dimensión	Nº Hilos	Tiempo ACC(ms)
100 x 100	1	4838,402033
100 x 100	2	2539,769888
100 x 100	4	1394,073009

Tabla 6. Multiplicación de matrices correcta ZCU102

En la Tabla 6 se observa la ejecución de la prueba, se aprecia que el tiempo de ejecución en el acelerador disminuye proporcionalmente a medida que aumenta el número de hilos, lo que indica un aprovechamiento adecuado del paralelismo. Además, los resultados obtenidos en ambos procesadores son correctos, lo que sugiere que el problema inicial ha sido resuelto. Esto podría indicar que el error estaba relacionado con la versión del compilador utilizada, ya que podría no estar interpretando correctamente las matrices.

4.6 Modificación del número de clusters y cores

En este apartado se detallan los procedimientos y configuraciones necesarias para modificar la cantidad de clústeres y núcleos en la plataforma HERO, con el objetivo de ajustar la configuración del hardware para mejorar el rendimiento de los programas ejecutados.

4.6.1 Modificación de ficheros HERO

A continuación, tras realizar una búsqueda intensiva en los ficheros que componen la plataforma HERO, dentro de la carpeta "Hardware" se encuentran ficheros que definen y configuran varios parámetros de interés para la modificación.

```
ivan.vitores@gdem18:/data/local/hero/hero/hardware/fpga/src$ ls
pulp.py  pulp.py.orig  pulp.template_v
```

Figura 31. Ruta pulp.py

En la figura 31 se muestra la ruta para poder acceder al fichero "pulp.py", el cual es un script de Python que genera configuraciones. A continuación, se muestra el contenido del script:

```
#!/usr/bin/env python3

from mako.template import Template
import math
import re

def clog2(x):
    return math.ceil(math.log2(x))

n_clusters = 2
pulp_template = Template(filename='pulp.template_v')
string = pulp_template.render(
    target='xilzu9eg', n_clusters=n_clusters,
    aw=64, dw=128, iw=3+clog2(n_clusters+1), uw=4,
    aw_pl2ps=49, iw_pl2ps=5, uw_pl2ps=1,
    aw_ps2pl=40, iw_ps2pl=17, uw_ps2pl=16,
    aw_lite=32, dw_lite=32,
)
string = re.sub(r'\s+$', '', string, flags=re.M)
print(string)
pulp.py (END)
```

Figura 32. Fichero pulp.py

En la figura 32 se muestra el contenido del script denominado "pulp.py", donde se define un valor para la variable "n_clusters". Para realizar esta prueba, se modifica el valor por defecto establecido en HERO, que originalmente configuraba un único clúster con ocho núcleos, y se ajusta para que la nueva configuración tenga dos clústers, cada uno con ocho núcleos.

```
ivan.vitores@gdem18:/data/local/hero/hero/hardware/src$ ls
apb              hero_axi_mailbox.sv  pulp_cluster_cfg_pkg.sv  pulp_ooc.sv  soc_bus.sv        soc_peripherals.sv
dmac_wrap_ooc.sv  l2_mem.sv           pulp_cluster_ooc.sv      pulp.sv      soc_ctrl_regs.sv
```

Figura 33. Ruta pulp_cluster_cfg_pkg.sv

En la figura 33 se muestra la ruta de acceso al archivo "Pulp_cluster_cfg_pkg.sv", un fichero de configuración escrito en SystemVerilog. A continuación, se muestra su contenido:

```
// Copyright 2019 ETH Zurich and University of Bologna.
// Copyright and related rights are licensed under the Solderpad Hardware
// License, Version 0.51 (the "License"); you may not use this file except in
// compliance with the License.  You may obtain a copy of the License at
// http://solderpad.org/licenses/SHL-0.51. Unless required by applicable law
// or agreed to in writing, software, hardware and materials distributed under
// this License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
// CONDITIONS OF ANY KIND, either express or implied. See the License for the
// specific language governing permissions and limitations under the License.

// Configuration package for PULP cluster OOC stub
package automatic pulp_cluster_cfg_pkg;
  // -- Decoupling of cluster clock domain
  localparam bit          ASYNC = 1'b0;
  localparam int unsigned DC_BUF_W = 8;
  // -- Cores
  localparam int unsigned N_CORES = 8; // must be a power of 2 and <= 8
  // -- AXI
  localparam int unsigned AXI_AW = 64; // [bit]
  localparam int unsigned AXI_DW = 64; // [bit]
  localparam int unsigned AXI_IW_MST = 6; // [bit]; do not change, seems to break instruction cache
  localparam int unsigned AXI_IW_SLV = 4; // [bit]
  localparam int unsigned AXI_UW = 4; // [bit]
  // -- DMA
  localparam int unsigned DMA_STREAMS = 1;
  localparam int unsigned DMA_MAX_BURST_SIZE = 2048; // [B], must be a power of 2
  // Maximum number of beats in a DMA burst on the SoC bus
  localparam int unsigned DMA_MAX_BURST_LEN = DMA_MAX_BURST_SIZE / (AXI_DW/8);
  // Maximum number of transactions the DMA can have in flight
  localparam int unsigned DMA_MAX_N_TXNS = N_CORES;
  localparam int unsigned N_DMAS = 4; // larger values seem to break the cluster
  // -- Instruction Cache
  localparam int unsigned ICACHE_SIZE = 4096; // [B], must be a power of 2
  // -- TCDM
  localparam int unsigned N_TCDM_BANKS = 2*N_CORES; // must be a power of 2
  localparam int unsigned TCDM_SIZE = 128*1024; // [B], must be a power of 2
  // -- L2 Memory (not inside cluster)
  localparam int unsigned L2_SIZE = 128*1024; // [B], must be a power of 2

  typedef logic      [AXI_AW-1:0] addr_t;
  typedef logic            [5:0] cluster_id_t;
  typedef logic      [AXI_DW-1:0] data_t;
  typedef logic    [DC_BUF_W-1:0] dc_buf_t;
  typedef logic  [AXI_IW_MST-1:0] id_mst_t;
  typedef logic  [AXI_IW_SLV-1:0] id_slv_t;
  typedef logic    [AXI_DW/8-1:0] strb_t;
  typedef logic      [AXI_UW-1:0] user_t;
endpackage
```

Figura 34. Fichero pulp_cluster_cfg_pkg.sv

En la figura 34 se puede observar el fichero "Pulp_cluster_cfg_pkg.sv", en el cual se realizan configuración de parámetros que afectan al número de núcleos, la interfaz de comunicación, la memoria caché y el control de DMA, lo cual es esencial para el funcionamiento del sistema. En él se puede observar la variable "N_CORES", la cual está establecida en el valor máximo, como se observa en la imagen, por lo tanto, no interesa la modificación de su valor ya que se busca una reducción en el tiempo de ejecución de los programas.

Después de la modificación de los ficheros se deben repetir los todos pasos establecidos en este proyecto, desde el capítulo "4.1.3 Configuración de variables y prototipado de aceleradores en Vivado" hasta el capítulo "4.3.2 Ejecución de programas en la tarjeta de desarrollo", a excepción de los capítulos sobre la configuración de la tarjeta de desarrollo.

4.6.2 Comprobación de los resultados después de la modificación

Tras completar los pasos necesarios para reconfigurar el hardware de la plataforma HERO, generar y copiar las imágenes en la tarjeta SD, así como configurar la conexión NFS, se procede a ejecutar el programa "helloworld.c". El objetivo de esta prueba es verificar si las modificaciones realizadas en los ficheros han sido efectivas y han generado cambios significativos en la ejecución del programa.

```
root@zcu102:/data/gdem18/hero/hero/openmp-examples/helloworld# ./helloworld
[11440.039997] PULP: Device opened.
Starting program execution.
>>> PRINTING BUFFER OF CORE 0:
Hello World, I am t[11440.263752] PULP: Device released.
hread 0 of 0
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 1:
Hello World, I am thread 0 of 1
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 2:
Hello World, I am thread 0 of 2
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 3:
Hello World, I am thread 0 of 3
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 4:
Hello World, I am thread 0 of 4
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 5:
Hello World, I am thread 0 of 5
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 6:
Hello World, I am thread 0 of 6
<<< END OF BUFFER
>>> PRINTING BUFFER OF CORE 7:
Hello World, I am thread 0 of 7
<<< END OF BUFFER
Done offloading, cycles to execute kernel: 12595!
```

Figura 35. Ejecución helloworld con dos clusters

Como se puede observar en la Figura 35, en comparación con la Figura 29 presentada anteriormente en este proyecto, la modificación de los ficheros no ha tenido el efecto esperado, ya que no se aprecia ningún cambio en la ejecución del programa "helloworld.c". En este punto, se esperaba la ejecución de dieciséis hilos que imprimieran el mensaje, dado que la nueva configuración del proyecto cuenta con dos clusters de ocho núcleos cada uno.

