Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

OpenDwarfs 2024: Refactorización de la benchmark suite OpenDwarfs

Realizado por **Juan José Ropero Cerro**

✠✠✠

Universidad de Valladolid

20 de septiembre de 2024

Tutor: Diego Rafael Llanos Ferraris

Tutor: Manuel de Castro Caballero

Resumen

La benchmark suite Open Dwarfs, desarrollada originalmente para la publicación *"OpenCL and the 13 Dwarfs: A Work in Progress"* en 2012, y continuada su desarrollo para la publicación *"On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms"* de 2015, se trata de una colección de diversos benchmarks, llamados dwarfs, desarrollados con el propósito de evaluar las capacidades de computación paralela de sistemas heterogéneos, empleando el estándar abierto de computación paralela, *OpenCL*. La colección de benchmarks presentes en este benchmark suite, se basa en la publicación *The Landscape of Parallel Computing Research: A View from Berkeley*. El propósito de este TFM ha consistido en llevar a cabo la refactorización de esta benchmark suite, de modo que este lista y disponible para ser empleada por cualquier usuario interesado, preservando el espíritu original con el que originalmente fue desarrollada esta benchmark suite.

Descriptores

OpenDwarfs, Sistemas paralelos, Computación heterogénea, Open CL, Benchmarking, GPU.

Índice general

Índice de figuras

Índice de tablas

1: Introducción

OpenDwarfs es una benchmark suite, o colección de benchmarks, compuesta originalmente por un total de 13 benchmarks, conocidos como los "13 Dwarfs de Berkeley" [1] de la publicación *"The Landscape of Parallel Computing Research: A View from Berkeley"* [2]. La primera instancia de esta suite se desarrolló con el uso del estándar de *OpenCL*. Esta se describe en el documento *"OpenCL and the 13 Dwarfs: A Work in Progress"*. La finalidad de la benchmark suite es determinar las capacidades de cómputo de un sistema, empleando un método que sea agnóstico al hardware que compone el mismo [3]. La versión más actual de este software fue desarrollada para los artículos *"On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms"* [4] y *"OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures"* [5], en donde explican que debido a la proliferación de plataformas heterogéneas para la computación, surgen nuevos restos para evaluar las capacidades de ejecución paralela de estas plataformas. En estos artículos, los autores proponen emplear la nueva versión de *OpenDwarfs* desarrollada para llevar a cabo estas evaluaciones de distintos sistemas de arquitecturas paralelas. En la última versión de esta benchmark suite aparece una nueva benchmark, aumentando el listado a un total de 14 benchmarks en la suite, con los que evaluar las capacidades de ejecución paralela de un sistema.

Tras las publicaciones y, con el paso del tiempo, la suite pasó a ser de código abierto (*Open Source*), para que cualquiera que los desee pueda contribuir a su desarrollo [6]. En los últimos 8 años, la mayoría de los programas del repositorio han dejado de ser mantenidos, a excepción de lo relativo a publicaciones hechas sobre el repositorio en lo que se refiere a cambios menores en la misma. También, revisando el historial de cambios del repositorio de esta benchmark suite, podemos ver cómo ha habido una gran diversidad de personas que han tratado de continuar con este proyecto. En la mayoría de casos solo se hicieron cambios menores sobre un *"fork"* (copias del repositorio sobre las que poder hacer cambios propios) de la última versión del repositorio en su momento, que después los desarrolladores de los mismos abandonaron.

Aunque actualmente es posible tomar esta benchmark suite y usarla para evaluar sistemas de cómputo, debido al avance tanto en aspectos software como hardware de los sistemas, así como a la escasa documentación que los autores proveen para cada uno de

los benchmarks presentes en la benchmark suite, puede resultar un proceso complicado. Además, el historial de "commits" del repositorio en donde se ha encontrado la benchmark suite original, así como los distintos "Issue Threads" del repositorio, parecen indicar que los desarrolladores de la benchmark suite han dejado de proveer soporte y mantenimiento a la misma.

Es por ello que, en este proyecto, se ha propuesto llevar a cabo un proceso de refactorización de la benchmark suite. Esta tarea consistirá en analizar el estado original en el que se encuentra esta benchmark suite, reparar los diversos defectos encontrados en cada uno de los benchmark, los cuales impidan o dificulten el uso de la misma. También deberemos de asegurarnos que, para cada benchmark incluido en *OpenDwarfs*, provea una salida, la cual sea razonable para la tarea asociada a cada uno de estos. Estas salidas deberán poder ser generadas en cada benchmark de forma opcional, con el propósito de tener un conjunto de datos que podamos emplear para verificar su correcto funcionamiento entre distintas plataformas en las que estos se ejecuten. Adicionalmente, realizaremos diversas tareas con el fin de proveer uniformidad de uso a la benchmark suite, y una mejor experiencia de usuario. Tras todo este proceso de refactorización, haremos una experimentación con cada uno de los benchmarks, de modo que se pueda hacer una comparación de los tiempos que obtengamos, en distintas plataformas de ejecución de *OpenCL*. Analizaremos las variaciones de tiempos que obtenemos de cada uno de estos benchmarks, dadas ciertas entradas. Para finalizar, discutimos las conclusiones del proyecto y presentamos el trabajo futuro derivado del mismo.

2: Estructura de la suite

En este próximo capítulo de nuestro proyecto, nos encargaremos de proporcionar una descripción, lo más completa posible, con el fin de mejor comprender el origen de esta benchmark suite y el propósito con el que OpenDwarfs fue desarrollada. Con el objetivo de brindar una mejor comprensión de todo este capítulo, hemos decidido dividir el contenido a explicar, en tres secciones diferentes, las cuales, pasaremos a describir brevemente.

- **Descripción general y filosofía**. En esta primera sección del capítulo, recopilamos todo lo que hemos podido hallar y comprender en lo relacionado con la benchmark suite de *OpenDwarfs*. Se proporcionará una descripción genérica de la misma, así como la filosofía que los autores tuvieron en mente, durante el proceso de desarrollo de *OpenDwarfs*. También incluiremos en esta sección cualquier otro descubrimiento adicional, asociados a estos aspectos a tratar.

- **Prerrequisitos de compilación** . Adjuntamos en esta sección, para proporcionar una breve y detallada guía de toda aquella herramienta que ha de instalarse en una máquina, para poder emplear de la colección de benchmarks de *OpenDwarfs*. Estos paquetes y herramientas son fundamentales no solo para la compilación de la benchmark suite, sino además, ser necesarios para el correcto funcionamiento

- **Proceso de compilación**. Seguido de los prerrequisitos de compilación, incluimos en esta sección una guía, lo más completa posible, del proceso de compilación de la benchmark suite en su totalidad. Esta guía se hizo a partir de la versión original de *OpenDwarfs*, con la cual comenzamos nuestro proyecto, pero, como no se alteró el código de este proceso para la versión de la benchmark suite a entregar, la guía también será útil para esta nueva version desarrollada.+

- **Descripción de los benchmarks**. En esta última sección del capítulo, presentamos una descripción de cada uno de los benchmarks, enfocándonos principalmente en los algoritmos o principios científicos, en los que los autores se fundamentaron, para el desarrollo de cada uno de los benchmarks de OpenDwarfs. También, se aborda el origen de cada benchmark presente en*OpenDwarfs*, los autores que colaboraron en el

desarrollo de cada uno de estos, y, en caso de ser necesario, las diferentes formas en las que se puede llevar a cabo ejecuciones de un benchmark. Para ello debimos hacer una búsqueda, no solo sobre la documentación existente en la benchmark suite, sino también, sobre las distintas publicaciones y otras fuentes, asociadas a cada uno de estos benchmarks.

Dada esta introducción de la totalidad de este capítulo del documento, pasaremos a desarrollar, más en profundidad, el contenido de cada sección, comenzando, con la descripción general de la benchmark suite y la filosofía que los autores emplearon, para el desarrollo de OpenDwarfs.

2.1. Descripción general y filosofía

El propósito original de esta benchmark suite, así como de las versiones actuales de cada uno de los benchmarks, radica en poder ser un punto de referencia para la computación heterogénea, a través del cual, se pueda evaluar la capacidad de paralelización de un sistema. Esto se llevó a cabo mediante la adaptación de los 13 Dwarfs de Berkeley, descritos en la publicación previamente mencionada [2], en la cual se definen cada uno de estos *dwarfs* (benchmarks de *OpenDwarfs*), como una serie de métodos algorítmicos que capturan patrones de comunicación y computación. Cuando hablamos de computación heterogénea, nos referimos a que esta benchmark suite se desarrolló, de modo que cada uno de los benchmarks que contiene, se puedan ejecutar en diversidad de dispositivos de computación, cuya lógica de funcionamiento y configuración de uso, son muy distintas entre ellos. Según comentan los autores en las documentaciones de la benchmark suite, así como en las distintas publicaciones, para la que *OpenDwarfs* fue desarrollada, estos benchmarks deben de poder ejecutarse en procesadores del estado del arte (CPUs multicore), tarjetas gráficas (GPUs), sistemas FPGA (Field Programmable Gate-Array) de la compañía Altera, y sistemas de arquitectura MIC de la compañía Intel (Many Integrated Core architecture).

Continuando a discutir la creación de los benchmarks de OpenDwarfs, para todos y cada uno de ellos se hizo uso de *OpenCL*. Desarrollado y mantenido, principalmente, por el grupo de Khronos *(Khronos Group)* [7], *OpenCL* es un estándar de programación abierta que permiten a los usuarios desarrolladores, la implementación de programas paralelos para su ejecución en sistemas heterogéneos (que cuenten con aceleradores hardware). Para ello, el desarrollador ha de crear uno o varios ficheros, conocidos como *Kernel*, los cuales se emplean en tiempo de ejecución por el programa que los llama, que son ejecutados en el acelerador hardware seleccionado. Este estándar presenta un lenguaje de programación similar a los lenguajes de programación de C/C++, con la particularidad de tener a su disposición, un conjunto de APIs *(Application Programming Interface)*, las cuales ayudan al programador a configurar las ejecuciones de las funciones de estos ficheros. Es gracias a este estándar abierto que muchos programadores, han podido diseñar programas capaces de aprovechar las capacidades de un sistema paralelo heterogéneo [8]. El grupo de Khronos ha desarrollado una diversidad de estándares abiertos para el público, a lo

largo de los años, desarrollados con el fin de realizar otro tipo de tareas particulares. Entre estos otros estándares, desarrollados por Khronos, podemos destacar WebGL, Vulkan, y SYCL [9] [10] [11].

En la ejecución de un *kernel* de *OpenCL* el trabajo se divide en *Work-Items*. Un *Work-Item* equivale a un hilo de ejecución (Thread) que ejecuta parte del algoritmo en paralelo a otros *Work-Items*. Los *Work-Items* se agrupan en *Work-Groups*. El tamaño de los *Work-Groups* se denomina *LocalSize* y el número total de *Work-Items* se denomina *GlobalSize*. Este modelo sigue principalmente una filosofía de programación y ejecución paralela en datos (data-parallel). En principio, OpenCL emplea todos los recursos hardware (ej. hilos hardware) disponibles en los aceleradores de forma eficiente. Sin embargo, la cantidad de recursos empleados en paralelo realmente y su gestión la abstrae el *runtime* de OpenCL y es opaca para el usuario.

Se ha de señalar que varios de estos benchmarks se tratan, según terminología informática y documentación de los autores, "ports" de versiones ya existentes en otras benchmark suites, con propósitos y filosofías similares. Al analizar los ficheros de cada benchmark de *OpenDwarfs*, que los autores categorizaron como "port", descubrimos el diverso origen de cada uno de estos. Diversos de los benchmarks provienen de una benchmark suite que existió previa a *OpenDwarfs*, siendo esta la benchmark suite de *Rodinia*. De acuerdo con las distintas fuentes relacionadas con esta benchmark suite [12] [13] [14], *Rodinia* se trataba de otra colección de benchmarks para la computación heterogénea. Los autores de esta benchmark suite, usaron como fundamento en el desarrollo de su benchmark suite, múltiples de los benchmarks descritos en la publicación *"The Landscape of Parallel Computing Research A View from Berkeley"* [2], misma publicación que los desarrolladores de *OpenDwarfs*, emplearon como base para la creación de su benchmark suite. Tras investigar estas publicaciones, los autores nunca dejaron, en ninguna de ellas, un enlace, a través del cual, pudiese descargarse esta benchmark suite. No obstante, pudimos encontrar un repositorio de GitHub, el cual contiene la última versión de la benchmark suite de *Rodina* [15]. Analizando la documentación asociada a este proyecto de GitHub, encontramos un enlace de una página web antigua, la cual ya no es posible acceder a ella, debido a la falta de soporte de los administradores de esta. Mediante el uso de la página de Web Archive [16] para acceder a una "snapshot" de esta página web, vemos que nos lleva a una antigua página de la Universidad de Virginia. Por otra parte, la benchmark suite de *OpenDwarfs*, fue desarrollada por Virginia Tech [3] [17], otra universidad, localizada en el estado de Virginia. En esta "snapshot" hemos podido encontrar un enlace, el cual aún parece funcionar, y descargar esta benchmark suite. Al compararlos contenidos descargados, y los existentes en el repositorio de GitHub, mencionado anteriormente [15], podemos asegurar de que el proyecto subido a esta plataforma, es la última versión del proyecto de Rodinia.

2.2. Prerrequisitos de compilación

A continuación, detallaremos los distintos prerrequisitos que hemos de cumplir, para poder, posteriormente, compilar la benchmark suite, y ejecutar cada uno de los benchmarks de la misma. Al igual que el proceso de compilación, que detallaremos posteriormente, estos requisitos se basan en aquellos dados, por los desarrolladores originales, en la documentación existentes en el repositorio original de *OpenDwarfs* [6], así como cualquier otro descubrimiento hecho durante la realización de este proceso. Así pues, los prerrequisitos a cumplir son los siguientes:

- **Tener instalado el compilador GCC/G++**. Según la documentación existente de OpenDwarfs, los autores nunca llegaron a especificar una versión particular de este compilador. Se ha optado por emplear las versiones de GCC/G++ 11.0, para nuestro proyecto, la cual es instalada por defecto con múltiples distros modernos de Linux. Uno de los propósitos de la refactorización consiste en asegurar que los distintos benchmarks presentes, sean compatibles con esta versión de GCC/G++, ya que, aunque no sea la versión más moderna del compilador, existente hasta la fecha, esta es una versión más madura que las otras versiones modernas del compilador, así como que los programas desarrollados en GCC/G++ 11 son aún compatibles con actuales y futuras versiones de este compilador. A fin de asistirnos en el desarrollo, y facilitar la identificación de errores, se ha utilizado, de manera ocasional, una versión anterior del compilador, para mejor determinar los errores que podían causar ciertos benchmarks, siendo estas la versión 7.0 de GCC/G++.

- **Instalar una versión de *OpenCL***. Como ya hemos mencionado previamente, *OpenCL* es un estándar de programación paralela, compatible con una amplia gama de dispositivos informáticos. Existen diversas versiones de este estándar, las cuales todas deberían de ser compatibles entre ellas y con la empleada, por los autores, durante el desarrollo de la benchmark suite. A pesar de que los autores no especificaron de qué distribuidor deberíamos descargar este software, sí que se especifica la versión de *OpenCL* a usar, siendo la 1.0. En nuestro caso, con el fin de trabajar en nuestra máquina local, hemos utilizado la variante de *OpenCL* distribuida por la compañía de Intel, a través de *oneApi*, mientras que para las máquinas en donde llevaremos a cabo nuestra experimentación, hemos empleado la versión de *OpenCL* presente en la librería de *CUDA*, de la compañía *Nvidia*.

- **Instalar la extensión de *autoconf***. Los autores optaron por la versión 2.63, o cualquiera que fuese superior a esta. Este paquete es uno de los múltiples paquetes de GNU/Linux, que permiten la creación de scripts de ejecuciones bash de forma automática. Estas configuraciones se llevan a cabo mediante el empleo de plantillas, las cuales tienen parámetros y configuraciones particulares para una amplia variedad de distros de GNU/Linux. Por lo encontrado, esta viene instalada por defecto con la mayoría de estas variantes de GNU/Linux.

- **Tener instalado una versión de la extensión de *autoheader***. Esta es una de las múltiples extensiones presentes en el paquete de *autoconf*, las cuales se instalan de forma automática, y asiste en la generación automática de los scripts del paquete. Este paquete permite la creación de plantillas de cabecera, para programas en lenguaje C [18].

- **Instalar el paquete *libtool***. Esta es una de las diversas variantes de paquetes similares, los cuales se aseguran de garantizar una correcta portabilidad de programas a otros sistemas operativos existentes. A través de este tipo de paquete, los programadores pueden fácilmente enlazar librerías existentes en un sistema, para un programa que las necesite, sin necesidad de conocer la ruta de directorios donde estas se almacenan.

- **Instalar una versión de la extensión de *automake***. Este es un paquete fundamental, que requerimos instalar para la compilación de los benchmarks. El paquete de *automake* se dedica a la generación automática del script *Makefile.in* del proyecto en el que se emplee. Para un uso adecuado de este paquete, se requiere tener instalada una versión de *autoconf* y *make*, en nuestro sistema.

