Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA
(Mención en Ingeniería de Software)

Integración de HIP/ROCm en un modelo de programación paralela heterogénea

---

Alumno: Javier Pericacho Ávila

Tutor: Arturo González-Escribano

Para mi familia, por estar siempre a mi lado.

Agradecimientos

A mi familia, por su apoyo incondicional para que consiga todas las metas que me propongo.

A mis amigos, por apoyarme y sacarme una sonrisa hasta en los peores momentos.

A mis compañeros y compañeras de clase, por compartir y hacer más divertido este camino.

A mi tutor Arturo, por su orientación y apoyo para que este proyecto salga adelante.

Resumen

Actualmente, las nuevas plataformas computacionales son cada vez más heterogéneas y paralelas, formadas a partir de diferentes tipos de dispositivos que se combinan e interconectan para formar sistemas cada vez más grandes y complejos. Esta tendencia la podemos observar en la lista TOP500, que recoge los 500 supercomputadores más potentes del mundo.

Por tanto, es muy relevante proveer a los programadores de herramientas de programación que puedan aprovechar unidades de cómputo altamente especializadas en sistemas heterogéneos, con flexibilidad para adaptarse al nivel de escala de la aplicación y la plataforma. La integración de estos dispositivos en un entorno de cómputo paralelo plantea desafíos complejos en términos de portabilidad y rendimiento. Diferentes propuestas de modelos de programación y compilación para sistemas heterogéneos se están formalizando y llevando a la práctica. Por ejemplo, Kokkos, HPX, Raja o el estándar SYCL con sus implementaciones prácticas. Sin embargo, estas soluciones tienen a día de hoy carencias para conseguir implementaciones eficientes de programas que utilicen simultáneamente diferentes tipos de dispositivos.

Controllers es un modelo de programación heterogéneo desarrollado por el Grupo de Investigación Trasgo del Departamento de Informática de la Universidad de Valladolid que permite la portabilidad entre diferentes dispositivos (FPGAs, GPUs, CPUs y aceleradores Xeon Phi). El sistema actual da soporte a las tarjetas gráficas AMD mediante OpenCL (Open Computing Language), API de propósito general, diseñada para poder ser utilizada en una variedad de dispositivos, lo que implica un nivel de abstracción añadido que podría impedir explotar el potencial específico de los dispositivos.

Este proyecto propone realizar un mantenimiento preventivo y perfectivo del sistema *Controllers* para dar soporte a las tarjetas gráficas AMD mediante la integración de la tecnología HIP/ROCm. ROCm está mantenido actualmente por AMD, por lo que se prevé que esté optimizada para sus dispositivos, ofrezca un mejor rendimiento y una mayor compatibilidad. Adenás se propone la realización de un estudio experimental para medir su eficacia.

Abstract

Nowadays, new computing platforms are becoming increasingly heterogeneous and parallel, made up of different types of devices that combine and interconnect to form ever larger and more complex systems. This trend can be seen in the TOP500 list of the 500 most powerful supercomputers in the world.

Therefore, it is very relevant to provide programmers with programming tools that can leverage highly specialised computing units in heterogeneous systems, with the flexibility to adapt to the scale level of the application and platform. The integration of these devices in a parallel computing environment presents complex challenges in terms of portability and performance. Different proposals for programming and compilation models for heterogeneous systems are being formalised and implemented. For example, Kokkos, HPX, Raja or the SYCL standard with its practical implementations. However, these solutions still have shortcomings in achieving efficient implementations of programs that simultaneously use different types of devices.

Controllers is a heterogeneous programming model developed by the Trasgo Research Group of the Computer Science Department of the University of Valladolid that allows portability between different devices (FPGAs, GPUs, CPUs and Xeon Phi accelerators). The current system supports AMD graphics cards through OpenCL (Open Computing Language), a general purpose API, designed to be used in a variety of devices, which implies an added level of abstraction that could prevent exploiting the specific potential of the devices.

This project proposes to perform preventive and perfective maintenance of the *Controllers* system to support AMD graphics cards by integrating HIP/ROCm technology. ROCm is currently maintained by AMD and is therefore expected to be optimised for their devices, offering improved performance and better compatibility. In addition, an experimental study is proposed to measure its efficiency.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación

Las nuevas plataformas computacionales son cada vez más heterogéneas y paralelas, formadas a partir de diferentes tipos de dispositivos que se combinan e interconectan para formar sistemas cada vez más grandes y complejos. Esta tendencia se puede observar en sistemas de cálculo intensivo, Big Data o inteligencia artificial, es clave para el objetivo de la computación Exascale y se relaciona también con el reto de minimizar el consumo por unidad de cómputo. Esta misma tendencia la podemos ver en la lista de los 500 supercomputadores más potentes del mundo, la lista TOP500 [1]. En ella podemos apreciar como los sistemas de computación más potentes en la actualidad presentan una gran heterogeneidad, con una variedad de plataformas y arquitecturas.

Por tanto, es muy relevante proveer a los programadores de herramientas de programación que puedan aprovechar unidades de cómputo altamente especializadas en sistemas heterogéneos, con flexibilidad para adaptarse al nivel de escala de la aplicación y la plataforma. Las diferentes unidades computacionales de un sistema paralelo heterogéneo (procesadores de diferentes tipos o familias, unidades de procesamiento gráfico o GPUs con núcleos especializados, aceleradores de hardware programable como las FPGAs, etc.) típicamente tienen diferentes requisitos y limitaciones. El uso combinado de dispositivos de diferentes tipos en un único cómputo paralelo plantea problemas complejos de portabilidad tanto funcional como de rendimiento.

Diferentes propuestas de modelos de programación y compilación para sistemas heterogéneos se están formalizando y llevando a la práctica. Por ejemplo, Kokkos, HPX, Raja, el estándar SYCL con sus implementaciones prácticas como Codeplay's ComputeCpp, Intel's oneAPI, triSYCL o hipSYCL [5]. Sin embargo, estas soluciones tienen a día de hoy carencias para conseguir implementaciones eficientes de programas que utilicen simultáneamente diferentes tipos de dispositivos (FPGAs, GPUs, CPU, etc.). Estos problemas surgen en la eficiencia de la gestión de tareas y sistema de ejecución, en los mecanismos de compilación y enlazado cruzado, en la distribución transparente de datos de forma equilibrada, o en los mecanismos para el movimiento eficiente y solapado de datos entre dispositivos.

El grupo de investigación Trasgo del Departamento de Informática de la Universidad de Valladolid [2] ha desarrollado un modelo de programación para sistemas heterogéneos de alto rendimiento denominado *Controllers* [3, 4, 5]. Este modelo implementa técnicas eficientes de ejecución y comunicación con diferentes dispositivos, entre ellos GPUs, con solapamiento automático de ta-

reas y trasferencias de datos. Se implementa en una biblioteca de funciones en lenguaje C/C++. La biblioteca implementa diferentes "backends" o mecanismos para, a partir del programa de alto nivel del usuario, generar código, compilarlo y enlazarlo con tecnologías específicas para el manejo de diferentes dispositivos CPU o aceleradores (GPUs o FPGAs). En concreto, se proveen backends para manejar GPUs de NVIDIA con CUDA, o GPUs de diversos fabricantes con OpenCL. Sin embargo, OpenCL es una API de propósito general, diseñada para poder ser utilizada en una variedad de dispositivos, lo que implica un nivel de abstracción añadido que podría impedir explotar el potencial específico de los dispositivos. Concretamente, AMD provee de un conjunto de herramientas y mecanismos de programación propios para sus GPUs (HIP/ROCm) similares a CUDA. ROCm está mantenido actualmente por AMD, por lo que se prevé que esté optimizada para sus dispositivos, ofrezca un mejor rendimiento y una mayor compatibilidad.

1.2. Objetivos

En este proyecto se propone llevar a cabo un mantenimiento preventivo y perfectivo del modelo de programación *Controllers* [3, 4, 5], para dar soporte a las tarjetas gráficas AMD con HIP/-ROCm, y verificar su eficiencia y utilidad. Uno de los objetivos complementarios es mantener actualizada la tecnología implicada en el sistema.

Los objetivos específicos de este proyecto son los siguientes:

- Analizar y comprender el sistema de partida *Controllers* [3, 4, 5] y su librería auxiliar *Hitmap* [6].
- Diseñar y desarrollar un backend en HIP/ROCm para las tarjetas gráficas AMD.
- Diseñar, implementar y ejecutar un estudio experimental que permita validar el modelo.
- Analizar y comparar los resultados del estudio experimental con el sistema de referencia y con el backend pre-existente para las tarjetas gráficas AMD, basado en OpenCL.
- Extraer conclusiones de los resultados obtenidos.

1.3. Estructura del trabajo

El resto del trabajo se estructura de la siguiente forma:

Capítulo 2: Conceptos previos. En este capítulo se detallan los conceptos previos, tecnologías implicadas y asociadas del proyecto.

Capítulo 3: Planificación. En este capítulo se realiza una estimación del trabajo a realizar y la organización para completarlo a tiempo. Se realiza un análisis de los posibles riesgos que podrían surgir durante el desarrollo del proyecto, así como el plan de acción para hacerlos frente. Además, se hace una estimación de los costes del proyecto.

Capítulo 4: Seguimiento del proyecto. En este capítulo se detalla como se ha desarrollado el proyecto, indicando los contratiempos que han surgido y comparándolo con la planificación inicial.

Capítulo 5: Ingeniería inversa. En este capítulo se desarrolla el proceso de ingeniería inversa para comprender el sistema *Controllers* y la librería auxiliar *Hitmap*.

Capítulo 6: Diseño de la solución. En este capítulo se presenta el diseño de la solución para implementar el nuevo backend de *Controllers*.

Capítulo 7: Implementación y pruebas. En este capítulo se detalla la implementación del nuevo backend de *Controllers*, los problemas a los que nos hemos enfrentado y las pruebas realizadas.

Capítulo 8: Estudio experimental. En este capítulo se presenta los resultados obtenidos del estudio experimental para validar el modelo propuesto.

Capítulo 9: Conclusiones. En este capítulo se detallan las conclusiones obtenidas tras el desarrollo del proyecto, los objetivos cumplidos, trabajos futuros y una valoración personal.

Bibliografía. Conjunto de fuentes bibliográficas empleadas en el proyecto.

Anexo A: Manual de usuario. En este anexo se adjunta un manual con los pasos a seguir para montar el proyecto y lanzar un programa de *Controllers* en el clúster del Grupo de Investigación Trasgo.

Anexo B: Manual de experimentación. En este anexo se adjunta un manual con los pasos a seguir para realizar un estudio experimental en el clúster del Grupo de Investigación Trasgo.

Anexo C: Tablas experimentación backend HIP. En este anexo se adjuntan las tablas con las métricas calculadas para la batería de pruebas compilando *Controllers* para el backend de HIP.

Anexo D: Tablas experimentación backend Openclamd. En este anexo se adjuntan las tablas con las métricas calculadas para la batería de pruebas compilando *Controllers* para el backend de Hip frente al de OpenclAmd.

Anexo E: Contenidos del CD-ROM. En este anexo se indica la estructura e información del CD entregado.

Capítulo 2

Conceptos previos

2.1. Conceptos previos

2.1.1. Ingeniería inversa

La ingeniería inversa es un proceso de análisis exhaustivo del sistema ya existente para comprender su funcionamiento interno, procesos y arquitectura. Es un proceso de análisis únicamente, en el que no se modifica la funcionalidad del sistema, simplemente se "desmonta" el sistema para comprenderlo. [7]

Algunos de los propósitos de la ingeniería inversa son:

- **Actualización de productos**, uno de los principales objetivos es actualizar productos antiguos para adaptarlos al uso actual. La ingeniería inversa ayuda a los ingenieros a comprender el funcionamiento del producto, lo que les permite utilizar este conocimiento para crear una versión actualizada, que cumpla los nuevos requisitos del usuario.

- **Reparación de productos**, también puede ayudar a los ingenieros a reparar productos que ya están en el mercado, especialmente si se desconoce la causa del problema. Los ingenieros pueden desmontar estos productos para ver si hay alguna forma de solucionar los problemas o mejorar la funcionalidad con una nueva tecnología o un diseño más óptimo.

- **Pruebas de errores**, otra finalidad es comprobar si hay errores o vulnerabilidades en el producto.

En el ámbito de los proyectos software, la ingeniería inversa se emplea en aquellos programas que necesiten mejoras o mantenimiento. Para llevar a cabo el desarrollo del nuevo backend se ha tenido que realizar un análisis exhaustivo de la estructura y funcionamiento del sistema de partida *Controllers*, ya que es nuevo para el alumno. Este análisis se realizó utilizando el enfoque de la ingeniería inversa.

2.1.2. Proyectos de mantenimiento

El mantenimiento del software [8] se refiere a cualquier modificación necesaria después de la entrega, ya sea para corregir errores, mejorar rendimiento o adaptarlo a nuevas necesidades.

Esta actividad es clave porque el ciclo de vida del software no finaliza con la entrega al cliente, ya que una vez se encuentra en operación se pueden encontrar fallos o errores que no se han tenido en cuenta en el desarrollo. También muchos sistemas tienen que ser mejorados o actualizados para mantener al día las necesidades del usuario.

Dentro de los proyectos de mantenimiento existen cuatro tipos de mantenimiento, detallados a continuación.

Tipos de mantenimientos:

- **Mantenimiento correctivo:**

  El objetivo de este tipo de mantenimiento es identificar y corregir los posibles defectos del programa, con el fin de evitar la posibilidad de que provoquen un fallo. Los fallos en un sistema software suelen ser:

  - **Fallos de procesamiento**, producidos cuando el programa genera una salida inesperada o incorrecta.
  - **Fallos de rendimiento**, se refieren a situaciones en las que el sistema tarda demasiado en responder a una solicitud del usuario o en procesar una tarea.
  - **Fallos de programación**, pueden deberse a errores de diseño o implementación.
  - **Fallos de documentación**, causados porque la documentación del sistema no se mantiene actualizada o no refleja con exactitud la funcionalidad real del software.

- **Mantenimiento adaptativo:**

  Este tipo de mantenimiento consiste en llevar a cabo modificaciones para adaptarlo a cambios en el entorno en el que se ejecuta, ya sea por cambios en el software o en el hardware. Estos cambios pueden incluir modificaciones en la arquitectura del sistema, en el sistema operativo o en el entorno de desarrollo del software.

  Los cambios en el entorno de desarrollo pueden ser:

  - En el entorno de datos, por ejemplo, migración de la base de datos.
  - En los procesos del sistema, como por ejemplo migrar a una plataforma con procesos distribuidos para mejorar el rendimiento y la escalabilidad del sistema.

- **Mantenimiento perfectivo:**

  También conocido como mantenimiento evolutivo, es el conjunto de actividades que se realizan para añadir o mejorar funcionalidades del software, con el objetivo satisfacer las necesidades del usuario.

  Este tipo de mantenimiento se divide en dos categorías:

  - **Mantenimiento de eficiencia**, con el objetivo de mejorar el rendimiento del sistema existente.
  - **Mantenimiento de ampliación**, con el objetivo de incorporar nuevas funcionalidades al sistema existente.

- **Mantenimiento preventivo:**

  El propósito de este tipo de mantenimiento es modificar el software para mejorar la calidad o mantenibilidad sin alterar la funcionalidad existente.

Este proyecto se enmarca dentro de un proyecto de mantenimiento perfectivo y preventivo, ya que se va a desarrollar un nuevo backend, con el objetivo de emplear las tecnologías más modernas paras las tarjetas gráficas de AMD y mejorar la mantenibilidad de *Controllers*.

2.2. Tecnologías implicadas

En esta sección se realiza una pequeña explicación de las tecnologías implicadas para realizar la implementación del proyecto.

2.2.1. CUDA

CUDA [9, 10] (Compute Unified Device Architecture) es una plataforma de cálculo paralelo y modelo de programación desarrollado por NVIDIA para sus dispositivos GPU. Proporciona una variedad de funcionalidades y herramientas, como un compilador nativo, interfaces para desarrollar programas en diferentes lenguajes de programación como C++, C, Fortran, etc.

Arquitectura

La arquitectura de CUDA se caracteriza porque un dispositivo ejecuta muchos hilos de forma simultánea, siguiendo el paradigma SIMD (Single Instruction Mltiple Data). Se definen bloques de cores, SM (*Stream multriprocessor*) que están formados por un conjunto de cores o SP (*Streaming Processors*) que comparten una memoria compartida (*shared memory*).

Este diseño permite que la programación sea más sencilla, declarando y ejecutando un kernel, función que se ejecuta de forma paralela en las tarjetas gráficas (*device*), en un conjunto de hilos (*threads*), que se agrupan en bloques (*blocks*) que a su vez se distribuyen en una malla (*grid*), como se muestra en la siguiente imagen: 2.1

Actualmente, *Controllers* cuenta con una implementación nativa en CUDA para dar soporte a las tarjetas gráficas de NVIDIA. Esta implementación se ha utilizado como punto de partida para desarrollar el backend en HIP debido a la gran similitud entre ambas tecnologías.

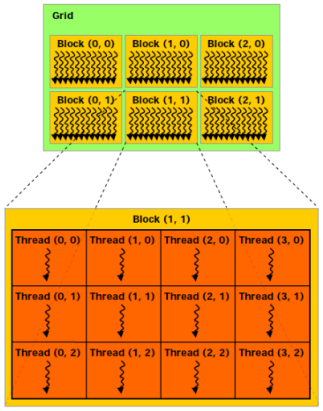

Figura 2.1: Jerarquía de hilos en una aplicación CUDA. Foto extraída de [10]

2.2.2. HIP/ROCm

HIP [11] (Heterogeneous-compute Interface for Portability) es una biblioteca de código abierto para la programación de GPU desarrollada por AMD para la plataforma ROCm. Proporciona una interfaz de programación en C ++ muy similar a la de CUDA de NVIDIA, que permite desarrollar programas directamente en HIP C++ y ejecutarlos en dispositivos NVIDIA o AMD.

La arquitectura de HIP es muy similar a la de CUDA 2.2.1, que permite la declaración y ejecucción de kernels en el device en un conjunto de hilos (*threads*) de forma paralela, agrupándose en bloques (*blocks*) que a su vez se distribuyen en una malla (*grid*) como se muestra en la imagen 2.1

Como HIP está desarrollado por AMD se prevé que aproveche las especificaciones de los dispositivos GPU de AMD para optimizar el rendimiento de los programas.

ROCM [12] (Radeon Open Compute Platform) es una plataforma de código abierto para programación de GPUs desarrollada por AMD. Diseñada para dar soporte y proporcionar herramientas para programación de GPUs y aceleración de aplicaciones científicas en sistemas basados en Linux. ROCm incluye una serie de herramientas de depuración y optimización, así como un kit de desarrollo de software (SDK).

Esta tecnología ha sido empleada para llevar a cabo la implementación del nuevo backend para *Controllers*.

2.2.3. Slurm

Slurm [13] (Simple Linux Utility for Resource Management) es un sistema de gestión de clusters y programación de trabajos de código abierto para supercomputadoras y clústeres Linux. Entre las funcionalidades que proporciona slurm está la planificación y priorización de tareas, gestión de recursos, gestión de usuarios, asignando acceso exclusivo y/o no exclusivo a los recursos, monitorización y seguimiento de los trabajos, entre otras.

Este sistema de colas ha sido empleado para lanzar los programas de prueba a la máquina de experimentación, *manticore*.

2.2.4. CMake

CMake [14] es un software de código abierto multiplataforma utilizado para gestionar el proceso de compilación de forma independiente del compilador.

Una de las funcionalidades más importantes es que permite separar el código fuente del código compilado, es decir, separa los archivos fuente de la aplicación (archivos .cpp, .h, etc.) del código objeto (archivos .o en Linux, .obj en Windows), realizando la compilación en una carpeta aparte normalmente llamada *build*. Esto facilita que el código fuente esté limpio y facilita la administración de proyectos, sobre todo cuando son grandes.

Para la compilación se emplea archivos de configuración simples llamados *CMakeLists.txt*, localizados en cada directorio del código fuente. En estos archivos se especifica la estructura, dependencias y las opciones de compilación del proyecto, y permiten generar archivos de compilación estándar (por ejemplo, projects/workspaces en Windows MSVC y makefiles en Linux).

Este diseño permite soportar jerarquías de directorios complejas y aplicaciones dependientes de varias bibliotecas, como es el caso de *Controllers*, sistema en el que se emplea esta herramienta para la compilación.

2.2.5. Doxygen

Doxygen [15] es una herramienta que permite generar documentación a partir de anotaciones en el código fuente C++, también compatible con otros legaujes como Java, C, Python, etc.

Entre las funcionalidades que proporciona Doxygen está la generación de documentación en línea en diversos formatos, como HTML, PDF, LaTeX, etc., permite mantener una documentación organizada y clara, generar diagramas y también permite realizar un análisis estático del código para detectar errores, facilitando la depuración.

Esta herramienta se ha empleado para complementar la documentación existente de *Controllers* y comprender la estructura del sistema en la fase de ingenería inversa.

2.2.6. Gnuplot

Gnuplot [16] es un programa basado en línea de comandos para Linux, OS/2, MS Windows, OSX, VMS, etc. que permite generar gráficos y diagramas.

Esta herramienta ha sido empleada para generar los gráficos de los resultados obtenidos en la experimentación 8.

2.2.7. OpenMP

OpenMP [17] (Open Multi-Processing) es una interfaz de programación de aplicaciones (API) que permite la programación multiplataforma y multiproceso en memoria compartida en C++, C o Fortran. Permite ejecutar programas paralelos en un conjunto de hilos, especificar como es el reparto de trabajo entre esos hilos, declarar variables, sincronizar hilos, etc.

OpenMP se emplea en *Controllers* para la sincronización de procesos. Además, se han utilizado las directivas que proporciona para medir los tiempos globales y de ejecución de los casos de estudio para las pruebas y la experimentación del proyecto.

2.3. Tecnologías asociadas

En esta sección se realiza una pequeña explicación de las tecnologías que han sido utilizadas como apoyo para la realización del proyecto.

2.3.1. Discord

Discord [18] es una plataforma de comunicación desarrollada por Jason Citron y Stanislav Vishnevskiy en 2015, que permite la comunicación vía texto, audio y vídeo. Permite una gran variedad de funcionalidades, creación de canales de texto y de voz/vídeo, compartición de archivos, compartición de pantalla, entre otras.

Se ha utilizado esta herramienta para la comunicación y compartición de archivos con el tutor.

2.3.2. Git/GitLab

Git [19] es un sistema de control de versiones desarrollado por Linus Torvalds en 2005, de código abierto y gratuito, que permite llevar un seguimiento de las modificaciones realizadas en el código fuente en el transcurso del desarrollo software.

GitLab [20] es un repositorio para la gestión de proyectos basado en Git, que permite la gestión de proyectos, repositorios, integración continua, entre otras funcionalidades a través de una interfaz gráfica.

Para este proyecto se ha empleado el GitLab del grupo de investigación Trasgo.

2.3.3. Astah

Astah [21] es una herramienta de modelado UML desarrollada por Change Vision, empleada para realizar los modelos y diagramas de la ingeniería inversa.

2.3.4. Overleaf

Overleaf [22] editor web que permite la creación y edición de documentos en LaTeX de forma online, herramienta empleada para la realización de la memoria del proyecto.

Capítulo 3

Planificación

3.1. Planificación inicial

Para llevar a cabo el mantenimiento preventivo y perfectivo de sistema, se dividirá el proyecto en dos fases diferenciadas.

- **Ingeniería inversa:** El sistema de partida es *Controllers*, luego será necesario comprender detalladamente el funcionamiento interno e identificar los cambios necesarios para implementar el nuevo backend. A su vez, es necesario una comprensión superficial de *Hitmap*, herramienta auxiliar en este sistema. Ambas herramientas son muy complejas, además de que el número de tecnologías implicadas en el proyecto es elevado, luego un conocimiento profundo en estas tecnologías y del sistema es una tarea costosa. Por ello, esta fase junto con la formación en las herramientas implicadas es una de las tareas críticas y dónde más tiempo se va a emplear.
- **Implementación y pruebas:** Fase de implementación y prueba del nuevo backend que se desea incorporar, esta tarea es delicada, debido a la complejidad conceptual asociada al sistema y las herramientas.

Por otro lado, otro de los objetivos de este proyecto es comparar el rendimiento del nuevo backend con el del actual, para ello se realizará un estudio experimental, que implica la realización de las siguientes tareas:

- Modificación de los scripts proporcionados por el Grupo de Investigación Trasgo.
- Lanzamiento del estudio.
- Cálculo de las métricas y elaboración de las gráficas.
- Análisis de los resultados.
- Extracción de conclusiones

Este estudio no debería de ser una tarea crítica una vez que el backend este implementado y probado, pero hay que tener en cuenta que es un proceso laborioso, en el que un error en el lanzamiento de la batería de pruebas implica reanudar de nuevo el estudio. Además, es importante considerar que se trata del primer acercamiento a este tipo de estudio y que el análisis de los resultados es otro punto que requiere tiempo.

En las siguientes tablas, se muestran las estimaciones de tiempo para las tareas que deben ser realizadas por el estudiante 3.1 y por el tutor 3.1.

Tareas	**Tiempo** (horas)
Formación	20
Ingeniería Inversa	80
Implementación backend	40
Pruebas backend	20
Estudio experimental	60
Reuniones seguimiento	30
Memoria	50

Cuadro 3.1: Estimación de tiempo estudiante.