Este resultado sugiere que aún existen otros ficheros, como los drivers, que requieren modificaciones para alcanzar el comportamiento deseado. Sin embargo, considerando la gran inversión de tiempo realizada hasta el momento para la realización de este proyecto, se propone que la adaptación del proyecto HERO para soportar diferentes configuraciones sea abordada en trabajos futuros.

5. Resultados

En este capítulo se presentan los resultados obtenidos, incluyendo los tiempos de ejecución medidos tanto en el acelerador RISC-V como en el host ARM, utilizando los programas diseñados en capítulos anteriores. Se realiza un análisis detallado de estos resultados y se extraen conclusiones que permiten caracterizar el sistema.

5.1 Script para la ejecución de pruebas

Los scripts mostrados a continuación permiten automatizar la compilación, ejecución y recolección de datos de los programas. El Script 10 genera diferentes versiones del ejecutable, variando el tamaño de los datos y el número de hilos, para cada combinación, compila el código y guarda el ejecutable con un nombre específico. Luego, el Script 11 ejecuta cada versión del programa veinte veces, capturando valores y almacenándolos en un archivo CSV para su análisis. Finalmente, el Script 12 automatiza la ejecución de todas las versiones del programa generadas en el Script 10, llamando al Script 11 para recolectar los datos.

```
#!/bin/bash

for f in 1 2 4 8 ;
do
    for s in 8 64 128 ;
    do
        echo "#define TAM ${s}" > constantes.h
        echo "#define NUM_THREADS ${f}" >> constantes.h
        make clean all && cp multmatrices multmatrices_${s}_t${f}
  done
done
```

Script 10. Generación de ejecutables

```
#!/bin/bash

ejecutable=$1
archivoSalida=/data/gdem18/hero/hero/openmp-examples/multmatrices/multmatrices.csv

for i in {1..20}
do
 echo "Ejecutando la iteracion $i..."
 salida=$(./$ejecutable 2>&1 | grep -E "CSV|kernel")
 scsv=$(echo ${salida}|sed -E -e 's/.*CSV;([0-9]+;[0-9]+;[0-9]+)\.([0-9]+).*/\1,\2/')
 sciclos=$(echo ${salida} | sed -E -e 's/.*cycles to execute kernel:([0-9]+).*/\1/')
 echo "${scsv};${sciclos}" >> $archivo_salida
done
```

Script 11. Ejecución y adquisición de datos

```
#!/bin/bash

for f in 1 2 4 8 ;
do
  for s in 8 64 128 ;
  do
        ./ejecucion.sh multmatrices_${s}_t${f}
  done
done
```

Script 12. Especificación del ejecutable

5.2 Resultados obtenidos con datos de tipo entero

En este apartado se presentan los resultados obtenidos tras la ejecución de los programas diseñados, utilizando valores de entrada de tipo "int", números enteros con signo de 32 bits. Las siguientes tablas muestran los valores mínimos obtenidos, ya que el procesador host ARM con sistema operativo Linux ejecuta varias tareas simultáneamente que están fuera de nuestro control, esto genera variaciones en los tiempos medidos, seleccionar el menor tiempo permite obtener un resultado más preciso.

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
64	1	222,189897	0,743151	14284
64	2	221,326943	0,633001	9018
64	4	220,855835	0,457048	6440
64	8	220,524127	0,460693	4776
4096	1	235,449076	0,803111	680035
4096	2	229,157925	0,684023	341715
4096	4	225,615978	0,501871	173000
4096	8	223,516941	0,50294	88166
16384	1	283,874989	0,887347	2740837
16384	2	257,227898	0,760794	1372478
16384	4	243,342161	0,656128	688317
16384	8	237,24103	0,670239	347075

Tabla 7. Ejecución Suma de vectores con datos enteros

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
8x8	1	220,669985	0	1050
8x8	2	220,40592	0	1050
8x8	4	220,242977	0	1050
8x8	8	220,14008	0	1050
64x64	1	223,675966	0	1051
64x64	2	223,004032	0	1051
64x64	4	222,511014	0	1051
64x64	8	222,172022	0	1051
128x128	1	229,635954	0	1052
128x128	2	229,106052	0	1052
128x128	4	228,742928	0	1052
128x128	8	228,540918	0	1052

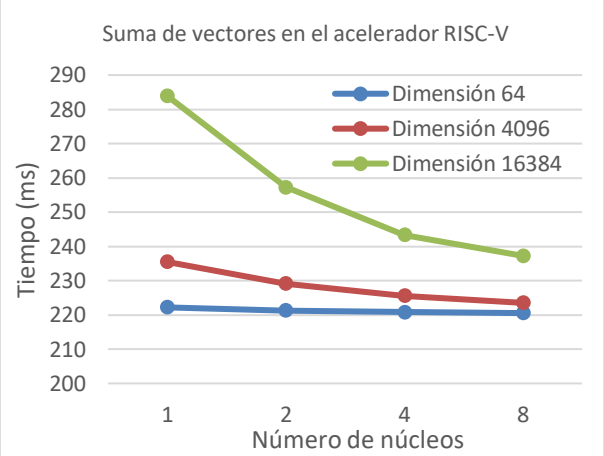
Tabla 8. Ejecución Arranque y movimiento de datos con datos enteros

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
8x8	1	224,341154	0,734782	181416
8x8	2	222,316902	0,614166	96025
8x8	4	221,276045	0,438929	52355
8x8	8	220,640898	0,443407	25870
64x64	1	1425,439119	3,16596	60133667
64x64	2	826,050043	1,964092	30186207
64x64	4	525,880098	1,442909	15100412
64x64	8	374,224901	1,446514	7580340
128x128	1	9873,673916	37,459135	482181334
128x128	2	5059,497118	19,402027	241490040
128x128	4	2669,167995	10,984898	121883040
128x128	8	1496,637106	10,991234	63343592

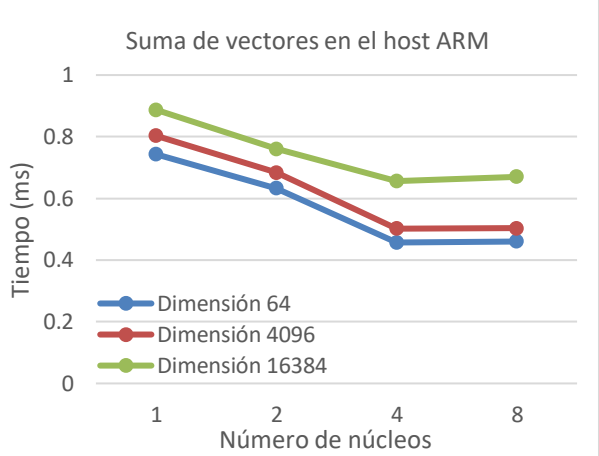
Tabla 9. Ejecución multiplicación de matrices con datos enteros

A continuación, se presentan los resultados obtenidos en la operación de suma de vectores tanto en el acelerador RISC-V como en el host ARM. La Tabla 7 muestra los tiempos de ejecución para diferentes tamaños de vectores y distintos números de hilos.

En las Gráficas 1 y 2 se visualiza la evolución del tiempo de ejecución a medida que aumenta el número de núcleos utilizados. Se observa que, en general, el tiempo de ejecución disminuye con un mayor paralelismo, aunque el comportamiento varía según la arquitectura y el tamaño del vector.



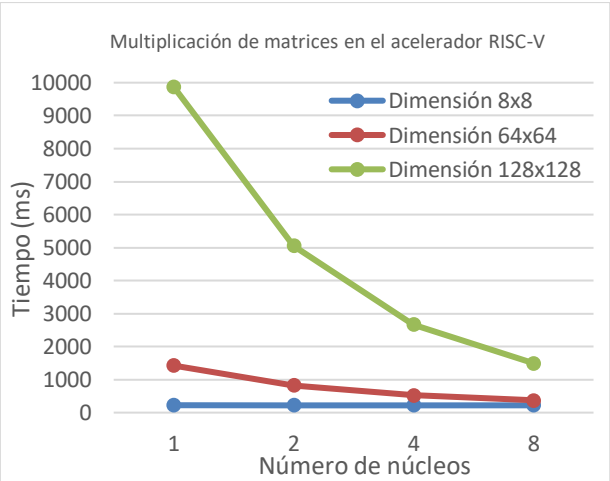
Gráfica 1. Suma vectores RISC-V con enteros



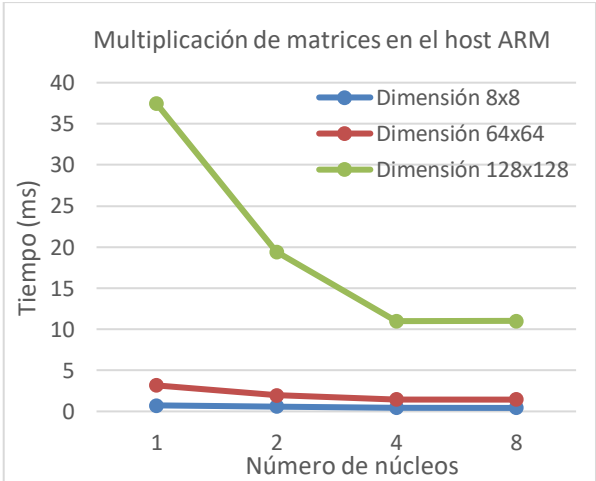
Gráfica 2. Suma vectores ARM con enteros

Ahora, se presentan los resultados obtenidos para la operación de multiplicación de matrices en el acelerador RISC-V y en el host ARM. La Tabla 9 muestra los tiempos de ejecución para distintas dimensiones de matrices y diferentes números de hilos.