- **Tener instalado una versión de la extensión de *maker***. A través de una exploración en el registro de paquetes y software que han existido y existe para GUN/Linux [19], se llegó a la conclusión de que, esta extensión en particular, nunca ha sido concebida y que probablemente los autores se referían a la extensión o paquete de *make*, comúnmente empleada para facilitar la compilación de programas, u otras similares. Esto tiene más sentido, ya que este es necesario para el uso del paquete de *automake*, mencionado anteriormente. Al instalar esta extensión vemos que no surgen ningún conflicto con otros, previamente instalados, y que podemos llevar a cabo el proceso de compilación de los benchmarks sin inconveniente alguno.

Adicionalmente, para un mayor fácil uso de los benchmarks, nosotros recomendamos del empleo adicional del paquete de *clinfo*. Este nos ayudará a ver las distintas versiones de *OpenCL*, que hemos instalado en el sistema y que podemos emplear para facilitarnos el uso de estos benchmarks. El paquete nos ayudará a identificar los identificadores de plataformas y dispositivos de ejecución de *OpenCL*, para posteriores ejecuciones de los benchmarks en estos.

2.3. Proceso de compilación

Una vez cumplidos todos los prerrequisitos, mencionadas anteriormente, continuaremos detallando el proceso de compilación de la benchmark suite, asumiendo que también la hemos descargado del repositorio *GitHub* donde se almacena [6], mencionado anteriormente. Afortunadamente, las instrucciones proporcionadas por los autores, en lo que respecta a la compilación de los benchmarks, son sencillas y fáciles de comprender, por lo que no debería haber muchos, y debería de ser relativamente sencillo de seguir y replicar en cualquier otro

sistema operativo. Los distintos pasos, que han de seguirse para compilar cada benchmarks presente en la suite de *OpenDwarfs*, de forma ordenada, son los siguientes:

1. Ejecutar el archivo binario *autogen.sh*, presente en la raíz del repositorio. Este generará automáticamente las cabeceras necesarias, así como otros ficheros indispensables, para el proceso de compilación de los benchmarks. También se ocupará de llevar a cabo configuraciones adicionales que se requieran o sean necesarias, para un correcto uso de la benchmark suite de *OpenDwarfs*. Este, por lo general, realiza las distintas configuraciones adicionales y generaciones de ficheros necesarios, siempre y cuando hayamos instalado los previos paquetes mencionados.

2. A continuación, nos trasladamos a la carpeta *build* del repositorio, para poder ejecutar el binario de *configure*, el cual ha debido de generarse previamente en el directorio raíz de la benchmark suite. En caso de no existir esta carpeta en el repositorio, podemos generarla manualmente con el comando *mkdir <nombre_de_carpeta>*. Recomendamos que esta carpeta crear se localice en el raíz del repositorio de *OpenDwarfs*.

3. Después de desplazarnos a la carpeta, donde generar los ficheros de los benchmarks, deberemos ejecutar el binario de *configure*. Este fichero ejecutable generará los diversos ficheros y archivos necesarios para generar los ejecutables de los benchmarks, en el directorio desde el cual lo ejecutemos, en nuestro caso el directorio*build*. Estos se tratan de cabeceras necesarias para cada benchmark, así como un fichero *Makefile*, con el que generar los ejecutables finales de cada benchmark de *OpenDwarfs*. Abriendo una terminal de comando en esta carpeta, ejecutaremos el binario con la línea de *../configure*, en un distro de GNU/Linux. Si existiese algún error, o falta de un paquete necesario para la compilación, el proceso se detendrá por completo, sin generar el fichero *Makefile*, para realizar la compilación final. Este ejecutable ofrece una variedad de opciones que se pueden emplear, tales como elegir qué ejecutables de los benchmarks específicos, se desean generar en el proceso de compilación final, empleando el siguiente comando: *–with-apps=<benchmark>,<benchmark>,...*. Asimismo, si por algún motivo no se ha reconocido el directorio en el que se encuentra la instalación de *OpenCL*, el programa permite especificar la ruta del SDK (System Development Kit) de *OpenCL*, con la entrada de *–with-opencl-sdk=<ruta>*. Existen una amplia variedad de opciones que los desarrolladores han proporcionado para usar este binario, aunque un listado claro y explicado de estas no se encuentra en ninguna parte de la documentación de la benchmark suite.

4. Finalmente, deberemos ejecutar el comando "make", empleando el *Makefile* generado con el binario de *configure* del paso anterior, para así poder generar los ejecutables de los benchmarks que hayamos indicado. Adicionalmente, se generará un fichero kernel de *OpenCL* en la compilación, el cual contiene todas y cada una de las funciones adicionales de los benchmarks que se encontraban en sus carpetas particulares, y que son fundamentales para el funcionamiento de estos. Dado que, en la última

versión de esta benchmark suite, presentaba diversos problemas de compilación en ciertos benchmarks, el haber generado el fichero *Makefile*, sin haberle especificado que benchmarks en particular se desean compilar, resultará en una versión del fichero que tratará de generar todos los ejecutables de la benchmark suite. A causa de los errores que hemos mencionado, el proceso de compilación se verá detenido y no generará los binarios de cada benchmark de forma correcta. No obstante, si se emplea la opción de *–with-apps* del ejecutable de *configure*, que mencionamos anteriormente, y seleccionamos benchmarks que sabemos pueden compilarse correctamente, la posterior ejecución del *Makefile* que obtendremos, si resultará en todos los ficheros y binarios necesarios para los benchmarks especificados.

Como ya se ha mencionado previamente, solo podemos generar los ejecutables de los benchmarks que aún funcionan, empleando la opción *–with-apps* del binario de *configure*. A través de esta opción, incluida por los desarrolladores, hemos podido ver el estado de cada benchmark de la suite y analizar los problemas que presentaban, aquellos benchmarks que no eran posible ser compilados, en el estado original en el que se encontró la benchmark suite. Comentaremos cada uno de estos fallos, así como las soluciones empleadas, y su razonamiento, en el próximo capítulo del documento.

2.4. Descripción de los benchmarks

Para acabar con este capítulo, haremos una breve descripción de cada uno de los benchmarks presentes en *OpenDwarfs*. De acuerdo con lo expuesto previamente, los autores de la benchmark suite fundamentan los diversos benchmarks creados en los "dwarfs" descritos en la publicación *" The Landscape of Parallel Computing Research: A View from Berkeley"* [2]. A través de un análisis minucioso de la benchmark suite, las publicaciones relacionadas con la benchmark suite y diversas fuentes halladas, comentaremos el fundamento de cada uno de los benchmarks, según nuestros descubrimientos. Hemos de recordar que los *dwarfs* se trataban de una serie de métodos algorítmicos creados con el propósito de capturar patrones de comunicación y computación.

Cabe señalar que, cada uno de estos benchmarks, ya fueron clasificados previamente por los autores de la suite, en la publicación "OpenCL and the 13 Dwarfs: A Work in Progress" [3]. En este documento haremos un breve comentario sobre las categorías, en cada uno de los benchmarks. Aconsejamos que se revisen las publicaciones hechas por los autores originales, relacionados con la benchmark suite de *OpenDwarfs*.

- ***BFS.*** **Dwarf *Graph Traversal***. Este benchmark se fundamenta en el algoritmo *Breathed First Search* o, como también se conoce, el algoritmo de *búsqueda por anchura*. La categoría de *Graph Traversal*, engloba los algoritmos cuyo propósito es recorre y evaluar los objetos de un grafo. El propósito del benchmark, y del algoritmo en el que se fundamenta, consiste en realizar el recorrido de un árbol particular, comenzando por el nodo raíz y, analizando en cada capa o nivel del árbol, los nodos

relacionados directamente con el nodo raíz. es uno de los algoritmos más conocidos y que resulta más fácil de implementar de modo secuencial. El benchmark genera, como resultado de su ejecución, un fichero en donde se listan el coste de cada uno de los nodos del árbol con el nodo raíz. Según los comentarios en el código fuente del benchmark, este solo funciona con árboles cuya distancia entre nodos es de coste uno. *BFS* es uno de los múltiples benchmarks que originalmente se desarrollaron para la benchmark de *Rodinia* [15]. Los autores solo realizaron un "port" de la versión de *OpenCL* existente en la benchmark suite de Rodinia, adaptándolo para su correcto uso en la benchmark suite de *OpenDwarfs*. Investigando la fuente de este benchmark, hallamos que este fue desarrollado por Pawan Harish, y que el algoritmo del benchmark se basa en uno que el autor describió en la publicación *"Accelerating Large Graph Algorithms on the GPU using CUDA"* junto con P. J. Narayanan [20]

- ***BWA_HMM.*** **Dwarf *Graphical Models***. Este benchmark se fundamenta en el algoritmo de *Baum-Welch* o, como también se conoce, el algoritmo de *maximización de expectaciones*. La categoría *Graphical Models*, se fundamentaban organizar modelos de modo que las variables fuesen nodos y las probabilidades condicionales fuesen conexiones entre los nodos. El propósito de este benchmark, consiste en hallar la mejor configuración de los parámetros para un *modelo oculto de Márkov(HMM)*, siendo estos parámetros, una matriz de transiciones de estados (A), una matriz de emisión (B) y el estado inicial de la distribución (π_0). El usuario puede proporcionar como entradas a este benchmark, una de las 3 configuraciones posibles y una cantidad de sus variaciones, seguidas del comando *-v* y el tipo de variación con las que trabajar con el modelo. Ejemplos del uso del benchmark, para cada una de estas entradas se mostrarán a continuación.

  - **Variaciones de estado**, donde el número de símbolos y observaciones del modelo son fijos con valores de 2 y 1000, respectivamente. (*.\bwa_hmm -n 20 -v n*)
  - **Variaciones de símbolos**, donde el número de estados y observaciones del modelo son fijos con valores de 60 y 1000, respectivamente. (*.\bwa_hmm -s 5 -v s*)
  - **Variaciones de observación**, donde el número de estados y símbolos del modelo son fijos con valores de 60 y 2, respectivamente. (*.\bwa_hmm -t 100 -v t*)

  Tras extensa experimentación del benchmark, y analizando el código del mismo, se llega a la conclusión de que, aunque el usuario le introduzca como entradas múltiples de las variaciones posibles, el benchmark solo aceptara una de ellas como variantes, manteniendo las demás entradas con los valores fijos, especificados previamente. Como ejemplo de esta explicación, observemos la siguiente entrada del benchmark:

  .\bwa_hmm -n 200 -s 10 -t 500 -v n

Según esta entrada, el usuario está especificando un modelo, con 200 variaciones de estado, 10 variaciones de símbolos y 500 variaciones de observaciones, pero, como únicamente se especifica al final, solo las variaciones de estado *-v n*, las otras 2 entradas se ignorarán. Esto se debe a que en el código, las 3 ejecuciones son exclusivas, de modo que, aunque se especificasen más a usar en la entrada -v, esto resultaría en una no ejecución del benchmark. En cuanto al origen o autoría de este benchmark, no ha podido hallarse durante la investigación de los ficheros y documentaciones, relacionadas con este benchmark.

- ***CFD.*** **Dwarf** ***Unstructured Grids*** . Este benchmark se fundamenta en la ciencia de la "Computación Dinámica de Fluidos" *(Computacional Fluids Dynamics)*. Este tipo de computaciones se distinguen por su intensidad, en cuanto a recursos de ordenadores. Según documentan los autores, la categoría de *Unstructured Grids* trabaja con estructuras de datos, como por ejemplo, una lista de punteros que contiene localizaciones, y que similar a dwarfs de *Sparse Linear Algebra*, realizar actualizaciones requieren de uso de varios niveles de memoria. El benchmark de *CFD* tiene como objetivo hallar soluciones a volúmenes tridimensionales de las ecuaciones de Euler, para la comprensión de la dinámica de fluidos. El benchmark requiere de un archivo sobre el que realizar las distintas computaciones matemáticas. El origen de estos archivos utilizados en el benchmark, no fue documentado por los autores, por lo que desconocen con certeza el programa empleado para la generación de estos. Como dato final del benchmark, este es un benchmark suite de *Rodinia* [12], que también fue un "port" para la benchmark de *OpenDwarfs*. Se investigó los contenidos del proyecto de Rodinia, en busca de más información sobre las entradas, pero lo único que pudimos hallar fue la publicación para la que originalmente fue creada este benchmark. La publicación original a la que pertenece este benchmark es*"Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware"*, por Andrew Corrigan, Fernando Camelli, Rainald Lohner and John Wallin [21]. Asimismo, encontramos en la benchmark suite de Rodinia, que la versión de *OpenCL* de este benchmark, fue adaptada de la versión original, por Jianbin Fang. No pudimos hallar información alguna, sobre los ficheros de entrada, empleados por este benchmark en sus ejecuciones.

- ***CRC.*****Dwarf** ***Combinational Logic*** . El algoritmo en el que se fundamenta este benchmark es uno muy conocido y utilizado en la transmisión de mensajes, siendo este el algoritmo de *Verificación Redundancia Cíclica (Cyclic Redundancy Check)*. La categoría de se creó para benchmarks que aprovechasen las capacidades de paralelismo a nivel de bit para procesar tareas con altos volúmenes de datos, de la forma más rápida posible. El algoritmo *CRC* consiste en añadir a un mensaje que se envíe un campo adicional, denominado campo *CRC*, en el cual se indica el número de bits del mensaje a transmitir. Si el receptor, al recibir el mensaje, determina que el número de bits del campo *CRC* y el del mensaje no coinciden, descarta dicho mensaje y se le vuelve a solicitar al emisor que repita el envío de dicho mensaje. En el caso de este benchmark, se le pasa un fichero particular que puede crearse con su

generador de ficheros, provisto por los desarrolladores de *OpenDwarfs*. La estructura de estos ficheros es una serie de N páginas, con M bytes por cada una de las páginas. Para cada página del archivo creado, se calcula el número de bits que contiene y se imprime este número calculado en formato hexadecimal, cuando el usuario le solicita verificar las operaciones del benchmark. La versión de este benchmark, se basa en la implementación de otro algoritmo adicional llamado *Slice-by-8*, creado originalmente por desarrolladores de la compañía Intel. De acuerdo con un comentario del código del benchmark *CRC*, los autores del benchmark tomaron una versión de este algoritmo, publicada por Stephan Brumme en su página web, y le aplicaron mínimas modificaciones, para su uso en este benchmark [22].

- ***CSR.*** **Dwarf *Sparse Linear Algebra***. La categoría de *Sparse Linear Algebra* es para benchmarks que, similares a los de *Dense Linear Algebra*, buscan solucionar problemas lineales de matemáticas, pero donde se emplean matrice sparse con pocos valores no nulos. El benchmark *CSR* se fundamenta en el algoritmo de *Compressed Sparse Row*, el cual posibilita el almacenamiento de una matriz, de manera eficiente, cuya la mayoría de sus celdas son celdas nulas o a valor 0. Posteriormente, esta se estructura en un conjunto de 3 vectores que almacenan la fila, la columna y el valor de las celdas no nulas de la matriz original. Para obtener entradas a poder emplear, este benchmark presenta un generador de ficheros, similar al del benchmark *CRC*, el cual genera un fichero de N matrices cuadradas de tamaño M, cada una de ellas. Como se ha mencionado previamente, esta técnica se enfoca en disminuir el uso de memoria, en donde la gran parte de las celdas de las matrices están vacías *(sparse matrix)*. En el fichero *README.md* asociado a este benchmark, proveen los nombres de los autores originales de los ficheros fuentes de este benchmark, siendo sus autores Nathan Bell y Michael Garland de la compañía Nvidia. Posteriormente, Joseph Banks lo adaptó para *OpenCL* y Tyler Kenney, de la compañía IBM, se encargó de adaptarlo para mejorar su uso en sistemas FPGA.

- ***FFT.*** **Dwarf *Spectral Methods***. Este benchmark se basa en el algoritmo de transformación de Furrier (Fast Furrier Transform). *Spectral Methods*, categoría a la que pertenece este benchmark, se fundamenta en transformaciones de datos a dominós temporales o espaciales. Este tipo de ejecuciones se caracteriza por diversas etapas de procesamiento, donde las dependencias en una etapa forman patrones mariposa de computación. Las mediciones de *Fast Fourier Transform* son importantes métodos de medición en las ciencias del sonido y acústicas, para tolerancia a fallos, control de calidad y monitorización de sistemas [23]. Los autores pretendían capturar estos patrones con el benchmark *FFT*. Según los autores, este benchmark es un "port" de la versión original, creada para la SHOC suite [24] [25]. Por lo que pudo hallarse, esta benchmark suite se diseñó con el fin de evaluar las capacidades y la estabilidad de sistemas con arquitecturas no tradicionales, dedicadas a computaciones de propósito general. Buscando en los archivos de este benchmark, descubrimos comentario donde indican que este fue programado, principalmente por Collin McCurdy, y mínimamente modificado por Jeremy Meredith, para añadir

tiempos de uso del bus PCIe del sistema donde se ejecute. El benchmark *FFT* está elaborado en *OpenCL*, para ser usado con sistemas CPU y GPU, y este debería poder ser usado en transformaciones unidimensionales o bidimensionales, según se especifique en las entradas de este benchmark.