Tareas	**Tiempo** (horas)
Reuniones seguimiento	30
Revisión memoria	10
Instalación HIP/ROCm	5

Cuadro 3.2: Estimación de tiempo tutor.

En la siguiente figura 3.1 se puede observar un diagrama de Gantt con una organización temporal de las tareas implicadas en el proyecto. Para el seguimiento con el tutor se realizará una reunión semanal, esta frecuencia puede variar dependiendo de la carga de trabajo y las necesidades del proyecto.

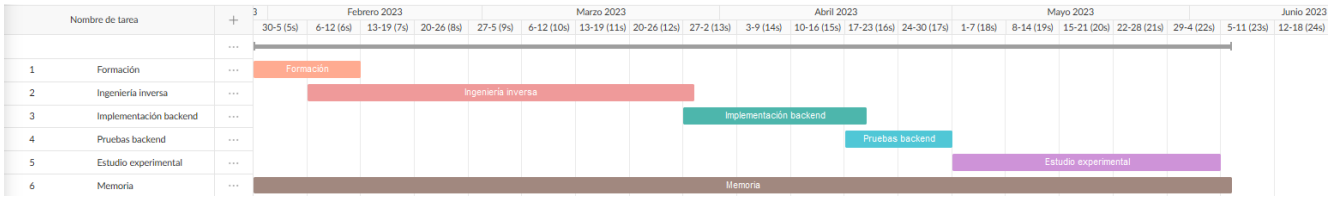

Figura 3.1: Diagrama de Gantt del proyecto

3.2. Planificación de riesgos

En esta sección se presenta una planificación de riesgos, con el objetivo de identificar los posibles riesgos que se pudieran ocasionar en la realización del proyecto y como va a ser el plan de acción para hacerlos frente.

Para llevar a cabo esta planificación se ha llevado a cabo los siguientes pasos, tomando como guía [23].

1. **Identificación de riesgos**, identificar y comprender los posibles riesgos que puedan materializarse en el proyecto.
2. **Análisis y priorización**, evaluar el nivel de prioridad de cada riesgo teniendo en cuenta la probabilidad y e impacto de cada riesgo.
3. **Planificación**, plan de acción frente a estos riesgos, para ello se llevará a cabo planes de mitigación y de contingencia para cada uno.
4. **Monitorización**, supervisión y control durante el desarrollo del proyecto.

Los aspectos a tener en cuenta para cada riesgo son los siguientes:

- **Descripción:** Explicación detallada del riesgo.
- **Probabilidad:** Medida en la que se espera que el riesgo pueda ocurrir. Se clasificará según los siguientes valores: Baja, Media y Alta.
- **Impacto:** Magnitud de los daños que la ocurrencia del riesgo tendría sobre el proyecto. Se clasificará según los siguientes valores: Bajo, Medio y Alto.
- **Plan de mitigación:** Conjunto de medidas preventivas para reducir la probabilidad de que ocurra el riesgo.
- **Plan de contingencia:** Conjunto de medidas correctivas para minimizar el impacto si el riesgo se materializa.

Los riesgos identificados son:

ID	RSK01
Nombre	Incapacidad médica
Descripción	Debido a una enfermedad, el estudiante no puede llevar a cabo las tareas en el periodo planificado
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	1. Evitar actividades y hábitos que pongan en riesgo la salud
Plan de contingencia	1. Aumento del esfuerzo diario 2. Replanificación de las tareas pendientes

Cuadro 3.3: RSK01: Incapacidad médica.

ID	RSK02
Nombre	Cambio en los requisitos
Descripción	Durante la realización del proyecto es necesario cambiar los requisitos para añadir nueva funcionalidad
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	1. Definición inicial detallada de los cambios a realizar en el sistema
Plan de contingencia	1. Aumento del esfuerzo diario 2. Replanificación de las tareas pendientes

Cuadro 3.4: RSK02: Cambio en los requisitos.

ID	RSK03
Nombre	Falta de formación
Descripción	Debido al elevado número de tecnologías implicadas en el proyecto, puede que el estudiante no tenga el conocimiento necesario para trabajar con ellas. Esto puede suponer retrasos en la planificación, errores o un código de mala calidad
Probabilidad	Media
Impacto	Alto
Plan de mitigación	1. Dedicar el tiempo necesario al comienzo del proyecto para formarse en las tecnologías implicadas
Plan de contingencia	1. Aumentar el esfuerzo diario y dedicarlo a la formación 2. Solicitar ayuda al tutor o a un experto

Cuadro 3.5: RSK03: Falta de formación.

ID	RSK04
Nombre	Desconocimiento del sistema
Descripción	El sistema y su arquitectura son nuevas para el estudiante, luego se requiere de una fase de aprendizaje, que si no se realiza correctamente puedo suponer retrasos en las demás tareas
Probabilidad	Media
Impacto	Medio
Plan de mitigación	1. Dedicar el tiempo necesario al comienzo del proyecto para familiarizarse con el sistema y su arquitectura
Plan de contingencia	1. Aumento del esfuerzo diario 2. Replanificación de las tareas pendientes 3. Solicitar ayuda al tutor o a los contribuidores de *Controllers*

Cuadro 3.6: RSK04: Desconocimiento del sistema.

ID	RSK05
Nombre	Fallo del clúster
Descripción	El desarrollo se va a realizar en el clúster del grupo de investigación, un fallo en este, podría implicar la perdida del progreso y retrasar las tareas
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	1. Utilizar herramientas de control de versiones como Git 2. Disponer de un equipo secundario
Plan de contingencia	1. Recuperar una versión anterior del clúster que almacena las copias de seguridad 2. Replanificación de las tareas pendientes 3. Avanzar con otra de las tareas a realizar que no sufra un bloqueo, como por ejemplo la memoria

Cuadro 3.7: RSK05: Fallo del clúster.

ID	RSK06
Nombre	Fallo del equipo personal
Descripción	Un fallo en el equipo podría implicar la pérdida del progreso y el retraso de las tareas debido a la imposibilidad de conectarse al clúster y continuar con la memoria
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	1. Utilización de herramientas online para la realización de la memoria como Overleaf 2. Disponer de un ordenador secundario
Plan de contingencia	1. Reemplazar el equipo

Cuadro 3.8: RSK06: Fallo del equipo personal.

ID	RSK07
Nombre	Fallo de la máquina de pruebas
Descripción	Posibilidad de que la máquina con las tarjetas gráficas AMD para realizar las pruebas, deje de funcionar temporalmente, se estropee o se encuentre ocupada. Esto puede impedir el lanzamiento de las pruebas y del estudio experimental, lo que puede suponer un retraso en las tareas mencionadas
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	1. Disponer otro ordenador secundario con tarjeta gráfica AMD
Plan de contingencia	1. Reemplazar el equipo 2. Solicitar acceso a otra máquina 3. Avanzar con otra de las tareas a realizar que no sufra un bloqueo, como por ejemplo la memoria

Cuadro 3.9: RSK07: Fallo de la máquina de pruebas.

ID	RSK08
Nombre	Fallo en la planificación
Descripción	La planificación inicial no tiene en cuenta algún factor limitante o se produce un retraso en alguna de las tareas
Probabilidad	Media
Impacto	Alto
Plan de mitigación	1. Ser realista con las estimaciones 2. Establecer periodos extra para afrontar situaciones imprevistas
Plan de contingencia	1. Aumento del esfuerzo diario 2. Replanificación de las tareas pendientes 3. Retraso fecha de entrega

Cuadro 3.10: RSK08: Fallo en la planificación.

ID	RSK09
Nombre	Funcionalidad limitada HIP/ROCm
Descripción	Posibilidad de que alguna funcionalidad implementada en el backend de CUDA no se pueda replicar con la tecnología HIP/ROCm.
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	1. Estudio inicial de funcionalidad compatible
Plan de contingencia	1. Buscar alternativas que puedan cubrir dicha funcionalidad 2. Solicitar ayuda al tutor o a un experto

Cuadro 3.11: RSK09: Funcionalidad limitada HIP/ROCm.

ID	RSK10
Nombre	Asignatura suspensa
Descripción	Una asignatura suspensa implicaría el retraso de la fecha de entrega del proyecto
Probabilidad	Baja
Impacto	Alto
Plan de mitigación	1. Constancia en el curso 2. Solicitar ayuda al profesor responsable de la asignatura si se presenta alguna dificultad
Plan de contingencia	1. Retraso fecha de entrega

Cuadro 3.12: RSK10: Asignatura suspensa.

3.3. Planificación de costes

En este apartado se detalla una planificación de costes con el objetivo de identificar y estimar los recursos necesarios para llevar a cabo el proyecto. Los recursos que se han tenido en cuenta son los siguientes:

Coste del personal

- **Sueldo tutor:** Se estima el salario anual de un profesor de Universidad en 43.000 €/año [24]. Si se considera que hay 250 días laborales al año y la jornada laboral es de 8 horas/día, nos da como resultado un sueldo de 21,5€/hora. Conforme a la planificación inicial del proyecto, las horas invertidas del tutor en el proyecto son 45 horas, lo que nos da un coste de 967,5 €.
- **Sueldo ingeniero software:** Se estima el salario anual de un Ingeniero de Software Junior en 21.000 €/año [25]. Si se considera que hay 250 días laborales al año y la jornada laboral es de 8 horas/día, lo que nos da un sueldo de 10,5 €/hora. Conforme a la planificación inicial del proyecto, la duración total del proyecto es de 300 horas, lo que se traduce en un coste de 3.150 €.

  Teniendo en cuenta estos valores, el coste total del personal es de 4.117,5 €, mientras que si se considera solo el sueldo del tutor es de 967,5 €.

Equipamiento

- **Equipo personal:** El ordenador utilizado es un Asus Rog Strix g15, con un coste de 1.400 €, su vida útil estimada es de 5 años, luego su coste asociado es de 0,03 €/hora, como el tiempo de uso de este será de 300 horas, nos da un coste de 9,59 €.
- **Clúster:** El equipo de gestión del clúster Trasgo ha estimado los costes de amortización, mantenimiento, operación y consumo en 0,05 €/CPU-Hour o Core-Hour (medida del tiempo de cálculo, 1 Core-Hour equivale a una hora de ejecución de la CPU [26]). Para el uso de GPUs, el coste se calcula como el tiempo de uso por un número de cores, este se calcula como el número de cores de la máquina dividido por el número de GPUs. Para nuestro caso, la máquina de pruebas es manticore, la cual dispone de 48 cores físicos, así que el uso de una hora de GPU equivale a 12 CPU-hours. Para el desarrollo de este proyecto se han reservado 10 kCPU-hours (10.000 CPU-hours). Así que se estima un coste de uso del clúster de 500 €. (10.000 CPU-Hour x 0,05 €/CPU-Hour)

  Luego, el coste total del equipamiento es de 509,59 €.

Gastos de servicios y tecnologías

- **Electricidad:** El consumo promedio de un portátil oscila entre los 200 y 250 W [27], se tomará como referencia la media de estos valores, es decir, 225 W. Teniendo en cuenta que la duración total del proyecto es de 300 horas, el consumo es de 67,5 kWh y considerando que el coste promedio de la luz en España es de 0,20875 €/kWh [28], nos da un coste de luz de 14,09 €.

No se ha considerado el consumo energético del clúster, ya que su medición y análisis es bastante complejo, debido a que el clúster está compuesto por diferentes dispositivos, cuyos consumos son diferentes y puede variar en función de otros factores como la carga de trabajo o el uso de los recursos del sistema. Además, este coste se ha considerado dentro del precio de 0,05 Eur/CPU-Hour mencionado con anterioridad.

- **Licencias:** La única licencia que ha sido necesaria para la realización de este proyecto es la de Astah, herramienta utilizada para el modelado UML, debido a que la licencia de uso para estudiantes es gratuita, el coste del software es de 0 €.

  Teniendo en cuenta estos valores, el coste total de los servicios y tecnologías es de 14,09 €.

Coste final

El coste total del proyecto se dividirá en un coste simulado, teniendo en cuenta el salario del estudiante y otro coste real, omitiendo este último coste.

Coste	**Coste (€)**
Personal	4.117,5
Equipamiento	509,5
Servicios y tecnologías	14,09
TOTAL	4.641,09

Figura 3.2: Coste simulado

Coste	**Coste (€)**
Personal	967,5
Equipamiento	509,5
Servicios y tecnologías	14,09
TOTAL	1.491,09

Figura 3.3: Coste real

Capítulo 4

Seguimiento del proyecto

Frente a la planificación inicial se ha tenido que realizar varias modificaciones debido a la materialización de algunos riesgos.

Uno de los retrasos ha sido causado por la materialización de los riesgos *RSK03: Falta de formación* 3.2 y *RSK08: Fallo en la planificación* 3.10 que han afectado en la finalización de las tareas de Implementación y Pruebas, los problemas específicos que han causado esta demora se recogen en la sección 7.1. El atraso producido para la fase de Implementación ha sido de aproximadamente una semana respecto a la fecha prevista en la planificación inicial.

Otro los de retrasos que han afectado al desarrollo de la fase de Pruebas, ha sido causado por la materialización del riesgo *RSK07: Fallo de la máquina de pruebas* 3.2, en los días 2, 3 y 4 de mayo, debido a un fallo en la configuración del sistema de colas SLURM del clúster. Ante esta situación se ha llevado a cabo uno de los planes de contingencia contemplados y se ha continuado con el resto de tareas pendientes, en este caso con la realización de la memoria. Esto, junto a lo comentado anteriormente, ha causado que la fase de pruebas finalizase el día 7 de mayo en vez del día 1, fecha de finalización prevista inicialmente.

Asimismo, se ha producido un retraso en el desarrollo del Estudio Experimental, debido a la necesidad de lanzar de nuevo la experimentación para el backend de Openclamd, causada por un fallo en la configuración del script que lanzaba los programas contra las tarjetas gráficas NVIDIA en vez de contra las AMD. Esto, junto a la materialización del riesgo *RSK07: Fallo de la máquina de pruebas* 3.2, desde el día 25 al 31 de mayo, a causa del fallo de una de las tarjetas gráficas AMD implicadas en el estudio, ha supuesto un retraso aproximado de una semana en la finalización de esta tarea, terminando de esta forma el día 7 de junio. Como plan de contingencia ante esta situación se ha proseguido con el desarrollo de la memoria, plan previsto en su planificación inicial.

Por último, se ha producido un retraso en la fecha de finalización de la memoria, debido a la materialización del riesgo *RSK08: Fallo en la planificación* 3.10, para el cual se ha llevado a cabo una replanificación de las tareas pendientes, en este caso solo del desarrollo de la memoria, plan de contingencia recogido en dicho riesgo. Por ello, la finalización de la memoria se ha realizado el día 18 de junio, en vez del 5 de junio, fecha prevista en la planificación inicial.

Luego el diagrama de Gantt con la organización temporal final por semanas de las tareas implicadas en el proyecto se puede observar en la siguiente figura 4.1:

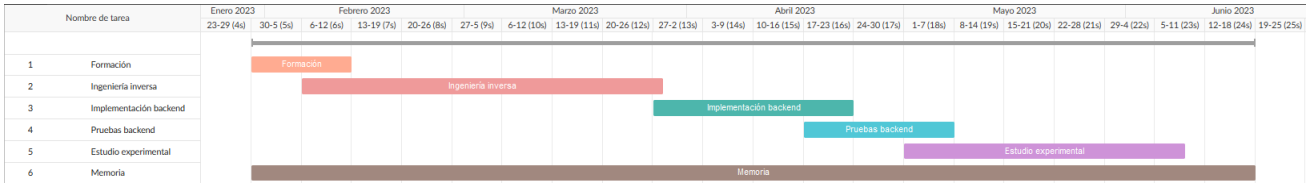

Figura 4.1: Diagrama de Gantt final del proyecto

Capítulo 5

Ingeniería inversa

En este capítulo se detallará el sistema de partida *Controllers* y la librería *Hitmap*, herramienta auxiliar de *Controllers*, tras realizar el proceso de la ingeniería inversa.

5.1. Hitmap

Hitmap [6] es una librería desarrollada por el Grupo de Investigación Trasgo del Departamento de Informática de la Universidad de Valladolid [2]. Está diseñada para programar en entornos distribuidos con abstracciones para ocultar los detalles relacionados con la gestión de datos distribuidos y el manejo de los sistemas de comunicación y sincronización necesarios en esos entornos. Ofrece funcionalidades para el particionado y mapeo automático de datos de forma eficiente en tiempo de ejecución. Proporciona una abstracción mediante una interfaz para el lenguaje de programación C que permite generar códigos independientes de las funciones de mapeo seleccionadas. Admite el particionado jerárquico de arrays con dominios densos, lo que permite la implementación de un paralelismo de datos y de tareas utilizando un modelo SPMD (Single Program, Multiple Data), modelo de programación paralela en el que todos los procesadores ejecutan el mismo programa, pero cada uno procesando datos diferentes.

En el contexto de *Controllers*, *Hitmap* se utiliza fundamentalmente por su gestión abstracta de los arrays.

Hitmap está diseñado con un enfoque orientado a objetos, utilizando internamente técnicas eficientes de MPI para la comunicación, centrándose en el rendimiento y en optimizaciones nativas del compilador. A pesar de que *Hitmap* ha sido diseñado con un enfoque orientado a objetos, la implementación ha sido realizada en C para explotar mejor la experiencia de los autores original en optimizaciones clásicas del compilador C. El desarrollo de una interfaz orientada a objetos en C++ es un esfuerzo sencillo.

La figura 5.1 muestra la Arquitectura de *Hitmap* en un diagrama de clases UML.

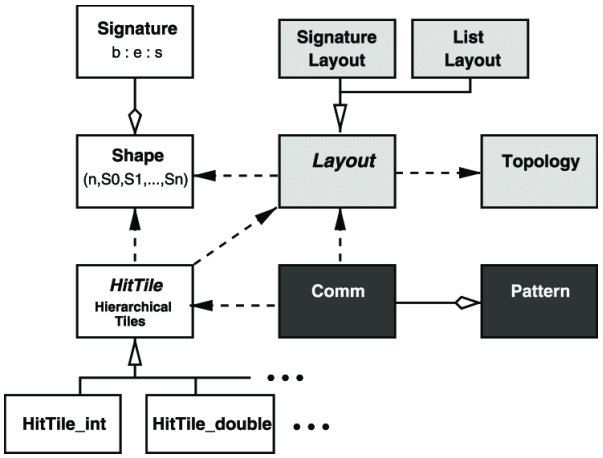

Figura 5.1: Arquitectura de *Hitmap*. Foto extraída de [6]

Conceptos clave:

Signature: Una signature S se define como un tupla de tres elementos que representan un subespacio de índices de una matriz unidimensional. Se parece a la notación clásica de MATLAB o Fortran90 para la selección de índices de una matriz. Su cardinalidad es el número de índices diferentes que existen en su dominio.

$$S \in Signature = (begin : end : stride)$$

$$Card(s \in Signature) = b(s.end - s.begin)/s.stridec$$

Shape: Definimos una Shape h como una n-tupla de Signatures. Representa la selección de un subespacio de índices de un array en un dominio multidimensional. Su cardinalidad es el número de combinaciones diferentes de índices en el dominio.

$$h \in Shape = (S_0, S_1, S_2, ..., S_{n-1})$$

$$Card(h \in Shape) = \prod_{i=0}^{n-1} Card(S_i)$$

Tile: Se define un Tile como una array de n dimensiones. Su dominio está definido por una Shape, y tiene un número de elementos de un tipo dado, dependiendo del lenguaje de programación utilizado.

$$Tile\ _{h \in Shape} : (S_0 \times S_1 \times S_2 \times ... \times S_{n-1}) \rightarrow < type >$$

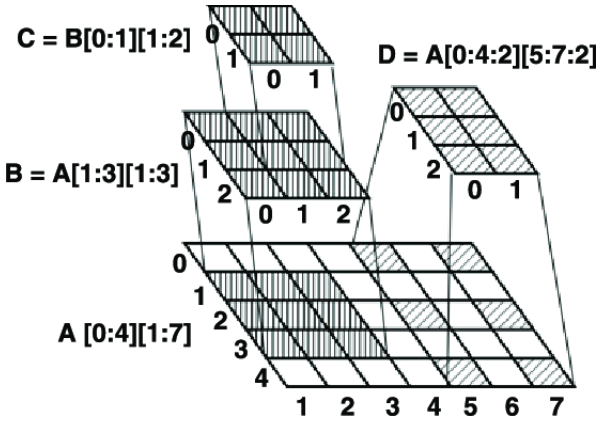

Figura 5.2: Creación de tiles a partir de un array. Foto extraída de [6]

Funciones

La librería *Hitmap* implementa funciones para crear, manipular, mapear y comunicar arrays jerárquicos para cálculos paralelos. La librería tres conjuntos de funcionalidades:

Funciones de tiling: se encargan de definir y manipular tanto arrays como tiles de manera individual. Estas funciones pueden ser utilizadas de manera separada de las demás, permitiendo mejorar la localidad en código secuencial y generar manualmente distribuciones de datos para la ejecución paralela. Los arrays y tiles se implementan con un tipo de dato abstracto: HitTile. Un ejemplo es la Figura 5.2 en la que la notación empleada para especificar dimensiones y rangos se asemeja a las convenciones de Fortran90 y MATLAB.

Esta se utiliza como base para las estructuras de datos en *Controllers*.

Funciones de mapeo: Funciones de distribución y mapeo de datos que particionan automáticamente los dominios de los arrays en tiles, dependiendo de la topología virtual seleccionada.

Funciones de comunicación: Creación de patrones de comunicación reutilizables para tiles jerárquicos distribuidos. Se trata de una abstracción de un modelo de paso de mensajes para comunicar tiles entre procesadores virtuales. Estas funciones pueden ser utilizadas junto con la información de mapeo para crear patrones de comunicación dependientes del mapeo. Como resultado, estas funciones proporcionan un handler (o controlador) que puede ser utilizado para comunicar repetidamente tiles entre procesadores.

5.2. Controllers

Los sistemas heterogéneos se componen de al menos un dispositivo de cómputo especializado o acelerador, al que denominaremos genéricamente "Dispositivo", conectado a la máquina de cómputo principal, a la que denominaremos "Host" en el resto de la memoria.

Controllers [3, 4, 5] es un sistema desarrollado por el Grupo de Investigación Trasgo del Departamento de Informática de la Universidad de Valladolid [2] que emplea como herramienta auxiliar, *Hitmap* [6], librería desarrollada por el mismo grupo.

Controllers es un modelo de programación heterogéneo que permite la portabilidad entre diferentes tipos de dispositivos de cómputo, como GPUs (mediante CUDA para tarjetas gráficas NVIDIA y OpenCL para tarjetas gráficas AMD), grupos de cores de CPU o CPUs (mediante OpenMP), FPGAs (mediante OpenCL) y aceleradores Xeon Phi (mediante la biblioteca Intel COI). Está desarrollada en C99 como una librería, lo que hace que sea compatible con cualquier compilador C/C++ y compatible con otras librerías y modelos de programación paralela. La biblioteca implementa diferentes "backends" o mecanismos para, a partir del programa de alto nivel del usuario, generar código, compilarlo y enlazarlo con tecnologías específicas para el manejo de diferentes dispositivos CPU o aceleradores (GPUs o FPGAs). Proporciona una entidad u objeto *Controller* que permite al usuario coordinar las actividades de ejecución de los bloques de código que se ejecutan en un dispositivo de cómputo específico, denominados kernels, y la gestión de la memoria en un acelerador o un conjunto de núcleos de una CPU.

Los arrays en *Controllers* se gestionan con HitTiles. Los HitTiles que en *Controllers* pueden tener una imagen de memoria solo en el dispositivo (Internal) o dos imágenes de memoria equivalentes, una en el host y otra en el dispositivo (tied). En estas últimas el sistema mantiene automáticamente la coherencia entre ambas. Los cambios en una imagen se reflejan automáticamente en la otra cuando es necesario.

En *Controllers* existen dos tipos de políticas de sincronización. La primera es la política síncrona, en la que las tareas se realizan en el mismo orden lógico en el que son lanzadas o invocadas desde el código del host, y en la que cada operación de cómputo o comunicación de datos se sincroniza y espera a la finalización de todas las anteriores antes de iniciarse. En este modo no existe solapamiento de cómputo en un dispositivo y operaciones de comunicación de datos entre el dispositivo y el host. Es el modo de ejecución más seguro y con menos probabilidades de que potenciales errores conlleven problemas de equivalencia secuencial (resultados equivalentes a los del mismo programa ejecutado en secuencial, sin paralelismo). Sin embargo, es el modo menos eficiente, ya que no permite ahorrar tiempo solapando tareas de cómputo y comunicación.

Por otra parte, en la política asíncrona, las tareas y operaciones de comunicación necesarias solo se sincronizan con las operaciones previas que impliquen una dependencia de datos. Es decir, cada operación solo espera para iniciarse a que terminen operaciones que están modificando sus datos de entrada o lectura, o leyendo sus datos de salida o escritura. Una vez sincronizada con el final de las operaciones con las que tiene dependencias, una operación se inicia en cuanto otras operaciones del mismo tipo (cómputo o comunicación) han terminado. Esto permite solapar cómputo en un dispositivo con operaciones de comunicación entre el mismo y el host. Este modo es más eficiente en tiempo de ejecución del programa si hay posibilidades de solapamiento.