En las Gráficas 3 y 4 se visualiza cómo varía el tiempo de ejecución al incrementar el número de núcleos utilizados. Se observa que el tiempo de ejecución disminuye con el aumento de núcleos, especialmente en matrices de mayor tamaño, aunque con diferencias significativas entre ambas arquitecturas.



Gráfica 3. Multiplicación matrices RISC-V con enteros



Gráfica 4. Multiplicación matrices ARM con enteros

A continuación, mediante las siguientes fórmulas, se calcula el tiempo y el número de ciclos kernel requeridos para realizar la operación de multiplicación de matrices en el acelerador, descontando aquellos debidos al arranque y al movimiento de datos. Los resultados obtenidos se pueden observar en la tabla 10.

$$T.operación\ ACC(ms) = T.multiplicación(ms) - T.movimiento(ms)$$

$$Ciclos\ Kernel\ operación = Ciclos\ kernel\ multiplicación - Ciclos\ kernel\ movimiento$$

Dimensión	N.º Hilos	T. operación ACC(ms)	T. operación Host(ms)	Ciclos kernel operación
8x8	1	3,671169	0,734782	180366
8x8	2	1,910982	0,614166	94975
8x8	4	1,033068	0,438929	51305
8x8	8	0,500818	0,443407	24820
64x64	1	1201,763153	3,16596	60132616
64x64	2	603,046011	1,964092	30185156
64x64	4	303,369084	1,442909	15099361
64x64	8	152,052879	1,446514	7579289
128x128	1	9644,037962	37,459135	482180282
128x128	2	4830,391066	19,402027	241488988
128x128	4	2440,425067	10,984898	121881988
128x128	8	1268,096188	10,991234	63342540

Tabla 10. Tiempo solo operación multiplicación de matrices con datos enteros

El siguiente cálculo permite observar el período (20 ns) y la frecuencia (50 MHz) de la FPGA. Además, nos ayuda a comprender el mensaje de ciclos kernel que arroja la plataforma HERO, mediante la terminal, al realizar una operación. Los ciclos kernel representan la cantidad de ciclos de reloj que tarda el procesador en ejecutar la parte del código correspondiente al cálculo principal, sin tener en cuenta el tiempo de arranque y el movimiento de datos. Los resultados obtenidos se pueden observar en la tabla 11.

$$Periodo\ FPGA = \frac{T.operación\ ACC(ms)}{Ciclos\ Kernel\ operación} \rightarrow Frecuencia\ FPGA = \frac{1}{Periodo\ FPGA}$$

Dimensión	N.º Hilos	T. operación ACC(ms)	Ciclos kernel operación	Periodo FPGA(ns)	Frec. FPGA(MHz)
8x8	1	3,671169	180366	20,35399687	49,1303996
8x8	2	1,910982	94975	20,12089497	49,6995785
8x8	4	1,033068	51305	20,13581522	49,6627521
8x8	8	0,500818	24820	20,17800161	49,5589216
64x64	1	1201,763153	60132616	19,98521323	50,0369943
64x64	2	603,046011	30185156	19,97823072	50,0544825
64x64	4	303,369084	15099361	20,09151805	49,7722471
64x64	8	152,052879	7579289	20,06162834	49,8464024
128x128	1	9644,037962	482180282	20,0008966	49,9977586
128x128	2	4830,391066	241488988	20,0025314	49,9936723
128x128	4	2440,425067	121881988	20,02285249	49,942934
128x128	8	1268,096188	63342540	20,01966116	49,9508954

Tabla 11. Cálculo periodo y frecuencia FPGA

Seguidamente, se presenta el cálculo de la cantidad de operaciones necesarias en la FPGA para realizar la multiplicación de una matriz, considerando cada una de las dimensiones elegidas en las pruebas realizadas.

$$Nº\ operaciones\ multiplicar\ matriz = N^3 multiplicaciones + \big((N-1)\cdot N^2\big) sumas$$

$$Matriz\ de\ dimensión\ 8x8 \rightarrow\ 8^3 + ((8-1)\cdot 8^2 = 960$$

$$Matriz\ de\ dimensión\ 64x64 \rightarrow\ 64^3 + ((64-1)\cdot 64^2 = 520192$$

$$Matriz\ de\ dimensión\ 128x128 \rightarrow\ 128^3 + ((128-1)\cdot 128^2 = 4194304$$

A continuación, se presentan los cálculos de operaciones por segundo realizadas. La tabla 12 muestra los resultados del acelerador RISC-V, mientras que la tabla 13 presenta los resultados del host ARM. Las diferencias significativas en los resultados son debidas a que el host opera a una frecuencia de 1,2 GHz, mientras que el acelerador trabaja a una frecuencia de 50 MHz.

$$Operaciones\ por\ segundo = \frac{Nº\ operaciones\ multiplicar\ matriz}{T.\ operación\ (s)}$$

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.
8x8	1	0,00367117	960	261497
8x8	2	0,00191098	960	502360
8x8	4	0,00103307	960	929271
8x8	8	0,00050082	960	1916864
64x64	1	1,20176315	520192	432857
64x64	2	0,60304601	520192	862607
64x64	4	0,30336908	520192	1714717
64x64	8	0,15205288	520192	3421126
128x128	1	9,64403796	4194304	434912
128x128	2	4,83039107	4194304	868316
128x128	4	2,44042507	4194304	1718678
128x128	8	1,26809619	4194304	3307560

Tabla 12. Cálculo operaciones por segundo realizadas en el acelerador con datos enteros

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.
8x8	1	0,00073478	960	1306510
8x8	2	0,00061417	960	1563095
8x8	4	0,00043893	960	2187142
8x8	8	0,00044341	960	2165054
64x64	1	0,00316596	520192	164307824
64x64	2	0,00196409	520192	264851137
64x64	4	0,00144291	520192	360516152
64x64	8	0,00144651	520192	359617674
128x128	1	0,03745914	4194304	111970124
128x128	2	0,01940203	4194304	216178650
128x128	4	0,0109849	4194304	381824574
128x128	8	0,01099123	4194304	381604468

Tabla 13. Cálculo operaciones por segundo realizadas en el host con datos enteros

El siguiente cálculo realiza una comparación del rendimiento del acelerador RISC-V y el procesador host ARM en términos de millones de operaciones por segundo. Para ello, se calcula la relación de frecuencias entre ambos procesadores y se calcula el rendimiento del acelerador a la frecuencia del host.