- ***GEM.*** **Dwarf *N-body Methods***. La agrupación de *N-body Methods* se compone de benchmarks, destinados a calcular interacciones entre múltiples puntos discretos. Estos puntos son caracterizados por un gran número de cálculos independientes, dado un paso del tiempo, seguido de una comunicación entre estos puntos. El benchmark de *GEM* se fundamenta en un paquete concebido para ser un programa "sandbox" con el objetivo de llevar a cabo experimentaciones de visualización de biomoléculas y computaciones de potencial electrostático. Dicho paquete es *Open Source* y disponible para la mayoría de plataformas científicas, según se especifica en la documentación de este benchmark. La versión de *GEM*, creada para la benchmark suite de *OpenDwarfs*, es una variante que no incorpora una interfaz gráfica. Se incluyen, junto con el benchmark, diversos ejemplos, los cuales deberían variar entre varios segundos de ejecución a una duración máxima de media hora, según documentaciones de los autores. A pesar de investigar por internet, en busca de más información de este paquete original o el benchmark desarrollado, no se ha logrado obtener más información, relevante para este proyecto.

- **Dwarf *Dense Linear Algebra* con instanciación *KMEANS***. Este es uno de los benchmarks que forman parte de la categoría de *Dense Linear Algebra*, categoría creada para benchmarks que realizan tareas matemáticas con una gran carga de trabajo.El benchmark de *KMEANS*, se basa en uno de los algoritmos básicos de agrupación de objetos por características (features) y comúnmente usados para casos de clasificación simples. *Dense Linear Algebra* es la categoría de benchmarks destinada a operaciones con matrices, donde existen una alta interdependencia de datos entre hilos de ejecución y una elevada carga de operaciones matemáticas. El benchmark primero establece, de manera aleatoria, los centroides de las agrupaciones de datos y, tras cada iteración, este relocaliza los centroides, con el fin de agrupar aquellos objetos con características similares. En la presente implementación de *KMEANS*, el usuario es capaz de indicarle el número de iteraciones máximas para recalcular la posición de cada centroide, el número de centroides máximos y mínimos a usar, entre otras más. Para hacer uso de este benchmark, el usuario debe proporcionar un fichero que contiene N cantidad de objetos, donde todos y cada uno de ellos ha de tener una cantidad M de características, *(features)*. Por lo que se ha podido analizar de estas entradas, estas características han de ser entradas numéricas, las cuales pueden ser decimales o números enteros. Este benchmark es un "port" de la versión del mismo benchmark que fue originalmente desarrollado para la benchmark suite de *Rodinia* [12]. En comentario del código, podemos encontrar los distintos autores que estuvieron trabajando en el desarrollo del benchmark, siendo así Weikeng Liao el autor original del benchmark, con ayuda de Jay Pisharath, ambos de la Universidad del Noroeste (Northwestern University), universidad privada que

se localiza en Evanston, Illinois [26]. Adicionalmente, encontramos que los autores Shuai Che, David Tarjan, Sang-Ha Lee, de la Universidad de Virginia, realizaron posteriores modificaciones a este benchmark

- **Dwarf *Dense Linear Algebra* con instanciación *LUD***. Este es otro de los benchmarks que forman parte de la categoría de *Dense Linear Algebra*, categoría creada para benchmarks que realizan tareas matemáticas con una gran carga de trabajo. El benchmark se fundamenta en el algoritmo de *Crout* y *Doolittle* o, como mejor se conoce, el algoritmo de *descomposición LU* [27]. Este algoritmo se emplea hoy en día en muchas de las diversas operaciones para el cálculo de matrices inversas, resolución de sistemas de ecuaciones lineales, etc. El benchmark de *LUD* toma una matriz cuadrada de números flotantes y genera una matriz, denominada internamente como matriz LU, la cual es una combinación de las matrices *Lower* y *Upper* que se calculan. Una vez obtenida esta matriz, se podrá emplear para llevar a cabo un procedimiento con el cual se puede verificar si, las matrices *Lower* y *Upper* generadas, producen la matriz original introducida en la ejecución del benchmark. Este, al igual que varios de los otros benchmarks de *OpenDwarfs*, es un "port" de una versión de este mismo benchmark, existente en la benchmark suite de *Rodinia* [15]. Investigando dentro de esta benchmark suite, los ficheros asociados a este benchmark, tanto para la versión de *CUDA* y *OpenCL*, presentes en Rodinia, hemos podido hallar que este fue desarrollado por Liang Wang. Podemos encontrar que este ha realizado diversas publicaciones [28], durante su estancia en la escuela de Virginia, y que actualmente traba como ingeniero software, en la compañía de META [29].

- **Dwarf *Branch and Bound* con instanciación *NQUEENS***. La categoría de *Branch and Bound* se creó para benchmarks que realizasen una búsqueda sobre un gran espacio, con el fin de hallar una opción global óptima. El benchmark de *NQUEENS* se fundamenta en uno de los desafíos más conocidos y complejos de la historia, basados en el juego del ajedrez, en el cual, mediante un tablero cuadrado de tamaño N, con N reinas, se ha de hallar todas y cada una de las posiciones en donde poder posicionar estas N reinas, sin que ninguna de ellas se encuentre en una posición que permitiese eliminar cualquiera de las otras reinas vecinas, en un solo movimiento. El origen de este benchmark, así como su autoría, procede de un foro de una página web, *Beyond3D* [30], que actualmente nos impide acceso a este foro en cuestión [31]. Como aún puede accederse a esta página web y a distintos foros existentes en esta web, pero no al foro particular que queremos, asumimos que este ha podido ser eliminado o archivado, explicando así la inaccesibilidad a este de cara al público. Si utilizamos la página de Web Archive [16], podemos encontrar una "snapshot" del foro en cuestión en donde se publicó la implementación actual, o una versión anterior del mismo, presente en *OpenDwarfs*. Como se puede apreciar, por comentarios en este foro, la versión de *OpenCL* del algoritmo, fue el resultado de hacer una conversión de una versión anterior hecha con la librería de *CUDA*. Se desconoce con claridad el nombre del autor del programa, ya que el foro únicamente nos proporciona el apodo o alias que el autor emplea en su página web, siendo este

pcchen, así como el lugar donde indica que vive, siendo este Taiwán. Por lo que puede verse, este sigue estando aún activo en el foro [32].

- **Dwarf *Dynamic Programming* con instanciación *NW/NEEDLE***. La categoría de *Dynamic programming* engloba a benchmarks que, para solucionar un problema de alta complejidad, lo consiguen a través de hallar la soluciones a problemas menores más simples. El benchmark de *NW* se basa en el algoritmo de Needleman-Wunsch, el cual permite calcular el alineamiento óptimo de 2 secuencias de ADN. Estas secuencias de ADN se generan y almacenan en matrices bidimensionales de dimensiones NxM, donde las puntuaciones de las celdas se estiman desde las celdas más al noroeste, hasta las celdas más al sureste de cada una de las matrices. Una vez calculadas estas puntuaciones, se hace una regresión de cada matriz para obtener la puntuación final que resulta en el mejor alineamiento posible, para estas 2 secuencias. Este benchmark fue originalmente creado para la benchmark suite de *Rodinia* [12]. Si revisamos esta benchmark suite, en busca de más información sobre el benchmark, no encontramos mucha más información relevante sobre el benchmark, pero sí vemos que existen una nota en el *README.md* de este benchmark, así como la documentación en *Rodinia*, en el que los autores comentan que el programa soporta únicamente 2 secuencias de ADN aleatorias, las cuales serán generadas con igual longitud, y que dichas longitudes de las cadenas han de poder ser divisibles por 16.

- **Dwarf *Structured Grids* con instanciación *SRAD***. La categoría de *Structured Grids* es para benchmarks con el propósito de organizar información en una malla multidimensional, donde las computaciones se procesan como una serie de actualizaciones de la malla, empleando para las actualizaciones de cada punto, los valores de sus vecinos. El benchmark *SRAD* se fundamenta en el proceso de *Speckle Reducing Anisotropic Diffusion* [33], una aplicación que tiene como objetivo eliminar, de forma segura, correlaciones locales de ruidos en imágenes de radar y ultrasónicas. El procedimiento se lleva a cabo a través de métodos de difusión basados en ecuaciones parciales diferenciales. El origen del benchmark de *SRAD*, presente en *OpenDwarfs* procede de un "port" de su versión original desarrollado para la benchmark suite de *Rodinia* [12]. Con ayuda de la plataforma de *Web Archive* [16], encontramos una página web asociada a la versión creada para *Rodinia*, donde vemos que *SRAD* es uno de los pasos iniciales que han de hacerse para la aplicación que ellos llaman como *Heart Wall application*. Esta aplicación sigue los movimientos del corazón de un ratón, a través de secuencias de 104 imágenes ultrasónicas de 609×590 que registra alteraciones causadas por respuestas a distintos estímulos [34]. Si examinamos la versión de este benchmark, para la plataforma de *CUDA*, presente en la benchmark suite de *Rodinia*, encontramos que existen 2 variantes de este benchmark. Al examinar los contenidos de los directorios asociados a cada variante, vemos que la versión 1 de este benchmark, está asociada al programa de *Heart Wall* que hemos mencionado, empleando imágenes como entrada para el benchmark. La versión 2 de este benchmark, desarrollado en *CUDA* y disponible en Rodinia, es la

que los autores han adaptado para la benchmark suite de *OpenDwarfs*, en donde las imágenes que se generan dentro del benchmark se realizan de forma aleatoria, empleando los inputs que el usuario ha de proporcionar al benchmark.

- **Dwarf *Dynamic Programming* con instanciación *SWAT***. Este es otro de los benchmarks, desarrollados para la categoría de *Dynamic Programming*, enfocada a solucionar de problemas complejos, a través de problemas menores. El benchmark de *SWAT* se basa en el algoritmo de Smith-Watterman, el cual es una variante del algoritmo de Needleman-Wunsch, previamente comentado. Este benchmark se enfoca en alinear 2 secuencias de ADN, y calcula la puntuación de coincidencia de las 2 cadenas genéticas que el usuario le da como entradas. Hasta la fecha, también se desconoce tanto el autor encargado de llevar a cabo el desarrollo de este benchmark, así como el origen de los distintos archivos presentes en la benchmark suite, que se emplean para las ejecuciones. Tampoco se sabe si es un "port" de otra versión, que los autores se encargaron de adaptar para *OpenDwarfs*, ya que tras una extensa investigación del benchmark, no conseguimos encontrar referencia alguna, que nos ampliase los conocimientos del benchmark. Se pueden encontrar comentarios en diversas secciones del código, y un comentario en el fichero *swat.cpp*, que indica que el benchmark es la implementación de la versión 27 de un programa no especificado, en el cual utiliza memoria estática para la puntuación de las matrices, y almacenamiento de las cadenas de caracteres de entrada.

- **Dwarf *Finite State Machine* con instanciación *TDM***. La categoría de *Finite State Machine* es para programas, diseñados para capturar el comportamiento de un sistema, definido por sus estados, transiciones definidas por sus entradas y el estado actual, y eventos asociados a transiciones o estados. El benchmark de *TDM*, se basa en una variante restrictiva de la regla de asociación, que se utiliza en la rama de minería de datos. De acuerdo con los autores, benchmarks similares capturan el comportamiento de un sistema definido por estados, transiciones definidas por entradas y estado actual, y eventos asociados a transiciones o estados. El objetivo del benchmark de TDM de *OpenDwarfs*, es descubrir correlaciones temporales empleando como fuente de información, ficheros que contienen eventos de encefalografía (EEG) del cerebro. Si analizamos los ficheros necesarios para la compilación de este benchmark descubrimos, a través de un comentario, localizado en el fichero *tdm_ocl.cpp*, que este benchmark es una adaptación de una versión previa de un programa de GPU Temporal Data Mining, existente en la librería de *CUDA*, y realizado por el desarrollador Sean Ponce. Buscando el nombre del desarrollador podemos encontrar una diversidad de publicaciones, relacionadas con el tema de *Temporal Data Mining*, como son *"Accelerator-Oriented Algorithm Transformation for Temporal Data Mining"* [35], *"Towards Algorithm Transformation for Temporal Data Mining on GPU"* [36], y *"Towards chip-on-chip neuroscience: fast mining of neuronal spike streams using graphics hardware"* [37]. En varias de estas publicaciones, encontramos, como coautor, a *Wu-chun Feng*, el cual fue uno de los autores en las publicaciones asociadas a la benchmark suite de *OpenDwarfs*, mencionadas

previamente en este documento [38]. Lo último que podemos comentar, relacionado con este benchmark, encontramos que los autores de *OpenDwarfs* categorizaron este benchmark como el único benchmark que se encontraba en versión *Alpha*, indicando que los desarrolladores de la benchmark suite, consideraban que la última versión que desarrollaron, aunque aún incompleta, estaba lista para empezar a realizarle pruebas, por un grupo reducido de usuarios.

3: Refactorización de la suite

En este capítulo de la memoria, vamos a examinar los diversos cambios que hemos implementado con el propósito de que la benchmark suite volviese a un estado funcional, y ampliar los límites de las entradas para ciertos benchmarks. De manera que se puedan comprender mejor estos cambios, hemos dividido el capítulo en 14 secciones, una para cada benchmark de la suite. El propósito de esta división, es poder aportar una guía clara, de los distintos cambios que se han estado haciendo a cada benchmark durante, el proceso de refactorización. También incluimos, para cada uno de estos cambios, el razonamiento detrás de cada uno de ellos, en la medida de lo posible. Ordenaremos las secciones de estos capítulos de forma alfabética, comenzando por el benchmark de *BFS* y finalizando con el benchmark de *TDM*.

Con el fin de poder asegurar que los cambios hechos a cada benchmark no causasen problemas inesperados en la suite, se hicieron pruebas de compilación y ejecución para cada benchmarks, después de ser modificado. Esto nos permitió, entre otras cosas, asegurarnos de que cambios en uno de los benchmarks no causan problemas en otros, por la existencia de alguna dependencia desconocida o no documentada por los autores.

3.1. BFS

Para comenzar con los cambios que se han realizado sobre el benchmark de BFS, es importante destacar que este, al igual que la gran parte de los otros benchmarks, funcionaban inicialmente de forma correcta, por lo que los cambio realizados sobre este han ido dirigidos principalmente a eliminar los mensajes de aviso (*warnings*), que lanzaba el compilador de GCC/G++. Aunque inicialmente los mensajes eran numerosos, estos avisos se debían principalmente a cambios en el estándar de *OpenCL* que sucedió con el tiempo, y que los desarrolladores no hubieran sido capaces de tener en cuenta durante el desarrollo. Debido a estas modificaciones, *OpenCL* no era capaz de identificar qué versión del estándar emplear en los benchmarks, por lo que empleaba por defecto la versión 2.0 del estándar.

Tras establecer el uso de esta versión del estándar, al compilar se reconocían que la mayoría de las APIs usadas, estaban deprecadas o que iban a dejar de ser soportadas en futuras versiones del estándar. Se debatió la idea de emplear la versión más moderna de *OpenCL*, durante el proceso de refactorización. Debido a que dichos cambios requería de grandes modificaciones a los códigos de cada benchmark, además de no poder garantizar completa y total compatibilidad con plataformas de Altera FPGAs de Intel, según su centro de soporte [39], se decidió emplear *OpenCL* 1.1 para toda la benchmark suite de *OpenDwarfs*. Para asegurar que *OpenCL* usase esta versión, había que incluir la definición del marco *CL_TARGET_OPENCL_VERSION*. Si establecemos el valor de 110, para este macro, durante el uso del estándar en el proceso de compilación, se interpretará que se desea emplear la versión 1.1 del estándar. Al completar el proceso de compilación, únicamente para el benchmark de *BFS*, vemos que desaparecieron los avisos que salían previamente. Al probar a realizar ejecuciones tanto en gráfica como en el procesador, vemos que el benchmark funciona correctamente y sin problema alguno. Aunque implementar este cambio a los ficheros principales de cada benchmark no es una tarea complicada, se vio que con solo añadirlo en uno de los ficheros de la carpeta *include* de la benchmark suite, haría que este cambio se propagase, ya que estos están presentes en la compilación de cada benchmark. Decidimos, entonces, incluir un *define* de este macro, al comienzo del fichero *rdtsc.h*, que se emplea para definir los distintos contadores de tiempos de cada benchmark de la suite. Una vez añadida esta línea, se aseguró que desaparecían todos los avisos asociados a este problema menor.

A continuación, se siguió analizando el benchmark, para descubrir si este presentaba algún otro problema, que pudiese afectar a su funcionamiento o uso del mismo, pero, bajo lo investigado y probado, este no presentó problema alguno de funcionamiento, según las pruebas efectuadas.

3.2. BWA_HMM

Cuando se comenzó el proceso de refactorización sobre el benchmark de BWA_HMM, inicialmente solo se detectaron como problema menor los mensajes de aviso de *OpenCL* que se lanzaban al compilar, problema que se solucionó tras aplicar los cambios comentados en el benchmark de *BFS*. Se pensó inicialmente que con este cambio el benchmark ya funcionaba correctamente.