Las principales funcionalidades que nos ofrece *Controllers* son:

- Gestión de kernels. Mecanismo para declarar, configurar y lanzar kernels u Host task para diferentes implementaciones. Nos proporciona mecanismos para optimizar automáticamente los parámetros de lanzamiento, como por ejemplo, las geometrías de los bloques de hilos. Además, nos ofrece la posibilidad de elegir entre una política asíncrona o síncrona en tiempo de ejecución, permitiendo al objeto *Controllers* ejecutar los movimientos del kernel y los datos con o sin solapamiento de computación y comunicación.

- Gestión de memoria. Mecanismos para realizar movimientos de memoria optimizados y de forma transparente al usuario entre el host y los dispositivos, y viceversa.

Arquitectura

Su arquitectura se representa en la figura 5.3. En ella podemos ver como una instancia de *Controllers* se asocia con un dispositivo específico para gestionarlo. Cuando se lanza una operación, esta se añade a la cola. A continuación, a partir de la información de eventos almacenada en los HitTiles, se transforma las peticiones a la tecnología adecuada asociada a la instancia de *Controllers*. Una vez estas operaciones son lanzadas a los dispositivos, los drivers correspondientes se encargan de coordinar la ejecución de los kernels y las transferencias de datos (3), siguiendo el orden implícito en los eventos programados por la política de ejecución y el adaptador de eventos.

Los módulos del sistema se representan en el diagrama de paquetes de la figura 5.4 y son los siguientes:

- Módulo Kernel. Este módulo se encarga de reescribir el código de usuario y transformarlo en código final específico para cada arquitectura.
- Módulo Core. Este módulo se encarga de la gestión y lanzamiento de las tareas, así como, de la cola de tareas si se encuentra activada.
- Módulo Architectures. En este módulo se encuentra la implementación de los diferentes backends en las tecnologías asociadas.

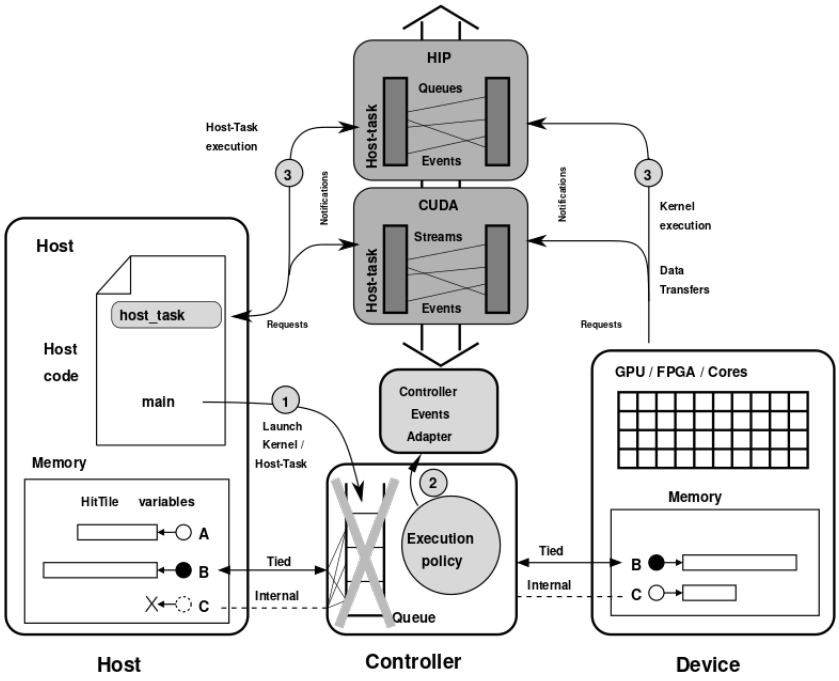

Figura 5.3: Arquitectura de *Controllers*. Foto extraída de [5].

Instrucciones

En *Controllers* existen dos tipos de intrucciones. Por un lado, las *declaraciones*, macro-funciones en lenguaje C que se transforman en declaraciones de más bajo nivel y prototipos de funciones.

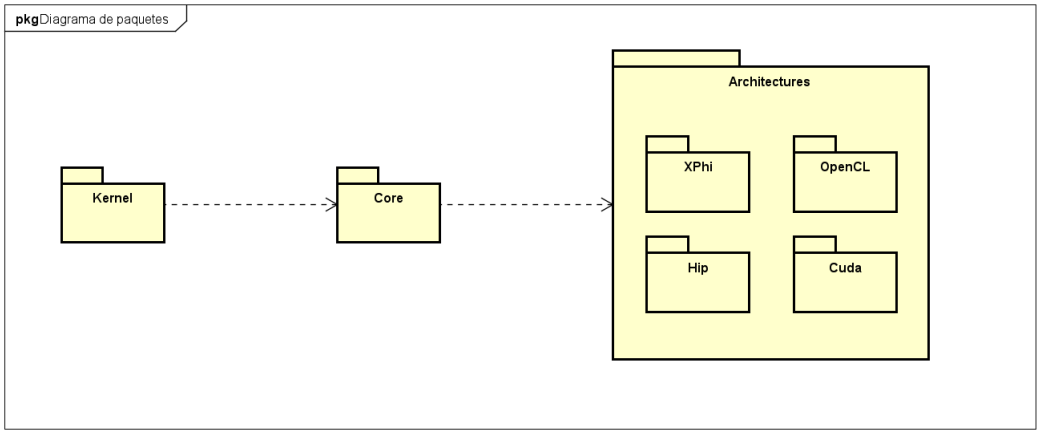

Figura 5.4: Diagrama de paquetes simplificado de *Controllers*.

Por otro, las funciones que son peticiones a los dispositivos asociados a la instancia de *Controllers*, es decir, a los backends que implementan las tecnologías específicas de cada dispositivo.

Declaraciones:

- **CTRL_KERNEL_PROTO(kernelName, implementationsList, parameterList).** Macro función que permite declarar un prototipo común para las diferentes implementaciones de un kernel. A esta macro función se le pasa la lista de implementaciones que se pretende declarar. La palabra clave GENERIC indica que se puede usar en cualquier backend, luego se declara para todas las implementaciones activas. El resto de parámetros son el nombre del kernel y la descripción de los parámetros del kernel, incluidos sus roles de entrada/salida, nombres y tipos. Las palabras clave IO (in/out), IN y OUT indica el rol del HitTile recibido como referencia. La palabra clave INVAL indica que el parámetro se pasa por valor.

- **CTRL_HOST_TASK_PROTO(kernelName, implementationsList, parameterList).** Macro función similar a *CTRL_KERNEL_PROTO*, pero en este caso permite declarar las tareas de host o Host task que se ejecutan en el dispositivo host.

- **CTRL_KERNEL(kernelName, backend, family, [paramType, paramName]..., code).** Macro función que permite declarar una implementación de un kernel. Los tres primeros argumentos son el kernel y la tupla (backend-family) que describe una de las implementaciones particulares declaradas en la función CTRL_KERNEL_PROTO(). A estos parámetros les sigue la lista de tipos y nombres de los parámetros del núcleo enumerados en la función CTRL_KERNEL_PROTO(). Finalmente, el último parámetro corresponde al código del kernel como bloque estructurado.

- **CTRL_HOST_TASK(kernelName, backend, family, [paramType, paramName]...).** Macro función similar a *CTRL_KERNEL* que permite declarar una implementación de una Host task. La diferencia con la anterior función es que el código a ejecutar es un bloque estructurado localizado justo después del macro, en vez de como parámetro.

- **CTRL BLOCK(n).** Macro función que declara el número de objetos *Controller* que se utilizarán dentro del bloque estructurado que sigue. Entre otras tareas, esta macro crea hilos OpenMP para gestionar los distintos controladores.

- **Ctrl_newType().** Macro función que permite definir tipos particulares de HitTile.

Funciones:

- int **Ctrl_SetHostAffinity(host_aff).** Función que permite elegir el nodo NUMA de la plataforma host que ejecutará el objeto *Controller*.
- int **Ctrl_ThreadInit(threads, size_x, size_y, size_z).** Función que permite definir e inicializar el bloque de hilos que se utilizará para dividir internamente el conjunto o malla de hilos lógicos seleccionado por el programador en porciones que se adapten al dispositivo y cómputo.
- Ctrl **Ctrl_Create(CTRL_TYPE, policy, device).** Función que crea el objeto *Controller*, seleciona el backend, la política de sincronización y el dispositivo.
- void **Ctrl_Launch(ctrl, kernel, threads, params).** Función que permite lanzar un kernel en el dispositivo asociado. Los drivers de los dispositivos se encargan de coordinar la ejecución de los kernels y las transferencias de datos. Se recibe como parámetros el objecto *Controller*, el kernel que se va a lanzar, el bloque de hilos *threads* con los que se ejecutará el kernel y los parámetros del kernel.
- void **Ctrl_HostTask(ctrl, kernel, params).** Función similar a *Ctrl_Launch* que permite ejecutar una Host task.
- void **Ctrl_Alloc(ctrl, tile).** Función que permite reservar memoria para una estructura de datos ya declarada en el host y en el dispositivo.
- void **Ctrl_DomainAlloc(ctrl, tile).** Función que declara y aloja espacio para la estructura de datos (tile). Tienen parámetros opcionales para seleccionar dónde se aloja la memoria, en uno o len ambos lados.
- void **Ctrl_Free(ctrl, tile).** Función para desalojar estructuras de datos.
- void **Ctrl_Destroy(ctrl).** Función que permite destruir el objeto *Controller*.
- void **Ctrl_MoveTo(ctrl, tile).** Función que permite transferir manualmente una estructura de datos del espacio de memoria del host al espacio de memoria del device.
- void **Ctrl_MoveFrom(ctrl, tile).** Función que permite transferir manualmente una estructura de datos del espacio de memoria del device al espacio de memoria del host.
- void **Ctrl_WaitTile(ctrl, p_task(tile)).** Función que espera a que todas las operaciones asociadas al tile que se pasa por parámetro finalicen.
- void **Ctrl_GlobalSync(ctrl).** Función que espera a que todas las operaciones que afectan los tiles asociados al objeto *Controller* finalizacen. Es una llamada siempre síncrona.
- void **Ctrl_HostTask_Sync().** Función que espera a que todas las Host task finalizen.
- void **Ctrl_Synchronize().** Función que espera a que todas las operaciones de todos los controladores finalicen.

- void **Ctrl_SetDependanceMode(ctrl, mode).** Función que permite cambiar el modo de dependencia para un objecto *Controller*. El modo de dependencia indica como se realizarán las transferencias de memoria. En *Controllers* hay dos modos de dependencia, el modo por defecto o implícito, en que el objeto *Controller* realiza transferencias de memoria automáticamente cuando es necesario. Por otro lado, el modo explícito, en que el objeto *Controller* solo realiza transferencias de memoria cuando se le indica explícitamente.

- HitTile **Ctrl_Select(ctrl, type, p_parent, shape, flags).** Función que declara un nuevo tile como un subconjunto de otro tile previamente reservado en memoria para ese *Controller*.

Modelo de sincronización

Como se ha mencionado anteriormente, *Controllers* ofrece la posibilidad de elegir entre una política asíncrona o síncrona en tiempo de ejecución, permitiendo al objeto *Controller* ejecutar los movimientos del kernel y los datos con o sin solapamiento de computación y comunicación.

En el modelo asíncrono, las Host tasks y los kernels pueden solaparse si las dependencias lo permiten y las transferencias de datos pueden solaparse tanto con la ejecución de las Host tasks y los kernels. Siempre preservando el orden parcial de ejecución del los kernels y las Host task. Las reglas que determinan si estas tareas pueden solaparse o no se realizan teniendo en cuenta los tipos de peticiones y sus parámetros de entrada/salida. Las dependencias generadas se encuentran recogidas en la figura 5.5. Siendo HL (Host-task Launch) el lanzamiento de una Host task, KL (Kernel Launch) el lanzamiento de un Kernel, HTD (Host To Device) movimiento de memoria del Host al Device y DTH (Device To Host) movimiento de memoria del Device al Host.

Procesos

Para implementar el nuevo backend, como se ha mencionado varias veces a lo largo de este documento, se toma como partida el backend de CUDA debido a la similitud entre ambas tecnologías. Para desarrollarlo es necesario replicar dos tipos de instrucciones de *Controllers*, por un lado, las declaraciones y por otro las funciones, explicadas anteriormente.

Para representar la sustitución de macros, de funciones, así como el flujo que realizan ambas operaciones, se van a emplear diagramas de actividades de UML. Luego, el primer paso a realizar, son los diagramas asociados al backend de CUDA, esto nos permitirá tener una mejor compresión de los procesos implicados en el sistema y del sistema en su conjunto. Estos diagramas serán empleados como punto de partida para realizar los diagramas del nuevo backend.

Comenzando con los diagramas de actividades de las declaraciones, en la figura 5.6, podemos ver el diagrama de actividades para CTRL_KERNEL_PROTO, que declara un prototipo común para las diferentes implementaciones de un kernel, permitiéndonos declarar diferentes implementaciones para un mismo kernel. Como podemos observar, el proceso comienza cuando se lanza la declaración CTRL_KERNEL_PROTO. Se hace una comprobación de si está activado el flag CTRL_FPGA_KERNEL_FILE, omitiendo la continuación del flujo en el que el flag está activado, ya que no es una parte que no influye en el desarrollo del nuevo backend. En caso de que no esté activada, el flujo se divide, por un lado, declara los prototipos del kernel para cada una de

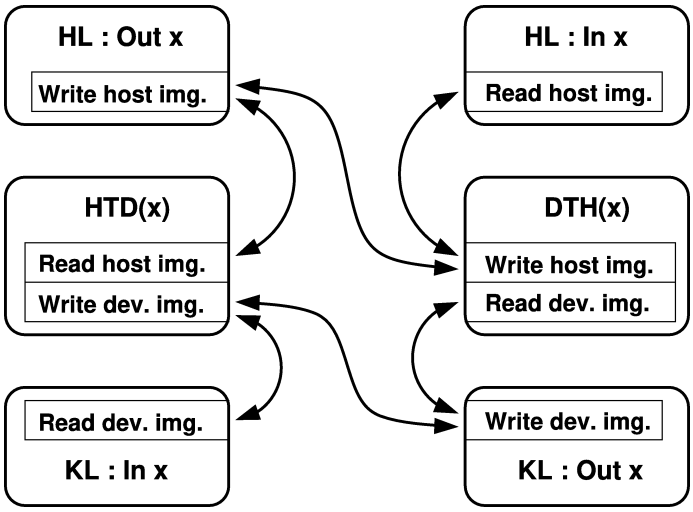

Figura 5.5: Dependencias entre tipos de solicitud. Las casillas redondeadas identifican los tipos de solicitud, utilizando la estructura de datos x como parámetro. Distinguimos entre peticiones KL o HL que utilizan x como de entrada o salida (In/Out). Para mayor claridad, en la figura se omiten las operaciones Wait y Alloc/Free. Dentro de cada recuadro, representamos en recuadros pequeños si la petición lee o escribe en la imagen de memoria del host o del dispositivo de x. Las flechas expresan las dependencias entre los tipos de solicitud implícitas por el uso (lectura o escritura) de las imágenes de memoria de x. Las peticiones que no están vinculadas con flechas pueden ejecutarse concurrentemente. Recuerde que este modelo no considera el caso de varios kernels concurrentes o varias tareas host concurrentes. Foto extraída de [5].

las implementaciones seleccionadas, o en caso de que el kernel sea genérico, para todas las implementaciones. Por otro lado, se declara la función que hace el lanzamiento de cada una de las implementaciones del kernel seleccionadas, o al igual que antes, si el kernel es genérico, de todas las implementaciones del kernel activas. Y por otro, se crea la tarea Ctrl_Task asociada. Este flujo es el más complejo. El flujo del resto de declaraciones son subconjuntos de este. Solo presentamos el diagrama de este ya que engloba a todos los otros.

Por otro lado, las funciones, simplemente realizan una serie de sustituciones hasta llamar a las funciones específicas de cada backend, unas declaradas anteriormente por las instrucciones de declaración y otras específicas de la tecnología del backend. En este caso, se presenta el diagrama de la función Ctrl_Create 5.7, función que crea el objeto *Controller*, selecciona el backend, la política de sincronización y el dispositivo asociado. Primero hace una comprobación del tipo de implementación para hacer la sustitución de funciones, se omite representar el flujo para el resto de implementaciones porque se perdería el foco y no influye para desarrollar el nuevo backend. Se realizan una serie de sustituciones de funciones, se comprueba si tiene streams o no, se llama a un *wrapper*, función contenedora que llama a la función de creación del objeto *Controller* específica del backend de CUDA y si están activadas las colas se inicializan llamando a la función Ctrl_TaskQueue_Init. El resto de funciones, salvo una de la que hablaremos a continuación, tienen un flujo similar, realizan las sustituciones y llamada de funciones hasta llegar a las funciones específicas del backend.

En cambio, el diagrama de la función Ctrl_Launch, representado en la figura 5.8, tiene un flujo ligeramente diferente al del resto de funciones, ya que realiza una serie de sustituciones de macros para cada backend y posteriormente hace las llamadas a las funciones específicas. Como podemos observar, previamente hace una sustitución de macros, comprobando las dimensiones para lanzar

el kernel con un bloque de hilos u otro, comprueba si las colas están activadas, en cuyo caso lanza la tarea a la cola. En caso contrario, evalúa la tarea, llamando a la función específica de cada backend y se llama al wrapper que lanza el kernel.

En este proyecto solo hay un único caso de uso, que se corresponde al lanzamiento de un programa, representado en el diagrama de secuencia 5.9. En él podemos ver la interacción del usuario para lanzar un programa. Existen dos alternativas, por un lado, que el programa esté compilado, entonces simplemente el usuario ejecutaría el programa y se le devuelve el resultado (el tiempo de ejecución del programa) y por otro, el caso en que no esté compilado el programa. Entonces el usuario tendría que compilar el programa enlazando con la librería *Controllers* y posteriormente realizar el proceso mencionado anteriormente.

Luego con esto hemos completado el análisis del sistema *Controllers* comprendiendo su funcionamiento interno, procesos y arquitectura.

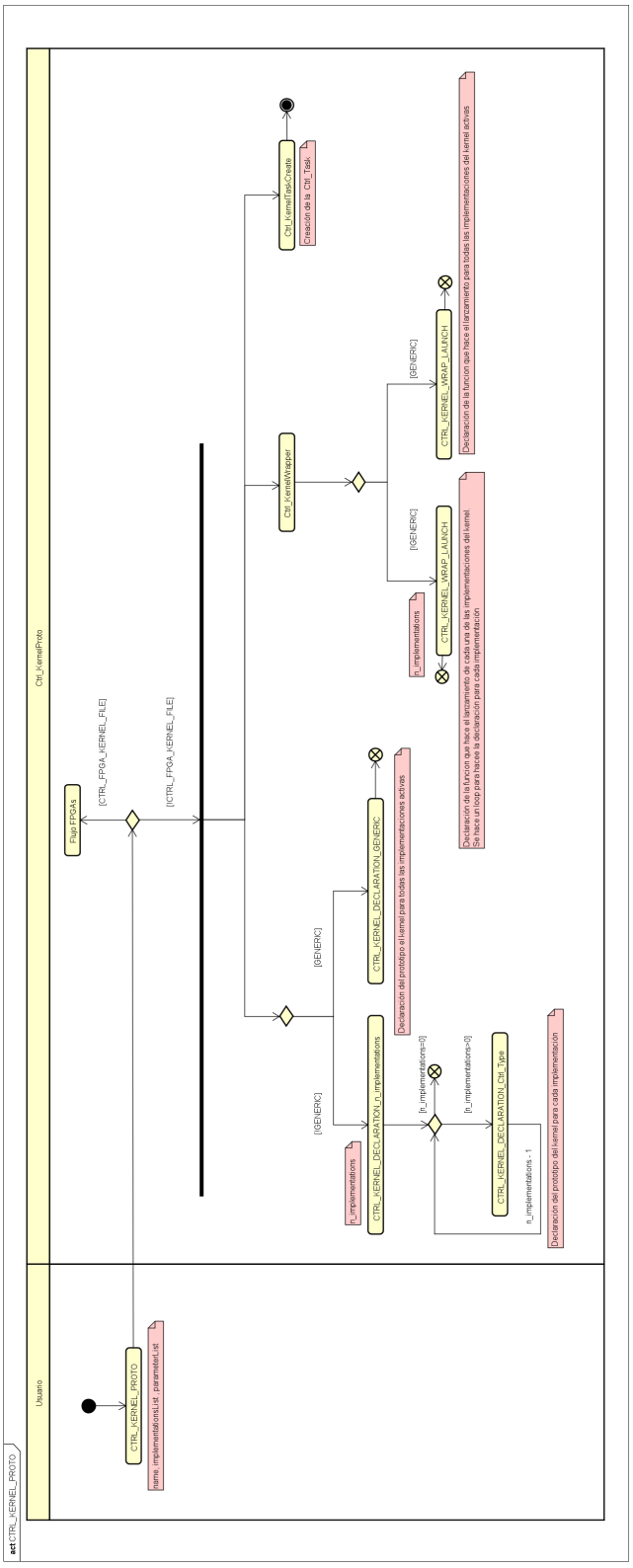

Figura 5.6: Diagrama de actividades de CTRL_KERNEL_PROTO.

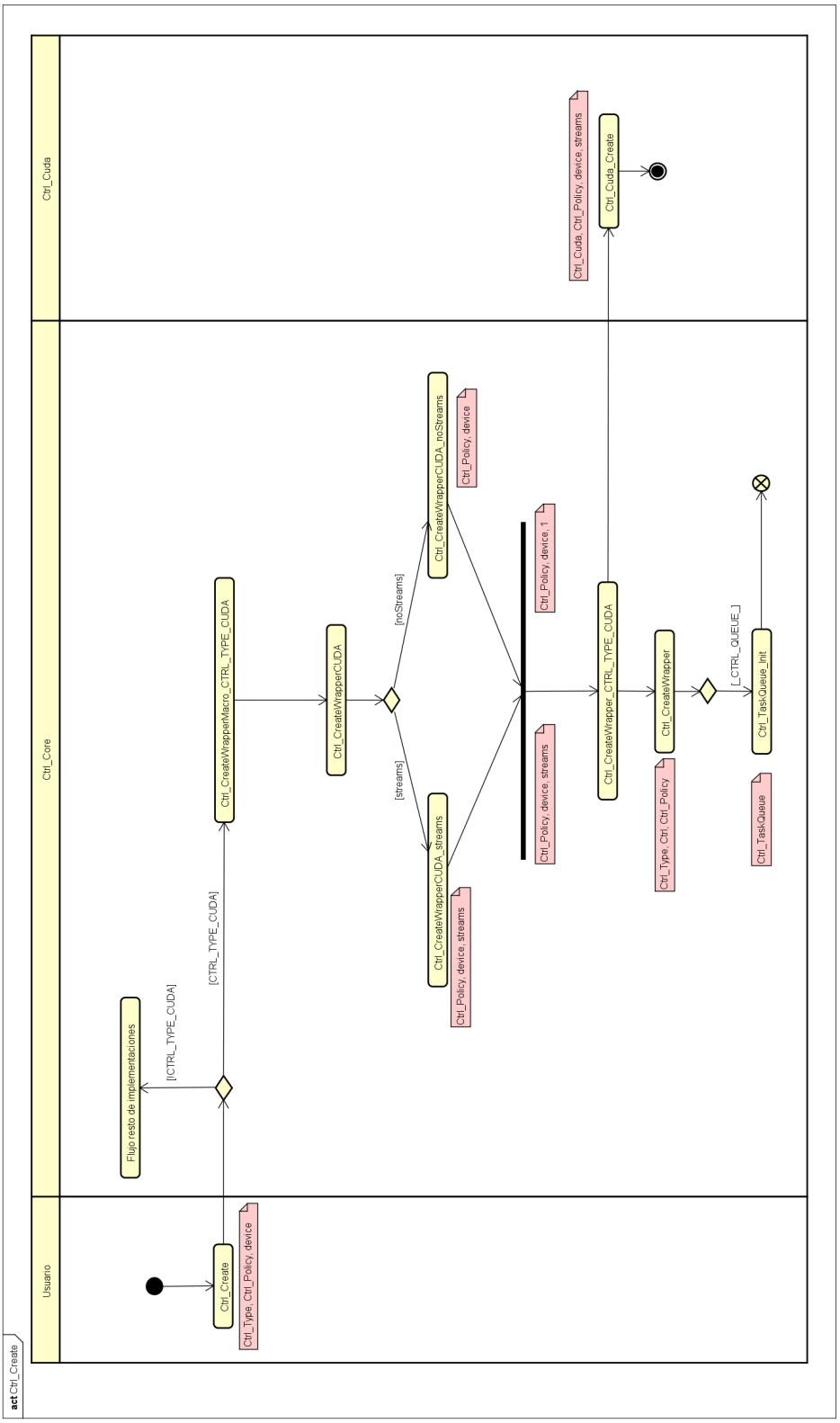

Figura 5.7: Diagrama de actividades de Ctrl_Create para Cuda.