En la tabla 14 se presentan los resultados de este cálculo para el acelerador RISC-V y en la tabla 15 se muestran los resultados para el procesador ARM, ambos operando a la misma frecuencia de 1,2 GHz. Al comparar los resultados, se observa que con un menor número de operaciones el acelerador produce mejores resultados, pero cuando el número de operaciones aumenta el host obtiene resultados bastante superiores, la diferencia en los resultados se debe a la eficiencia de la arquitectura ARM y cómo cada procesador maneja las operaciones y el paralelismo.

$$Relación\ de\ frecuencias = \frac{Frecuencia\ host\ ARM}{Frecuencia\ acelerador\ RISC-V} = \frac{1.2\ GHz}{50MHz} = 24$$

$$Operaciones\ por\ segundo\ con\ frec.host = \ operaciones\ por\ segundo\ \cdot Rel.de\ frecuencias$$

$$Millones\ de\ op.por\ segundo\ con\ frec.host = \frac{Operaciones\ por\ segundo\ con\ frec.host}{10^6}$$

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.	Millones op. por seg. Con la frec. del host
8x8	1	0,00367117	960	261497	6,27593009
8x8	2	0,00191098	960	502360	12,0566285
8x8	4	0,00103307	960	929271	22,3025009
8x8	8	0,00050082	960	1916864	46,0047363
64x64	1	1,20176315	520192	432857	10,3885761
64x64	2	0,60304601	520192	862607	20,7025795
64x64	4	0,30336908	520192	1714717	41,1531981
64x64	8	0,15205288	520192	3421126	82,1070149
128x128	1	9,64403796	4194304	434912	10,3971055
128x128	2	4,83039107	4194304	868316	20,7581702
128x128	4	2,44042507	4194304	1718678	41,0871374
128x128	8	1,26809619	4194304	3307560	79,071352

Tabla 14. Cálculo millones de op. por segundo en el acelerador con frec. Host con datos enteros

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.	Millones op. por seg.
8x8	1	0,00073478	960	1306510	1,30650996
8x8	2	0,00061417	960	1563095	1,56309532
8x8	4	0,00043893	960	2187142	2,18714188
8x8	8	0,00044341	960	2165054	2,16505378
64x64	1	0,00316596	520192	164307824	164,307824
64x64	2	0,00196409	520192	264851137	264,851137
64x64	4	0,00144291	520192	360516152	360,516152
64x64	8	0,00144651	520192	359617674	359,617674
128x128	1	0,03745914	4194304	111970124	111,532741
128x128	2	0,01940203	4194304	216178650	215,334202
128x128	4	0,0109849	4194304	381824574	380,333072
128x128	8	0,01099123	4194304	381604468	380,113825

Tabla 15. Cálculo millones de op. por segundo en el host con datos enteros

Ahora se procede a realizar el cálculo de SpeedUp que se muestra en la fórmula, el cual es una medida utilizada en computación para evaluar el rendimiento de un sistema al aumentar la cantidad de recursos de procesamiento, como el número de hilos o procesadores.

En la tabla 16 se observa el SpeedUp del acelerador, hay una mejora significativa en el rendimiento, con un SpeedUp cercano al ideal. Esto indica que esta arquitectura maneja bien el paralelismo y reduce el tiempo de ejecución de manera eficiente.

En contraste, en la tabla 17 se observa el SpeedUp del host, en este caso no escala de manera tan efectiva. Aunque con 2 y 4 hilos hay una mejora notable, con 8 hilos el rendimiento empeora, esto es debido a que el procesador host ARM cuenta tan solo con cuatro núcleos y se generan problemas en la gestión de hilos, sincronización y acceso a memoria.

$$SpeedUp = \frac{Operaciones\ por\ segundo_{(N\ hilos)}}{Operaciones\ por\ segundo_{(N/2\ hilos)}}$$

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.	SpeedUp
8x8	1	0,00367117	960	261497	-
8x8	2	0,00191098	960	502360	1,92109279
8x8	4	0,00103307	960	929271	1,84981089
8x8	8	0,00050082	960	1916864	2,06276102
64x64	1	1,20176315	520192	432857	-
64x64	2	0,60304601	520192	862607	1,99282211
64x64	4	0,30336908	520192	1714717	1,98783107
64x64	8	0,15205288	520192	3421126	1,99515489
128x128	1	9,64403796	4194304	434912	-
128x128	2	4,83039107	4194304	868316	1,99653263
128x128	4	2,44042507	4194304	1718678	1,9793232
128x128	8	1,26809619	4194304	3307560	1,92447916

Tabla 16. Cálculo SpeedUp en el acelerador con datos enteros

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.	SpeedUp
8x8	1	0,00073478	960	1306510	-
8x8	2	0,00061417	960	1563095	1,19638962
8x8	4	0,00043893	960	2187142	1,39923805
8x8	8	0,00044341	960	2165054	0,98990098
64x64	1	0,00316596	520192	164307824	-
64x64	2	0,00196409	520192	264851137	1,61192042
64x64	4	0,00144291	520192	360516152	1,36120296
64x64	8	0,00144651	520192	359617674	0,9975078
128x128	1	0,03745914	4194304	111970124	-
128x128	2	0,01940203	4194304	216178650	1,93068153
128x128	4	0,0109849	4194304	381824574	1,76624553
128x128	8	0,01099123	4194304	381604468	0,99942354

Tabla 17. Cálculo SpeedUp en el host con datos enteros

En esta ocasión, se presenta una estimación del ancho de banda entre el procesador host ARM y el acelerador RISC-V, definido como la cantidad de datos transferidos por unidad de tiempo entre ambos componentes del sistema.

Para realizar la estimación se emplea un método iterativo que permite lograr la convergencia del ancho de banda, estabilizándose en el valor de 23,43 megabytes por segundo.

$$T(8x8) = Tiempo\ movimiento\ y\ arranque\ matriz\ de\ 8x8\ (8\ hilos)$$

$$T(64x64) = Tiempo\ movimiento\ y\ arranque\ matriz\ de\ 64x64\ (8\ hilos)$$

$$T(64x64) - T(8x8) = Diferencia\ de\ tiempo\ entre\ el\ mov.de\ matrices = 222.17 - 220.14 = 2.03$$

$$D(8x8) = Nº\ de\ Bytes\ en\ la\ matriz\ de\ 8x8 = 8 \cdot 8 \cdot 32/8 = 256$$

$$D(64x64) = Nº\ de\ Bytes\ en\ la\ matriz\ de\ 64x64 = 64 \cdot 64 \cdot 32/8 = 16384$$

$$D(64x64)/D(8x8) = Factor\ de\ escala\ entre\ el\ tamaño\ de\ las\ matrices = 256/16384 = 64$$

$$T\_mov(8x8) = Tiempo\ movimiento = (T(64x64) - T(8x8) - Tmov(8x8)ant.)/(D(64x64)/D(8x8)$$

$$T(128x128) = Tiempo\ movimiento\ y\ arranque\ matriz\ de\ 128x128\ (8\ hilos)$$

$$D(128x128) = Nº\ de\ Bytes\ en\ la\ matriz\ de\ 128x128 = 128 \cdot 128 \cdot 32/8 = 65536$$

$$D(128x128)/D(8x8) = Factor\ de\ escala\ entre\ el\ tamaño\ de\ las\ matrices = 65536/256 = 256$$

$$T\_mov\_CALC(128x128) = Cálculo\ tiempo\ movimiento = (T(128x128) - T(8x8)) + T\_mov(8x8)ant.$$

$$T\_mov\_EST(128x128) = Estimación\ tiempo\ movimiento = (T(128x128) - T(8x8)) + T\_mov(8x8)ant$$

$$T.Error = Error\ entre\ estimación\ y\ cálculo = T\_mov\_CALC(128x128) - T\_mov\_EST(128x128)$$

$$BW = Ancho\ de\ banda = ((3 \cdot (D(8x8)/T\_mov(8x8))) \cdot 1000)/(1024^2)$$

	1ª Iteración	2ª Iteración	3ª Iteración	4ª Iteración	5ª Iteración
T(8x8) (ms)	220,1401	220,1401	220,1401	220,1401	220,1401
T(64x64) (ms)	222,1720	222,1720	222,1720	222,1720	222,1720
T(64x64) - T(8x8) (ms)	2,0319	2,0319	2,0319	2,0319	2,0319
D(8x8) (Bytes)	256	256	256	256	256
D(64x64) (Bytes)	16384	16384	16384	16384	16384
D(64x64) / D(8x8)	64	64	64	64	64
T_mov(8x8) (ms)	0,03174909	0,03125301	0,03126077	0,03126064	0,03126065
T(128x128) (ms)	228,5409	228,5409	228,5409	228,5409	228,5409
D(128x128) (Bytes)	65536	65536	65536	65536	65536
D(128x128) / D(8x8)	256	256	256	256	256
T_mov_CALC(128x128) (ms)	8,4008	8,4326	8,4321	8,4321	8,4321
T_mov_EST(128x128) (ms)	8,1278	8,0008	8,0028	8,0027	8,0027
T. Error (ms)	0,2731	0,4318	0,4293	0,4294	0,4294
BW(MB/s)	23,07	23,44	23,43	23,43	23,43

Tabla 18. Estimación del ancho de banda

5.3 Resultados obtenidos con datos de tipo coma flotante

En este apartado se presentan los resultados obtenidos tras la ejecución iterativa de los programas diseñados, utilizando datos de tipo "float", números en coma flotante de 32 bits. Este tipo de datos se maneja en unidades conocidas como FPU (Floating Point Unit), las cuales se pretende estudiar con estas pruebas. Al igual que en el apartado de los datos enteros, las siguientes tablas muestran los valores mínimos obtenidos.