Después, con el propósito de proveer más información de las ejecuciones de este benchmark, se hizo imprimir los parámetros que se calculaban durante las ejecuciones, y que no se mostraban por pantalla, siendo estos los parámetros de los modelos ocultos de Márkov (HMM) que se comentaron él en capítulo previo (A, B y π_0), así como guardar estas salidas en un fichero. Tras este cambio menor, se pasaron a realizar pruebas de ejecución del benchmark. Después de diversas pruebas realizadas sobre el benchmark, se detectó un problema que, hasta la fecha, se desconoce de manera concreta la causa por la que este ocurría, pero al que pudimos una ponerle solución. Este problema sucedía cuando le proporcionamos una entrada de datos relativamente grande para la configuración de

variedad de estados (*-n*), comentada en el previo capítulo. Se observó que, a partir de un tamaño superior o igual a 4000, el benchmark retornaba extrañas e inconsistentes salidas, desde ejecuciones correctas; retorno de valores *nan*, para las salidas que este generaba; o mensajes de *Segmentation Fault*. Estos 3 resultados se retornaban para distintas ejecuciones con el mimo tamaño de entrada que mencionamos, lo cual hacía este problema más difícil de entenderlo. Al investigarlo, se determinó que lo más probable fuese a que se estuviese accediendo a la *HEAP* del sistema, de forma aleatoria, lo que explicaría el porqué daba salidas tan inconsistentes. Al usarse un debugger para detectar dónde se daba el problema de *Segmentation Fault*, se descubrió que este ocurría en una de las funciones del kernel, durante la ejecución de uno de los *threads* que este generaba. Examinando la función y efectuando los cambios que se consideraban necesarios, alterando lo más mínimo el funcionamiento de esta función del benchmark, se logró resolver los problemas previamente encontrados. Tras repetidas pruebas y ver que estos no volvían a ocurrir, con las entradas que causaban problemas previos, se pasó a probar con entradas más grandes. Originalmente, estas estaban limitadas a 8000 para la variación de estados, pero una vez que se hicieron los cambios, se pudieron superar estos límites y probarse con ejecuciones más grandes.

Una vez arreglado el problema, y visto que ya no encontrábamos más fallos en el benchmark se dio, por el momento, terminado el trabajo a hacer sobre.

3.3. CFD

El benchmark de *CFD* solo tenía problemas en detectar la versión de *OpenCL* que se debía emplear, de la misma que tenía cada uno de los benchmarks anteriores. El benchmark ya generaba salidas de ejecuciones en ficheros, siendo estos los ficheros *density.txt*, *momentum.txt* y *density_energy.txt*. Llevando a cabo ejecuciones, con el propósito de descubrir qué otros problemas podía haber en cuanto al uso del mismo, no se pudo hallar ningún tipo de error con los ficheros que nos aportaron los autores de *OpenDwarfs*. Dado que no se detectaron más errores en cuanto a las ejecuciones del benchmark, se dio por terminado la refactorización en este benchmark, y proseguir con otros benchmarks que se incluyen dentro de *OpenDwarfs*.

3.4. CRC

Este benchmark también tuvo los mismos problemas que mencionamos con anterioridad en cuanto a la identificación de la versión de *OpenCL* a usar en la compilación del mismo.

Al emplear esta benchmark y preparar entradas para las experimentaciones a realizar, se observó un problema no dado anteriormente. Durante la toma de tiempos de GPU, en las pruebas, se creía inicialmente que este funcionaba de forma correcta, pero tras realizar pruebas en nuestra máquina de experimentación, las ejecuciones se interrumpían con un error en el fichero de kernel de este benchmark. Curiosamente, cuando se fue probando en todas las GPUs a nuestra disposición para hacer las pruebas, siendo estas gráficas integradas de *Intel*, tarjetas de *AMD* y *Nvidia*. Se observó el problema solo se presentaba

en las tarjetas de *Nvidia*, en particular. El fallo en cuestión se debía a un problema en una línea de una de las funciones del fichero kernel de este benchmark, la cual no causaba problemas en otras GPUs o ejecuciones en CPU. Al analizar el problema que resultaba de la ejecución, se veía que el este era causado por un *casting* o conversión de una variable a *unsigned char**. Investigando esta conversión, se probó a hacerla al tipo de variable *uchar*, siendo este el mismo tipo al que se trató de convertir previamente, y que se recomienda emplear en tarjetas de *Nvidia*. Esto solucionó el problema en esta plataforma, pero hacía que toda otra plataforma de ejecución de *OpenCL* no pudiese realizar estas ejecuciones, lanzando el mismo error y en la misma línea. Esto nos llevó a entender que el problema en cuestión, se trataba de una mínima diferencia entre la versión de *OpenCL* para tarjetas de *Nvidia*, y las otras versiones de *OpenCL* para otras plataformas de ejecución. Se descubrió finalmente que, quitando dicha conversión y dejando que el compilador la interpretase y actuase de forma propia, era la solución a aplicar a este problema, ya que esto nos permitió el uso de este benchmark en cualquier CPU y GPU de cualquier proveedor, sin ningún inconveniente.

Se llevaron a cabo algunas ejecuciones con las que comparar las salidas del benchmark, de modo que estuviéramos seguros de que no hubiese ningún fallo. Empleando el comando *-a* del benchmark, el cual realiza verificaciones de ejecuciones empleando 1 solo núcleo del procesador, nos permitió llevar a cabo estas comparaciones de funcionamiento. Según nuestras pruebas, las ejecuciones del benchmark no se veían afectadas, y para una misma entrada, se obtenía siempre la misma salida, independiente de la plataforma de ejecución.

En cuanto a más problemas asociados al benchmark, se halló un problema, que surgió comparando el tiempo real de duración del benchmark, y el tiempo que indicaba que duraba, en realizar grandes ejecuciones. Se observó que, por cómo estaban diseñados los contadores y los usos dados para este benchmark, estos no eran capaces de registrar correctamente la duración de este benchmark, y solo registraban el tiempo de ejecución del último hilo que se lanzaba. Este se debe a que los temporizadores, se definían dentro de un bucle, haciendo que el contador se reiniciase en cada iteración. La solución que aplicamos a este fallo era mover las definiciones, de inicio y fin de estos temporizadores, a una línea del código previa y posteriormente al bucle principal de las ejecuciones. Esto solucionó el temporizador general, dándonos tiempos más realistas, pero no los temporizadores, designados a cronometrar tareas específicas de la ejecución (contadores H2D, D2H, etc.). Comprendimos que la causa del problema se debía a múltiples entradas a esta función, las cuales no almacenaba tiempos que se obtenían de previas entradas, registrando así el tiempo de ejecución del último hilo de *OpenCL* lanzado.Por inseguridad de causar problemas, en la temporización de los otros benchmarks, así como insuficiente tiempo, se decidió no modificarlos, y añadir un mensaje de aviso, el cual indicase porque el tiempo total del benchmark, no coincidía con la suma de los contadores desglosados.

Una vez que todas las configuraciones del benchmark se hubieran realizado y, estuviese todo preparado para ejecutarse, no llegó a encontrarse más problemas asociados al benchmark, y pudo continuar con la extracción de tiempos para las pruebas, las cuales comentaremos más adelante.

3.5. CSR

Este benchmark fue un ejemplo curioso en cuanto al estado en el que se encontraba. Inicialmente, se pensaba que solo presentaba el mismo problema que hemos mencionado en los otros benchmarks hasta ahora, siendo esto los problemas de reconocimiento de la versión de *OpenCL* a usar, como en los otros benchmarks.

Cuando se trataban de hacer ejecuciones del benchmark vimos que, aunque la compilación generaba el ejecutable principal del benchmark, no se llegaba a generar el fichero de kernel, con las funciones a ejecutar a través de *OpenCL*. Analizando la salida de compilación, se observó que el proceso fracasaba en la creación del ejecutable de *createcsr*, el cual se trataba del generador de ficheros para este benchmark. Esto debía ser solucionado debido a que, sin ninguno de estos ficheros y ejecutables, este benchmark no se podría utilizar. Analizando dicho error que se presentaba tras el proceso de compilación, vimos que el problema se daba por la ausencia de un símbolo en la compilación, según los siguientes mensajes de salida por consola: *undefined reference to symbol exp* y *error adding symbols: DSO missing from command line*. Cuando se investigó en busca de estos una solución a estos problemas, se descubrió que este podía deberse a que no se estaba empleando el flag *-lm* durante el proceso de compilación. Lo peculiar de este problema es que, según las instrucciones de compilación, definidas por los autores en el Makefile de este benchmark, este flag de compilación estaban siendo incluido, pero durante el proceso de compilación, se ignoraba por alguna razón. Se consideró que realmente que el problema podría ser otro, aunque cuando se probó a compilar este fichero en particular, de forma manual, se pudo generar el ejecutable del benchmark y del compilador de ficheros, sin problema alguno. Al final consiguió encontrarse una solución simple a este problema de compilación del benchmark. Durante la generación automática del *Makefile* general de la benchmark suite, se define una variable, llamada *LIBS*, la cual es empleada en la compilación de todos y cada uno de los ejecutables de esta suite. Añadiendo el flag de *-lm* a esta variable, la compilación podía completarse sin problema alguno, y el benchmark funcionaba como debía.

Aparte de volver a hallar el problema que encontramos con los contadores de tiempos, el cual se detectó inicialmente en el benchmark de *CRC*, no se llegó a ver ningún otro fallo. Aplicando a este problema, la misma solución que aplicamos para *CRC*, se arregló el temporizador general del benchmark, de modo que podíamos mejor obtener los tiempos de duración del benchmark. Tras estos cambios, no se detectaron más problemas asociados a *CSR*, y se decidió continuar con la refactorización de los otros benchmarks de *OpenDwarfs*.

3.6. FFT

Este benchmark presentaba un problema particular a la hora de poder usarse. Antes de hablar de este extraño fallo en el benchmark, hemos de comentar que pudimos observarlo más en detalle, una vez que corregimos los mensajes de avisos del uso de métodos obsoletos de *OpenCL*, aplicando los mismos cambios realizados sobre el benchmark de *BFS*.

Tras esto, proseguimos analizando el problema asociado al benchmark *FFT*. De acuerdo con nuestros experimentos e investigaciones, el benchmark era compatible con versiones 10.0 del compilador G++ y anteriores, pero en el momento en que se desease pasar a una versión más reciente que fuese la versión 11 o posterior, el benchmark dejaba de compilar y poder usarse. Esto se debe a cómo decidieron hacer la gestión de excepciones durante sus ejecuciones. Originalmente, el lenguaje C++ disponía de métodos con los que el desarrollador era capaz de crear y captar excepciones de ejecución. Pero debido a cambios, a partir la versión 11.0 del compilador de G++, dichos métodos pasaron a ser deprecados [40], de modo que todo software que quisiese emplear esta versión del lenguaje C++, u otras versiones más nuevas, debían de modificar sus programas para cumplir con el estándar impuesto en esta versión. Afortunadamente, este es solo un problema que ocurre con este benchmark, y que no ha surgido con ninguno de los otros benchmarks de *OpenDwarfs*. Como hemos comentado anteriormente, este no compila con la versión 11 del compilador, o posteriores, y el error que se lanza es el siguiente: *ISO C++17 does not allow dynamic exceptions specifications*. En versiones previas a esta, como la versión GCC 9 o 7 que se usaron varias veces al inicio del proyecto, el benchmark se compila correctamente, pero el compilador lanza el siguiente aviso: *dynamic exception definitions are deprecated in ISO C++11*. Tras varias pruebas para solucionar este problema, se vio que la única solución era eliminar ciertas líneas del código, así como la cabecera de *exceptions.h* que se incluía. Se pensó que este cambio afectaría radicalmente al benchmark, necesitando de tener que cambiar por completo todo el código del benchmark, para el empleo de este cambio. Afortunadamente, tras diversas pruebas hechas sobre el mismo en varias de las versiones del compilador de GCC (versiones 7, 9 y 11 en particular), vimos que no había ningún problema en ninguna de las ejecuciones, y este funcionaba correctamente, tanto en CPU como en GPU.

Tras estos cambios comentados, no encontramos más problemas que afectasen a las ejecuciones del benchmark, según nuestras experimentaciones, por lo que decidimos continuar con los otros benchmarks, pendientes a revisar.

3.7. GEM

Este es otro de los pocos casos curiosos de problemas en los benchmarks de *OpenDwarfs*. Según los autores que lo crearon, este se considera como un benchmark que fue estaba terminado y en una versión estable *(stable)*, lo cual quiere decir que este debería funcionar sin problema alguno y de forma correcta. Lamentablemente, este es uno de los benchmarks que no es posible compilar, sin tener que hacer cambios sobre el código fuente del mismo, dado que este reporta errores durante el proceso de compilación. A diferencia de los mensajes de aviso de compilación *(warnings)*, que permiten la compilación y ejecución de un programa, pudiendo este ser inestable o tener problemas en ejecuciones particulares, los errores de compilación no permiten ejecutar el fichero binario que resulte del proceso, ya que el compilador termina de forma inesperada el proceso y no genera el binario ejecutable.

Para solucionar la causa de estos errores, se tenía que descubrir, primero, donde comienzan a dar errores, puesto que problemas como mala definición de métodos o variables lanzan errores en toda y cada una de las líneas donde se hacen uso de estos, lo cual puede hacer confuso hallar la raíz de estos errores. Leyendo el registro de errores de compilación, vemos que el problema, que está afectando a este benchmark, es las definiciones de ciertas funciones como *macros*, los cuales tenían conflictos con otros existentes en una librería de C++. Entendiendo que estos fallos no surgían en versiones previas donde el benchmark sí era operativo, nos lleva a asumir que los autores emplearon una versión mucho más antigua del compilador G++, la cual debe de ser actualmente obsoleta, y ha dejado de ser mantenida, ya que ni en la versión 7 de G++, el benchmark no compilaba. Si eliminamos y cambiamos los usos de estos *macros*, por otros métodos similares provistos en otras librerías, y nos aseguramos que las nuevas funciones son empleadas la compilación, vemos que los errores dejan de aparecer y que el benchmark, vuelve a poder compilarse y usarse sin problema alguno.

Además de cambios menores, en casting de diversas variables, se ha de comentar que se hizo un pequeño cambio en cuanto a la gestión de opciones de entradas de este benchmark. Durante la investigación de la suite y los benchmarks en ella era que, en ciertos casos, la documentación comentaba que existían un determinado listado de opciones a poder usar, pero el benchmark reportaba tener otra lista de opciones, que eran distintas a las dadas en la documentación. A veces estas distinciones podían ser menores, como en *GEM*, o podían ser totalmente distintas, como en *NQUEENS*, pero en la mayor parte de los benchmarks, no había distinción alguna entre estas. Con el fin de hacer que las opciones de este benchmark fuesen más precisas y coherentes con la documentación, se rehizo la función encargada de la gestión de estas opciones, facilitando el futuro mantenimiento que pueda e interese hacerse sobre esta. Esta era originalmente una sola función que gestionaba tanto las entradas que eran necesarias como las opcionales. Tras nuestro cambio, no solo se redujo el número de líneas de código, debido en parte a que se eliminaron opciones no documentadas, sino que esta se dividió en 2 funciones, para mejor mantenimiento del sistema, siendo estas *verify_initial_args* y *read_opt_arg_cmd_line*.

El último cambio aplicado sobre el benchmark, se hizo con el propósito de eliminar los mensajes de *warning* que el compilador lanzaba, al emplear la versión de 2.0 de OpenCL. Esta pequeña inconveniencia se solucionó tras la implementación de los cambios, durante la refactorización de *BFS*. Tras estos no se detectó ningún error que pudiese afectar, gravemente, a las ejecuciones de este benchmark.

3.8. KMEANS

Este benchmark solo presentaba el problema de mensajes de los avisos de *OpenCL* durante el proceso de compilación de ficheros, los cuales se solucionaron con el cambio realizado durante el proceso de refactorización hecho sobre el benchmark de *BFS*. Problemas adicionales relacionados en cuanto a ejecuciones no se hallaron ninguno según las pruebas realizadas sobre el mismo. El único cambio adicional que se realizó sobre este fue hacer

que, cuando se le pida proveer los centroides y sus posiciones resultantes tras la ejecución del benchmark, estos se almacenarán en un fichero de texto, en vez de mostrarlas por pantalla, de modo que fuese más sencillo poder analizar y verificar las salidas. También se realizaron modificaciones mínimas, en cuanto a la entrada de ficheros para las ejecuciones, necesitando ahora especificarle la opción *-i* antes de proveerle la ruta del fichero a emplear, como lo hacen varios de los otros benchmarks de *OpenDwarfs*. Asimismo, se le implementó, al igual que otros benchmarks de la suite, la opción de modificar el tamaño de *LocalSize*, para sus ejecuciones en *OpenCL*.

3.9. LUD

Este es otro de los benchmarks a los cuales no fue necesario realizarle mucho mantenimiento, ya que lo único más problemático que llegó a detectarse eran los mensajes de avisos del compilador, con relación al reconocimiento de la versión de *OpenCL* a usar. Durante pruebas y experimentaciones, solo se detectó un problema, el cual aún no ha podido encontrarse solución durante le proyecto.

Como comentamos anteriormente, este benchmark se encarga de hacer descomposiciones LU, de matrices de puntos flotantes. El inconveniente es que, si se trata de aplicar el proceso de verificación que proveen los desarrolladores de este benchmark, se observa que los resultados que se generan del benchmark son erróneos o que el proceso de verificación de resultados retorna fracasos, para matrices de tamaño 256x256 o superiores. Esto se debía a que, cuando se comparaba la matriz original, con la que se generaba en el proceso de verificación de las salidas, estas deberían de ser iguales, o con una diferencia inferior a 5 decimales, pero como el proceso de verificación demuestra, estas son distintas.