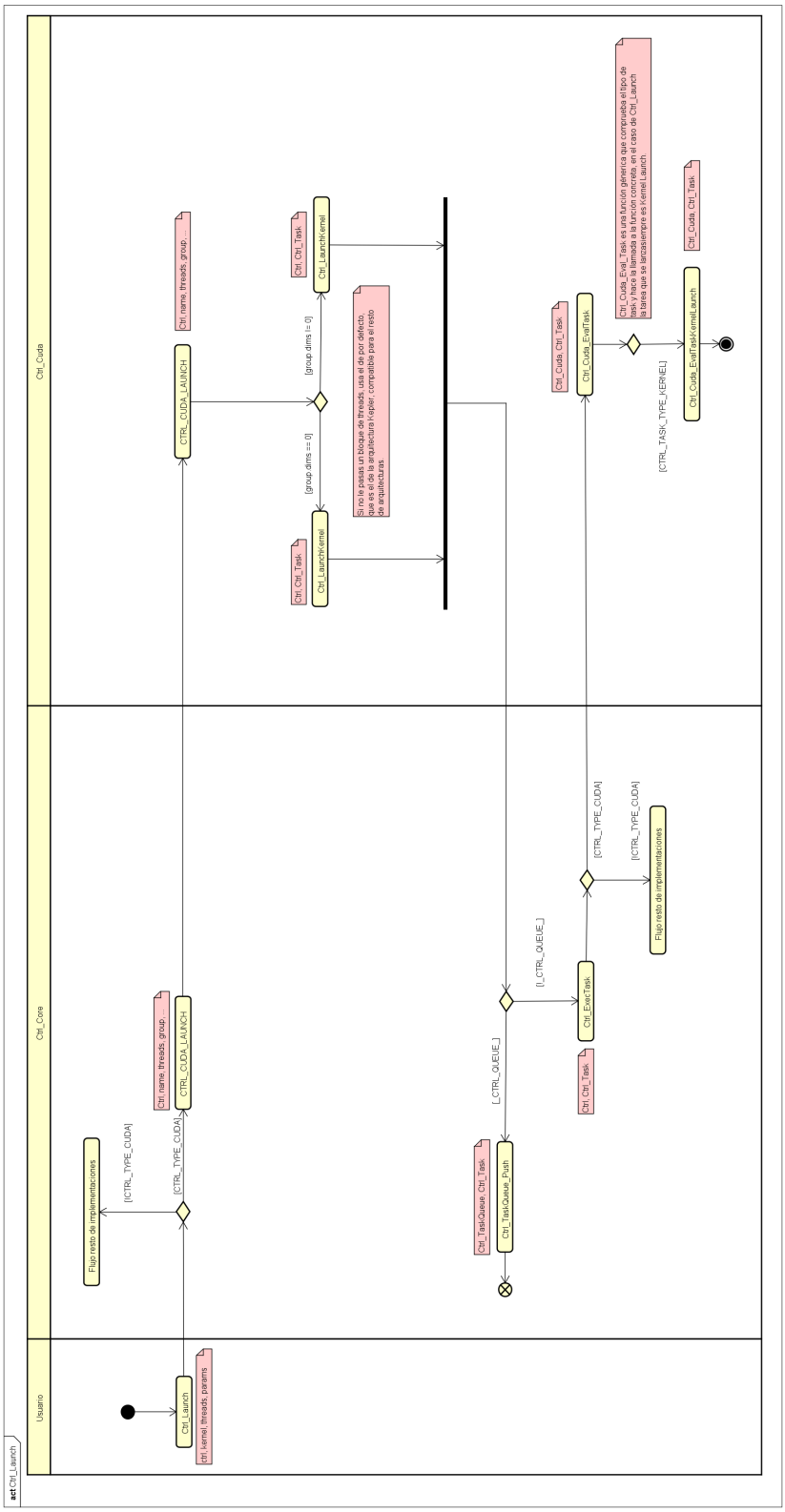

Figura 5.8: Diagrama de actividades de Ctrl_Launch para Cuda.

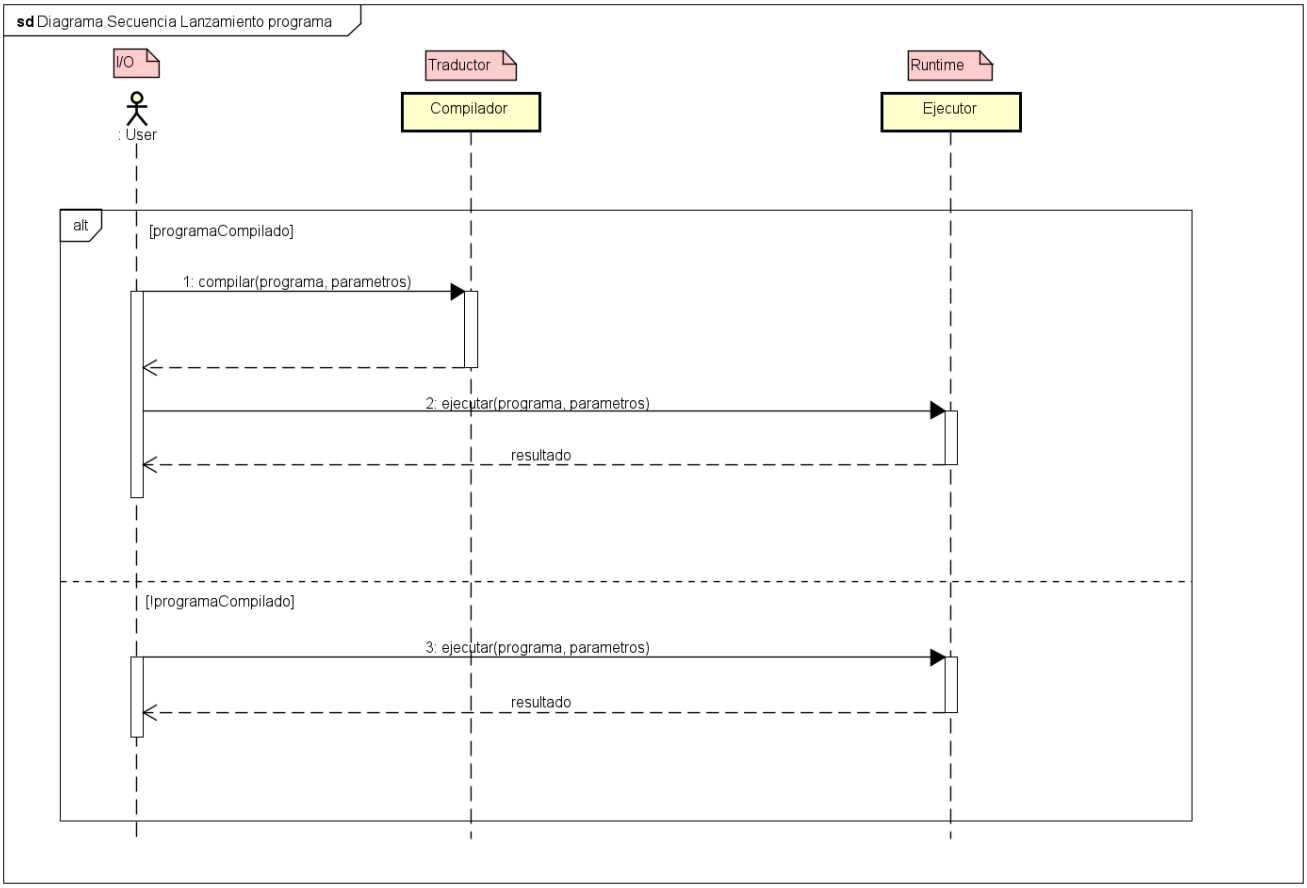

Figura 5.9: Diagrama de secuencia Lanzamiento de un programa de *Controllers*

Capítulo 6

Diseño de la solución

En esta sección se presenta la propuesta de solución para implementar el nuevo backend una vez analizado el sistema de partida *Controllers*.

Debido a la similitud entre CUDA e HIP y a la estructura de *Controllers*, el proceso de replicar las diferentes operaciones es sencillo. El flujo de las operaciones es el mismo, ya que la estructura es la misma y el objetivo es portar la tecnología de un backend a otra, lo que hace que los diagramas sean prácticamente idénticos. Solo difieren en los nombres de tipos y macros que se construyen para el nuevo backend. Simplemente, hay que crear los ficheros, macros y funciones para que se realicen las sustituciones adecuadas para declarar las funciones y wrappers apropiados y que se llamen las funciones asociadas a la tecnología del nuevo backend (HIP).

Por ello, el diagrama de actividades para CTRL_KERNEL_PROTO, es el mismo que el del backend de CUDA 5.6, al ser el flujo idéntico. Luego, internamente se tendrá que declarar el nuevo tipo de implementación y los macros con las sustituciones adecuadas.

Por otro lado, para las funciones, ocurre algo similar, simplemente se tiene que declarar el nuevo tipo, funciones y macros asociados para que las llamadas se hagan al nuevo backend implementado. Internamente, se tendrán que portar las funciones de CUDA a las funciones homólogas de HIP. En las figuras 6.1 y 6.1, podemos ver los nuevos diagramas de actividades para el nuevo backend. Como podemos ver, estos solo difieren de los anteriores en los nombres de las funciones, en el tipo de implementación y tipo de objeto *Controller*.

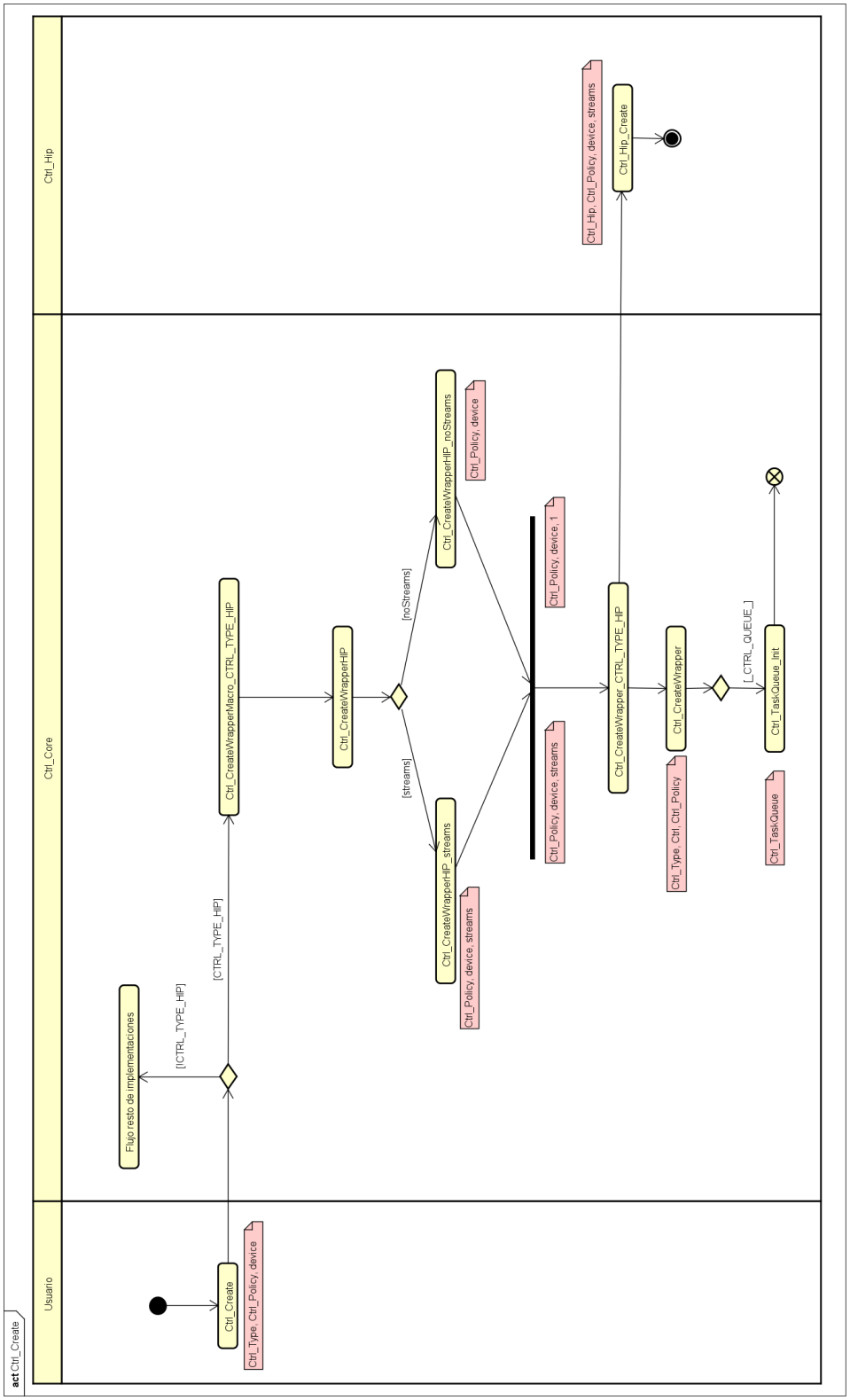

Figura 6.1: Diagrama de actividades de Ctrl_Create para Hip.

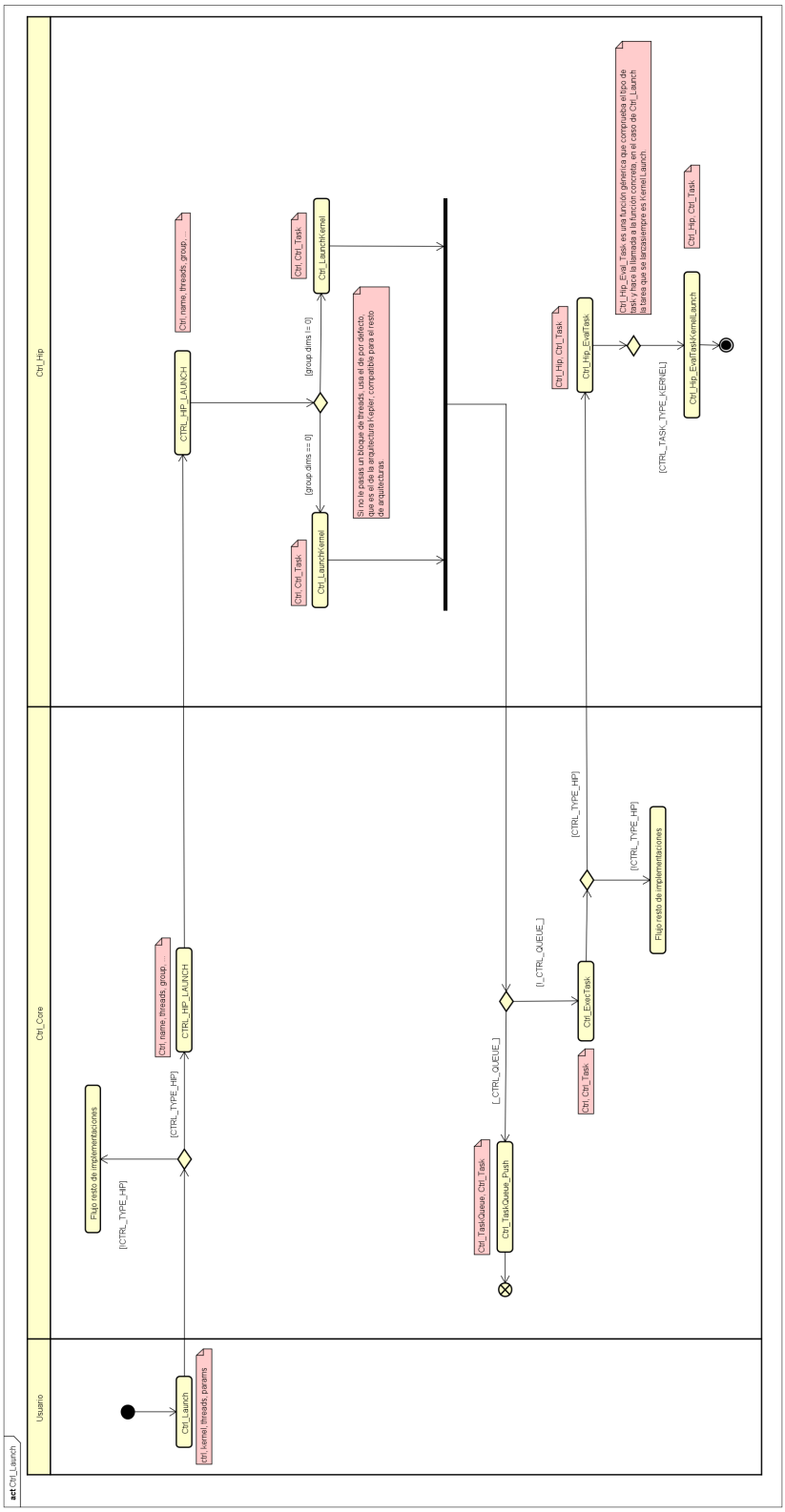

Figura 6.2: Diagrama de actividades de Ctrl_Launch para Hip.

Capítulo 7

Implementación y pruebas

7.1. Implementación del Backend

El backend de partida elegido es el de CUDA para las tarjetas gráficas NVIDIA, debido a la gran similitud entre HIP y CUDA. Esta similitud junto con la estructura modular de *Controllers* ha permitido que la portabilidad entre ambas tecnologías sea más sencilla y directa.

Para llevar a cabo el desarrollo se han ido adaptando los diferentes módulos de *Controllers*, comenzando con el módulo *Kernel*, dónde se encuentran definidas las declaraciones, que realizan la transformación del código fuente de alto nivel de abstracción a otro código fuente que usa una tecnología más específica y de más bajo nivel, a través de la sustitución de macros. Posteriormente, el siguiente módulo a adaptar ha sido el *Core*, módulo que se encarga de la gestión y ejecución de las tareas de *Controllers*. Finalmente, se ha portado el *Backend*, duplicando los ficheros del backend de partida y renombrando primero las funciones de *Controllers* para posteriormente portar las funciones específicas de CUDA a HIP.

Por último, una vez desarrollado el nuevo código de *Controllers*, se han llevado a cabo las modificaciones necesarias en los ficheros de compilación *CMakeList.txt* del proyecto, para adaptar la integración de esta nueva tecnología y se ha desarrollado la documentación Doxygen en el código fuente. La versión empleada de Doxygen para realizar y generar la documentación ha sido la v1.8.5.

Problemas en la implementación

El primer problema al que nos hemos enfrentado, es que CUDA e HIP a pesar de su gran similitud, tienen funciones y estructuras de datos en la que los parámetros se ordenan de diferente forma o tienen parámetros diferentes. Para solucionar este problema se ha tenido que ir función por función mirando cuidadosamente si el orden de los parámetros era el adecuado. Para algunos de los casos se han empleado macros proporcionados por la tecnología para declarar las estructuras de datos, también proporcionados por CUDA, pero que permiten abstraer el ordenamiento de estos parámetros.

Otro de los problemas a los que nos hemos enfrentado para lanzar los programas a la máquina de pruebas, es que al compilar con las opciones por defecto, el sistema de compilación de HIP

intenta detectar qué dispositivos están instalados en la máquina, compilando el kernel solo para ese tipo de dispositivo (el código máquina compilado es diferente incluso para GPUs de diferente familia aunque el fabricante sea el mismo). Al compilar previamente los programas en la máquina frontend del clúster (única máquina accesible para compilar y lanzar los programas), el compilador no detecta ningún dispositivo GPU para hacer *offloading* y no compila el kernel. Así que en el fat-binary generado está el código de host, pero no hay un kernel apropiado para las tarjetas gráficas AMD que están localizadas solo en los nodos de cómputo. Por eso, cuando el programa se ejecuta en un nodo de cómputo, intenta lanzar el kernel, la función intermediaria intenta cargar un kernel que no encuentra y falla.

Este problema se ha solucionado usando "cross-compilation", compilación del programa para una arquitectura en concreto aunque no esté disponible en la máquina donde se compila. Para ello se ha utilizado la opción `--offload-arch`, indicándole la arquitectura asociada para las tarjetas gráficas AMD que se encuentran en el clúster donde se realizan las pruebas y experimentación. En este caso concreto, la opción adecuada es `--offload-arch=gfx900`.

Otro problema al que se ha tenido que hacer frente es a un fallo de *segmentation fault*, que se producía cuando se lanzaba un programa de prueba de *Controllers* del backend de HIP. El problema surgía en la gestión de hilos de *Controllers*, que utiliza OpenMP para crear y gestionar diferentes hilos de ejecución para el código de host, las Host-Task y cada instacia de un controlador. Al parecer, todos los hilos se identificaban con el mismo índice 0. Lo que hacía que no se lanzará la función privada Ctrl_Thread_HostTask_Thread, función que inicializa la variable Ctrl_TaskQueue, que luego al ser referencia esta era *null*.

Este error se producía porque al compilar con CMake, se estaba enlazando contra 2 librerías distintas de OpenMP, la de gnu (libgomp) y la de rocm (libomp), debido a que CMake localiza ambas al buscar dependencias automáticamente. Para solucionar este problema se ha cambiado la configuración de CMake para forzar a enlazar solo una de estas librerías dependiendo de sí el backend de HIP estaba activado o no.

Finalmente, el problema más grande al que nos hemos enfrentado es la imposibilidad de compilar simultáneamente el backend de CUDA junto al de HIP. Esto está causado porque ambas tecnologías comparten ficheros de encabezado, dónde se definen estructuras de datos y funciones que comparten el mismo nombre. Esto produce que se lance un error al compilar *Controllers* para ambas tecnologías, ya que el compilador detecta que estás intentando definir o declarar nuevamente una función o estructura de datos ya definida.

Este problema requeriría una reestructuración completa del sistema o una modificación de las librerías enlazadas en tiempo de ejecución. Ambas soluciones implicarían un gran esfuerzo y una alta complejidad, que abarca conceptos avanzados que van más allá del nivel de conocimiento del alumno, y del propósito del proyecto. Debido al escaso tiempo restante cuando se detectó el problema, se ha optado por una solución más sencilla y directa. Se ha añadido una lógica adicional en los ficheros de compilación que evitan y avisan al usuario que no se pueden compilar ambas tecnologías simultáneamente, compilando por defecto para HIP si el usuario intenta compilar *Controllers* para HIP y CUDA al mismo tiempo. Las modificaciones asociadas a la realización de una solución que permita la compilación simultánea de ambas tecnologías se deja como trabajo futuro.

7.2. Pruebas del Backend

En esta sección se recogen las pruebas realizadas para comprobar el correcto funcionamiento del backend, para ello se han llevado a cabo unas pruebas de sistema, las cuales se centran en comprobar la integración de los distintos módulos y el funcionamiento del programa en su conjunto.

No se han llevado a cabo unas pruebas unitarias, las cuales se centran en probar los módulos de forma individual, debido a que las funciones de *Controllers* tienen una alta relación unas con otras, con una dependencia del resultado de funciones previas que se deben ejecutar en un orden específico. Comprobar el funcionamiento de forma aislada no tiene sentido, en la mayor parte de los casos.

Para las pruebas de sistema se ha utilizado la batería de pruebas del Grupo de Investigación Trasgo, formada por los siguientes programas, que recogen todas las funcionalidades de *Controller* [3, 4, 5]:

7.2.1. Matrix Power

La potencia de matrices es una extensión de los programas 2 y 3 mm de PolyBench Benchmarks [29]. Este programa genera una cadena de multiplicaciones de matrices de longitud variable k, siguiendo el siguiente proceso. En el dispositivo, se calcula iterativamente la multiplicación de la matriz original por el resultado parcial de la matriz anterior:

$$C^k = C^{k-1} \times A : k \in [1 : n] \qquad \text{dónde } C^0 = A$$

El kernel está optimizado para utilizar memoria compartida para aprovechar los recursos de la GPU. Cada resultado parcial es trasferido al host, dónde se normaliza la matriz y se guarda en otro búfer. Para realizar la normalización de la matriz se realizan los siguientes pasos:

1. Determinación de los valores máximos y máximo de la matriz.
2. Restar el mínimo de cada elemento de la matriz, y dividir cada elemento por el máximo.
3. Calcular la norma de los elementos como la raíz cuadrada de la suma de cada elemento al cuadrado.
4. Dividir cada elemento de la matriz por la norma de los elementos.

Este caso se caracteriza por la gran carga de cómputo necesaria para realizar la multiplicación de matrices y una menor carga en las tareas de transferencia de datos, creciendo esta proporción de carga de cómputo a medida que crece el tamaño de la matriz.

7.2.2. Hotspot

Este programa es una extensión del código Hotspot de la suite de benchmarks Rodinia [30, 31]. Calcula el punto de estabilidad de la Ecuación Diferencial Parcial (EDP) de Poisson para la difusión de calor en 2 dimensiones. Se trata de un programa del denominado tipo ILP (Iterative Loop Stencil). Estos programas ejecutan de forma iterativa una actualización de cada celda de una matriz en función de los valores de ciertas celdas vecinas. Las celdas vecinas se escogen con un stencil o patrón determinado por las ecuaciones del problema concreto a resolver. En este caso se trata de un stencil de 4 puntos, que utiliza las cuatro celdas vecinas. En cada iteración se realiza un cálculo para determinar el nuevo valor para cada celda utilizando los valores de las cuatro celdas adyacentes. El kernel explota la memoria *shared* para la localidad espacial y temporal. La matriz de resultados se transfiere al host después de cada operación del kernel del dispositivo, guardándola en un búfer diferente del host mediante una tarea del host. De este modo, los resultados podrían utilizarse para comprobar resultados parciales o para crear una animación de la evolución del cálculo. Esto da lugar a oportunidades de superposición de comunicación y computación.

Este caso se caracteriza por su mayor carga en las tareas de transferencia de datos y una menor carga de cómputo del kernel.

7.2.3. Sobel YUV

El operador Sobel es un algoritmo de detección de bordes en imágenes, se basa en la utilización de dos filtros. El resultado de estos filtros se utiliza para calcular la magnitud y dirección del gradiente en cada punto de la imagen, lo que permite detectar los bordes.