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
64	1	225,204134	0,752972	14435
64	2	224,845992	0,703584	9131
64	4	224,629988	0,634535	6538
64	8	224,47491	0,63244	5153
4096	1	247,249914	0,79298	708865
4096	2	240,903965	0,684476	380146
4096	4	237,452048	0,502594	180734
4096	8	235,113978	0,510137	91525
16384	1	322,206001	0,873089	3083446
16384	2	297,20605	0,760794	1572695
16384	4	280,310017	0,662088	783886
16384	8	275,112158	0,671124	400194

Tabla 19. Ejecución Suma de vectores con datos en coma flotante

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
8x8	1	220,669985	0	1050
8x8	2	220,40592	0	1050
8x8	4	220,242977	0	1050
8x8	8	220,14008	0	1050
64x64	1	223,675966	0	1051
64x64	2	223,004032	0	1051
64x64	4	222,511014	0	1051
64x64	8	222,172022	0	1051
128x128	1	229,635954	0	1052
128x128	2	229,106052	0	1052
128x128	4	228,742928	0	1052
128x128	8	228,540918	0	1052

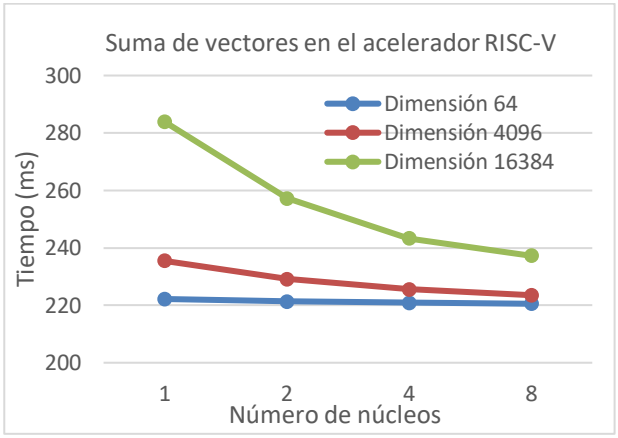
Tabla 20. Ejecución Arranque y movimiento de datos con datos en coma flotante

Dimensión	N.º Hilos	Tiempo ACC(ms)	Tiempo Host(ms)	Ciclos kernel
8x8	1	278,983133	0,982673	2599040
8x8	2	254,870125	0,740051	1599310
8x8	4	243,472561	0,608921	1014361
8x8	8	236,039694	0,61322	720314
64x64	1	1633,811832	4,252195	68680547
64x64	2	954,072677	2,537966	34567007
64x64	4	592,878055	1,721859	16914810
64x64	8	396,339512	1,739378	7886759
128x128	1	13646,76337	46,422005	666289574
128x128	2	6678,732634	24,338007	318655691
128x128	4	3163,332748	13,330936	144459916
128x128	8	1606,669544	13,673841	73371778

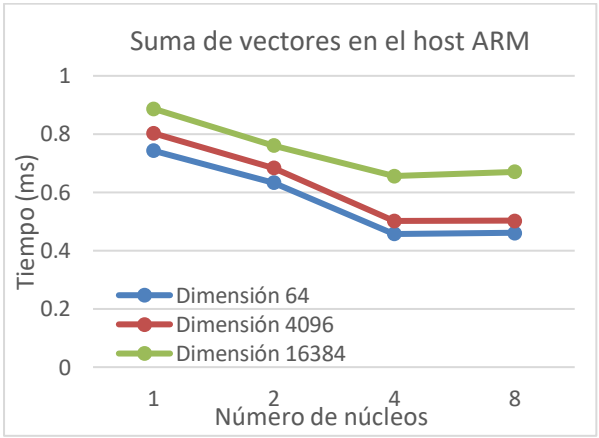
Tabla 21. Ejecución multiplicación de matrices con datos en coma flotante

En este apartado se presentan los resultados obtenidos al ejecutar la operación de suma de vectores utilizando datos en coma flotante. Siguiendo la misma metodología que en el caso de los datos enteros, la Tabla 19 muestra los tiempos de ejecución para diferentes tamaños de vectores y distintos números de hilos, tanto en el acelerador RISC-V como en el host ARM.

En las Gráficas 5 y 6 se visualizan los tiempos de ejecución en función del número de núcleos, se puede apreciar que, en términos generales, a medida que aumenta el paralelismo, el tiempo de ejecución tiende a reducirse, aunque el patrón varía dependiendo de la arquitectura y las dimensiones del vector.



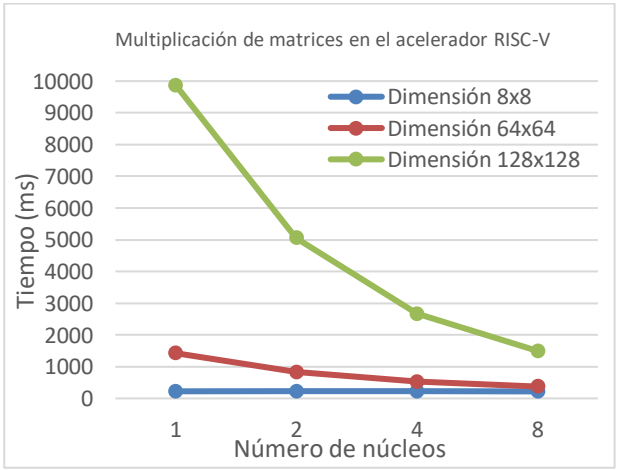
Gráfica 5. Suma vectores RISC-V con coma flotante



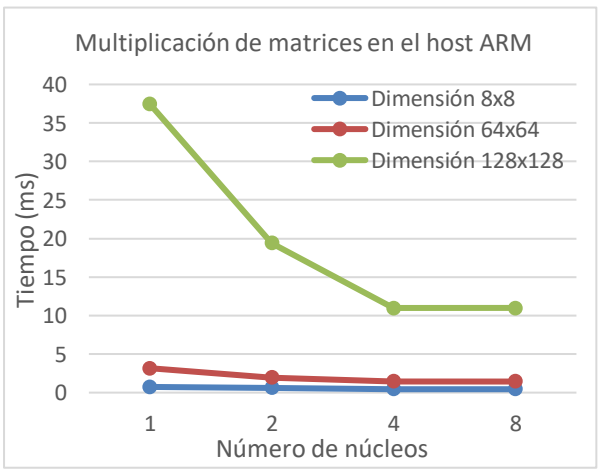
Gráfica 6. Suma vectores ARM con coma flotante

A continuación, se presentan los resultados obtenidos en la operación de multiplicación de matrices tanto en el acelerador RISC-V como en el host ARM. La Tabla 21 resume los tiempos considerando distintas dimensiones de matrices y configuraciones de hilos.

En las Gráficas 7 y 8 se observa cómo varía el tiempo de ejecución al aumentar la cantidad de núcleos utilizados. Los datos indican que el tiempo de ejecución tiende a reducirse conforme se incrementan los núcleos, especialmente en matrices de mayor tamaño, aunque se notan diferencias notables entre las dos arquitecturas.



Gráfica 7. Multiplicación matrices RISC-V con coma flotante



Gráfica 8. Multiplicación matrices ARM con coma flotante

Siguiendo la misma metodología, se calculan el tiempo y los ciclos kernel necesarios para realizar la operación multiplicación de matrices en el acelerador y el host, excluyendo los tiempos de arranque y movimiento de datos. Los resultados se muestran en la tabla 22.

Dimensión	N.º Hilos	T. operación ACC(ms)	T. operación Host(ms)	Ciclos kernel operación
8x8	1	58,313148	0,982673	2597990
8x8	2	34,464205	0,740051	1598260
8x8	4	23,229584	0,608921	1013311
8x8	8	15,899614	0,61322	719264
64x64	1	1410,13587	4,252195	68679496
64x64	2	731,068645	2,537966	34565956
64x64	4	370,367041	1,721859	16913759
64x64	8	174,16749	1,739378	7885708
128x128	1	13417,1274	46,422005	666288522
128x128	2	6449,62658	24,338007	318654639
128x128	4	2934,58982	13,330936	144458864
128x128	8	1378,12863	13,673841	73370726

Tabla 22. Tiempo solo operación multiplicación de matrices con datos en coma flotante

A continuación, se presentan los cálculos de operaciones por segundo. La tabla 23 muestra los resultados del acelerador RISC-V y la tabla 24, los del host ARM. Como se comentó anteriormente las diferencias tan significativas se deben a la distintas frecuencias de trabajo de ambas arquitecturas.