Tras semanas tratando de analizar y solucionar este problema existente en el benchmark, se determinó que era un problema al que, con los conocimientos y experiencias actuales, no sería posible ponerle solución en este proyecto. Esto se debe a que, el error de cálculo con los valores flotantes, que se observó, se propagaba por las celdas de las matrices. Se pensó que inicialmente era causado por las matrices de entradas, pero como el benchmark fallaba con las matrices originales, se descartó esta posibilidad. Lo mejor que pudo hacerse, para ayudar a personas futuras que trabajen en este benchmark, es proveer, como salida del proceso de verificación, la *matriz LU* que se calcula en el benchmark, así como la matriz resultante del proceso de verificación. Originalmente, estos eran generados por pantalla y eran difíciles de poder analizar, en casos de matrices grandes. Es por ello que se hizo que, por medio de entradas en líneas de comandos, se pudiese hacer que estos proceso de verificación, guardaran las matrices resultantes en un fichero, llamándolo con *-v*. Este es solo una variante del proceso de verificación original, que estaba implementado con este comando, y que tras nuestro cambio se omiten la generación de salidas por pantalla. Aun así, se mantuvo el funcionamiento original de esta verificación, pero se alteró el comando necesario para llamarlo, a través del comando *-w*. Esta otra opción también genera los ficheros previamente mencionados, cuando se emplea en las ejecuciones del benchmark.

Tras estos cambios, y ver que no había nada más que hacer por nuestra parte, se optó por continuar con los otros benchmarks de la suite.

3.10. NQUEENS

Este, como otros de los múltiples benchmarks de la suite, solo tenía problemas en cuanto a los mensajes de avisos de *OpenCL* que surgen.

Una vez arreglados estos menores errores, se probó todas y cada una de las opciones que se proveían para el uso del benchmark. Nada más empezar con este proceso, lo primero que se descubrió era que existían muchas más opciones de las que aparecían en la documentación del benchmark. Lo siguiente que se encontró, es un fallo que abarcaba 2 de las opciones del benchmark. El benchmark permite establecer un tamaño de *blocksize* y una cantidad de *threads* a usar, según indica su documentación. Analizando y estudiando el código, así como el funcionamiento del estándar de *OpenCL*, se concluyó que estas afectaban a los tamaños de *GlobalSize* y *LocalSize*. Estos son valores que se pueden indicar en las ejecuciones de funciones con *OpenCL*, las cuales tienen un efecto sobre los recursos que se dedican a cada función definida para ejecuciones de *OpenCL*. El problema que se encontró era que, si se trataba de usar un número de *threads*, menor al número de *Blocksize*, el programa emitía un mensaje *Segmentation Fault*. Según lo investigado y probado, concluimos que este se debía a que el tamaño de *LocalSize* era menor que el tamaño de *GlobalSize* y casaban estos problemas. Modificando la gestión de opciones haciendo que, en caso de ser *threads* menor a *Blocksize*, se optaría por establecer ambos al valor de *threads* que el usuario estableció.

En cuanto a más cambios a realizar sobre este benchmark, no se llevaron ninguno a cabo, dado que no hubo ningún problema de ejecuciones que se detectasen.

3.11. NW/NEEDLE

El benchmark de *NW*, o *needle* según el nombre del ejecutable resultante del benchmark, inicialmente solo presentaba problemas a la hora de detectar la versión de *OpenCL* que se debía emplear, igual que los otros benchmarks, previamente comentados.

Una vez arreglados estos mensajes de avisos, se hizo un cambio menor, de modo que las salidas de datos, necesarias para verificar el correcto funcionamiento del benchmark, se imprimiesen en un fichero de texto. En cuanto a problemas de ejecución del benchmark, los cuales hallamos durante nuestras ejecuciones en la máquina de experimentación a emplear, se vio que con entradas muy grandes, superiores a tamaños de 42000, este acaba dando errores de *Segmentation Fault*. Se creyó, en un principio, que este problema debería ser fácil de solucionar y que, con un poco de tiempo y ayuda de un debugger, podríamos ver el problema que estaba causando estos fallos y corregirlo para este proyecto. Sin embargo, tras dos semanas de trabajo sobre el benchmark, se constató que el problema que presentaba, requería de más tiempo del que disponíamos, para cumplir con el plazo de entrega de este

proyecto, por lo que decidió notificarse y apuntarlo como un problema a continuar a seguir investigando para solucionarlo futuramente. Este benchmark requiere, adicionalmente, de proveerle como entrada un valor de penalización, que se aplica a la hora de calcular los resultados de la ejecución. Analizando el código de*NW* esta penalización que el usuario ha de introducir, debía ser un valor entero positivo, ya que cuando el benchmark interpreta la entrada, el símbolo del número se invierte, aplicando siempre un valor negativo. Los autores del benchmark añadieron, adicionalmente, una verificación de la entrada, para asegurar que el usuario solo podía proporcionarle valores positivos. Según nuestras pruebas con este parámetro, el valor de penalización que le introduzcamos al benchmark no afecta en ningún modo a los tiempos de ejecución del benchmark.

Para concluir con la refactorización del benchmark, comentar que se le han hecho 2 cambios menores, adicionales a este benchmark, los cuales fueron enfocados a mejorar la gestión de sus entradas por líneas de comandos, así como un cambio al mensaje de uso del benchmark, y aportando un ejemplo de entrada correcta.

3.12. SRAD

En lo que respecta al benchmark de *SRAD*, este también solo tenía problemas en cuanto a los mensajes de avisos de *OpenCL* que aparecen al intentar compilarlo. Una vez que estos mensajes se solucionaron, no se encontraron ningún otro defecto que existiese con respecto a este benchmark, o que afectasen a las ejecuciones. Se pensó que quizá podría haber alguna limitación o problema, no especificado, en cuanto a las entradas que este benchmark podría recibir para sus ejecuciones. Pero, por lo que pudo analizarse en pruebas adicionales realizadas sobre el benchmark, no se llegó a localizar ningún tipo de problema o error que pudieran afectar a las ejecuciones del benchmark de *SRAD*.

3.13. SWAT

Este benchmark únicamente presentaba la problemática de mensajes de avisos de *OpenCL* durante el proceso de compilación de sus ficheros, los cuales se solucionaron con el cambio realizado durante el mantenimiento hecho sobre el benchmark de BFS.

Un cambio significativo que se hizo sobre este fue un aumento del tamaño de las entradas para las secuencias de ADN con las que trabaja el benchmark. Originalmente, solo podían recibir como entradas 2 cadenas de entradas: la primera no podía ser más larga de 5000 caracteres, mientras que la segunda entrada podía ser de 25 millones de caracteres longitud. Aunque esto no era un obstáculo para las pruebas con los ficheros de entrada provistos por la benchmark suite, para alguien que desee probar con sus propios ficheros, los cuales debían tener la misma estructura y no superar estos límites de tamaños, esto podría ser una gran limitación y un impedimento para un uso de este benchmark. Por ello se decidió ampliar el tamaño límite de las entradas, con el fin de tratar de que se diese esta limitación. Cuando se trató de aplicar esta modificación, su implementación inicial causaba diversos problemas haciendo que el benchmark dejara de funcionar en sistemas GPU, y

entradas con las que originalmente funcionaba el benchmark, pasaron a dar problemas, no vistos previos a las ejecuciones, como mensajes de *Segmentation Fault* los cuales tenían la peculiaridad de ocurrir de forma aleatoria. Tras intentos, pruebas y errores, se vio cuál podía ser el problema de verdad y, tras implementar una solución a este, el benchmark ha quedado modificado, permitiendo una entrada de ficheros de mayor tamaño, sin causar problemas a previas ejecuciones, u otras nuevas ejecuciones, según nuestras pruebas.

Asimismo, durante el proceso de refactorización se incluyeron dos programas menores adicionales con los que crear ejemplos de entradas a probar en el benchmark: *createswat_query* y createswat_sampledb, los cuales generan secuencias de ADN donde cada componente sigue las probabilidades y reglas de las secuencias humanas. Detallaremos más adelante, el propósito de estos generadores, así como los problemas encontrados durante el desarrollo de los mismos, en el próximo capítulo del documento.

3.14. TDM

Este es otro de los benchmarks que, solo tenía problemas en cuanto a los mensajes de avisos de *OpenCL*, que surgen cuando se compilaba este benchmark.

Una vez solucionados estos mensajes de avisos, no hubo nada más que hacer con este benchmark que se haya detectado y que poder realizar sobre el mismo, ya que este funciona con los ficheros de entrada que estaban originalmente en la benchmark suite. En cuanto a las posibles entradas que proveerle, encontramos que en el directorio */test/finite-state-machine/tdm/* una serie de ficheros distintos, los cuales nos permiten variar la primera entrada, de las 3 entradas de ficheros, que se han de proveer para este benchmark. Encontramos también, en este directorio, un fichero llamado *ref-31-episodes-CPU.txt*. Inicialmente, pensamos que este sería una variante de las posibles entradas a proveer al benchmark, pero, analizando la estructura del archivo, con el fichero de salida, *tdm-gpu-csw.txt*, generado por el benchmark, vemos que es una colección de ejecuciones. Considerando que esta estaba provista, originalmente, con los ficheros de ejecución de la benchmark suite, creemos que es un ejemplo de resultados de ejecuciones, generados a través de los otros ficheros existentes. Si analizamos el fichero, veremos que incorpora cabeceras, previos a la aportación de resultados, empleando los ficheros *sim-64-size-50.csv*, *sim-64-size-100.csv*, *sim-64-size-200.csv* y *sim-64-size-500.csv*. Se consideró la oportunidad de desarrollar e incorporar un generador de ficheros para el benchmark, con el fin de generar este tipo de documentos, y llevar a cabo pruebas de ejecución de mayor tamaño. Sin embargo, debido a la falta de comprensión de estos ficheros o de encontrar un patrón para replicarlos, se optó por emplear los ficheros incluidos para las ejecuciones y continuar trabajando con los otros benchmarks de *OpenDwarfs*.

4: Mejora de la usabilidad

En este capítulo del documento detallaremos los trabajos adicionales realizados durante el proyecto. Los diferentes cambios y trabajos, que se llevaron a cabo después del proceso de refactorización de la benchmark suite, los presentaremos en las siguientes 2 secciones de este capítulo del documento:

- **Adaptaciones para permitir una evaluación experimental sistemática**. Esta sección del capítulo se centra en exponer los aspectos más relevantes de las ejecuciones experimentales que hemos realizado con la benchmark suite, detallando los tipos de experimentos que hemos realizado.
- **Invocaciones por líneas de comandos**. Estos son cambios menores que hemos efectuado sobre los benchmarks, previos a la realización de las pruebas experimentales, con el fin de hacer las ejecuciones de los benchmarks de la forma más uniforme posible. Algunas de estas modificaciones más específicas han podido ser comentadas en capítulos anteriores, pero trataremos de desarrollar con mayor profundidad en esta futura sección.

Una vez detallada la estructura en la que se segmentará este capítulo del documento, podremos proceder ahora a desarrollar, en mayor detalle, cada una de estas secciones, comenzando así, con la sección de pruebas experimentales.

4.1. Adaptaciones para permitir una evaluación experimental sistemática

En esta sección del capítulo se presentarán de las diversas pruebas experimentales finales, realizadas sobre los diversos benchmarks, y el proceso de pruebas y verificaciones que se llevó a cabo con el fin de preparar dichos experimentos. El objetivo de llevar a cabo esta experimentación, era preparar un conjunto de pruebas, para cada benchmark, las cuales simulasen ciertos niveles de carga de trabajo. Estableceremos tres input sets

diferentes similares a los establecidos en su momento por el benchmark *SPEC CPU*[41]. Esos tres input sets son los siguientes:

- **Test input set:** Los experimentos que se realizaron para este set de entradas, fueron ejecuciones cuya duración fuese de 1 segundo, aproximadamente. El propósito de estas ejecuciones era llevar a cabo unas pruebas rápidas, las cuales sirvan para asegurar que los benchmarks se han instalado de manera adecuada y funcionen correctamente. La carga de trabajo que estas ejecuciones han de realizar ha de ser equivalente a una baja carga de trabajo.
- **Train input set:** Esta categoría de experimentos se diseñó para que simularan una carga de trabajo moderada. Se acordó que las duraciones de estas ejecuciones deberían ser próximas a 30 segundos para garantizar que las simulaciones fueran correctas. El propósito de estas es de simular una carga moderada de trabajo.
- **Reference input set:** Estas se tratan de nuestra categoría final de ejecuciones a preparar y con las que proveer a usuarios de la benchmark suite. Este tipo de ejecuciones deberían tener una duración de 1 minuto, y suponer una carga elevada, así como realista, de ejecuciones para cada benchmark, similar a cómo podían probar los usuarios finales de la benchmark suite. Emplearemos estas entradas, y los tiempos de ejecución obtenidos, para la evaluación del funcionamiento de los benchmarks, en distintas plataformas de ejecuciones. Los resultados de la experimentación, se presentarán en el siguiente capítulo del documento. Detallar que, por limitaciones de algunos benchmark, en ciertos casos no han sido posible sacar ejecuciones de 1 minuto para algunos de estos.

Los tiempos estimados para cada tipo de prueba fueron obtenidos a través de los temporizadores de ejecuciones, incluidos en cada uno de los benchmarks, por los desarrolladores originales de *OpenDwarfs*. Los tiempos base de ejecuciones, que se han detallado previamente, serán obtenidos con las ejecuciones de *OpenCL* en CPU. Nuestro objetivo es observar cómo de bien escalan las ejecuciones de cada benchmark, en función de la plataforma de OpenCL que se usará. Después de determinar las ejecuciones de CPU, pasaríamos a repetir estas mismas entradas especificadas, con la diferencia de detallar que estas se realizasen sobre una plataforma GPU. Para mejor observar este cambio de tiempos de ejecución, planeamos emplear los tiempos obtenidos, con los distintos input sets, y ver como se comportan los benchmarks. De acuerdo con nuestra hipótesis, planteada durante el desarrollo y configuración de estas pruebas, debería poder apreciarse una mejoría en los tiempos de ejecución cuando estas se llevasen a cabo sobre la plataforma GPU, en comparación con las ejecuciones en CPU. Dichas mejorías se deberían apreciar en las 3 experimentaciones, y en mayor medida sobre la experimentación *reference* realizada. Consideramos que sucederá esta mejoría en tiempos de ejecución, ya que, muchos de los benchmarks existentes en *OpenDwarfs*, como son *CSR*, *GEM* o *LUD*, llevan a cabo una gran cantidad de cálculos matemáticos, los cuales deberían de realizarse, de forma mucho más eficiente, sobre las plataformas GPU. Suponemos que, como este tipo de

sistemas, se caracterizan por ser capaces de realizar estos procesos, que requieren de un alto nivel de cómputo, de forma mucho más rápida y paralela, los tiempos medios, que obtengamos en ejecuciones de GPU, deberían ser menores que, los tiempos de CPU medios que obtengamos para cada benchmark de *OpenDwarfs*.

Para calcular el tiempo promedio de ejecución, de cada benchmark y en los distintos dispositivos de ejecución, se calculará la media aritmética de estos tiempos de ejecución que hayamos obtenido, mientras que para calcular el *speedup* de la benchmark suite, emplearíamos el cálculo de la media geométrica. Esta última trata de calcular la media de un conjunto de datos, de forma similar a la media aritmética. La diferencia fundamental que se puede observar es que, la media geométrica se calcula haciendo la raíz n-ésima del resultado que se obtiene, obtenido de hacer el producto de los N datos. Esta diferencia del procedimiento de cálculo, resultará en una media que no se ve influenciada, o sufre poca influencia, de los tiempos anómalos *(outliers)*, que pueden existir en un conjunto de datos, originados en este tipo de experimentos. Junto con esta media de tiempos, a obtener de cada benchmark, aplicaremos lo que se conoce como el Teorema Central del Límite, en nuestra experimentación a llevar a cabo. Este determina que, dado un conjunto suficientemente grande de datos, estos deberían acabar organizándose, de forma que la función de distribución con la que se pueden representar, se asemeja a la conocida distribución Normal o campana de Gauss [42]. Aplicando el principio de este teorema, determinamos que realizando un mínimo de 30 ejecuciones de cada uno de los benchmark, y para cada plataforma de *OpenCL* a usar, nos resultaría en una correcta medición de los tiempos de ejecución de cada benchmark, así como una buena observación del speedup entre estas 2 plataformas.

Pasaremos, pues, a la siguiente, y última, sección del capítulo, en el que hablaremos de cambios en invocaciones por líneas de comandos, para los benchmarks de *OpenDwarfs*.

4.2. Invocaciones por línea de comandos

En esta sección haremos una breve descripción de cambios adicionales que se han realizado, de mejorar la usabilidad de estos benchmarks. Estos cambios han sido enfocados, principalmente, a mejorar la calidad de uso de estos benchmark, en el lanzamiento de ejecuciones de los benchmarks terminales de ejecución. Primero detallaremos los cambios que se han efectuado con el finde mejor poder indicarle el uso de la plataforma particular de *OpenCL* a usar. Seguidamente, hablaremos de cambios realizados en cuanto a sus entradas por líneas de comandos en los benchmarks, y finalizaremos la sección, y por consiguiente este capítulo, comentando que cambios o inclusiones se realizaron, relacionados con generadores de ficheros, para diversos benchmarks de OpenDwarfs.