En nuestro caso, la implementación que procesa iterativamente fotogramas de un vídeo en formato YUV. El programa lee un flujo de vídeo YUV de entrada desde un archivo, fotograma a fotograma. Cada fotograma está formado por tres componentes que se comunican al dispositivo, el filtro Sobel se aplica a cada componente, dando lugar a unas imágenes que se transfieren de vuelta al host para almacenarlas en un archivo de vídeo de salida.

Para probar el funcionamiento del filtro Sobel, se han utilizado los siguientes escenarios:

- **Sobel File To File (FTF):** Las imágenes se leen de un fichero y se guardan en un fichero, una situación de E/S limitada.
- **Sobel File To Mem. (FTM):** Las imágenes se leen de un fichero y se guardan en memoria, para simular la fase inicial de una pipeline de tratamiento de vídeo con varias fases o filtros encadenados.
- **Sobel Mem. To File (MTF):** Las imágenes se leen de un búfer de memoria y se guardan en un fichero, para simular la última etapa de una pipeline de tratamiento de vídeo.
- **Sobel Mem. To Mem. (MTM):** Las imágenes se leen de un búfer de memoria y se guardan en memoria, para simular una etapa en medio de una pipeline, una situación de memoria limitada.

Este caso se caracteriza porque la carga de cómputo y de transferencia de datos son muy pequeñas, con comunicaciones bidireccionales con varios kernels, por lo que hay una mayor posibilidad de solapamiento de tareas.

7.2.4. Realización de las pruebas

Una vez implementados los programas de pruebas se realizaron las pruebas de sistema para el nuevo backend de *Controllers*. La tabla 7.1 recoge los parámetros de entrada que se emplearon para lanzar la batería de pruebas.

Casos de estudio	Iteraciones	Tamaño de entrada	Escenarios
Hotspot	1, 2, 500	1024x1024, 1000x1000	-
Matrix Power	1, 2, 500	1024x1024, 1000x1000	-
Sobel	1, 2, 100	1920x1080	FTF FTM MTF MTM

Cuadro 7.1: Parámetros de entrada para los programas de la batería de pruebas.

Estos parámetros de entrada fueron escogidos por el personal del Grupo de Investigación Trasgo al diseñar las aplicaciones de prueba. Los diferentes números de iteraciones están escogidos para probar los casos límite del bucle principal y para tener un número de iteraciones suficiente para detectar problemas de sincronización en los solapamientos de comunicaciones y cómputo tanto en las versiones síncronas como asíncronas. En el caso de Sobel el tamaño de entrada es fijo por el diseño del programa, que trabaja con videos HD de la resolución indicada. En los otros dos programas se escogen dos tamaños. Uno alinea las filas y columnas de la matriz con las líneas de memoria del dispositivo, con múltiplos de dos del tamaño de bloque y con el tamaño máximo de memoria compartida del dispositivo. El otro prueba el caso en el que no existe dicha alineación. En el caso de Hotspot eso obliga al sistema de ejecución a introducir *padding* (datos de relleno inútiles) en la memoria para alinearla, y permite comprobar si el sistema de ejecución ignora correctamente esos datos. En el caso de Matrix Power el programa debe fallar, ya que el uso interno que hace el kernel de la memoria compartida del dispositivo obliga a que el tamaño sea potencia de dos.

Para verificar que los resultados obtenidos tras la ejecución de los programas para el nuevo backend de HIP eran correctos, se compararon con los resultados obtenidos por el backend previo de *Controllers* para dispositivos AMD, basado en OpenCL y los programas que utilizan directamente la tecnología HIP. Una vez realizada esta comparación se comprobó que los resultados eran los mismos y que los programas no lanzaban ningún tipo de error. Debido a los problemas presentados en la sección 7.1, esta batería se ejecutó correctamente el día 7 de mayo.

Capítulo 8

Estudio Experimental

8.1. Objetivos

El objetivo de este estudio experimental es realizar una comparativa del rendimiento del nuevo backend desarrollado para *Controllers* frente a versiones de referencia que utilizan directamente la tecnología del backend y frente a las versiones que utilizan el backend previo de *Controllers* para dispositivos AMD, basado en OpenCL. Esto permitirá verificar la eficiencia del uso de Hip en la solución propuesta para el desarrollo del nuevo backend.

Para ello se tomarán medidas de los tiempos de ejecución en segundos, empleando las funciones que proporciona OpenMP. Los programas miden dos tiempos, por un lado, uno global, que incluye la inicialización del programa, y por otro, uno de ejecución, que solo considera el tiempo en el que se realizan cálculos. Los tiempos empleados para este estudio corresponden a los de ejecución, ya que son los más significativos para medir la eficiencia de los mecanismos de sincronización y comunicación de un backend de Controllers. Solo la operación de creación se ejecuta antes de la inicialización de los datos, tarea que se realiza manualmente solo en el código de host.

8.2. Métricas

En esta sección, se van a presentar las métricas calculadas para realizar el estudio. Estas métricas nos permiten tener una mayor compresión de los resultados.

Las métricas seleccionadas son la media, desviación típica e intervalo de confianza al 95 %. Cada una de estas métricas nos dan una visión diferente de los datos, permitiéndonos evaluar su variabilidad y tendencia central. Por un lado, la media nos proporcionan una visión de la tendencia central. Por otro lado, la desviación típica nos permite observar la variabilidad de los datos, es decir, cuánto se alejan los valores atípicos de la media. Esta ha sido utilizada en una primera instancia para ver si los datos obtenidos tenía mucha dispersión y decidir si era necesario analizar más en detalle los resultados. Cuanto mayor sea este valor, mayor es la dispersión y viceversa. Finalmente, el intervalo de confianza al 95 %, nos indica un rango en el que el 95 % de las veces se obtendría un valor dentro de ese rango.

8.3. Casos de estudio

Los programas escogidos para llevar a cabo la experimentación son los mismos que han sido utilizados como batería de pruebas para la fase de Pruebas, introducidos en la sección 7.2.

Cada uno de estos programas presenta unas características específicas en términos de carga computacional y transferencia de datos, que permite observar el comportamiento del sistema en una amplia variedad de escenarios. Esto nos permitirá evaluar mejor la eficiencia y rendimiento bajo diferentes cargas de trabajo.

Para los programas con las implementaciones nativas de Hip, se han desarrollado dos versiones, una síncrona y otra asíncrona, para cada uno de los programas de estudio. En el caso de los programas de *Controllers*, se dispone de una única implementación, ya que la decisión de emplear una política síncrona o asíncrona se toma en tiempo de ejecución, como se ha explicado en el capítulo 5.

Se van a realizar dos tipos de pruebas, por un lado, por iteraciones (*iter*), para un tamaño de entrada fijo, modificamos el número de iteraciones y por otro, por tamaño de entrada (*size*), para un número fijo de iteraciones, modificamos los tamaños de entrada. Los parámetros con los que hemos lanzado los programas Matrix Power y Hotspot se muestran en la tabla 8.1, mientras que para Sobel YUV a entrada es un vídeo de alta resolución de tamaño de imagen fijo (1920x1080 píxeles), por lo que solo varía el número de fotogramas como se muestra en la tabla 8.2.

Para cada caso de estudio se ejecutan 30 repeticiones, con el objetivo de obtener muestras lo suficientemente grandes para poder aplicar el Teorema del Límite Central (TLC), consiguiendo que la distribución de la media de las muestras se aproxime a una distribución normal.

Casos de estudio	Iteraciones	Tamaño de entrada
Matrix Power	40	1024 1536 2048 4096 6144 8192
	20 40 60 80 100 120	2048
Hotspot	500	1000 2500 5000 10000 15000
	100 200 300 400 500 600	10000

Cuadro 8.1: Parámetros de entrada para los programas Matrix Power y Hotspot.

Casos de estudio	Resolución (píxeles)	Fotogramas size	Fotogramas iter
Sobel YUV	1920x1080	100	20 30 40 50 60 70 100

Cuadro 8.2: Resolución y número de fotogramas para Sobel YUV

8.4. Estudio

8.4.1. Entorno de experimentación

La máquina del clúster del grupo Trasgo empleada para la experimentación se denomina *manticore*. Sus especificaciones son las siguientes:

- Sistema operativo: Linux CentOS v7.5
- Procesador: 2 x Xeon Platinum 8160 @ 2.10GHz.
- Hilos: 98 (2 CPUs, 24 cores, 2 HT).
- Memoria RAM: 256 GB.
- Disco duro HDD: 1 TB.
- GPU NVIDIA: 2 x Nvidia Volta V100.
- GPU AMD: 2 x AMD Radeon 9600.

Respecto a las versiones de las herramientas implicadas en la experimentación y en el análisis de los datos, son las siguientes:

- GCC v10.3
- SLURM v18.08.3
- CMake v3.17.5
- HIP/ROCm v5.4.3
- Gnuplot v5.4
- OpenMP v4.5

8.4.2. Resultados

Los anexos C y D recogen todas las métricas calculadas para el desarrollo de este estudio experimental.

Antes de comenzar a analizar los resultados y sacar conclusiones, hay que tener en cuenta si los resultados muestran diferencias significativas, es decir, si se solapan o no los resultados y si podemos concluir que las medias obtenidas para dos versiones del mismo programa implementado con diferentes tecnologías indican realmente que una versión es significativamente mejor que la otra.

Por ello primero hay que fijarse en los intervalos de confianza de los resultados que vamos a comparar, si ambos intervalos de confianza no se superponen, podemos decir al 95 % que ambos intervalos no comparten un rango de valores, luego son significativamente diferentes. Esto nos permite concluir que uno de los resultados es mejor o peor al 95 % de seguridad.

En cambio, si los intervalos de confianza se superponen, no podemos sacar esta conclusión tan fácilmente. Es necesario realizar un análisis para saber si los resultados son significativamente diferentes para poder sacar conclusiones con seguridad.

Se ha realizado un Test de Levene para comprobar que las varianzas de las muestras son diferentes. En nuestros casos de estudio las varianzas son diferentes. Luego, para verificar si las

medias de las muestras son significativamente diferentes vamos a calcular el intervalo de confianza para la diferencia de las medias con varianzas diferentes [32]. Para poder calcular este intervalo se tiene que cumplir una serie de condiciones, las muestras tienen que tener una distribución normal, que se consigue con el TLC y han de ser muestras independientes. Para calcular este intervalo de confianza se emplea la fórmula 8.1 y para calcular los grados de libertad se emplea la fórmula 8.2.

De esta forma, una vez calculado el intervalo de confianza, podemos concluir si las muestras son significativamente diferentes o no, ya que si el 0 no se encuentra dentro de este intervalo significa que hay evidencias estadísticas para concluir que las medias de las dos muestras son significativamente diferentes. Por otro lado, si el 0 se encuentra dentro del intervalo, no hay suficiente evidencia para concluir que son significativamente diferentes.

$$\bar{X}_1 - \bar{X}_2 \pm t_{\frac{\alpha}{2},v} \cdot \sqrt{\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}} \tag{8.1}$$

$$v = \frac{\left(\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2}\right)^2}{\frac{\left(\frac{{s_1}^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{{s_2}^2}{n_2}\right)^2}{n_2 - 1}} \tag{8.2}$$

Dónde:

- n_1 y n_2 son los tamaños de las muestras.
- $\bar{X}_1$ y $\bar{X}_2$ son las medias de las muestras.
- s_1 y s_2 son las desviaciones típicas de las muestras.
- $t_{\frac{\alpha}{2},v}$ es el valor de la distribución t de Student con v grados de libertad y un nivel de significancia del $\frac{\alpha}{2}$.

En la comparativa del nuevo backend de *Controllers* con las implementaciones nativas, hemos observado algún caso en el que los intervalos de confianza de ambas muestras se superponían. Por ejemplo, en el caso de estudio Matrix Power con tamaño de entrada (tamaño de la matriz) 2048, número de iteraciones 40 y política asíncrona. Luego para poder sacar conclusiones hemos calculado el intervalo de confianza para la diferencia de las medias. Las métricas calculadas las podemos ver en la tabla C.1, siendo el caso 1 *Controller Hip Async* y el caso 2 *Ref Hip Async*, los resultados son los siguientes:

$$\bar{X}_1 = 0{,}7054; \bar{X}_2 = 0{,}7092; n_1 = 30; n_2 = 30; s_1 = 0{,}0155; s_2 = 0{,}012; \alpha = 0{,}05$$

Con estos valores obtenemos el intervalo de confianza del 95 %:

$$\text{IC} = [-0{,}007308, -0{,}000292]$$

Como podemos observar, el 0 no se encuentra dentro del intervalo luego como se ha explicado anteriormente, esto significa que las medias son significativamente diferentes. Por tanto, tenemos evidencias estadísticas para concluir que una de las muestras tiene mejor rendimiento frente a la otra. En este caso concreto, como el intervalo es negativo, esto significa que la muestra 1 tiene un mejor rendimiento, correspondiente a la versión compilando *Controllers* para el backend de Hip, o de forma alternativa que la muestra 2 tiene una media más grande y un peor rendimiento que la 1.

Para determinar cuán grande es esta diferencia de rendimiento, podemos calcular el porcentaje de mejora o pérdida, restando el resultado de una de las versiones por la otra y dividiendo por la primera. Para este caso, restando la versión de referencia a la versión de *Controllers* y dividiendo por la primera, obtenemos una mejora del rendimiento del 0,53 %.

A continuación se presentan los resultados analizados. Los casos donde las medias de las muestras no son significativamente diferentes no presentan ningún problema, ya que indican el caso ideal donde las soluciones utilizando diferentes tecnologías para resolver el mismo problema usando la misma máquina y dispositivo obtienen rendimientos similares. En el resto de la sección se hacen observaciones y se extraen conclusiones para los casos donde se aprecian diferencias significativas.

Controllers Hip frente a versiones de referencia de Hip

Primero vamos a hacer una comparativa de *Controllers* compilando con el backend de Hip con los programas escritos con las implementaciones nativas de Hip. En esta sección nos referiremos como versión de referencia a los programas con la implementación nativa de Hip.

Las figuras 8.1, 8.2 y 8.3/8.4.2 muestran los resultados de rendimiento obtenidos para las diferentes versiones de los casos de estudio Hotspot, Matrix Power y Sobel, respectivamente. Los gráficos presentan los tiempos de ejecución en el eje y y los distintos tamaños de entrada o número de iteraciones en el eje x.

Primero comencemos analizando los resultados obtenidos del caso de estudio Hotspot 8.1, en la gráfica de la izquierda, se muestran los resultados por iteraciones (*iter*), el objetivo de esta prueba es observar que a partir de un número de iteraciones, las diferencias proporcionales de rendimiento no varían. De esta forma nos podemos centrar en la variación de los tamaños de entrada, para un número arbitrario de iteraciones fijo, por ejemplo 500. En esta gráfica de la derecha podemos observar que las diferencias entre ambas versiones son muy pequeñas, incluso llegando a tener un mejor rendimiento en algunos casos la versión de *Controllers*. Este comportamiento podría sorprender, ya que *Controllers* añade un nivel más de abstracción.

Hemos deducido que la gestión de memoria *pinned* (memoria bloqueada para evitar paginación y permitir a los dispositivos DMA de acceso direct a memoria optimizar las transferencias de memoria entre host y dispositivo) en *Controllers* es un poco más eficiente que la de las referencias. Será necesario un estudio más en profundidad y diseñar un estudio experimental propio para verificarlo

Por otro lado, algo similar ocurre para Matrix Power 8.2. En la gráfica de la izquierda, se muestran los resultados por iteraciones, vemos como las diferencias proporcionales de rendimiento no varían y son unos rendimientos similares. En la gráfica de la derecha, variando el tamaño de entrada, observamos que para tamaños de entrada muy grandes, el rendimiento de la versión de

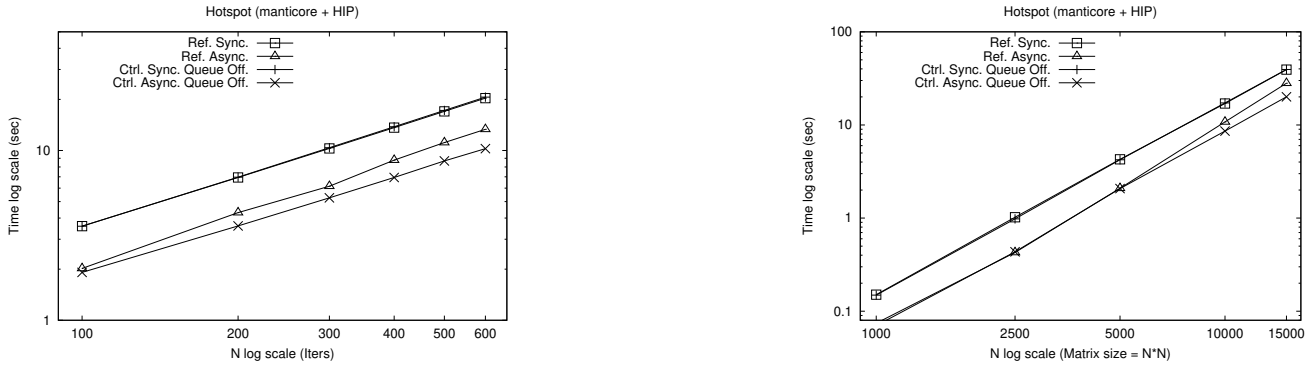

Figura 8.1: Resultados experimentales del caso de estudio Hotspot para *Controllers* frente a las implementaciones nativas de Hip.

Controllers tiene un mejor rendimiento, tanto para las versiones síncronas como asíncronas. Este comportamiento se aprecia más en Hotspot, ya que la carga del kernel es menor comparada con Matrix Power, luego cualquier ineficiencia en sincronizaciones o gestión de memoria hace que se note más.

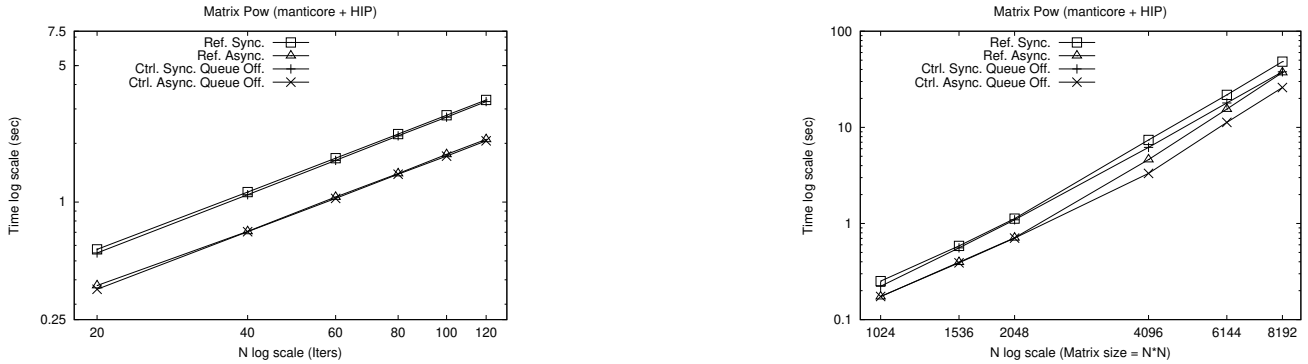

Figura 8.2: Resultados experimentales del caso de estudio Matrix Power para *Controllers* frente a las implementaciones nativas de Hip.

Continuado para el caso de estudio Sobel 8.3 variando el número de fotogramas para los diferentes escenarios, podemos observar un comportamiento diferente al esperado. El objetivo de esta prueba, como se ha mencionado anteriormente, es demostrar que a partir de un número de iteraciones (fotogramas), las diferencias proporcionales no varían. En estos casos podemos ver como para un número de fotogramas pequeño, la versión de *Controllers* tiene un mejor rendimiento. A medida que se va aumentando el número de fotogramas, se va notando como el rendimiento de *Controllers* se va reduciendo en comparación con el de referencia, llegando a tener un rendimiento similar con 40 fotogramas. A partir de este número, se empieza a observar el overhead que incluye las versiones *Controllers* al añadir un nivel más de abstracción, notándose como los mecanismos de sincronización más genéricos de *Controllers* son menos eficientes cuando se usa la tecnología Hip directamente.

Finalmente, para el caso de estudio Sobel 8.4.2 con un tamaño de entrada dado, es decir, un número de fotogramas fijo, 100. Podemos apreciar mejor la diferencia de rendimiento entre las versiones de referencia y *Controllers* mencionada anteriormente. Como se ha visto anteriormente, si se hubiera tomado un tamaño de fotogramas diferente, en este caso el resultado podría variar, por ejemplo, para un número de fotogramas inferior a 40 probablemente veríamos un mejor rendimiento

de *Controllers*. Para determinar el origen de este comportamiento que se aprecia solo en el caso de este programa concreto es necesario un estudio más en profundidad y diseñar un estudio experimental propio para verificarlo.

Luego, la solución propuesta incurre en poco o ningún sobrecoste, en comparación con las implementaciones nativas de Hip, observándose un mejor rendimiento en muchos de los casos, no llegando a un sobrecoste aproximado del 10 % de media, salvo para algunas excepciones. A esto hay que añadir que utilizar la librería *Controllers* requiere menos esfuerzo de desarrollo en comparación con implementar manualmente el código en las tecnologías de referencia para lograr el solapamiento de tareas.

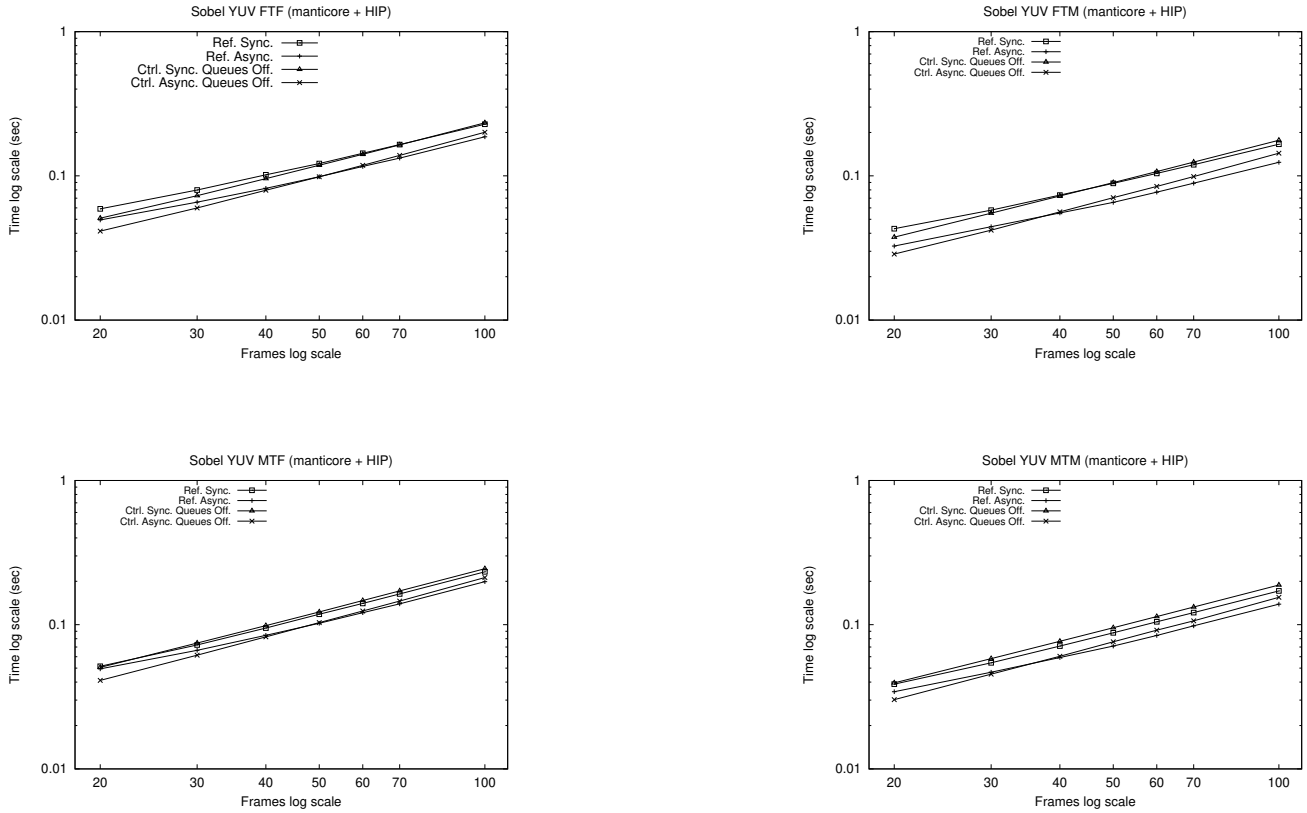

Figura 8.3: Resultados experimentales del caso de estudio Sobel YUV para *Controllers* frente a las implementaciones nativas de Hip. Los escenarios son FTF (File To File), FTM (File To Memory), MTF (Memory To File) y MTM (Memory To Memory).

Controllers Hip frente a Openclamd

A continuación, una vez comparado el rendimiento frente a las versiones nativas de Hip, se va a comparar el rendimiento del nuevo backend de Hip frente al backend anterior de Controller para las tarjetas AMD, basado en OpenCL. En esta sección nos referiremos como versión de referencia al backend de Openclamd ya existente en el sistema de partida.

Las figuras 8.5, 8.6 y 8.7/8.8 muestran los resultados de rendimiento obtenidos para las diferentes versiones de los casos de estudio Hotspot, Matrix Power y Sobel, respectivamente. Al igual que en la anterior sección, en los gráficos se presentan los tiempos de ejecución en el eje y y los distintos tamaños de entrada o número de iteraciones en el eje x.