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.
8x8	1	0,05831315	960	16462
8x8	2	0,03446421	960	27854
8x8	4	0,02322958	960	41326
8x8	8	0,01589961	960	60378
64x64	1	1,41013587	520192	368894
64x64	2	0,73106865	520192	711550
64x64	4	0,37036704	520192	1404531
64x64	8	0,17416749	520192	2986734
128x128	1	13,4171274	4194304	311387
128x128	2	6,44962658	4194304	647777
128x128	4	2,93458982	4194304	1423681
128x128	8	1,37812863	4194304	3031589

Tabla 23. Cálculo operaciones por segundo realizadas en el acelerador con datos en coma flotante

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.
8x8	1	0,00098267	960	976927
8x8	2	0,00074005	960	1297207
8x8	4	0,00060892	960	1576559
8x8	8	0,00061322	960	1565506
64x64	1	0,0042522	520192	122334935
64x64	2	0,00253797	520192	204964133
64x64	4	0,00172186	520192	302110684
64x64	8	0,00173938	520192	299067828
128x128	1	0,04642201	4194304	89998697
128x128	2	0,02433801	4194304	171662372
128x128	4	0,01333094	4194304	313400349
128x128	8	0,01367384	4194304	305541069

Tabla 24. Cálculo operaciones por segundo realizadas en el host con datos en coma flotante

Se continúa con el cálculo para comparar el rendimiento del acelerador RISC-V y el procesador host ARM en términos de millones de operaciones por segundo.

En la tabla 25 se presentan los resultados de este cálculo para el acelerador RISC-V y en la tabla 26 se muestran los resultados para el procesador ARM, ambos operando a la misma frecuencia de 1,2 GHz. En esta ocasión al realizar la comparación de los resultados, se observa que utilizando datos en coma flotante el procesador host obtiene mejores resultados independientemente del número de operaciones, lo que vuelve a demostrar la eficiencia de la arquitectura ARM.

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.	Millones op. por seg. Con la frec. del host
8x8	1	0,05831315	960	16462	0,39510815
8x8	2	0,03446421	960	27854	0,6685197
8x8	4	0,02322958	960	41326	0,99183868
8x8	8	0,01589961	960	60378	1,44909178
64x64	1	1,41013587	520192	368894	8,8534788
64x64	2	0,73106865	520192	711550	17,0772035
64x64	4	0,37036704	520192	1404531	33,7087446
64x64	8	0,17416749	520192	2986734	71,681621
128x128	1	13,4171274	4194304	311387	7,47328969
128x128	2	6,44962658	4194304	647777	15,5466489
128x128	4	2,93458982	4194304	1423681	34,1683459
128x128	8	1,37812863	4194304	3031589	0,39510815

Tabla 25. Cálculo millones de op. por seg. en el acelerador con frec. Host con datos en coma flotante

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.	Millones op. por seg.
8x8	1	0,00098267	960	976927	0,97692722
8x8	2	0,00074005	960	1297207	1,2972079
8x8	4	0,00060892	960	1576559	1,57655919
8x8	8	0,00061322	960	1565506	1,56550667
64x64	1	0,0042522	520192	122334935	122,334935
64x64	2	0,00253797	520192	204964133	204,964133
64x64	4	0,00172186	520192	302110684	302,110684
64x64	8	0,00173938	520192	299067828	299,067828
128x128	1	0,04642201	4194304	89998697	89,9986978
128x128	2	0,02433801	4194304	171662372	171,662372
128x128	4	0,01333094	4194304	313400349	313,400349
128x128	8	0,01367384	4194304	305541069	305,541069

Tabla 26. Cálculo millones de op. por seg. en el host con datos en coma flotante

En la tabla 27 se observa el SpeedUp del acelerador, en esta ocasión se puede observar que la mejora es más notable en las dimensiones más grandes, lo que sugiere que para datos en coma flotante el acelerador es más eficiente con cargas de trabajo mayores. Aun así, para valores grandes a medida que aumenta el número de hilos, se observa un incremento en el SpeedUp, lo que indica que el acelerador aprovecha bien la paralelización.

En la tabla 28 se observa el SpeedUp del host, al igual que con valores enteros no escala de manera efectiva. Se sigue apreciando el resultado de tener solo cuatro núcleos puesto que con 8 hilos el rendimiento empeora.

Los resultados sugieren que el acelerador es más adecuado para tareas altamente paralelizables y computacionalmente intensivas.

Dimensión	N.º Hilos	T. operación ACC(s)	N.º Operaciones	Operaciones por seg.	SpeedUp
8x8	1	0,05831315	960	16462	-
8x8	2	0,03446421	960	27854	1,69197595
8x8	4	0,02322958	960	41326	1,48364746
8x8	8	0,01589961	960	60378	1,46100612
64x64	1	1,41013587	520192	368894	-
64x64	2	0,73106865	520192	711550	1,92886865
64x64	4	0,37036704	520192	1404531	1,97390345
64x64	8	0,17416749	520192	2986734	2,02649917
128x128	1	13,4171274	4194304	311387	-
128x128	2	6,44962658	4194304	647777	2,08029545
128x128	4	2,93458982	4194304	1423681	2,09779584
128x128	8	1,37812863	4194304	3031589	2,02940227

Tabla 27. Cálculo SpeedUp en el acelerador con datos en coma flotante

Dimensión	N.º Hilos	T. operación Host(s)	N.º Operaciones	Operaciones por seg.	SpeedUp
8x8	1	0,00098267	960	976927	-
8x8	2	0,00074005	960	1297207	1,32784538
8x8	4	0,00060892	960	1576559	1,21534789
8x8	8	0,00061322	960	1565506	0,9929898
64x64	1	0,0042522	520192	122334935	-
64x64	2	0,00253797	520192	204964133	1,67543419
64x64	4	0,00172186	520192	302110684	1,47396854
64x64	8	0,00173938	520192	299067828	0,98992801
128x128	1	0,04642201	4194304	89998697	-
128x128	2	0,02433801	4194304	171662372	1,90738728
128x128	4	0,01333094	4194304	313400349	1,82567878
128x128	8	0,01367384	4194304	305541069	0,97492256

Tabla 28. Cálculo SpeedUp en el host con datos en coma flotante

6. Presupuesto

Todos los recursos de hardware y software utilizados en el desarrollo de este proyecto serán proporcionados y financiados íntegramente por el Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM). Sin embargo, con el objetivo de proporcionar una estimación aproximada del coste que implicaría desarrollar un proyecto de estas características, se presenta un presupuesto detallado a continuación.

Con el fin de obtener una mejor organización se han desarrollado dos tablas que muestran el coste de **recursos materiales y recursos humanos**, estas tablas incluyen los costes asociados al personal necesario y a los elementos tecnológicos requeridos para la implementación del proyecto.

Concepto	Unidades	Precio/unidad	Total
Ordenador personal	1	1200€	1200€
Tarjeta de desarrollo Xilinx ZCU102	1	3539,51€	3539,51€
Adaptador tarjeta SD a USB	1	8,60€	8,60€
Tarjeta SD 64 GB	1	16,99€	16,99€
Cable ethernet RJ45	1	3,50€	3,50€
Cable USB a Micro USB	2	5,10€	10,20€
Licencia Vivado	1	5427,83	5427,83
Total recursos materiales			**10206,63€**

Tabla 29. Presupuesto recursos materiales

Concepto	Horas	Precio/hora	Total
Mano de obra Ingeniero Junior	360	16,5€	5940€
Total recursos humanos			**5940€**

Tabla 30. Presupuesto recursos humanos

Como se observa en las tablas anteriores, el coste total asociado al desarrollo del proyecto asciende a **16146,63 €**. En caso de que el proyecto fuese destinado a su comercialización, podría añadirse un beneficio comercial de entorno al 15%, lo que incrementaría el coste total.

7. Impacto del proyecto

A continuación, se hace un estudio sobre las implicaciones sociales, de salud y seguridad, ambientales, económicas, tecnológicas o industriales [8] que están relacionadas con este proyecto:

Implicaciones sociales:

Este proyecto puede tener un impacto social positivo al basarse en una plataforma de código abierto y una arquitectura abierta basada en el conjunto de instrucciones reducido RISC-V. Estas características permiten que la tecnología sea accesible para una amplia comunidad de desarrolladores y países con recursos limitados, eliminando la dependencia de licencias propietarias costosas. Además, al reducir los costos de implementación y fomentar la innovación colaborativa, HERO puede facilitar el desarrollo de soluciones tecnológicas que respondan a necesidades locales, contribuyendo a reducir la brecha digital y promoviendo la inclusión tecnológica en regiones menos desarrolladas.