Comenzaremos, detallando esta sección, los cambios que se realizaron sobre el sistema de selección de plataforma de *OpenCL* para cada uno de los benchmarks. Hemos de mencionar que el sistema ya fue originalmente implementado por los desarrolladores de esta benchmark suite, empleando los métodos que *OpenCL* incorpora, para leer y seleccionar sobre qué plataforma se ha de realizar la ejecución. Aunque es cierto que este

sistema funciona, en la mayoría de los casos correctamente, debido a la documentación existente, la cual nos causó alguna que otra confusión, en los inicios del proyecto. Tras adaptarnos a la estructura y funcionamiento de la benchmark suite, se llevaron a cabo diversas pruebas para verificar y probar cada una de las opciones, de modo que aportaremos una breve descripción del funcionamiento y uso de ellas.

- Para seleccionar una plataforma específica en la que ejecutar, el usuario ha de introducirle la plataforma e id del dispositivo establecido en dicha plataforma. Después de indicarle estas entradas, el usuario le podrá indicar parámetros de entrada particulares del benchmark, indicando como separador entre estas opciones una doble línea `--`. Las opciones de selección de dispositivo y plataforma de *OpenCL* son las siguientes:

  - **-p** x **-d** y: Estas son 2 opciones que deben de ejecutarse conjuntamente para poder elegir correctamente el dispositivo. Con la opción *-p* se le puede indicar el #id de la plataforma a usar, mientras que, con *-d*, le indicamos el #id del dispositivo particular que se desea usar de la plataforma previamente indicada. Para ver los #id de plataforma y dispositivos de nuestras máquinas, que soportan *OpenCL*, podemos instalar y emplear el comando de *clinfo*, que comentamos en capítulos anteriores.
  - **-t** n: Con este parámetro podemos indicarle el tipo de dispositivo que deseásemos emplear en nuestras ejecuciones. La lista de entradas para estas opciones es: #0 para usar CPU #1 para usar GPU #2 para usar sistemas MICs #3 para usar un sistema FPGA. Los autores reconocen que si con esta entrada no se le añade el parámetro *-p*, a la hora de ejecutar el benchmark, se buscará el dispositivo especificado, en la plataforma 0 de *OpenCL* del sistema.

  En lo que respecta a estas opciones, según nuestras pruebas e investigaciones, estas funcionan correctamente y no fue necesario realizarles ninguna modificación o adaptación, para su uso en ninguno de los benchmarks, presentes en *OpenDwarfs*.

- En caso de no especificarle el dispositivo particular, el benchmark por defecto como sistema de ejecución, la CPU de la plataforma #0 del sistema. Aquí se implementó un pequeño cambio, de modo que se garantizase el correcto funcionamiento de esta alternativa, de acuerdo con las especificaciones de los desarrolladores. Ahora, con esta opción, se analizan todas las plataformas y dispositivos en cada una de ellas, con el fin de encontrar una CPU con soporte de *OpenCL*, donde poder realizar una ejecución indicada. Esto se hizo debido a que, en casos como nuestro sistema local de desarrollo, para indicar correctamente el empleo de la CPU de nuestro sistema, habría que indicarle el uso del dispositivo #0 de la plataforma #1. Este cambio menor debería poder facilitar el uso de los benchmarks en sistemas, y evitar posibles confusiones de utilización, con sistemas cuyas configuraciones de *OpenCL* fuesen similares.

El comando resultante de una ejecución de uno de estos benchmark, sobre una tarjeta gráfica en particular, se mostrará a continuación, para un caso ejemplo del benchmark de *LUD*. En este ejemplo del comando, se considerará que nuestro dispositivo, sobre el que deseamos llevar a cabo la ejecución del benchmark, se identifica en nuestra configuración de *OpenCL* con el *#id de plataforma* 2, y el *#id de dispositivo* 3. Asimismo, asumimos que el ejecutable del benchmark *LUD*, ha sido compilado en el directorio de *build* de nuestra benchmark suite, y que la máquina en la que tratamos de realizar la ejecución de este benchmark es una máquina común, y no una máquina clúster, como la que detallamos para nuestras experimentaciones.

./lud -p 2 -d 3 -- -i ../test/dense-linear-algebra/lud/2048.dat

Con respecto a modificaciones implementadas del uso de cada benchmark, se decidió que, para aquellos benchmarks que requerían entradas de un solo fichero, nos asegurásemos de que se necesita del uso de la opción de *-i*, para llevar a cabo sus ejecuciones. Esta adición se hizo debido a que, varios benchmarks de *OpenDwarfs* hacían uso de este tipo de comandos, para indicar la entrada de sus ficheros de ejecución. Realizar este cambio menor para ciertos benchmarks que no la empleasen, ayudaría a causar menos confusiones a la hora de usar estos benchmarks, así como mejorar la homogeneidad de los benchmarks en la medida de lo posible. Para otros benchmarks que no requieren de ficheros para sus ejecuciones, y solo emplean las entradas aportadas en la línea de comandos, se les ha provisto un poco más de especificidad de qué entrada se está aportando, como en el caso del benchmark *NW*, que ahora requiere de comandos para especificar sus entradas para una ejecución.

Otra modificación menor que se implementó, en cada benchmark que no lo incluyese, fue que el usuario pudiese establecerse un tamaño particular de *LocalSize*. Esta opción ya estaba incluida en un grupo menor de benchmarks, por lo que se decidió expandir a aquellos que no la tuvieran en la medida de lo posible. Originalmente, se planeó que el usuario fuese capaz de modificar el número de *threads* que *OpenCL* usase, pero según lo investigado durante estas modificaciones, *OpenCL* no trabaja de esa forma y, la opción más cercana, eran llevar a cabo los cambios de los tamaños de *LocalSize* y *GlobalSize*. La opción para el usuario de, modificar los tamaños de *GlobalSize*, no se llegó a implementar debido a que, hacer este cambio, suponían de modificar completamente cómo se llevaban a cabo las ejecuciones de cada uno de los benchmarks. Independientemente de esta opción, se pudo implementar la opción de modificar el tamaño de *LocalSize* en aquellos benchmarks que no lo incorporasen, aunque no podemos garantizar que no cause problemas en los benchmarks, por un uso indebido. Esto se debe a que, si el usuario establece un tamaño de *LocalSize*, que no es compatible con la experimentación a realizar, dadas las otras entradas, la ejecución del benchmark puede resultar en errores. Por esta razón, a pesar de incluir estas opciones de entradas para diversos benchmarks, y realizar pruebas con el fin de validar el efecto de esta en los benchmarks, se ha de señalar que estas aún requieren de pruebas más extensas, con el fin de garantizar que no existan problemas graves de ejecución en los benchmarks donde se empleen.

Como últimos cambios a destacar, realizados en *OpenDwarfs*, se añadieron a diversos de los benchmarks, que requieren de ficheros de entradas, generadores de ficheros. A través de estos se pueden generar ficheros de mayor tamaño a emplear en ejecuciones de benchmarks que requieres estas entradas. Estos son similares a los ejecutables de *createcrc* y *createcsr* que los desarrolladores incluyeron originalmente en los benchmarks de *CRC* y *CSR*, respectivamente. Se planeó entregar un generador de ficheros para cada benchmark que los necesitase, con el fin de poder realizar pruebas adicionales sobre estos benchmarks, pero, por falta de documentación sobre el origen de estos ficheros, que permitiese replicarlos, solo se han podido aportar los siguientes generadores:

- **createswat_query** y **createswat_sampledb**. Estos dos generadores de ficheros se crearon para el benchmark *SWAT*, uno de los que más se desconoce de sus orígenes. Estos permiten generar las entradas que se necesitan aportar para poder hacer ejecuciones de este benchmark. Originalmente, estos dos generadores eran parte de otro, el cual generaba inicialmente ambas entradas, necesarias para este benchmark. Pero, como se analizó posteriormente en pruebas de ejecución de *SWAT*, que los 2 ficheros de entrada no tenían por qué ser necesariamente del mismo tamaño, se optó por mejor separarlos en dos ejecutables independientes. Durante el desarrollo del generador original, hubo ciertos problemas en la generación de ficheros para la entrada de *sampledb*, ya que, dos de los tres ficheros para esta entrada, eran ficheros que debían de ser escritos en lenguaje binario. Dado que no se conocía el programa original que los desarrolladores emplearon para crear estos ficheros, lo mejor que pudo hacerse fue tratar de comprender e interpretar estos ficheros, con ayuda de los originales, que ya se encontraban el *OpenDwarfs*. Tras una semana de pruebas, se consiguió crear una versión de este generador que producía ficheros que el benchmark de *SWAT*, podía interpretar y emplear en sus ejecuciones. En la implementación original de estos generadores, los programas producían secuencias de ADN, de forma aleatoria, sin ningún patrón o regla de combinación alguna entre los distintos componentes presentes en las cadenas genéticas. Así que, tras una investigación más en detalle sobre posibles reglas de estas secuencias, se optó por alterar estos generadores, implementando reglas de composición, basadas en los principios de *Erwin Chargaff* para las secuencias de ADN humanas [43][44][45].
- **createkmeans**: Este es generador de ficheros para el benchmark de *KMEANS*. Inicialmente, se planeó desarrollar este a mano, pero tras investigar dentro de la benchmark suite de Rodinia de la cual pertenecía la implementación original de este benchmark, se dio con un generado de ficheros que esta benchmark usaba para su implementación. Probando diversos ficheros que este generaba, en la implementación de *KMEANS* de *OpenDwarfs*, se vio que el benchmark no tenía ningún problema en aceptarlos y trabajar con ellos, por lo que se decidió traer el código fuente de este generador, y editar el *Makefile* del benchmark, de modo que se generase un ejecutable a partir de este código. Una mínima alteración a realizar sobre el benchmark fue la de indicarle el valor de entrada, para cada ejecución del generador, y con un comando particular (*-o* para indicar el número de objetos a generar, y *-n* para

indicar el número de características(*features* para cada objeto). En caso de que estas características de los objetos se deseasen representar con valores flotantes, en vez de números enteros, podemos indicarle al programa, en sus entradas por la consola, la opción *-f* que los desarrolladores ya tenían implementada.

- **createbfs**: Este es otro generador, se realizó para el benchmark de *BFS*. Este fue uno de los generadores que, inicialmente, se crearon a mano, comprendiendo y tratando de replicar los ficheros originales que traían esta benchmark suite. Tras estar cerca de una semana de trabajo, se consiguió aportar uno, el cual generaba ficheros que podían ser ejecutados en el benchmark, y producían correctos resultados cuando se empleaban en sus ejecuciones. También, durante el trabajo que se hizo con el programa de *createkmeans*, se halló un generador de ficheros para este benchmark, en la benchmark suite de *Rodinia*. Por lo experimentado y probado, los ficheros generados con este binario, son compatibles con la implementación de *BFS* en *OpenDwarfs*. Con el fin de poder aportar la mejor experiencia para futuros usuarios de la benchmark suite, así con tal de no desperdiciar tiempo de trabajo dedicado, se decidieron incorporar los 2 generadores de ficheros a la benchmark suite, siendo *createbfs* el que se desarrolló manualmente, y *createbfs_rodinia* siendo este segundo el que se localizó en la benchmark suite de *Rodinia*.

- **createlud**. Este es un generador de ficheros, para el benchmark *LUD*, el cual crea ficheros de matrices flotantes, las cuales deben de tener descomposiciones LU. Inicialmente, este generador de ficheros solo creaba matrices con valores aleatorios, las cuales no tenían garantía de tener descomposición LU. La versión actual, que se entrega con la benchmark suite, se asegura que si tengan descomposiciones LU, ya que, para generar la salida, se producen inicialmente una matriz *Lower* y *Upper*, que después se emplean para en un producto para generar la matriz final. El problema que tiene esta implementación es que, para matrices de grandes tamaños, esta puede tardar en realizarse. Se trató de mejorar estos tiempos de generación de las matrices, haciendo uso de la extensión de OpenMP, la cual incorpora, similares a las APIs de *OpenCL*, instrucciones con la que poder realizar la paralelización de programas, con la limitación de que OpenMP solo puede implementar paralelizaciones para CPUs. En las últimas revisiones de investigación realizadas sobre los benchmarks de *OpenDwarfs*, se descubrió que, la benchmark suite de *Rodinia*, incorporaba un generador de estas matrices, almacenado en el código asociado a los benchmarks de *CUDA*. Este ejecutable, generado a través del código del fichero *get_input.c*, realiza el mismo proceso, descrito previamente, donde se generan dos matrices triangulares, que luego después se multiplican para generar una matriz que se debe poder descomponer por LU.

Tras esta última explicación de cambios hechos a los benchmarks, concluimos este capítulo del documento, y pasaremos al siguiente capítulo, en el cual mostraremos los resultados de nuestra experimentación, realizada sobre los benchmarks de OpenDwarfs.

5: Resultados experimentales

En esta sección queremos mostrar los resultados obtenidos tras llevar a cabo una experimentación completa con los distintos input sets desarrollados, comentados en el anterior capítulo. Antes de mostrarlos, detallaremos la estructura de la máquina empleada en la experimentación. Seguidamente, proveeremos una tablas, en donde reflejamos los distintos input sets empleados para cada experimentación. Más adelante, presentamos dos secciones donde se detallan los dos tipos distintos de experimentos realizados. En cada una de estas secciones, se muestran las gráficas de tiempos y de speedups generadas a partir de las entradas, y se discuten las conclusiones derivadas a partir de los resultados.

Vamos, primeramente, a detallar la máquina de experimentación que hemos decidido emplear. Dado nuestro tipo de evaluación a realizar, se optó por usar las máquinas clúster de experimentación de la UVa. Este tipo de máquinas, son una agrupación de distintos recursos informáticos, los cuales tras ser configurados, de modo se pueda interactuar como si de un único sistema se tratase. De esta manera, el sistema que resulta presenta capacidades mucho más elevadas que las de un sistema común, de modo que tareas más pesadas o complejas puedan realizarse eficientemente en esta máquina. Los sistemas clúster, son empleados diariamente en una multitud de trabajos y tareas, y pueden ser clasificados en clústeres HTC y clústeres HPC. En ambas configuraciones de estos sistemas, se requiere de una colección de software intermediario, *middleware*, necesario para la gestión de recursos[46]. Por lo analizado e investigado, las máquinas del clúster que vamos a usar, están equipadas con hardware específico, con el fin de realizar tareas de experimentación científica que se le demanden, por lo que estos entrarían dentro de la categoría de clústeres HPC, así como de emplear la pila de software de SLURM como middleware [47]. Este sistema de clúster será ideal para probar el estado final en el que se encuentra la benchmark suite, tras el proceso de refactorización que hemos llevado a cabo.

La máquina clúster, en nuestro caso, se divide en diversos nodos, en los cuales pueden llevarse a cabo ejecuciones de programas. Para el caso de nuestras experimentaciones a realizar con cada uno de los benchmarks de OpenDwarfs, se decidió, inicialmente, que se optarían a emplear los nodos de *Manticore* y *Gorgon*. Al comienzo de preparación de nuestras experimentaciones, se planeó usar el nodo *Manticore*, presente en estas máquinas,

debido a que estaba menos ocioso de tareas a realizar, dándonos mayor libertad a la hora de realizar la configuración de nuestras pruebas. Aun así, debido a problemas de inconsistencia y extravagante de tiempos obtenidos, se optó por cambiar al nodo de *Gorgon*. Tras volver a lanzar nuestras pruebas iniciales, vimos que, los tiempos que reportaba este nodo, eran más coherentes que los otros lanzados sobre *Manticore*. Así pues, las especificaciones de este nodo de la máquina clúster son las siguientes:

- 2× *procesadores EPYC 7713 de AMD*, cada uno con 64 núcleos físicos y 128 *threads* o subprocesos (128 núcleos y 256 *threads* en total).
- 2× *GPU Nvidia RTX 4500*, 1× *GPU NVIDIA A100* y 1× *GPU AMD W7800*. Planeamos llevar a cabo experimentación utilizando las 3 tarjetas gráficas disponibles en el sistema.
- 1 TB de memoria RAM.