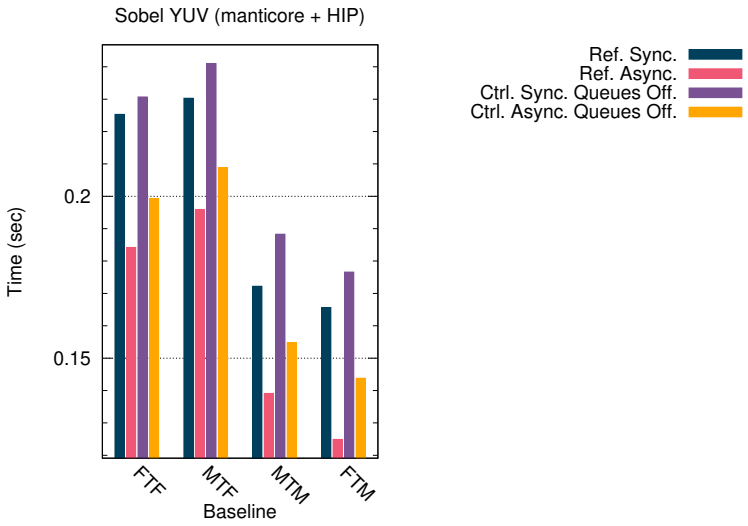

Figura 8.4: Resultados experimentales del caso de estudio Sobel YUV con 100 fotogramas para *Controllers* frente a las implementaciones nativas de Hip. Los escenarios son FTF (File To File), FTM (File To Memory), MTF (Memory To File) y MTM (Memory To Memory).

Comenzando con el caso de estudio Hotspot 8.5, en el que la carga de cómputo es menor y la carga en las tareas de transferencia de datos es mayor. Podemos ver como en la gráfica de la izquierda, variando el número de iteraciones, las diferencias proporcionales de rendimiento no se ven afectadas con la variación de este parámetro. Por otro lado, en la gráfica de la derecha, en la que se varía el tamaño de entrada (o matriz), el rendimiento del backend de Hip es mejor, tanto para las versiones síncronas como asíncronas. Luego, esto indica que los mecanismos de sincronización son más eficientes usando Hip que usando OpenCL.

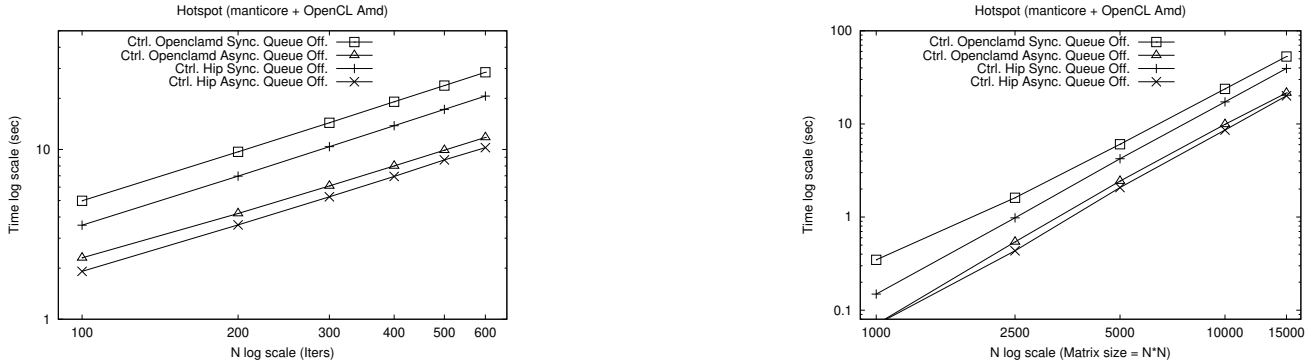

Figura 8.5: Resultados experimentales del caso de estudio Hotspot para *Controllers* frente a las implementaciones nativas de Hip.

Continuamos analizando el caso de estudio Matrix Power 8.6. Este caso se caracteriza por la gran carga de cómputo necesaria para realizar la multiplicación de matrices y una menor carga en las tareas de transferencia de datos. Por un lado, en la gráfica de la izquierda, la variación de las iteraciones no afecta a la diferencia proporcional de rendimiento. Para un tamaño fijo de iteraciones, por ejemplo 40, podemos observar como para un tamaño de entrada pequeño, la versión de openclamd para una política asíncrona tiene un mejor rendimiento. En cambio, para las versiones síncronas, el rendimiento de Hip es mejor para tamaños pequeños. A medida que se va aumentando el tamaño de entrada, esta diferencia de rendimiento se va reduciendo, teniendo

un rendimiento similar para tamaños grandes, incluso apreciándose un ligero mejor rendimiento para Hip en las versiones asíncronas.

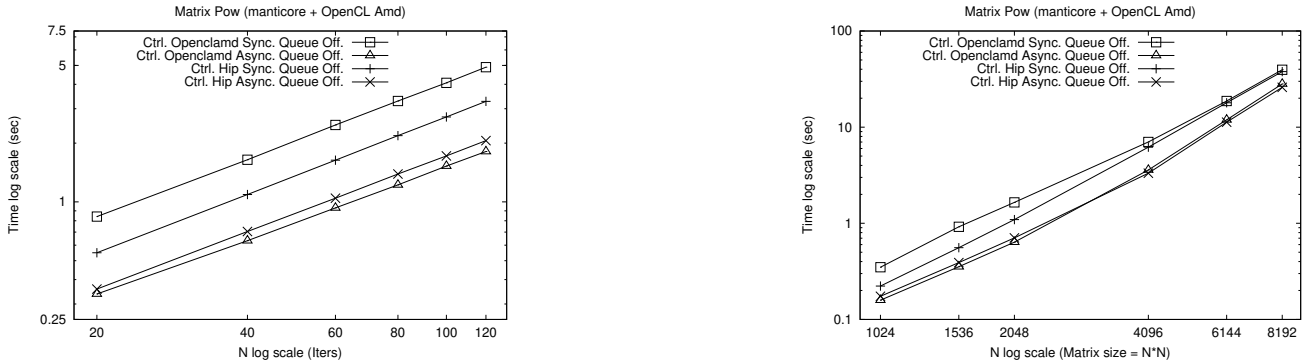

Figura 8.6: Resultados experimentales del caso de estudio Matrix Power para *Controllers* frente a las implementaciones nativas de Hip.

Para el caso de estudio Sobel 8.7, para los diferentes escenarios, variando el número de fotrogramas de entrada, podemos observar un comportamiento diferente. Podemos ver como las diferencias de rendimiento son proporcionales al aumento de iteraciones. En este caso, al igual que en el resto de casos, podemos apreciar como las versiones con política síncrona, tienen un mejor rendimiento para el nuevo backend. Pero, para el caso de las versiones con política asíncronas, ocurre un comportamiento diferente, en los escenarios FTF y MTF, en los que la salida es por fichero, el rendimiento del nuevo backend es peor. También destacar que los resultados para versiones con política síncrona y asíncrona son bastante similares en estos escenarios. En cambio, para las versiones en las que la salida es por memoria, FTM y MTM, el rendimiento es mejor en comparación con el backend existente.

Se ha deducido que este comportamiento se puede deber a un mal rendimiento de la Host task encargada de la escritura, el mecanismo de sincronización de tareas o una diferente gestión de memoria pinned. Será necesario un estudio más en profundidad y diseñar un estudio experimental propio para verificarlo.

Finalmente, nos centraremos en Sobel 8.8 con un número fijo de fotogramas, 100 en este caso. En esta gráfica podemos ver mejor el comportamiento mencionado con anterioridad, viendo como la diferencia de rendimiento entre ambos backends, para los escenarios FTF y MTF, en el que el rendimiento de Openclamd es mejor. Respecto a los escenarios FTM y MTM, podemos ver como el rendimiento de Hip es mejor.

Luego la solución propuesta presenta un mejor rendimiento en líneas generales para la mayoría de casos. Para aquellos casos en el que el rendimiento es menor, el sobrecoste aproximado que no supera el 10 % de media, salvo en algunas excepciones, como por ejemplo para los casos Sobel YUV en los escenarios FTF y MTF.

8.5. Conclusiones

Para finalizar el estudio experimental podemos concluir que la solución propuesta para integrar Hip en *Controllers*, presenta ventajas respecto a las versiones de referencia, tanto las implementaciones nativas de Hip, como el backend ya existente de Openclamd. Aunque en algunos casos

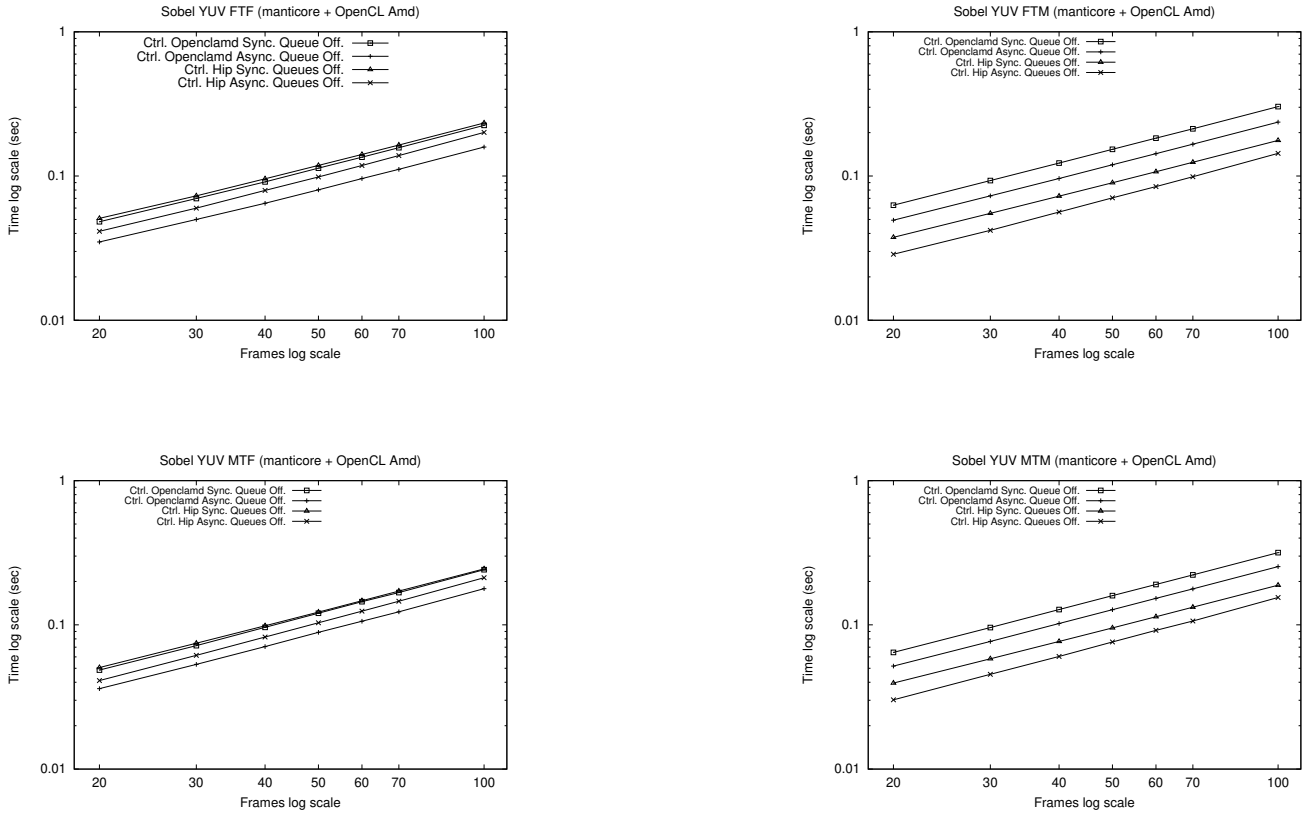

Figura 8.7: Resultados experimentales del caso de estudio Sobel YUV para *Controllers* Hip frente Openclamd. Los escenarios son FTF (File To File), FTM (File To Memory), MTF (Memory To File) y MTM (Memory To Memory).

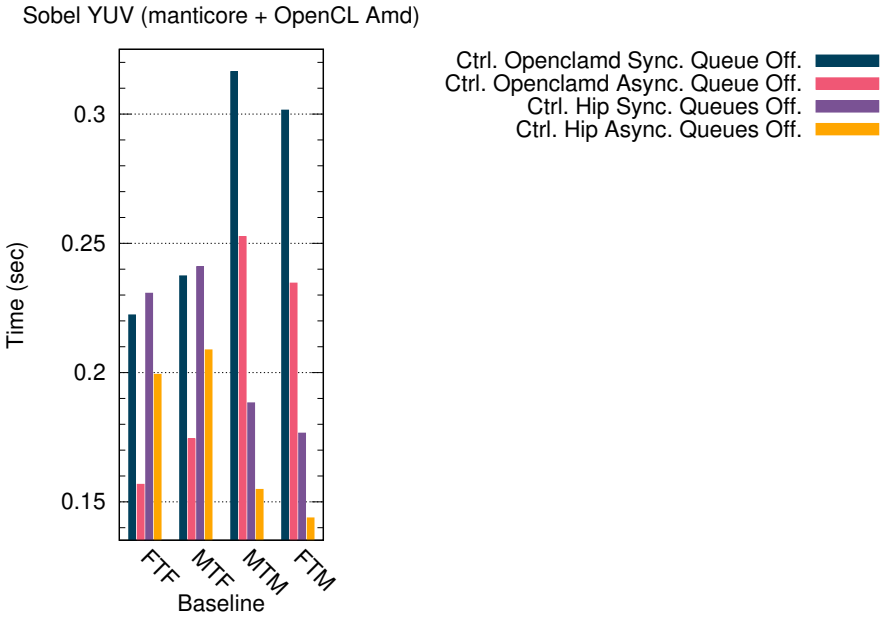

Figura 8.8: Resultados experimentales del caso de estudio Sobel YUV con 100 fotogramas para *Controllers* Hip frente Openclamd. Los escenarios son FTF (File To File), FTM (File To Memory), MTF (Memory To File) y MTM (Memory To Memory).

el rendimiento es menor, nuca supone un overhead superior al 10 % aproximadamente de media, salvo en algunas excepciones. Además, como se ha mencionado, el menor esfuerzo de desarrollo necesario para implementar la solución utilizando la librería *Controllers* con respecto a las versiones nativas de Hip, es un factor importante a considerar.

Otro factor a tener en cuenta, es el objetivo de mantener actualizada la tecnología implicada en *Controllers*, esto junto con las mejoras de rendimiento que hemos visto en varios casos, compensa los overhead observados.

Es necesario en cualquier caso seguir investigando los casos donde el nuevo backend no obtiene el rendimiento esperado, para deducir el origen de las diferencias y mejorar el uso que hace *Controllers* de la tecnología Hip.

Capítulo 9

Conclusiones

En esta sección se recogen las conclusiones extraídas, así como el trabajo a futuro que se podría realizar y una pequeña valoración personal de este proyecto.

9.1. Objetivos cumplidos

Se han cumplido todos los objetivos propuestos al comienzo del proyecto, enumerados a continuación:

- Se ha analizado y comprendido el sistema de partida *Controllers* [3, 4, 5] y la librería auxiliar *Hitmap* [6] realizando una ingeniería inversa.
- Se ha diseñado y desarrollado un nuevo backend del sistema *Controllers* utilizando tecnología HIP/ROCm para las tarjetas gráficas AMD.
- Se ha diseñado, implementado y ejecutado un estudio experimental para validar el modelo.
- Se ha analizado y comparado los resultados del estudio experimental con los programas de referencia y con el backend existente para las tarjetas gráficas AMD para comprobar la eficiencia de la implementación.

9.2. Líneas futuras

Durante el desarrollo del proyecto nos hemos dado cuenta de algunas modificaciones o ampliaciones que se podrían llevar a cabo:

- Actualmente, el backend de HIP, solo permite ejecutar los programas de *Controllers* en las tarjetas gráficas AMD. Una posible ampliación o línea futura podría consistir en ampliar la compatibilidad de *Controllers* para que los programas compilados para el backend de HIP puedan ejecutarse también en las tarjetas gráficas NVIDIA. Esto abriría nuevas posibilidades, ya que las tarjetas gráficas de NVIDIA son ampliamente empleadas en el campo de la

computación paralela y tienen un gran soporte por los desarrolladores. Además de que una tecnología actualizada nos proporciona acceso a nuevas oportunidades, innovaciones y soluciones, esta podría suponer una mejora del rendimiento y eficiencia de los cálculos.

- Por otro lado, durante el desarrollo de la fase de Implementación nos dimos cuenta de un problema, no era posible compilar simultáneamente *Controllers* para el backend de CUDA e HIP. Esto no es un desafío sencillo y requiere de unos conocimientos técnicos avanzados, que junto con el poco tiempo para realizar dicha modificación se ha dejado como una posible línea futura.

  Como se ha explicado en la sección 7.1, se podría abordar de dos formas, por un lado, se podría llevar a cabo una reestructuración completa del sistema, que evitará que los ficheros de encabezado de HIP y CUDA que comparten funciones y estructuras de datos se enlacen simultáneamente. Por otro lado, se podrían modificar las librerías enlazadas en tiempo de ejecución, pero esta sería probablemente una solución temporal.

- También hemos observado en el desarrollo del Estudio Experimental, como curiosamente en algunos casos *Controllers* tenía un rendimiento mejor que las implementaciones nativas de referencia y como se ha indicado en la sección 8.4.2, se ha deducido que la gestión de memoria pinned en *Controllers* es un poco más eficiente que la de las referencias. Adicionalmente, se ha visto como para el caso de Sobel YUV, variando el número de iteraciones, las diferencias de rendimiento son proporcionales al aumento de iteraciones. También nos hemos dado cuenta de ciertas situaciones anómalas, en la comparación de *Controllers* para el backend de openclamd frente al de Hip. Luego, para tener una compresión más detallada de la causa de estos comportamientos, es necesario realizar un análisis profundo y exhaustivo, que se deja como estudio futuro.

9.3. Valoración personal

Una vez realizado el proyecto, me gustaría expresar en esta pequeña sección mis sensaciones a lo largo de estos meses. No ha sido un trabajo sencillo, ya que se ha tenido que compaginar con las prácticas de empresa y la asignatura correspondiente al último cuatrimestre de la carrera, que han supuesto un desgaste que no había tenido en cuenta. A pesar de ello, he visto como poco a poco el esfuerzo y dedicación ha dado sus frutos y he ido superando los retos que han ido surgiendo.

Además, gracias a esta oportunidad, me he podido acercar un poco al mundo de la investigación académica que tiene una estructura completamente diferente a los proyectos software en los que he participado a lo largo de estos años y que han supuesto una formación complementaria que considero muy valiosa, proporcionándome un contexto muy útil si deseo continuar desarrollando mi vida laboral y académica en este ámbito.

Otra de las cosas útiles es que me ha permitido aplicar conocimientos desarrollados en asignaturas como *Computación Paralela*, *Fundamentos de Sistemas Operativos*, *Estructura de Sistemas Operativos*, entre otras. También he podido aplicar y ampliar conocimientos de asignaturas como *Diseño de Software* de la rama de Ingeniería de Software, con la introducción de nuevos conceptos como el de la Reingeniería e Ingeniería Inversa.

Bibliografía

[1] Top500. Top 500 main page. [Online]. Available: https://www.top500.org. Último acceso: 01-06-2023.

[2] Grupo Trasgo. Controller. [Online]. Available: https://trasgo.infor.uva.es/controller/ Último acceso: 30-05-2023.

[3] A. Moreton-Fernandez, H. Ortega-Arranz, A. Gonzalez-Escribano, Controllers: An abstraction to ease the use of hardware accelerators, The International Journal of High Performance Computing Applications (IJHPCA) 32 (2018) 838–853. doi: 10.1177/1094342017702962.

[4] G. Rodriguez-Canal, Y. Torres, F. J. Andjar, A. Gonzalez-Escribano, Efficient heterogeneous programming with fpgas using the controller model, The Journal of Supercomputing 77 (2021) 1-16. doi: 10.1007/s11227-021-03792-7.

[5] Y. Torres, Torres, F. J. Andjar, A. Gonzalez-Escribano, D. Llanos, Supporting efficient overlapping of host-device operations for heterogeneous programming with CtrlEvents. Journal of Parallel and Distributed Computing (2023) 179. doi: 10.1016/j.jpdc.2023.04.009.

[6] A. Gonzalez-Escribano, Y. Torres, J. Fresno, D. Llanos, An Extensible System for Multilevel Automatic Data Partition and Mapping, IEEE Transactions on Parallel and Distributed Systems 25 (2014) 1145–1154. doi:10.1109/TPDS.2013.83.

[7] Indeed Editorial Team. (2023, Feb 04). What is Reverse Engineering? With Process Steps and Examples. [Online]. Available: https://www.indeed.com/career-advice/career-development/reverse-engineering. Último acceso: 02-05-2023.

[8] Tema 9: Mantenimiento. class notes for Ingeniería de Proyectos Informáticos, Universitat de València, 2020. http://informatica.uv.es/iiguia/2000/IPI/material/tema7.pdf.

[9] César Represa Pérez, José María Cámara Nebreda, Pedro Luis Sánchez Ortega. Introducción a la programación en CUDA. class notes for Departamento de Ingeniería Electromecánica, Universidad de Burgos, 2016. https://riubu.ubu.es/bitstream/handle/10259/3933/Programacion_en_CUDA.pdf;jsessionid=2CE1E649205CA1D03436FCC34704D56F?sequence=1. Último acceso: 18-05-2023.

[10] Jason Sanders, Edward Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 1st edition, 2010.

[11] AMD ROCm Documentation. Introduction to HIP Installation Guide v5.5. [Online]. Available: https://docs.amd.com/bundle/HIP-Installation-Guide-v5.5/page/Introduction_to_HIP_Installation_Guide_v5.5.html. Último acceso: 18-05-2023.

[12] AMD ROCm Documentation. ROCm Installation Guide v5.4.3. [Online]. Available: https://docs.amd.com/bundle/ROCm-Installation-Guide-v5.4.3/page/Introduction_to_ROCm_Installation_Guide_for_Linux.html. Último acceso: 18-05-2023.

[13] Slurm. About. [Online]. Available: https://slurm.schedmd.com/overview.html. Último acceso: 16-05-2023.

[14] Cmake. About Overview. [Online]. Available: https://cmake.org/overview/. Último acceso: 16-05-2023.

[15] Doxygen. About. [Online]. Available: https://www.doxygen.nl/. Último acceso: 16-05-2023.

[16] gnuplot. Info. [Online]. Available: http://www.gnuplot.info/. Último acceso: 16-05-2023.

[17] OpenMP. Home. [Online]. Available: https://www.openmp.org/. Último acceso: 16-05-2023.

[18] Discord. Discord. [Online]. Available: https://discord.com/. Último acceso: 11-05-2023.

[19] Git. About. [Online]. Available: https://git-scm.com/about. Último acceso: 11-05-2023.

[20] GitLab. GitLab. [Online]. Available: https://git-scm.com/. Último acceso: 11-05-2023.

[21] Astah. About. [Online]. Available: https://astah.net/about/. Último acceso: 11-05-2023.

[22] Overleaf. Overleaf. [Online]. Available: https://es.overleaf.com/. Último acceso: 11-05-2023.

[23] Bob Hughes y Mike Cotterell, Software Project Management, Fifth Edition, McGraw-Hill 2009.

[24] Jobted. ¿Cuánto cobra un Profesor de Universidad en España?. [Online]. Available: https://www.jobted.es/salario/profesor-universidad Último acceso: 10-04-2023.

[25] Glassdoor. Sueldo: Ingeniero de Software junior en España. [Online]. Available: https://www.glassdoor.es/Sueldos/ingeniero-de-software-junior-sueldo-SRCH_KO0,28.htm Último acceso: 10-04-2023.

[26] Kevin Chan. (2019, Sep 17). What are Core-Hours? How are they estimated?. [Online]. Available: https://support.onscale.com/hc/en-us/articles/360013402431-What-are-Core-Hours-How-are-they-estimated- Último acceso: 10-04-2023.

[27] Javier Jiménez. (2022, Feb 11). Esto es lo que consume tu ordenador encendido. [Online]. Available: https://www.redeszone.net/noticias/power/consumo-ordenador-encendido/ Último acceso: 15-03-2023.

[28] Tarifaluzhora. Precio de la luz por horas. [Online]. Available: https://tarifaluzhora.es/?tarifa=pcb&fecha=15%2F03%2F2023 Último acceso: 15-03-2023.

[29] Pouchet L-N, Grauer-Gray S (2015) PolyBench: the polyhedral benchmark suite (2011). [Online]. Available: http://web.cs.ucla.edu/~pouchet/software/polybench/. Último acceso: 15-04-2023.

[30] H. R. Zohouri, High Performance Computing with FPGAs and OpenCL, CoRR, vol. abs/1810.09773, 2018. arXiv: 1810.09773. `http://arxiv.org/abs/1810.09773`.

[31] Zohouri HR, Maruyama N, Smith A, Matsuda M, Matsuoka S (2016) Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs. In: SC'16: Proceedings of the Inter- national Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, pp 409–420

[32] Moore, D. S, Estadística aplicada básica, Second Edition, Antoni Bosch editor 2009.

Apéndice A

Manual de usuario

En este manual se indica los pasos a seguir para montar el proyecto y lanzar un programa de prueba, este manual está realizado para el clúster del grupo de investigación Trasgo.