Implicaciones de salud y seguridad:

Este proyecto basado en la plataforma HERO tiene el potencial de ser clave en el desarrollo de aplicaciones de salud avanzadas, como el monitoreo remoto y el procesamiento eficiente de datos biomédicos. Un ejemplo, es el trabajo desarrollado en el Centro de Investigación en Tecnologías Software y Sistemas Multimedia para la Sostenibilidad (CITSEM), donde se emplean tecnologías de imagen hiperespectral para la detección de cáncer durante procedimientos quirúrgicos. Este proyecto podría aprovechar las capacidades de procesamiento rápido y eficiente para analizar imágenes hiperespectrales en tiempo real, permitiendo la localización precisa de tumores de forma no invasiva y mejorando significativamente la precisión y eficacia de los procedimientos médicos.

Implicaciones ambientales:

HERO está basado en la arquitectura RISC-V y esto utiliza un conjunto de instrucciones reducido por lo que permite un ahorro energético significativo en comparación con otras arquitecturas más complejas. Además, el uso de lógica programable mediante FPGA reduce la necesidad de múltiples microprocesadores para realizar tareas equivalentes, disminuyendo el consumo de recursos y el impacto ambiental asociado a la fabricación de hardware. Por otro lado, los test realizados en la plataforma HERO para identificar configuraciones óptimas contribuyen a maximizar el rendimiento, optimizando el consumo energético y garantizando un uso más eficiente de los recursos tecnológicos.

Implicaciones económicas:

Este proyecto basado en la plataforma HERO tiene importantes implicaciones económicas al ofrecer una plataforma modular y eficiente en costos. Como se explicó en el punto anterior al utilizar lógica programable mediante FPGA se reduce la necesidad de múltiples procesadores para la realización de la misma tarea, lo que disminuye tanto el consumo de recursos como los costos hardware. Un aspecto clave de HERO es su arquitectura abierta basada en RISC-V, que elimina la dependencia de licencias propietarias costosas. Esto no solo reduce barreras económicas para su adopción, sino que permite a empresas y desarrolladores acceder libremente al diseño de hardware y software sin restricciones impuestas por modelos comerciales cerrados. Las pruebas realizadas para identificar configuraciones óptimas contribuyen aún más a la optimización de recursos, maximizando la eficiencia y minimizando los costos en el proceso de desarrollo y producción.

Implicaciones tecnológicas e industriales:

HERO es una plataforma de investigación avanzada con un gran potencial para generar un impacto significativo en diversas áreas tecnológicas e industriales. Su capacidad para integrar arquitecturas heterogéneas basadas en FPGA con procesadores RISC-V permite avanzar en el diseño de sistemas embebidos, permitiendo el desarrollo de soluciones innovadoras que aprovechen al máximo el procesamiento paralelo de datos. Además, al permitir un procesamiento rápido y eficiente de datos podría tener implicaciones en campos como el de la salud y tecnologías emergentes como la IA, todo esto podría acelerar la digitalización de procesos industriales, mejorando la eficiencia y reduciendo los tiempos de desarrollo e innovación lo que producirá avances significativos en distintas áreas tecnológicas e industriales.

Ahora se procede a exponer la posible aportación a los ODS (Objetivos de Desarrollo Sostenible)[9] con la implementación de este proyecto:

ODS 4 – Educación de calidad:

Al ser una plataforma de código abierto, HERO facilita el acceso al conocimiento y garantiza una educación inclusiva, equitativa y de calidad, lo que puede ser utilizado por instituciones educativas, investigadores y estudiantes para aprender y avanzar en áreas relacionadas con hardware, software y diseño de sistemas.

ODS 7 – Energía asequible y no contaminante:

Debido al uso de software basado en instrucciones reducidas (RISC-V) y al estudio para conseguir la configuración más eficiente entre procesadores y clusters, producimos una reducción de recursos hardware en la FPGA siendo más eficiente y reduciendo el consumo energético, de esta forma se contribuye a reducir el impacto ambiental.

ODS 9 – Industria, innovación e infraestructura:

Mediante el uso de HERO y el testeo de una configuración eficiente este proyecto profundiza en el desarrollo de sistemas computacionales avanzados, optimiza la infraestructura tecnológica mediante prototipos más eficientes y todo ello contribuye directamente al objetivo de mejorar la industrialización realizando una modernización tecnológica en múltiples sectores.

ODS 12 – Producción y Consumo Responsable:

Al prototipar la plataforma HERO en la FPGA del chip XCZU9EG que contiene la tarjeta de desarrollo ZCU102, se podría sustituir por el uso de múltiples microprocesadores específicos, esto disminuiría la necesidad de aumentar la producción de microprocesadores y generaría un consumo responsable de recursos naturales, contribuyendo a un modelo más sostenible de producción y consumo tecnológico.

ODS 17 – Alianzas para lograr los objetivos:

El proyecto HERO está diseñado para fomentar la colaboración en investigación y desarrollo a través de plataformas compartidas como GitHub, involucrando a múltiples usuarios interesados y comunidades académicas.

8. Conclusiones

En este capítulo se presentan las conclusiones del proyecto, destacando los logros, hallazgos, limitaciones y se evalúa el impacto de la plataforma HERO. Además, se proponen futuras líneas de investigación centradas en mejorar la caracterización del sistema, optimizar la comunicación entre el host y el acelerador.

8.1 Conclusiones

A lo largo de este proyecto, se ha llevado a cabo tanto el prototipado de la plataforma HERO como el diseño e implementación de un banco de pruebas que ha permitido caracterizar la comunicación y el rendimiento del sistema multiprocesador heterogéneo basado en las arquitecturas ARM y RISC-V. Tras la ejecución de las pruebas se han obtenido las conclusiones que a continuación se exponen:

- En primer lugar, las pruebas con datos en coma flotante fueron diseñadas para analizar la eficiencia de la unidad de punto flotante, la cual, como es normal, muestra un aumento en el tiempo de ejecución en ambos procesadores. No obstante, los resultados indican que esta unidad es más eficiente en el procesador ARM, ya que la diferencia de tiempos entre las operaciones con enteros y con coma flotante es menos pronunciada, en comparación con el acelerador RISC-V, cuyo rendimiento en la unidad de punto flotante no logra escalar de manera óptima en ciertas configuraciones.

- En segundo lugar, la comparativa realizada en operaciones por segundo muestra, independientemente del tipo de dato, que el procesador ARM tiene una clara ventaja debido a la gran diferencia en su frecuencia de trabajo. Sin embargo, al observar la comparativa realizada al igualar la frecuencia del acelerador RISC-V con la del host ARM, este último sigue manteniendo la ventaja debido a su arquitectura.

- En tercer lugar, se observa que el rendimiento mejora significativamente con el aumento del paralelismo, como se muestra en el análisis del SpeedUp. Sin embargo, estas mejoras son más notorias en el acelerador RISC-V cuando se manejan grandes cargas de trabajo. No obstante, se debe tener en cuenta la limitación del host ARM, que cuenta con cuatro núcleos Cortex-A53 y no soporta 'Multithreading', lo cual le impide aumentar a ocho hilos.

- En cuarto lugar, tras calcular el ancho de banda, se observa que la comunicación entre el procesador host y el acelerador es un factor crítico en el rendimiento global del sistema. Se ha determinado un ancho de banda de aproximadamente 23,43 MB/s, lo cual podría generar latencia o cuello de botella debido a la imposibilidad de aprovechar correctamente la velocidad total de ejecución del sistema, dado que ambos procesadores operan a una frecuencia mayor.

8.2 Trabajos futuros

Una posible línea de trabajo futuro sería el desarrollo y finalización del apartado propuesto en este proyecto “4.6 Modificación del número de clústeres y núcleos”, que no pudo completarse debido a limitaciones de tiempo y carga de trabajo.

Este apartado, centrado en la modificación del número de clústeres y núcleos en la plataforma HERO, abriría la puerta a una caracterización más detallada del sistema, permitiendo el estudio de parámetros clave bajo configuraciones adaptadas a distintos tipos de carga de trabajo. De esta manera, sería posible optimizar el rendimiento de los programas ejecutados, ya que se podrían explorar diferentes combinaciones de recursos para encontrar la configuración más eficiente en función de las necesidades específicas de cada tarea.

Además, una correcta implementación de esta modificación facilitaría la exploración de arquitecturas heterogéneas optimizadas. Obtener la configuración óptima entre clústeres y núcleos ayudaría a reducir latencias en la ejecución de tareas computacionales intensivas, mejorando la eficiencia del sistema y garantizando un uso más efectivo de los recursos disponibles.

9. Referencias

[1] AMD Inc., "ZCU102 Evaluation Board User Guide," UG1182 (v1.7), 21-Feb-2023. [En línea]: https://docs.amd.com/v/u/en-US/ug1182-zcu102-eval-bd.

[2] Xilinx, "Zynq UltraScale+ MPSoc Base Targeted Reference Design User Guide," UG1221 (v2020). [En línea]: https://docs.amd.com/v/u/en-US/ug1221-zcu102-base-trd.