Al momento de realizar estas ejecuciones en el nodo del clúster, ha sido necesario comprender con claridad la configuración implementada sobre estos nodos, con el fin de lanzar ejecuciones al nodo a emplear, de forma correcta. Para hacer uso de un nodo configurado en el clúster, a través del middleware de SLURM, hemos de emplear la opción *-w* e indicar el nombre de dicho nodo, *Gorgon* en nuestro caso, a la hora de emplear el comando de *srun*. Para poder asegurar que, durante la ejecución de uno de los benchmarks se le dediquen todos los recursos que este tiene a su disposición, tenemos también que especificarle al comando de *srun* la opción de ejecución *--exclusive*. Mostraremos ahora un ejemplo de ejecución del benchmark NQUEENS, usando todo lo detallado hasta el momento.

srun -w gorgon --exclusive ./nqueens 15

Esta sería la estructura final de cada línea de lanzamiento de cada uno de los benchmarks en los scripts de *BASH* desarrollados para la experimentación, para los diversos benchmarks a ejecutar y las entradas asociadas, en cada experimentación que se planea realizar. Con el propósito de preservar los tiempos resultados de las ejecuciones, así como datos adicionales que los benchmarks muestran en sus salidas, se decidió redireccionar estas salidas a ficheros de texto, del cual poder después extraer los tiempos totales de ejecución. Las salidas de cada experimentación, y de cada benchmark, se guardan en un directorio y ficheros distintos, de modo que evitemos confusiones a la hora de organizar todas y cada una de las salidas de los benchmarks. Así pues, las ejecuciones del el benchmark de *NQUEENS* en la CPU, para una ejecución del input set de *Train*, se almacenarían en la ruta */tiemposSalida/TRAIN/CPU/NQUEENS.txt*, mientras que, ejecuciones del input set de *Reference*, realizadas sobre la GPU seleccionada, se almacenaran en */tiemposSalida/REF/GPU/NQUEENS.txt*. Para evitar que, en cada ejecución, este fichero se vea sobrescrito, hemos de direccionarlo empleando el comando *»*, el cual añadirá los resultados

de una nueva ejecución, al final del fichero. Finalmente, a cada una de estas ejecuciones, incorporaremos el símbolo de ampersand &, al final de la ejecución del comando. Esta opción de ejecución, por lo normal, permite mandar ejecuciones en segundo plano a nuestro dispositivo, permitiendo al usuario poder seguir empleando la instancia de la terminal, a través de la cual lanzó una ejecución. Gracias al software de *SLURM*, estas ejecuciones entrarán directamente a la cola de procesos a ejecutar, sin tener que abrir una segunda conexión *SSH*, y poder seguir trabajando sin ninguna interrupción. Tras definir parte de estas entradas como variables de nuestro script de *BASH*, con el fin de tener un mejor control a la hora de realizar nuestras ejecuciones y fácilmente alterar los parámetros de ejecución comunes de cada benchmark, obtendremos las siguientes líneas de ejecución:

srun $srunParam ./nqueens 15 »<directorio>/NQUEENS.txt &

Para realizar las ejecuciones, de cada benchmark, sobre la GPU designada, hemos tenido que añadirle una variable adicional a este comando, con la que especificar cuál es la que se usaba en particular. Aportaremos una explicación más detallada, del funcionamiento de esta entrada, con el fin de especificar la plataforma de ejecución OpenCL del benchmark, en la próxima sección del capítulo. Por el momento, solo mostraremos la variante del comando previamente mostrado.

srun $srunParam ./nqueens $GPU 15 » $OutPath/NQUEENS.txt &

Tras presentar las máquinas de experimentación que se han empleado, y la configuración adicional a realizar sobre las ejecuciones de los benchmarks en la máquina, presentamos las entradas usadas. Como comentamos anteriormente, estas se tratan de las entradas particulares de cada benchmark, para las experimentaciones de los input sets *Test*, *Train* y *Reference* planeadas. Estos se han de realizar no solo en la CPU del sistema, sino también en las distintas GPU que la máquina *Gorgon* tiene a su disposición

Benchmark	Entrada usada	Tipo de entrada
BFS	graph4096.txt	Fichero
BWA_HMM	-t 100 -v t	Comando
CFD	fvcorr.domn.097K	Fichero
CRC	crcfile_N50000_S2048_24-7-23-9-14	Fichero
CSR	csrmatrix_R1000_N800_D10000_S01_24-7-24-13-25	Fichero
FFT	--pts 2	Comando
GEM	Mb.Hhelix.bondi 80 1 0	Ambos
KMEANS	1275_47.txt	Fichero
LUD	512.dat	Fichero
NQUEENS	10	Comando
NW	-d 512 -p 23	Comando
SRAD	512 512 0 32 0 32 0.5 1	Comando
SWAT	query1K1 sampledb1K1	Fichero
TDM	sim-64-size50.csv ivl.txt 30-episodes.txt 256	Ambos

Tabla 5.1: Entradas de Input set *Test*

Benchmark	Entrada usada	Tipo de entrada
BFS	graph32M.txt	Fichero
BWA_HMM	-n 6500 -v n	Comando
CFD	fvcorr.domn.193K	Fichero
CRC	crcfile_N400000_S2048_24-7-26-10-58	Fichero
CSR	csrmatrix_R10000_N1024_D5000_S01_24-7-23-12-8	Fichero
FFT	--pts 5	Comando
GEM	nucleosome 80 1 0	Ambos
KMEANS	2000000_45.txt.txt	Fichero
LUD	12288.dat	Fichero
NQUEENS	19	Comando
NW	-d 40000 -p 11	Comando
SRAD	7584 7584 333 5537 849 6169 0.5 100	Comando
SWAT	query3K1 sampledb3K1	Fichero
TDM	sim-64-size200.csv ivl.txt 30-episodes.txt 256	Ambos

Tabla 5.2: Entradas de Input set *Train*

Benchmark	Entrada usada	Tipo de entrada
BFS	graph72M.txt	Fichero
BWA_HMM	-n 9500 -v n	Comando
CFD	missile.domn.0.2M	Fichero
CRC	crcfile_N600000_S4096_24-7-27-11-48	Fichero
CSR	csrmatrix_R20000_N1024_D5000_S01_24-7-29-9-46	Fichero
FFT	--pts 7	Comando
GEM	capsid 80 1 0	Ambos
KMEANS	4500000_45.txt	Fichero
LUD	16384.dat	Fichero
NQUEENS	20	Comando
NW	-d 42000 -p 500	Comando
SRAD	25072 25072 1585 10947 7523 15987 0.5 60	Comando
SWAT	query5K1 sampledb5K1	Fichero
TDM	sim-64-size500.csv ivl.txt 30-episodes.txt 256	Ambos

Tabla 5.3: Entradas de Input set *Reference*

Como ya mencionamos en una sección anterior, para medir los tiempos de cada benchmark en su correspondiente experimentación, se ha decidido realizar 30 repeticiones de las ejecuciones, y obtener la media de estas. De esta forma, reducimos el ruido en los resultados, y obtenemos mediciones que representan mejor el comportamiento de los benchmarks. Esta cantidad de repeticiones resulta en, aproximadamente, 6300 ejecuciones a realizar en la máquina; traduciéndose, a su vez, en alrededor de 50 horas de experimentación.

En las próximas secciones describiremos los distintos experimentos realizados.

5.1. Experimentación en distintas arquitecturas hardware

En estos experimentos, se ejecutan todos los benchmarks con todos los input sets desarrollados, para los distintos tipos de arquitecturas hardware disponibles: CPU y GPU. La experimentación se realiza en todas las arquitecturas hardware presentes en *Gorgon*: una CPU, y tres GPUs. Con esta experimentación, podemos analizar el comportamiento de la benchmark suite en arquitecturas hardware de distinta naturaleza, de forma acorde a su propósito original.

A continuación, pasaremos a mostrar los tiempos medios que hemos obtenido de cada input set de las ejecuciones de los benchmarks. Con el fin de clarificar las gráficas de tiempos que mostraremos, el eje Y representarán los tiempos de ejecuciones, en segundos, mientras que el eje X identifica cada uno de los benchmarks existentes. Para cada una de las gráficas proveeremos los tiempos medios de CPU y las GPUs de *Gorgon*, siendo estas la Nvidia RTX 4500 (GPU1), Nvidia A100 (GPU2) y la AMD W7800 (GPU3).

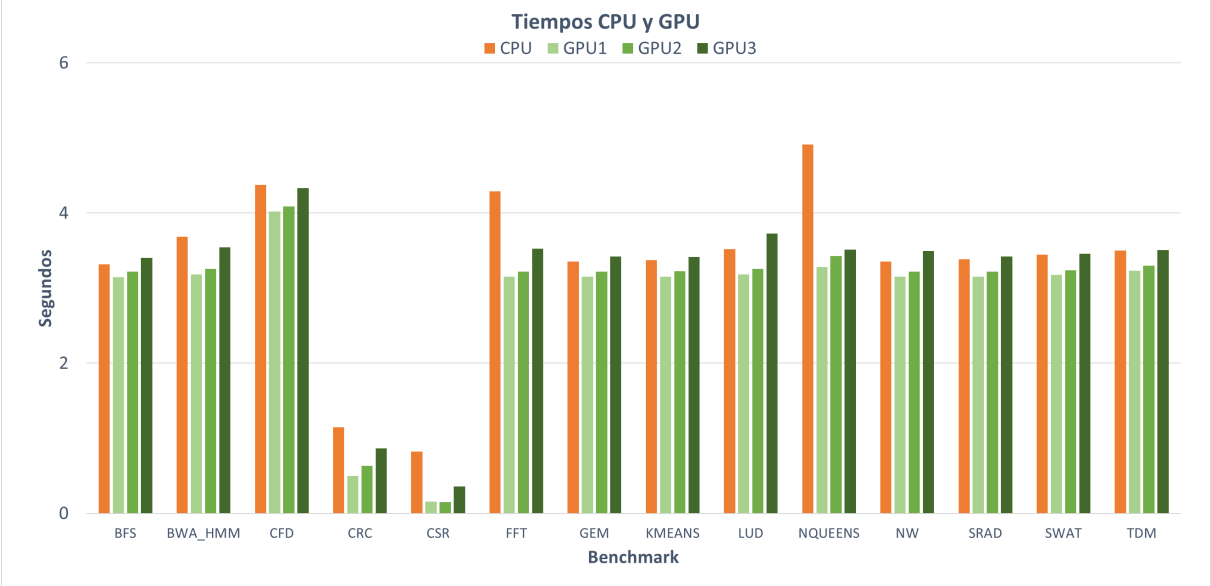

Figura 5.1: Tiempos medios del input set *Test*

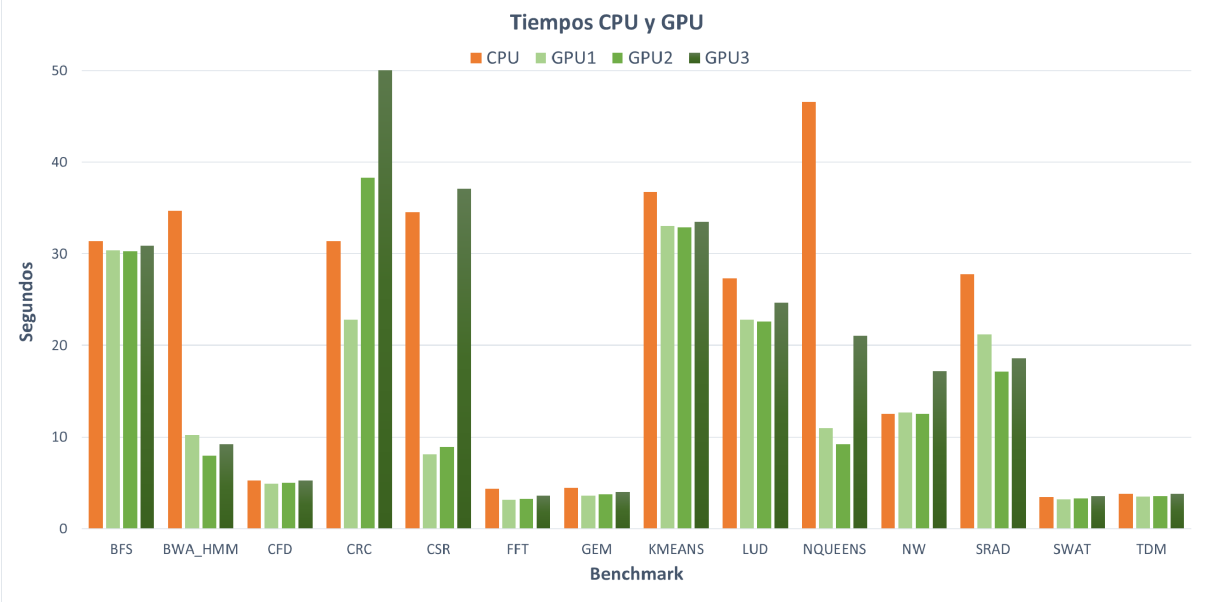

Figura 5.2: Tiempos medios del input set *Train*

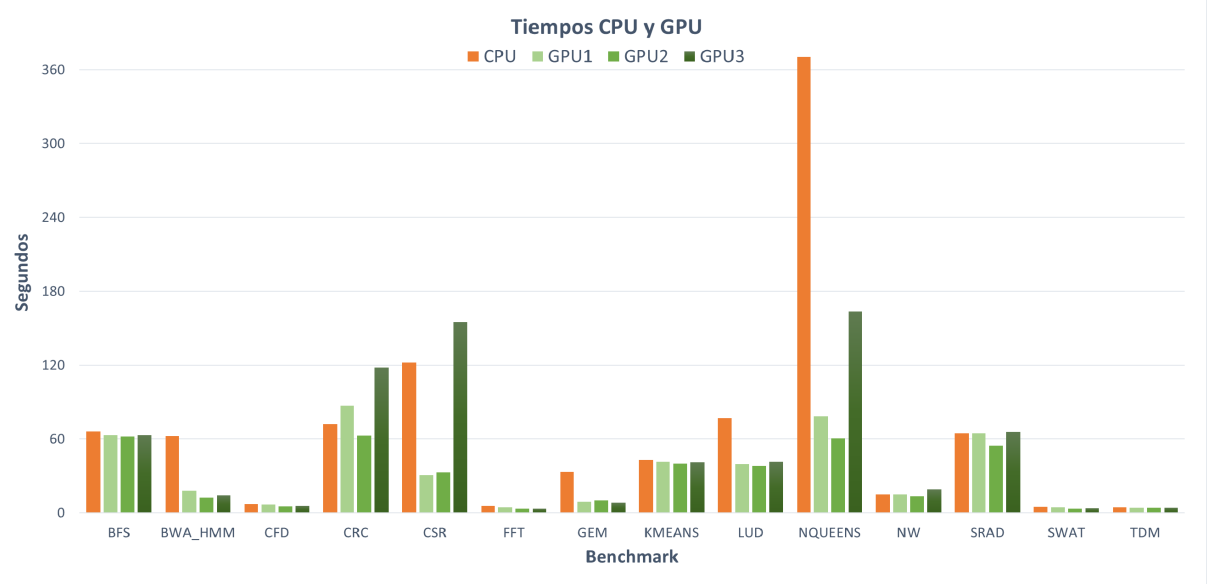

Figura 5.3: Tiempos medios del input set *Reference*

Tras presentar esta media de tiempos obtenidos, nos gustaría mostrar, adicionalmente, el *speedup* de las ejecuciones de cada benchmark. Este nos servirá para mostrar mejor las variaciones de tiempos entre las ejecuciones de las distintas GPUs con respecto a la CPU. Adicionalmente, se añadió a la gráfica, un último campo (Geo Mean Speedup), que refleja la media geométrica de los speedups de cada benchmark.