El primer paso será descargar el código fuente de *Controllers* del repositorio, para ello habrá que lanzar el siguiente comando:

```
1 git clone -b backend/hip
2     https://repo.infor.uva.es/waxa/controllers.git
```

Listing A.1: Descarga *Controllers*

Una vez descargado el código fuente de *Controllers*, hay que descargar el código fuente de la librería auxiliar *Hitmap*, para ello hay que ejecutar los comandos:

```
1 cd ./controllers/extern
2 git clone https://repo.infor.uva.es/trasgo-2018/hitmap.git
```

Listing A.2: Descarga *Hitmap*

El primer paso a realizar antes de comenzar con el compilado del proyecto es cargar el módulo de MPICH:

```
1 #move to frontend dir
2 cwdfront
3
4 # load modules
5 module purge
6 module load mpich/prod/3.4.2/gcc-10.3
```

Listing A.3: Módulo MPICH

El siguiente paso a seguir es compilar el proyecto, en nuestro caso se compilará para las tarjetas gráficas AMD:

```
1 # CMake (3.17 or higher).
2 mkdir -p build
3 cd build
4 cmake -DSUPPORT_HIP=ON ..
```

```
5 make
```

Listing A.4: Compilado *Contollers*

La documentación de la biblioteca *Controllers* puede generarse utilizando Doxygen:

```
1 cd Ctrl/doc
2 doxygen Doxyfile
```

Listing A.5: Documentación Doxygen

Doxygen genera la documentación en formato HTML y latex, en subcarpetas dentro del directorio *Ctrl/doc*, para acceder al formato HTML basta con abrir el fichero *index.html*.

Para ejecutar un ejemplo en local:

```
1 cd ./build/examples/Matrix_Power
2 make
3 ./Matrix_Power_Hip_Ctrl 100 1 0 0 1
```

Listing A.6: Ejemplo lanzamiento Matrix Power en local

Para lanzar un programa a la máquina de pruebas del clúster del grupo Trasgo:

```
1 cd ./build/examples/Matrix_Power
2 make
3
4 #move to frontend dir
5 cwdfront
6
7 srun -w manticore ./Matrix_Power_Hip_Ctrl 100 1 0 0 1
```

Listing A.7: Ejemplo lanzamiento Matrix Power en la máquina manticore del clúster

Apéndice B

Manual de experimentación

En este manual se indica los pasos a seguir para llevar a cabo el estudio experimental para el clúster del grupo de investigación Trasgo.

Para realizar la experimentación se han utilizado scripts proporcionados por el Grupo de Investigación Trasgo [2], disponibles en la ruta *controllers/exp/scripts* del repositorio.

El primer paso a realizar es lanzar el script `run_exp.sh`, script en bash que permite lanzar la experimentación, algo a destacar es que lanza la experimentación de forma independiente para cada arquitectura, es decir, compila y lanza las pruebas para cada una de las arquitecturas seleccionadas. Este script llama a su vez a `compile.sh`, encargado de compilar el proyecto.

Para lanzar las pruebas en segundo plano y evitar que se interrumpa el proceso al cerrar la sesión se ha utilizado `nohup` y `&`.

```
1 nohup bash run_exp.sh -a hip,openclamd &
```

Listing B.1: Lanzamiento experimentación para hip y openclamd

Este script genera los resultados en unos ficheros con extensión .log en una carpeta results. Para calcular las métricas necesarias para la experimentación se utiliza el programa en python `parse_exp.py`.

```
1 python3 parse_exp.py
```

Listing B.2: Cálculo métricas

Posteriormente, para obtener las gráficas se ha utilizado GNUPlot, herramienta que permite generar gráficos a partir de los datos generados por los programas. El script usado para crear las gráficas es `generate_plots.sh`, script que llama para cada una de las experimentaciones a los programas de GNUPlot que generan las gráficas para cada programa de la batería de pruebas.

```
1 bash generate_plots.sh
```

Listing B.3: Generación gráficas

Adicionalmente, se podría utilizar ROCTX, herramienta que permite añadir marcas de tiempo y anotaciones para obtener información sobre el tiempo de ejecución, memoria utilizada, número de hilos y otros datos de rendimiento relevantes del código. Los programas desarrollados se han implementado para poder ejecutar esta herramienta.

```
rocprof --roctx-trace -d outdir -o results.csv
    ./Matrix_Power_Hip_Ctrl 100 1 0 0 1
```

Listing B.4: Ejemplo de uso ROCTX

Apéndice C

Tablas experimentación backend HIP

En este anexo se incluyen todas las tablas con las métricas calculadas en la experimentación para el backend de *Controllers* basado en Hip, frente a las implementaciones nativas en Hip (versión de referencia).

matrix_pow (manticore, hip)					
size	versión	media	mediana	desviación	IC 95 %
1024	Ctrl Hip Sync	0.2225	0.2226	0.0008	[0.2221, 0.2228]
	Ref Sync	0.2516	0.2516	0.0008	[0.2512, 0.2519]
	Ref Async	0.1734	0.1735	0.0011	[0.1730, 0.1738]
	Ctrl Hip Async	0.1742	0.1741	0.0013	[0.1737, 0.1747]
1536	Ctrl Hip Sync	0.5575	0.5573	0.0026	[0.5565, 0.5585]
	Ref Sync	0.5852	0.5858	0.0025	[0.5843, 0.5861]
	Ref Async	0.3954	0.3932	0.0078	[0.3925, 0.3982]
	Ctrl Hip Async	0.3899	0.3899	0.0028	[0.3888, 0.3910]
2048	Ctrl Hip Sync	1.0934	1.0910	0.0110	[1.0893, 1.0974]
	Ref Sync	1.1246	1.1245	0.0024	[1.1237, 1.1255]
	Ref Async	0.7092	0.6999	0.0155	[0.7034, 0.7149]
	Ctrl Hip Async	0.7054	0.7021	0.0120	[0.7009, 0.7098]
4096	Ctrl Hip Sync	6.1863	6.1929	0.0465	[6.1694, 6.2032]
	Ref Sync	7.3948	7.3950	0.0053	[7.3927, 7.3968]
	Ref Async	4.6193	4.6193	0.0047	[4.6176, 4.6211]
	Ctrl Hip Async	3.3179	3.3187	0.0057	[3.3158, 3.3200]
6144	Ctrl Hip Sync	17.9212	17.9037	0.0828	[17.8910, 17.9513]
	Ref Sync	21.7721	21.7763	0.0265	[21.7623, 21.7819]
	Ref Async	15.4830	15.4919	0.0204	[15.4754, 15.4906]
	Ctrl Hip Async	11.2645	11.2803	0.0453	[11.2474, 11.2816]
8192	Ctrl Hip Sync	37.8335	37.8542	0.0968	[37.7982, 37.8687]
	Ref Sync	48.2148	48.2203	0.0341	[48.2019, 48.2276]
	Ref Async	37.1778	37.1753	0.0170	[37.1715, 37.1841]
	Ctrl Hip Async	25.9487	25.9487	0.0031	[25.9475, 25.9499]

Cuadro C.1: Resultados de matrix_pow hip. Tiempos en segundos.

matrix_pow (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ref Async	0.3737	0.3764	0.0123	[0.3693, 0.3782]
	Ctrl Hip Async	0.3577	0.3572	0.0023	[0.3568, 0.3586]
	Ctrl Hip Sync	0.5487	0.5480	0.0026	[0.5477, 0.5497]
	Ref Sync	0.5725	0.5724	0.0011	[0.5720, 0.5729]
40	Ref Async	0.7077	0.7016	0.0163	[0.7016, 0.7137]
	Ctrl Hip Async	0.7049	0.7025	0.0110	[0.7009, 0.7089]
	Ctrl Hip Sync	1.0907	1.0897	0.0046	[1.0889, 1.0925]
	Ref Sync	1.1252	1.1250	0.0032	[1.1240, 1.1263]
60	Ref Async	1.0614	1.0564	0.0255	[1.0521, 1.0707]
	Ctrl Hip Async	1.0440	1.0376	0.0160	[1.0381, 1.0499]
	Ctrl Hip Sync	1.6347	1.6325	0.0087	[1.6315, 1.6380]
	Ref Sync	1.6787	1.6788	0.0038	[1.6773, 1.6801]
80	Ref Async	1.4001	1.3930	0.0273	[1.3900, 1.4102]
	Ctrl Hip Async	1.3894	1.3770	0.0209	[1.3818, 1.3970]
	Ctrl Hip Sync	2.1836	2.1737	0.0227	[2.1754, 2.1919]
	Ref Sync	2.2308	2.2311	0.0049	[2.2290, 2.2326]
100	Ref Async	1.7548	1.7395	0.0406	[1.7400, 1.7696]
	Ctrl Hip Async	1.7197	1.7207	0.0085	[1.7164, 1.7231]
	Ctrl Hip Sync	2.7229	2.7082	0.0281	[2.7127, 2.7331]
	Ref Sync	2.7832	2.7820	0.0070	[2.7807, 2.7857]
120	Ref Async	2.0952	2.0942	0.0216	[2.0872, 2.1032]
	Ctrl Hip Async	2.0598	2.0577	0.0126	[2.0547, 2.0648]
	Ctrl Hip Sync	3.2715	3.2772	0.0325	[3.2596, 3.2833]
	Ref Sync	3.3335	3.3345	0.0068	[3.3310, 3.3359]

Cuadro C.2: Resultados de matrix_pow hip. Tiempos en segundos.

hotspot (manticore, hip)					
size	versión	media	mediana	desviación	IC 95 %
1000	Ref Sync	0.1505	0.1505	0.0005	[0.1504, 0.1507]
	Ref Async	0.0742	0.0740	0.0016	[0.0736, 0.0748]
	Ctrl Hip Sync	0.1489	0.1489	0.0003	[0.1488, 0.1491]
	Ctrl Hip Async	0.0701	0.0700	0.0007	[0.0698, 0.0704]
2500	Ref Sync	1.0208	1.0208	0.0006	[1.0205, 1.0210]
	Ref Async	0.4273	0.4284	0.0109	[0.4233, 0.4313]
	Ctrl Hip Sync	0.9810	0.9806	0.0110	[0.9770, 0.9850]
	Ctrl Hip Async	0.4346	0.4371	0.0149	[0.4292, 0.4401]
5000	Ref Sync	4.2755	4.2748	0.0220	[4.2675, 4.2835]
	Ref Async	2.0916	2.0684	0.0534	[2.0698, 2.1134]
	Ctrl Hip Sync	4.2344	4.2288	0.0229	[4.2261, 4.2428]
	Ctrl Hip Async	2.0762	2.0776	0.0123	[2.0716, 2.0809]
10000	Ref Sync	16.9964	16.9898	0.0134	[16.9915, 17.0014]
	Ref Async	10.7868	11.1342	1.1651	[10.3628, 11.2109]
	Ctrl Hip Sync	17.2317	17.2324	0.0453	[17.2149, 17.2484]
	Ctrl Hip Async	8.5947	8.6527	0.1138	[8.5532, 8.6361]
15000	Ref Sync	39.3125	39.3154	0.0637	[39.2893, 39.3357]
	Ref Async	28.0799	28.3071	0.5806	[27.8568, 28.3031]
	Ctrl Hip Sync	39.4214	39.4098	0.0691	[39.3958, 39.4469]
	Ctrl Hip Async	20.0356	20.0400	0.0446	[20.0188, 20.0524]

Cuadro C.3: Resultados de hotspot hip. Tiempos en segundos.

hotspot (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
100	Ctrl Hip Async	1.9139	1.9269	0.0240	[1.9051, 1.9226]
	Ctrl Hip Sync	3.5801	3.5957	0.0262	[3.5705, 3.5896]
	Ref Sync	3.5821	3.5811	0.0029	[3.5810, 3.5831]
	Ref Async	2.0192	1.8552	0.2481	[1.9289, 2.1095]
200	Ctrl Hip Async	3.5957	3.6119	0.0392	[3.5814, 3.6099]
	Ctrl Hip Sync	6.9654	6.9845	0.0544	[6.9456, 6.9852]
	Ref Sync	6.9342	6.9332	0.0045	[6.9325, 6.9359]
	Ref Async	4.3046	4.5685	0.5666	[4.0984, 4.5108]
300	Ctrl Hip Async	5.2585	5.2997	0.0693	[5.2333, 5.2837]
	Ctrl Hip Sync	10.3832	10.4246	0.0793	[10.3544, 10.4121]
	Ref Sync	10.2871	10.2833	0.0084	[10.2839, 10.2902]
	Ref Async	6.1596	6.2159	0.6970	[5.9059, 6.4132]
400	Ctrl Hip Async	6.9339	6.9832	0.0814	[6.9043, 6.9636]
	Ctrl Hip Sync	13.7948	13.8519	0.1000	[13.7584, 13.8312]
	Ref Sync	13.6396	13.6372	0.0091	[13.6362, 13.6431]
	Ref Async	8.7790	9.1928	0.8686	[8.4629, 9.0952]
500	Ctrl Hip Async	8.6758	8.6787	0.0210	[8.6672, 8.6844]
	Ctrl Hip Sync	17.2155	17.2884	0.1189	[17.1722, 17.2587]
	Ref Sync	16.9901	16.9838	0.0135	[16.9850, 16.9952]
	Ref Async	11.1365	11.2563	0.6472	[10.8828, 11.3901]
600	Ctrl Hip Async	10.2551	10.3399	0.1348	[10.2052, 10.3050]
	Ctrl Hip Sync	20.6327	20.7041	0.1384	[20.5824, 20.6831]
	Ref Sync	20.3473	20.3423	0.0163	[20.3412, 20.3533]
	Ref Async	13.3236	13.8160	0.9385	[12.9760, 13.6712]

Cuadro C.4: Resultados de hotspot hip. Tiempos en segundos.

sobel_yuv (manticore, hip)					
size	versión	media	mediana	desviación	IC 95 %
FTF	Ref Async	0.1842	0.1834	0.0028	[0.1831, 0.1852]
	Ctrl Hip Sync	0.2307	0.2308	0.0007	[0.2305, 0.2310]
	Ref Sync	0.2253	0.2254	0.0006	[0.2251, 0.2255]
	Ctrl Hip Async	0.1993	0.1993	0.0017	[0.1987, 0.1999]
MTF	Ref Async	0.1958	0.1960	0.0013	[0.1953, 0.1962]
	Ctrl Hip Sync	0.2410	0.2410	0.0004	[0.2409, 0.2412]
	Ref Sync	0.2303	0.2300	0.0018	[0.2297, 0.2310]
	Ctrl Hip Async	0.2088	0.2092	0.0019	[0.2081, 0.2095]
MTM	Ref Async	0.1391	0.1389	0.0017	[0.1384, 0.1397]
	Ctrl Hip Sync	0.1883	0.1882	0.0004	[0.1881, 0.1884]
	Ref Sync	0.1722	0.1720	0.0011	[0.1718, 0.1726]
	Ctrl Hip Async	0.1548	0.1545	0.0012	[0.1543, 0.1552]
FTM	Ref Async	0.1249	0.1246	0.0019	[0.1242, 0.1256]
	Ctrl Hip Sync	0.1766	0.1765	0.0004	[0.1765, 0.1768]
	Ref Sync	0.1656	0.1657	0.0007	[0.1653, 0.1658]
	Ctrl Hip Async	0.1438	0.1438	0.0010	[0.1435, 0.1442]

Cuadro C.5: Resultados de sobel_yuv hip. Tiempos en segundos.

sobel_yuv (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Async	0.0414	0.0414	0.0003	[0.0413, 0.0415]
	Ref Sync	0.0387	0.0386	0.0004	[0.0385, 0.0388]
	Ref Async	0.0343	0.0344	0.0007	[0.0341, 0.0346]
	Ctrl Hip Sync	0.0376	0.0376	0.0003	[0.0375, 0.0377]
30	Ctrl Hip Async	0.0599	0.0599	0.0004	[0.0598, 0.0601]
	Ref Sync	0.0544	0.0545	0.0003	[0.0544, 0.0545]
	Ref Async	0.0468	0.0472	0.0011	[0.0463, 0.0472]
	Ctrl Hip Sync	0.0551	0.0551	0.0002	[0.0550, 0.0552]
40	Ctrl Hip Async	0.0795	0.0794	0.0004	[0.0793, 0.0796]
	Ref Sync	0.0711	0.0711	0.0003	[0.0710, 0.0713]
	Ref Async	0.0592	0.0593	0.0009	[0.0588, 0.0595]
	Ctrl Hip Sync	0.0725	0.0724	0.0002	[0.0724, 0.0726]
50	Ctrl Hip Async	0.0986	0.0987	0.0010	[0.0983, 0.0990]
	Ref Sync	0.0878	0.0878	0.0004	[0.0877, 0.0880]
	Ref Async	0.0711	0.0712	0.0008	[0.0708, 0.0714]
	Ctrl Hip Sync	0.0898	0.0898	0.0003	[0.0897, 0.0899]
60	Ctrl Hip Async	0.1184	0.1184	0.0013	[0.1179, 0.1189]
	Ref Sync	0.1048	0.1048	0.0005	[0.1046, 0.1049]
	Ref Async	0.0842	0.0844	0.0014	[0.0837, 0.0847]
	Ctrl Hip Sync	0.1071	0.1072	0.0003	[0.1070, 0.1072]
70	Ctrl Hip Async	0.1386	0.1382	0.0015	[0.1380, 0.1391]
	Ref Sync	0.1213	0.1213	0.0006	[0.1210, 0.1215]
	Ref Async	0.0982	0.0982	0.0009	[0.0978, 0.0985]
	Ctrl Hip Sync	0.1246	0.1246	0.0003	[0.1245, 0.1247]
100	Ctrl Hip Async	0.2007	0.2009	0.0016	[0.2001, 0.2013]
	Ref Sync	0.1715	0.1715	0.0008	[0.1712, 0.1718]
	Ref Async	0.1389	0.1392	0.0009	[0.1385, 0.1392]
	Ctrl Hip Sync	0.1766	0.1767	0.0005	[0.1764, 0.1768]

Cuadro C.6: Resultados de sobel_yuv hip FTF. Tiempos en segundos.

sobel_yuv (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Async	0.0287	0.0287	0.0003	[0.0286, 0.0288]
	Ref Sync	0.0387	0.0386	0.0004	[0.0385, 0.0388]
	Ctrl Hip Sync	0.0376	0.0376	0.0003	[0.0375, 0.0377]
	Ref Async	0.0343	0.0344	0.0007	[0.0341, 0.0346]
30	Ctrl Hip Async	0.0420	0.0419	0.0004	[0.0419, 0.0422]
	Ref Sync	0.0544	0.0545	0.0003	[0.0544, 0.0545]
	Ctrl Hip Sync	0.0551	0.0551	0.0002	[0.0550, 0.0552]
	Ref Async	0.0468	0.0472	0.0011	[0.0463, 0.0472]
40	Ctrl Hip Async	0.0563	0.0563	0.0008	[0.0561, 0.0566]
	Ref Sync	0.0711	0.0711	0.0003	[0.0710, 0.0713]
	Ctrl Hip Sync	0.0725	0.0724	0.0002	[0.0724, 0.0726]
	Ref Async	0.0592	0.0593	0.0009	[0.0588, 0.0595]
50	Ctrl Hip Async	0.0706	0.0705	0.0008	[0.0703, 0.0709]
	Ref Sync	0.0878	0.0878	0.0004	[0.0877, 0.0880]
	Ctrl Hip Sync	0.0898	0.0898	0.0003	[0.0897, 0.0899]
	Ref Async	0.0711	0.0712	0.0008	[0.0708, 0.0714]
60	Ctrl Hip Async	0.0845	0.0847	0.0009	[0.0841, 0.0848]
	Ref Sync	0.1048	0.1048	0.0005	[0.1046, 0.1049]
	Ctrl Hip Sync	0.1071	0.1072	0.0003	[0.1070, 0.1072]
	Ref Async	0.0842	0.0844	0.0014	[0.0837, 0.0847]
70	Ctrl Hip Async	0.0989	0.0989	0.0009	[0.0985, 0.0992]
	Ref Sync	0.1213	0.1213	0.0006	[0.1210, 0.1215]
	Ctrl Hip Sync	0.1246	0.1246	0.0003	[0.1245, 0.1247]
	Ref Async	0.0982	0.0982	0.0009	[0.0978, 0.0985]
100	Ctrl Hip Async	0.1437	0.1437	0.0009	[0.1434, 0.1440]
	Ref Sync	0.1715	0.1715	0.0008	[0.1712, 0.1718]
	Ctrl Hip Sync	0.1766	0.1767	0.0005	[0.1764, 0.1768]
	Ref Async	0.1389	0.1392	0.0009	[0.1385, 0.1392]

Cuadro C.7: Resultados de sobel_yuv hip FTM. Tiempos en segundos.

sobel_yuv (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Async	0.0411	0.0412	0.0002	[0.0411, 0.0412]
	Ref Sync	0.0387	0.0386	0.0004	[0.0385, 0.0388]
	Ctrl Hip Sync	0.0395	0.0395	0.0002	[0.0394, 0.0396]
	Ref Async	0.0343	0.0344	0.0007	[0.0341, 0.0346]
30	Ctrl Hip Async	0.0615	0.0615	0.0003	[0.0614, 0.0616]
	Ref Sync	0.0544	0.0545	0.0003	[0.0544, 0.0545]
	Ctrl Hip Sync	0.0582	0.0581	0.0001	[0.0581, 0.0582]
	Ref Async	0.0468	0.0472	0.0011	[0.0463, 0.0472]
40	Ctrl Hip Async	0.0823	0.0824	0.0005	[0.0821, 0.0825]
	Ref Sync	0.0711	0.0711	0.0003	[0.0710, 0.0713]
	Ctrl Hip Sync	0.0767	0.0767	0.0002	[0.0766, 0.0768]
	Ref Async	0.0592	0.0593	0.0009	[0.0588, 0.0595]
50	Ctrl Hip Async	0.1034	0.1031	0.0008	[0.1031, 0.1037]
	Ref Sync	0.0878	0.0878	0.0004	[0.0877, 0.0880]
	Ctrl Hip Sync	0.0953	0.0953	0.0003	[0.0952, 0.0954]
	Ref Async	0.0711	0.0712	0.0008	[0.0708, 0.0714]
60	Ctrl Hip Async	0.1245	0.1245	0.0009	[0.1241, 0.1248]
	Ref Sync	0.1048	0.1048	0.0005	[0.1046, 0.1049]
	Ctrl Hip Sync	0.1139	0.1140	0.0004	[0.1138, 0.1141]
	Ref Async	0.0842	0.0844	0.0014	[0.0837, 0.0847]
70	Ctrl Hip Async	0.1458	0.1460	0.0013	[0.1453, 0.1462]
	Ref Sync	0.1213	0.1213	0.0006	[0.1210, 0.1215]
	Ctrl Hip Sync	0.1326	0.1326	0.0004	[0.1324, 0.1327]
	Ref Async	0.0982	0.0982	0.0009	[0.0978, 0.0985]
100	Ctrl Hip Async	0.2123	0.2125	0.0015	[0.2118, 0.2129]
	Ref Sync	0.1715	0.1715	0.0008	[0.1712, 0.1718]
	Ctrl Hip Sync	0.1884	0.1885	0.0003	[0.1883, 0.1885]
	Ref Async	0.1389	0.1392	0.0009	[0.1385, 0.1392]

Cuadro C.8: Resultados de sobel_yuv hip MTF. Tiempos en segundos.

sobel_yuv (manticore, hip)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Async	0.0302	0.0302	0.0004	[0.0300, 0.0303]
	Ctrl Hip Sync	0.0395	0.0395	0.0002	[0.0394, 0.0396]
	Ref Async	0.0343	0.0344	0.0007	[0.0341, 0.0346]
	Ref Sync	0.0387	0.0386	0.0004	[0.0385, 0.0388]
30	Ctrl Hip Async	0.0454	0.0455	0.0006	[0.0452, 0.0456]
	Ctrl Hip Sync	0.0582	0.0581	0.0001	[0.0581, 0.0582]
	Ref Async	0.0468	0.0472	0.0011	[0.0463, 0.0472]
	Ref Sync	0.0544	0.0545	0.0003	[0.0544, 0.0545]
40	Ctrl Hip Async	0.0604	0.0602	0.0008	[0.0601, 0.0607]
	Ctrl Hip Sync	0.0767	0.0767	0.0002	[0.0766, 0.0768]
	Ref Async	0.0592	0.0593	0.0009	[0.0588, 0.0595]
	Ref Sync	0.0711	0.0711	0.0003	[0.0710, 0.0713]
50	Ctrl Hip Async	0.0761	0.0762	0.0009	[0.0757, 0.0764]
	Ctrl Hip Sync	0.0953	0.0953	0.0003	[0.0952, 0.0954]
	Ref Async	0.0711	0.0712	0.0008	[0.0708, 0.0714]
	Ref Sync	0.0878	0.0878	0.0004	[0.0877, 0.0880]
60	Ctrl Hip Async	0.0916	0.0914	0.0008	[0.0913, 0.0919]
	Ctrl Hip Sync	0.1139	0.1140	0.0004	[0.1138, 0.1141]
	Ref Async	0.0842	0.0844	0.0014	[0.0837, 0.0847]
	Ref Sync	0.1048	0.1048	0.0005	[0.1046, 0.1049]
70	Ctrl Hip Async	0.1064	0.1064	0.0011	[0.1060, 0.1068]
	Ctrl Hip Sync	0.1326	0.1326	0.0004	[0.1324, 0.1327]
	Ref Async	0.0982	0.0982	0.0009	[0.0978, 0.0985]
	Ref Sync	0.1213	0.1213	0.0006	[0.1210, 0.1215]
100	Ctrl Hip Async	0.1549	0.1550	0.0013	[0.1544, 0.1553]
	Ctrl Hip Sync	0.1884	0.1885	0.0003	[0.1883, 0.1885]
	Ref Async	0.1389	0.1392	0.0009	[0.1385, 0.1392]
	Ref Sync	0.1715	0.1715	0.0008	[0.1712, 0.1718]

Cuadro C.9: Resultados de sobel_yuv hip MTM. Tiempos en segundos.