[3] ETH Zurich, "Project Information,", About PULP Platform. [En linea]: https://www.pulp-platform.org/projectinfo.html

[4] PULP Platform, "HERO: Open Heterogeneous Research Platform," PULP Platform Website. [En linea] : https://pulp-platform.org/hero.html.

[5] PULP Platform, "HERO: Heterogeneous Embedded Research Platform," README.md file, HERO Repository, GitHub. [En linea]: https://github.com/pulp-platform/hero

[6] ARM, "Instruction Set Architecture (ISA)," ARM Glossary. [En linea]: https://www.arm.com/glossary/isa

[7] P. Lobo, “Issues booting HERO in ZCU102,” PULP Community Forum. [En línea]: https://pulp-platform.org/community/showthread.php?tid=311

[8] Fernández Aller, Celia; Miñano, Rafael. "Guía para trabajar la Responsabilidad Social y Ambiental (GRSA)". ETSI Sistemas Informáticos, Universidad Politécnica Madrid. [En línea]: https://oa.upm.es/35542/1/Guia_Responsabilidad_Social_y_Ambiental-V2-1.pdf.

[9] Naciones Unidas, "Objetivos de Desarrollo Sostenible," Naciones Unidas. [En línea]: https://un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/.

10. Bibliografía

[10] S.Sharma, “Understanding RISC-V: The Open Standard Instruction Set Architecture,” Wevolver Website, [En línea]. https://wevolver.com/article/risc-v-instruction-set.

[11] S.Sharma, “RISC-V Architecture: A comprehensive Guide to open-Source ISA” Wevolver Website, [En línea]. https://www.wevolver.com/article/risc-v-architecture.

[12] S.Sharma, “RISC-C vs ARM: A Comprehensive Comparison of Processor Architectures” Wevolver Website, [En línea]. https://www.wevolver.com/article/risc-v-vs-arm.

[13] D. Patterson y A. Waterman,”Guía Práctica de RISC-V: El atlas de una Arquitectura Abierta”, 1ª ed. [En línea]. https://riscvbook.com/spanish/.

[14] The Linux Foundation, "Introduction to RISC-V (LFD110)," The Linux Foundation Training, Online Course. https://training.linuxfoundation.org/training/introduction-to-riscv-lfd110/.

[15] J. Izquierdo Juárez, "Prototipado en FPGA de sistemas multiprocesador heterogéneos ARM / RISC-V," PFG (CITSEM), ETSIST Universidad Politécnica de Madrid, España, jul. 2023. https://oa.upm.es/77388/1/PFG_IZQUIERDO_JUAREZ_JOSE.pdf.

[16] R. Rendek, "Ubuntu 20.04 Guide," LinuxConfig.org, 11 de abril de 2020. [En línea]: https://linuxconfig.org/ubuntu-20-04-guide.

[17] Canonical, "The Linux command line for beginners," Ubuntu Tutorials. [En línea]: https://ubuntu.com/tutorials/command-line-for-beginners.

[18] T. Mattson y L. Meadows, "A 'Hands-on' Introduction to OpenMP," Intel Corporation, 2025. [En línea]: https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf.

[19] OpenMP, “OpenMP Application Programming Interface Examples” Versión 4.5. [En línea]: https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf.

[20] Kenny Yip Coding, "How to setup VS Code for C/C++ programming in Windows” *YouTube*, [En línea]: https://www.youtube.com/watch?v=Smh9tzol7QI

[21] LLVM Project, "LLVM: Home," *LLVM*, [En linea]: https://llvm.org

Anexo

En este apartado se van a incluir los códigos fuente o las partes más significativas que permitan explicar la importancia de las directivas proporcionadas por la API openMP. Estas directivas son una parte fundamental de la paralelización de procesos a lo largo de este proyecto, ya que facilitan la división de tareas y la ejecución en paralelo, lo que resulta esencial para mejorar el rendimiento, optimizar el manejo de recursos y contribuir significativamente a la eficiencia del sistema.

A.1 helloworld.c

```
#include <hero-target.h>
#include <omp.h>

#pragma omp declare target
void helloworld(void)
{
    #pragma omp parallel
    printf("Hello World, I am thread %d of %d\n", omp_get_thread_num(), omp_get_num_threads());
}
#pragma omp end declare target

int main(int argc, char *argv[])
{
    #pragma omp target device(BIGPULP_MEMCPY)
    helloworld();

  return 0;
}
```

- **#include <hero-target.h>**: Esta línea define una biblioteca con funciones que permiten interactuar con la FPGA y lógica programable del proyecto HERO.
- **#include <omp.h>:** Esta línea define la biblioteca para el uso de funciones y directivas openMP.
- **#pragma omp declare target** y **#pragma omp end declare target:** Estas dos líneas son directivas de la API openMP, definen una sección de código que será ejecutado en un dispositivo distinto, en este caso los cluster y núcleos de la FPGA.
- **#pragma omp parallel:** Esta directiva de openMP indica al compilador que el código de debajo será ejecutado en paralelo por múltiples hilos.
- **omp_get_thread_num()**: Esta función de openMP devuelve el ID del hilo.
- **omp_get_num_threads():** Esta función de openMP devuelve el número total de hilos.
- **#pragma omp target device(BIGPULP_MEMCPY):** Esta directiva de openMP indica que el código de debajo debe ejecutarse en el dispositivo “BIGPULP_MEMCPY”, definido dentro de la biblioteca “hero-target.h”.

A.2 sumaVectores.c

```
#include <hero-target.h>
#include <omp.h>

void genVectores(int vector1[], int vector2[])
{
   #pragma omp parallel for
   for(int i=0; i<LONG_VECTOR; i++)
   {
        …
   }
}
```

- **#pragma omp parallel for:** Esta directiva de openMP se utiliza para paralelizar bucles, divide las iteraciones del bucle entre los hilos disponibles.

```
void sumVectores(int vector1[], int vector2[], int vector3[])
{
 #pragma omp target data map(from: vector3[0:LONG_VECTOR]) map(to: vector2[0:LONG_VECTOR], vector1[0:LONG_VECTOR])
 {
    #pragma omp target device(BIGPULP_MEMCPY)
    {
        #pragma omp parallel for num_threads(NUM_THREADS)
        for(int i=0; i<LONG_VECTOR; i++)
        {
           vector3[i]=vector1[i]+vector2[i];
        }
    }
 }
}
```

- **#pragma omp target data map(from: vector3[0:LONG_VECTOR]) map(to: vector2[0:LONG_VECTOR], vector1[0:LONG_VECTOR]):** Esta directiva de openMP gestiona la transferencia de datos entre el host ARM y el acelerador de FPGA.
- **#pragma omp target device(BIGPULP_MEMCPY):** Esta directiva de OpenMP se usa para indicar que un bloque de código debe ejecutarse en el dispositivo BIGPULP_MEMCPY.
- **#pragma omp parallel for num_threads(NUM_THREADS):** Esta directiva de OpenMP paraleliza la ejecución de un bucle permitiendo que las iteraciones se dividan entre un número específico de hilos.

A.3 multMatrices.c

```
#include <omp.h>
#include <hero-target.h>

void genMatrizA(int fil, int col, int *matriz)
{
   #pragma omp parallel for
   for(int i = 0; i < fil; i++) {
      for(int j = 0; j < col; j++){
         …
      }
   }
}
```

```
Void multMatrices (const int fil, const int col, const int* matrizA, const int* matrizB, int* matrizR)
{
 #pragma omp target data map(from: matrizR[0:fil*col]) map(to: matrizA[0:fil*col], matrizB[0:fil*col])
    #pragma omp target device(BIGPULP_MEMCPY)
    {
       #pragma omp parallel for num_threads(NUM_THREADS)
       for(int i = 0; i < fil; i++){
           for(int j = 0; j < col; j++) {
                  for(int k = 0; k < col; k++) {
                     matrizR[i*columnas + j] += matrizA[i*columnas + k] * matrizB[k*columnas + j];
                  }
           }
       }
    }
}
```

```
int main()
{
   omp_set_default_device(BIGPULP_MEMCPY);
   …
}
```

- **omp_set_default_device(BIGPULP_MEMCPY):** Esta función de openMP establece el dispositivo por defecto en el que se ejecutarán las regiones #pragma.

A.4 movYarranque.c

```
Void multMatrices (const int fil, const int col, const int* matrizA, const int* matrizB, int* matrizR)
{
 #pragma omp target data map(from: matrizR[0:fil*col]) map(to: matrizA[0:fil*col], matrizB[0:fil*col])
    #pragma omp target device(BIGPULP_MEMCPY)
    {
      matrizR[0]=matrizA[0]+matrizB[0];
    }
}
```