Apéndice D

Tablas experimentación backend Openclamd

En este anexo se incluyen todas las tablas con las métricas calculadas en la experimentación para el backend de *Controllers* basado en Hip, frente al backend de *Controllers* basado en OpenCL.

matrix_pow (manticore, openclamd)					
size	versión	media	mediana	desviación	IC 95 %
1024	Ctrl Hip Sync	0.2225	0.2226	0.0008	[0.2221, 0.2228]
	Ctrl Openclamd Sync	0.3494	0.3478	0.0240	[0.3407, 0.3582]
	Ctrl Openclamd Async	0.1587	0.1582	0.0017	[0.1581, 0.1593]
	Ctrl Hip Async	0.1742	0.1741	0.0013	[0.1737, 0.1747]
1536	Ctrl Hip Sync	0.5575	0.5573	0.0026	[0.5565, 0.5585]
	Ctrl Openclamd Sync	0.9190	0.9138	0.0339	[0.9066, 0.9313]
	Ctrl Openclamd Async	0.3532	0.3525	0.0043	[0.3516, 0.3548]
	Ctrl Hip Async	0.3899	0.3899	0.0028	[0.3888, 0.3910]
2048	Ctrl Hip Sync	1.0934	1.0910	0.0110	[1.0893, 1.0974]
	Ctrl Openclamd Sync	1.6485	1.6433	0.0414	[1.6335, 1.6636]
	Ctrl Openclamd Async	0.6362	0.6351	0.0085	[0.6331, 0.6393]
	Ctrl Hip Async	0.7054	0.7021	0.0120	[0.7009, 0.7098]
4096	Ctrl Hip Sync	6.1863	6.1929	0.0465	[6.1694, 6.2032]
	Ctrl Openclamd Sync	7.0202	7.0211	0.0179	[7.0133, 7.0271]
	Ctrl Openclamd Async	3.5840	3.5845	0.0119	[3.5796, 3.5884]
	Ctrl Hip Async	3.3179	3.3187	0.0057	[3.3158, 3.3200]
6144	Ctrl Hip Sync	17.9212	17.9037	0.0828	[17.8910, 17.9513]
	Ctrl Openclamd Sync	18.7215	18.7157	0.0337	[18.7086, 18.7345]
	Ctrl Openclamd Async	11.9083	11.9083	0.0102	[11.9045, 11.9122]
	Ctrl Hip Async	11.2645	11.2803	0.0453	[11.2474, 11.2816]
8192	Ctrl Hip Sync	37.8335	37.8542	0.0968	[37.7982, 37.8687]
	Ctrl Openclamd Sync	39.5574	39.5589	0.0420	[39.5412, 39.5735]
	Ctrl Openclamd Async	28.1653	28.1594	0.0296	[28.1539, 28.1767]
	Ctrl Hip Async	25.9487	25.9487	0.0031	[25.9475, 25.9499]

Cuadro D.1: Resultados de matrix_pow Hip y openclamd. Tiempos en segundos.

matrix_pow (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Sync	0.5487	0.5480	0.0026	[0.5477, 0.5497]
	Ctrl Openclamd Async	0.3375	0.3370	0.0071	[0.3349, 0.3401]
	Ctrl Openclamd Sync	0.8405	0.8412	0.0414	[0.8254, 0.8555]
	Ctrl Hip Async	0.3577	0.3572	0.0023	[0.3568, 0.3586]
40	Ctrl Hip Sync	1.0907	1.0897	0.0046	[1.0889, 1.0925]
	Ctrl Openclamd Async	0.6327	0.6329	0.0083	[0.6296, 0.6357]
	Ctrl Openclamd Sync	1.6430	1.6440	0.0477	[1.6253, 1.6606]
	Ctrl Hip Async	0.7049	0.7025	0.0110	[0.7009, 0.7089]
60	Ctrl Hip Sync	1.6347	1.6325	0.0087	[1.6315, 1.6380]
	Ctrl Openclamd Async	0.9314	0.9270	0.0124	[0.9268, 0.9359]
	Ctrl Openclamd Sync	2.4762	2.4869	0.0522	[2.4572, 2.4952]
	Ctrl Hip Async	1.0440	1.0376	0.0160	[1.0381, 1.0499]
80	Ctrl Hip Sync	2.1836	2.1737	0.0227	[2.1754, 2.1919]
	Ctrl Openclamd Async	1.2213	1.2193	0.0102	[1.2173, 1.2253]
	Ctrl Openclamd Sync	3.2866	3.3008	0.0548	[3.2663, 3.3069]
	Ctrl Hip Async	1.3894	1.3770	0.0209	[1.3818, 1.3970]
100	Ctrl Hip Sync	2.7229	2.7082	0.0281	[2.7127, 2.7331]
	Ctrl Openclamd Async	1.5252	1.5196	0.0187	[1.5184, 1.5320]
	Ctrl Openclamd Sync	4.0692	4.0664	0.0718	[4.0430, 4.0953]
	Ctrl Hip Async	1.7197	1.7207	0.0085	[1.7164, 1.7231]
120	Ctrl Hip Sync	3.2715	3.2772	0.0325	[3.2596, 3.2833]
	Ctrl Openclamd Async	1.8117	1.8087	0.0126	[1.8070, 1.8165]
	Ctrl Openclamd Sync	4.8937	4.8912	0.0853	[4.8621, 4.9253]
	Ctrl Hip Async	2.0598	2.0577	0.0126	[2.0547, 2.0648]

Cuadro D.2: Resultados de matrix_pow Hip y openclamd. Tiempos en segundos.

hotspot (manticore, openclamd)					
size	versión	media	mediana	desviación	IC 95 %
1000	Ctrl Hip Async	0.0701	0.0700	0.0007	[0.0698, 0.0704]
	Ctrl Hip Sync	0.1489	0.1489	0.0003	[0.1488, 0.1491]
	Ctrl Openclamd Async	0.0706	0.0706	0.0004	[0.0704, 0.0707]
	Ctrl Openclamd Sync	0.3475	0.3473	0.0167	[0.3414, 0.3535]
2500	Ctrl Hip Async	0.4346	0.4371	0.0149	[0.4292, 0.4401]
	Ctrl Hip Sync	0.9810	0.9806	0.0110	[0.9770, 0.9850]
	Ctrl Openclamd Async	0.5417	0.5420	0.0049	[0.5399, 0.5435]
	Ctrl Openclamd Sync	1.6103	1.6080	0.0139	[1.6053, 1.6154]
5000	Ctrl Hip Async	2.0762	2.0776	0.0123	[2.0716, 2.0809]
	Ctrl Hip Sync	4.2344	4.2288	0.0229	[4.2261, 4.2428]
	Ctrl Openclamd Async	2.4335	2.4306	0.0171	[2.4272, 2.4398]
	Ctrl Openclamd Sync	6.0550	6.0547	0.0169	[6.0488, 6.0613]
10000	Ctrl Hip Async	8.5947	8.6527	0.1138	[8.5532, 8.6361]
	Ctrl Hip Sync	17.2317	17.2324	0.0453	[17.2149, 17.2484]
	Ctrl Openclamd Async	9.9103	9.9070	0.0486	[9.8920, 9.9287]
	Ctrl Openclamd Sync	23.7885	23.7861	0.0629	[23.7648, 23.8123]
15000	Ctrl Hip Async	20.0356	20.0400	0.0446	[20.0188, 20.0524]
	Ctrl Hip Sync	39.4214	39.4098	0.0691	[39.3958, 39.4469]
	Ctrl Openclamd Async	21.5052	21.4950	0.0801	[21.4761, 21.5344]
	Ctrl Openclamd Sync	52.9733	52.9775	0.1472	[52.9187, 53.0278]

Cuadro D.3: Resultados de hotspot Hip y openclamd. Tiempos en segundos.

hotspot (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
100	Ctrl Openclamd Sync	4.9829	4.9833	0.0152	[4.9773, 4.9885]
	Ctrl Hip Async	1.9139	1.9269	0.0240	[1.9051, 1.9226]
	Ctrl Hip Sync	3.5801	3.5957	0.0262	[3.5705, 3.5896]
	Ctrl Openclamd Async	2.2990	2.3028	0.0305	[2.2879, 2.3101]
200	Ctrl Openclamd Sync	9.6862	9.6919	0.0289	[9.6753, 9.6971]
	Ctrl Hip Async	3.5957	3.6119	0.0392	[3.5814, 3.6099]
	Ctrl Hip Sync	6.9654	6.9845	0.0544	[6.9456, 6.9852]
	Ctrl Openclamd Async	4.1955	4.1989	0.0299	[4.1842, 4.2068]
300	Ctrl Openclamd Sync	14.3643	14.3768	0.0739	[14.3374, 14.3912]
	Ctrl Hip Async	5.2585	5.2997	0.0693	[5.2333, 5.2837]
	Ctrl Hip Sync	10.3832	10.4246	0.0793	[10.3544, 10.4121]
	Ctrl Openclamd Async	6.0972	6.1001	0.0331	[6.0845, 6.1100]
400	Ctrl Openclamd Sync	19.0959	19.0917	0.0533	[19.0758, 19.1160]
	Ctrl Hip Async	6.9339	6.9832	0.0814	[6.9043, 6.9636]
	Ctrl Hip Sync	13.7948	13.8519	0.1000	[13.7584, 13.8312]
	Ctrl Openclamd Async	7.9955	7.9960	0.0336	[7.9823, 8.0086]
500	Ctrl Openclamd Sync	23.7750	23.7884	0.1205	[23.7312, 23.8189]
	Ctrl Hip Async	8.6758	8.6787	0.0210	[8.6672, 8.6844]
	Ctrl Hip Sync	17.2155	17.2884	0.1189	[17.1722, 17.2587]
	Ctrl Openclamd Async	9.9001	9.8935	0.0479	[9.8817, 9.9185]
600	Ctrl Openclamd Sync	28.5008	28.5188	0.0984	[28.4644, 28.5373]
	Ctrl Hip Async	10.2551	10.3399	0.1348	[10.2052, 10.3050]
	Ctrl Hip Sync	20.6327	20.7041	0.1384	[20.5824, 20.6831]
	Ctrl Openclamd Async	11.7728	11.7704	0.0555	[11.7511, 11.7946]

Cuadro D.4: Resultados de hotspot Hip y openclamd. Tiempos en segundos.

sobel_yuv (manticore, openclamd)					
size	versión	media	mediana	desviación	IC 95 %
FTF	Ctrl Hip Sync	0.2307	0.2308	0.0007	[0.2305, 0.2310]
	Ctrl Openclamd Sync	0.2223	0.2210	0.0028	[0.2213, 0.2233]
	Ctrl Hip Async	0.1993	0.1993	0.0017	[0.1987, 0.1999]
	Ctrl Openclamd Async	0.1568	0.1568	0.0006	[0.1566, 0.1570]
MTF	Ctrl Hip Sync	0.2410	0.2410	0.0004	[0.2409, 0.2412]
	Ctrl Openclamd Sync	0.2374	0.2377	0.0024	[0.2365, 0.2383]
	Ctrl Hip Async	0.2088	0.2092	0.0019	[0.2081, 0.2095]
	Ctrl Openclamd Async	0.1744	0.1744	0.0006	[0.1741, 0.1746]
MTM	Ctrl Hip Sync	0.1883	0.1882	0.0004	[0.1881, 0.1884]
	Ctrl Openclamd Sync	0.3164	0.3163	0.0012	[0.3159, 0.3168]
	Ctrl Hip Async	0.1548	0.1545	0.0012	[0.1543, 0.1552]
	Ctrl Openclamd Async	0.2526	0.2525	0.0006	[0.2524, 0.2529]
FTM	Ctrl Hip Sync	0.1766	0.1765	0.0004	[0.1765, 0.1768]
	Ctrl Openclamd Sync	0.3015	0.3015	0.0011	[0.3011, 0.3019]
	Ctrl Hip Async	0.1438	0.1438	0.0010	[0.1435, 0.1442]
	Ctrl Openclamd Async	0.2346	0.2346	0.0005	[0.2344, 0.2348]

Cuadro D.5: Resultados de sobel_yuv Hip y openclamd. Tiempos en segundos.

sobel_yuv (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Async	0.0414	0.0414	0.0003	[0.0413, 0.0415]
	Ctrl Openclamd Sync	0.0482	0.0481	0.0006	[0.0480, 0.0484]
	Ctrl Hip Sync	0.0509	0.0510	0.0004	[0.0508, 0.0511]
	Ctrl Openclamd Async	0.0349	0.0349	0.0002	[0.0348, 0.0350]
30	Ctrl Hip Async	0.0599	0.0599	0.0004	[0.0598, 0.0601]
	Ctrl Openclamd Sync	0.0699	0.0699	0.0007	[0.0696, 0.0702]
	Ctrl Hip Sync	0.0729	0.0728	0.0002	[0.0728, 0.0730]
	Ctrl Openclamd Async	0.0500	0.0500	0.0005	[0.0498, 0.0502]
40	Ctrl Hip Async	0.0795	0.0794	0.0004	[0.0793, 0.0796]
	Ctrl Openclamd Sync	0.0911	0.0909	0.0011	[0.0907, 0.0915]
	Ctrl Hip Sync	0.0957	0.0957	0.0005	[0.0955, 0.0959]
	Ctrl Openclamd Async	0.0647	0.0647	0.0004	[0.0646, 0.0649]
50	Ctrl Hip Async	0.0986	0.0987	0.0010	[0.0983, 0.0990]
	Ctrl Openclamd Sync	0.1134	0.1134	0.0013	[0.1129, 0.1139]
	Ctrl Hip Sync	0.1185	0.1185	0.0004	[0.1183, 0.1186]
	Ctrl Openclamd Async	0.0802	0.0802	0.0005	[0.0801, 0.0804]
60	Ctrl Hip Async	0.1184	0.1184	0.0013	[0.1179, 0.1189]
	Ctrl Openclamd Sync	0.1351	0.1349	0.0015	[0.1346, 0.1357]
	Ctrl Hip Sync	0.1412	0.1412	0.0004	[0.1411, 0.1414]
	Ctrl Openclamd Async	0.0960	0.0960	0.0004	[0.0959, 0.0961]
70	Ctrl Hip Async	0.1386	0.1382	0.0015	[0.1380, 0.1391]
	Ctrl Openclamd Sync	0.1572	0.1569	0.0016	[0.1566, 0.1578]
	Ctrl Hip Sync	0.1641	0.1641	0.0003	[0.1640, 0.1643]
	Ctrl Openclamd Async	0.1113	0.1115	0.0006	[0.1111, 0.1115]
100	Ctrl Hip Async	0.2007	0.2009	0.0016	[0.2001, 0.2013]
	Ctrl Openclamd Sync	0.2246	0.2237	0.0024	[0.2237, 0.2254]
	Ctrl Hip Sync	0.2332	0.2332	0.0007	[0.2329, 0.2335]
	Ctrl Openclamd Async	0.1589	0.1589	0.0009	[0.1586, 0.1593]

Cuadro D.6: Resultados de sobel_yuv FTF Hip y openclamd. Tiempos en segundos.

sobel_yuv (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Hip Sync	0.0376	0.0376	0.0003	[0.0375, 0.0377]
	Ctrl Hip Async	0.0287	0.0287	0.0003	[0.0286, 0.0288]
	Ctrl Openclamd Sync	0.0628	0.0629	0.0005	[0.0626, 0.0630]
	Ctrl Openclamd Async	0.0494	0.0494	0.0003	[0.0492, 0.0495]
30	Ctrl Hip Sync	0.0551	0.0551	0.0002	[0.0550, 0.0552]
	Ctrl Hip Async	0.0420	0.0419	0.0004	[0.0419, 0.0422]
	Ctrl Openclamd Sync	0.0930	0.0929	0.0004	[0.0928, 0.0931]
	Ctrl Openclamd Async	0.0729	0.0728	0.0002	[0.0728, 0.0729]
40	Ctrl Hip Sync	0.0725	0.0724	0.0002	[0.0724, 0.0726]
	Ctrl Hip Async	0.0563	0.0563	0.0008	[0.0561, 0.0566]
	Ctrl Openclamd Sync	0.1230	0.1231	0.0004	[0.1229, 0.1232]
	Ctrl Openclamd Async	0.0962	0.0961	0.0003	[0.0960, 0.0963]
50	Ctrl Hip Sync	0.0898	0.0898	0.0003	[0.0897, 0.0899]
	Ctrl Hip Async	0.0706	0.0705	0.0008	[0.0703, 0.0709]
	Ctrl Openclamd Sync	0.1532	0.1531	0.0004	[0.1530, 0.1533]
	Ctrl Openclamd Async	0.1198	0.1198	0.0003	[0.1197, 0.1199]
60	Ctrl Hip Sync	0.1071	0.1072	0.0003	[0.1070, 0.1072]
	Ctrl Hip Async	0.0845	0.0847	0.0009	[0.0841, 0.0848]
	Ctrl Openclamd Sync	0.1831	0.1832	0.0006	[0.1829, 0.1833]
	Ctrl Openclamd Async	0.1429	0.1431	0.0005	[0.1427, 0.1431]
70	Ctrl Hip Sync	0.1246	0.1246	0.0003	[0.1245, 0.1247]
	Ctrl Hip Async	0.0989	0.0989	0.0009	[0.0985, 0.0992]
	Ctrl Openclamd Sync	0.2126	0.2126	0.0003	[0.2125, 0.2127]
	Ctrl Openclamd Async	0.1664	0.1664	0.0005	[0.1662, 0.1666]
100	Ctrl Hip Sync	0.1766	0.1767	0.0005	[0.1764, 0.1768]
	Ctrl Hip Async	0.1437	0.1437	0.0009	[0.1434, 0.1440]
	Ctrl Openclamd Sync	0.3029	0.3027	0.0008	[0.3026, 0.3032]
	Ctrl Openclamd Async	0.2367	0.2367	0.0006	[0.2365, 0.2369]

Cuadro D.7: Resultados de sobel_yuv FTM Hip y openclamd. Tiempos en segundos.

sobel_yuv (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Openclamd Sync	0.0486	0.0485	0.0004	[0.0485, 0.0487]
	Ctrl Openclamd Async	0.0361	0.0361	0.0001	[0.0361, 0.0362]
	Ctrl Hip Async	0.0411	0.0412	0.0002	[0.0411, 0.0412]
	Ctrl Hip Sync	0.0506	0.0506	0.0002	[0.0505, 0.0507]
30	Ctrl Openclamd Sync	0.0718	0.0719	0.0008	[0.0715, 0.0721]
	Ctrl Openclamd Async	0.0532	0.0532	0.0003	[0.0531, 0.0533]
	Ctrl Hip Async	0.0615	0.0615	0.0003	[0.0614, 0.0616]
	Ctrl Hip Sync	0.0746	0.0746	0.0003	[0.0745, 0.0747]
40	Ctrl Openclamd Sync	0.0961	0.0965	0.0010	[0.0958, 0.0965]
	Ctrl Openclamd Async	0.0707	0.0706	0.0004	[0.0705, 0.0708]
	Ctrl Hip Async	0.0823	0.0824	0.0005	[0.0821, 0.0825]
	Ctrl Hip Sync	0.0985	0.0984	0.0005	[0.0983, 0.0987]
50	Ctrl Openclamd Sync	0.1205	0.1203	0.0012	[0.1201, 0.1210]
	Ctrl Openclamd Async	0.0888	0.0889	0.0004	[0.0887, 0.0890]
	Ctrl Hip Async	0.1034	0.1031	0.0008	[0.1031, 0.1037]
	Ctrl Hip Sync	0.1226	0.1227	0.0004	[0.1225, 0.1228]
60	Ctrl Openclamd Sync	0.1448	0.1451	0.0015	[0.1442, 0.1453]
	Ctrl Openclamd Async	0.1060	0.1062	0.0008	[0.1057, 0.1063]
	Ctrl Hip Async	0.1245	0.1245	0.0009	[0.1241, 0.1248]
	Ctrl Hip Sync	0.1471	0.1471	0.0007	[0.1469, 0.1474]
70	Ctrl Openclamd Sync	0.1676	0.1679	0.0021	[0.1668, 0.1683]
	Ctrl Openclamd Async	0.1232	0.1231	0.0007	[0.1229, 0.1235]
	Ctrl Hip Async	0.1458	0.1460	0.0013	[0.1453, 0.1462]
	Ctrl Hip Sync	0.1714	0.1714	0.0005	[0.1712, 0.1716]
100	Ctrl Openclamd Sync	0.2414	0.2415	0.0022	[0.2406, 0.2422]
	Ctrl Openclamd Async	0.1782	0.1781	0.0006	[0.1780, 0.1784]
	Ctrl Hip Async	0.2123	0.2125	0.0015	[0.2118, 0.2129]
	Ctrl Hip Sync	0.2445	0.2446	0.0010	[0.2442, 0.2449]

Cuadro D.8: Resultados de sobel_yuv MTF Hip y openclamd. Tiempos en segundos.

sobel_yuv (manticore, openclamd)					
iter	versión	media	mediana	desviación	IC 95 %
20	Ctrl Openclamd Sync	0.0644	0.0644	0.0002	[0.0643, 0.0645]
	Ctrl Hip Sync	0.0395	0.0395	0.0002	[0.0394, 0.0396]
	Ctrl Openclamd Async	0.0518	0.0518	0.0003	[0.0516, 0.0519]
	Ctrl Hip Async	0.0302	0.0302	0.0004	[0.0300, 0.0303]
30	Ctrl Openclamd Sync	0.0958	0.0957	0.0004	[0.0956, 0.0960]
	Ctrl Hip Sync	0.0582	0.0581	0.0001	[0.0581, 0.0582]
	Ctrl Openclamd Async	0.0767	0.0767	0.0003	[0.0766, 0.0768]
	Ctrl Hip Async	0.0454	0.0455	0.0006	[0.0452, 0.0456]
40	Ctrl Openclamd Sync	0.1277	0.1276	0.0005	[0.1275, 0.1279]
	Ctrl Hip Sync	0.0767	0.0767	0.0002	[0.0766, 0.0768]
	Ctrl Openclamd Async	0.1022	0.1022	0.0004	[0.1020, 0.1023]
	Ctrl Hip Async	0.0604	0.0602	0.0008	[0.0601, 0.0607]
50	Ctrl Openclamd Sync	0.1594	0.1595	0.0007	[0.1592, 0.1597]
	Ctrl Hip Sync	0.0953	0.0953	0.0003	[0.0952, 0.0954]
	Ctrl Openclamd Async	0.1275	0.1275	0.0004	[0.1273, 0.1276]
	Ctrl Hip Async	0.0761	0.0762	0.0009	[0.0757, 0.0764]
60	Ctrl Openclamd Sync	0.1905	0.1904	0.0005	[0.1903, 0.1907]
	Ctrl Hip Sync	0.1139	0.1140	0.0004	[0.1138, 0.1141]
	Ctrl Openclamd Async	0.1526	0.1525	0.0006	[0.1524, 0.1528]
	Ctrl Hip Async	0.0916	0.0914	0.0008	[0.0913, 0.0919]
70	Ctrl Openclamd Sync	0.2220	0.2217	0.0011	[0.2216, 0.2224]
	Ctrl Hip Sync	0.1326	0.1326	0.0004	[0.1324, 0.1327]
	Ctrl Openclamd Async	0.1777	0.1777	0.0005	[0.1775, 0.1779]
	Ctrl Hip Async	0.1064	0.1064	0.0011	[0.1060, 0.1068]
100	Ctrl Openclamd Sync	0.3169	0.3167	0.0014	[0.3164, 0.3174]
	Ctrl Hip Sync	0.1884	0.1885	0.0003	[0.1883, 0.1885]
	Ctrl Openclamd Async	0.2535	0.2536	0.0007	[0.2532, 0.2537]
	Ctrl Hip Async	0.1549	0.1550	0.0013	[0.1544, 0.1553]

Cuadro D.9: Resultados de sobel_yuv MTM Hip y openclamd. Tiempos en segundos.

Apéndice E

Contenidos del CD-ROM

El CD entregado cuenta con los siguientes directorios:

- Controllers: Directorio se encuentra el código fuente de *Controllers* así como el código de la librería auxiliar *Hitmap* (en la carpeta *extern*).
- Batería de pruebas: Directorio que contiene los ficheros fuente desarrollados para las pruebas y experimentación de este proyecto.
- Scripts: Directorio con los scripts empleados en la experimentación.
- Results: Directorio con los resultados de la experimentación, con los siguientes subdirectorios:
  - raw: Con los ficheros .log con los resultados obtenidos de la ejecución de la experimentación.
  - stats: Con los ficheros .csv con las métricas calculadas a partir de los resultados.
  - plots: Con todas las gráficas obtenidas tras la experimentación.
- Memoria: Esta misma memoria en formato PDF.
- Diagramas: Los diagramas del proyecto en formato ASTA.