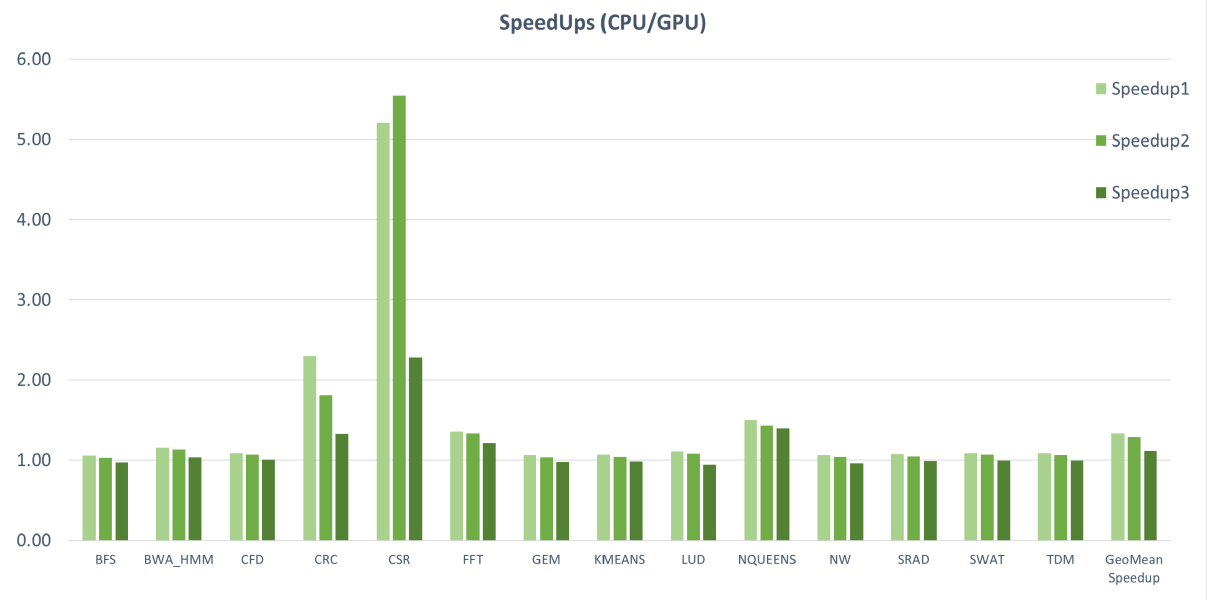

Figura 5.4: Speedups del input set *Test*

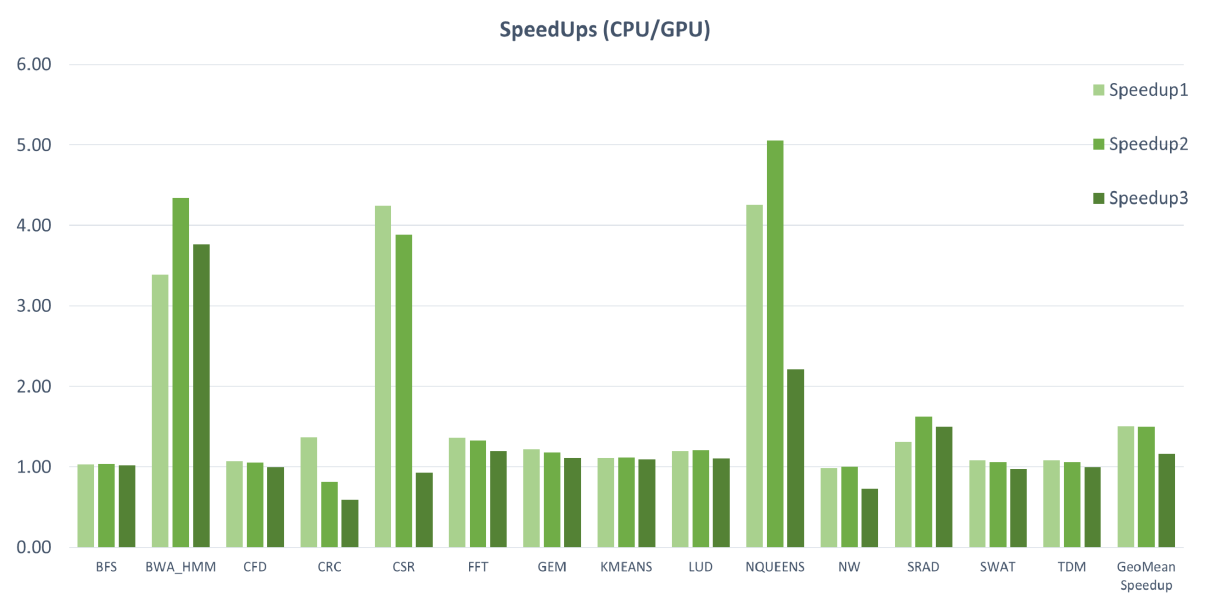

Figura 5.5: Speedups del input set *Train*

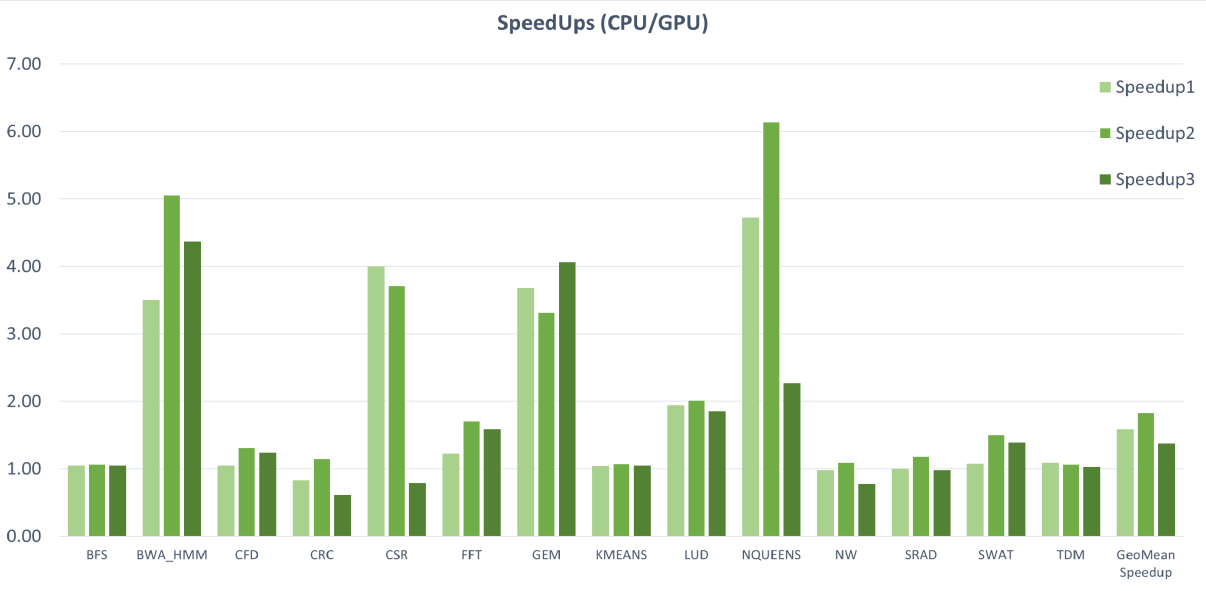

Figura 5.6: Speedups del input set *Reference*

Tras mostrar los resultados de tiempos y speedup de esta experimentación, extraemos las siguientes conclusiones:

- **Las ejecuciones de los benchmarks son más rápidas en GPU que en CPUs.** Esto es algo que esperamos obtener de las ejecuciones y, ocurre en una buena parte

de los benchmarks. Aun así, en muchos de estos, la diferencia es tan pequeña que no es apreciable. Esto puede deberse a diferentes factores. En primer lugar, cabe recordar que las versiones de CPU no se corresponden con la ejecución secuencial en un único hilo. Como OpenCL no nos ofrece la posibilidad de elegir manualmente los hilos a utilizar, la versión de CPU de cada benchmark se ejecuta sobre todos los hilos disponibles, que en Gorgon son 256, lo que hace más difícil obtener speedups con GPUs. Por otra parte, los distintos benchmarks son tareas que presentan distinto grado de paralelismo, por lo que su eficiencia no escala de la misma forma con el número de hilos de ejecución. Alternativamente, también puede ser debido a que el código paralelo que se ejecuta podría optimizarse aún más, resultando en un mejor uso de las capacidades de los sistemas de ejecución.

- **Inconsistencias o anomalías en los speedups entre experimentos:** Como hemos visto en las gráficas, la variación de tiempos entre ejecuciones en CPU y ejecuciones en GPU, así como variaciones en los speedups, va ampliándose a medida que aumenta el tamaño del problema a resolver. Esto puede deberse a que, en los tamaños mayores, existe un mayor número de tareas potencialmente paralelizables, lo que permite aprovechar mejor las capacidades de las GPUs utilizadas. En la siguiente sección, se profundizará en estos temas.

5.2. Experimentación de escalabilidad

En estos experimentos, se ejecutan todos los benchmarks con exclusivamente con el input set de *Train*, en CPU, variando la cantidad de *threads* a utilizar. Con esta experimentación, podemos analizar la escalabilidad de las distintas benchmarks de la suite. De esta forma, pretendemos profundizar en el comportamiento general de los benchmarks ante sistemas computacionales con distinta cantidad de recursos, así como tratar de explicar ciertos comportamientos detectados con los experimentos tratados en la sección anterior.

El comando de *srun* de *SLURM* nos provee una opción con la que poder determinar el número de *threads* disponible a un programa durante su ejecución,*--cpus-per-task=N*. Hemos decidido realizar las ejecuciones haciendo uso de esa opción, para emplear 1, 32, 64, 128 y 256 *threads* para las ejecuciones.

Esta experimentación se realiza exclusivamente con el input set de *Train* debido a limitaciones de tiempo: se estimó que la ejecución de la misma experimentación (incluyendo todas las repeticiones para cada benchmark) con el input set de *Reference* sería tan elevado que superaría el tiempo planificado de uso de las máquinas clúster. Esta estimación se justifica basándose en que, al ser experimentos que hacen uso de menos hilos de CPU, se esperaban tiempos mayores que los obtenidos en los experimentos descritos en la sección 5.1. Por los mismos motivos, se decidió prescindir de experimentos que hicieran uso de menos de 64 hilos de CPU (exceptuando los que utilizan 1 hilo, que se usan como referencia secuencial).

En las gráficas de tiempos de este experimento, hemos decidido emplear una escala logarítmica en base 2 para facilitar la lectura y ver mejor las diferencias de tiempos obtenidos.

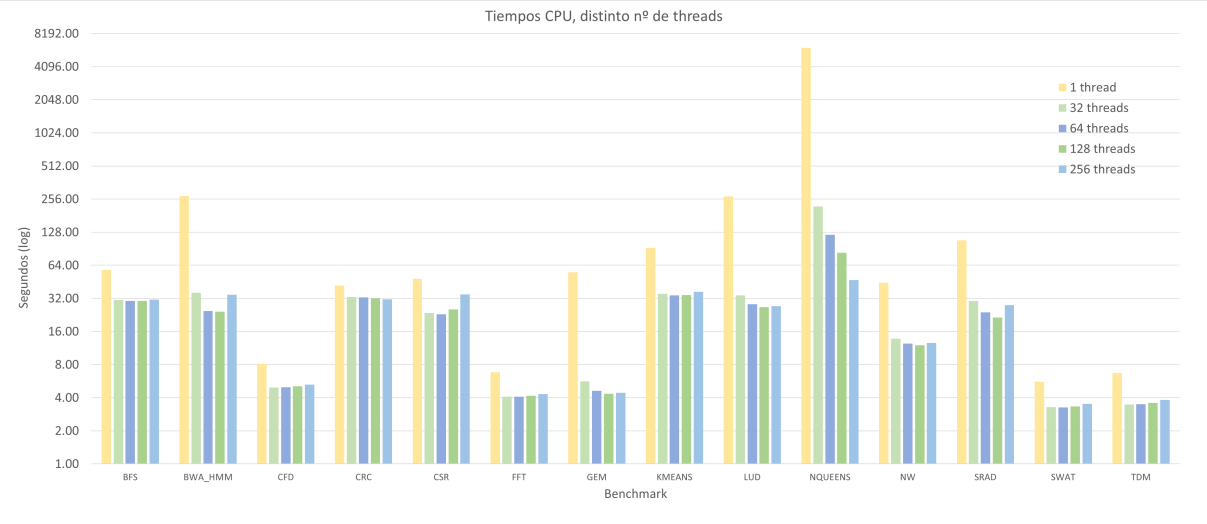

Figura 5.7: Escalabilidad de CPU (tiempos)

Tras presentar esta media de tiempos obtenidos, nos gustaría mostrar, adicionalmente, el *speedup* de las ejecuciones de cada benchmark, con respecto a la ejecución en 1 hilo. Estas gráficas también incluyen un campo, al final, que refleja la media geométrica de los speedups de cada benchmark (Geo Mean Speedup).

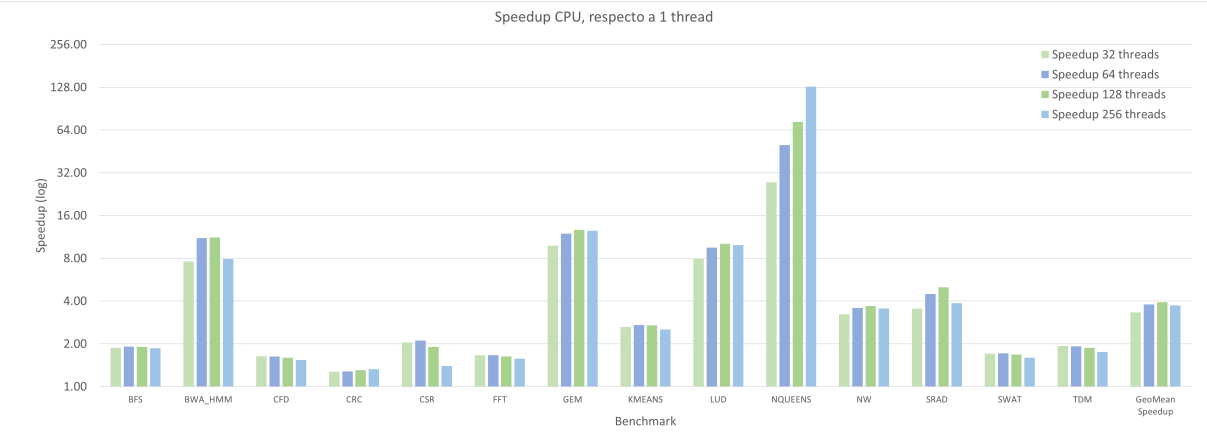

Figura 5.8: Escalabilidad de CPU (speedup)

Tras mostrar los resultados de tiempos y *speedup* de esta experimentación, extraemos las siguientes conclusiones:

- **La mayoría de benchmarks no presentan una escalabilidad ideal.** La benchmark que mejor escala es *NQUEENS*, que obtiene un speedup de $\times 128$ al utilizar 256 hilos. La mayoría de benchmarks, sin embargo, presentan una escalabilidad considerablemente más baja. Esto parece indicar que la mayoría de los *dwarfs* (las distintas benchmarks) no presentan un grado de paralelismo alto.
- **Para algunas benchmarks, maximizar el número de hilos usados incrementa el tiempo de ejecución.** Los casos más notorios de este fenómeno son *BWA_HMM* y *CSR*. Esto puede deberse a varios factores: optimización subóptima de los códigos, desbalanceo de carga entre los distintos hilos CPU, tamaño de tareas demasiado pequeño como para compensar el sobrecoste asociado a generar y coordinar los hilos, contenciones en los accesos a memoria... Para determinar la causa exacta en cada benchmark se requeriría un análisis más profundo, haciendo uso de herramientas de *profiling*. Tal análisis, sin embargo, no entra dentro de los objetivos de este trabajo. En cualquier caso, estos fenómenos refuerzan la idea de que algunos de los *dwarfs* parecen presentar características que inherentemente dificultan su paralelización.

6: Conclusiones y trabajo futuro

Para acabar con el documento, expondremos las conclusiones obtenidas a lo largo de todo este proyecto, y comentando trabajos adicionales que deberá de hacerse para mejorar la benchmark suite, de cara a futuro.

En cuanto a nuestras conclusiones sobre lo realizado al proyecto, se han realizado los siguientes cambios:

- Se ha realizado la refactorización de la benchmark suite, solucionando problemas más agravantes que se detectaron, durante la evaluación preliminar de la suite. Se modificaron los códigos que forman el benchmark, de modo de asegurar su compatibilidad con la versión 11.0 del compilador GCC/G++.
- Se añadieron múltiples generadores de ficheros, bien creados a mano o haciendo un "port" de aquellos que se encontraron en otras benchmark suite, incorporándolos en los respectivos benchmarks que forman la suite *OpenDwarfs*.
- Se ha documentado extensamente el propósito de cada benchmark de la suite, así como todos los cambios realizados, para facilitar la labor de los futuros desarrolladores que utilicen la nueva suite para sus propios fines.

Una vez resumidas las labores realizadas a lo largo del proyecto, nos gustaría comentar aspectos de futuro trabajo, que consideramos que deberían llevar a cabo con el fin de mejorar el estado actual de la benchmark suite. Los trabajos que principalmente discutiremos son aquellos que no han podido realizarse con mayor profundidad en nuestro proyecto, o que, por el contrario, no pudieron llevarse a cabo por escasez de tiempo y conocimientos. Los trabajos que podrían realizarse en un futuro mantenimiento, serían:

- **Poder modificar la cantidad de recursos de ejecución.** Debido a falta de tiempo y falta de conocimientos de la API de *OpenCL*, esto no ha sido posible llevar a cabo en este proyecto. El propósito de este cambio sería poder permitir que los futuros usuarios de la benchmark suite puedan seleccionar la cantidad de recursos

del sistema informático a poder asignar durante las ejecuciones de estos benchmarks. De acuerdo a lo investigado y probado en lo relacionado con *OpenCL*, la única forma de poder modificar la cantidad de recursos, sin apenas modificación de la estructura del código, sería pudiendo alterar el tamaño que se asigna de *LocalWorkSize* que se puede asignar a los *kernels* de ejecuciones. Como trabajo futuro, se debería poder dar la opción al usuario a tener mayor control sobre los campos asociados a estos aspectos de ejecuciones de *OpenCL*. Esto permitirá aumentar las capacidades y versatilidades de evaluación del sistema a probar con la benchmark suite.

- **Asegurar compatibilidad con sistemas FPGA.** Esto es otro aspecto que debería de verificarse de cara a futuro. Originalmente, la benchmark suite estuvo pensada para que funcionase sobre FPGAs de Altera, con el fin de garantizar aún más las capacidades heterogéneas de la benchmark suite. Dado que, durante el proceso de refactorización, se ha establecido el uso de la versión 1.1 de *OpenCL*, esta debería de seguir siendo compatible, pero, debido a la falta de recursos que poder dedicarle a este proyecto, no pudo llegar a garantizarse que los cambios realizados fuesen realmente compatibles con dichos sistemas. Es por ello que otro trabajo que debería de realizarse de cara a futuro es verificar esta compatibilidad con este tipo de sistemas, con el fin de que la benchmark suite siga cumpliendo con la filosofía de computación heterogénea de sus autores originales. Esto consistiría en hacer diversas pruebas y ejecuciones de los benchmarks, sobre FPGAs de distintos vendedores, (Altera, Xilinx, Digilent, etc.), para asegurar la compatibilidad de dispositivos con la benchmark suite.

- **Ampliar las entradas disponibles a usar con los benchmarks**. Esto es una tarea que pudo llevarse a cabo para un conjunto de benchmarks de *OpenDwarfs*, a lo largo del proyecto, y que debería de ampliarse y mejorarse. Esta tarea adicional tuvo como objetivo crear diversos generadores de ficheros, los cuales pueden ser empleados para crear ficheros que usar en sus respectivos benchmarks. Como comentamos previamente, aún existen varios que, por falta de tiempo, así como por escasa documentación de las entradas de estos benchmarks, no pudieron hacerse para todos y cada uno de estos benchmarks que requieren de estas entradas. El trabajo futuro, que se trata de expresar en este punto, es que debería continuarse con esta tarea, con el fin de aportar a cada benchmarks que necesitase de ficheros de entradas, un generador válido y compatible con cada benchmark. Adicionalmente, se debería revisar los ya incorporados a lo largo de este proyecto, y los proporcionados por los autores originales, en caso de existir errores no localizados durante este proyecto.

- **Corrección de comportamientos anómalos de los benchmarks**. Esta tarea, a realizar de cara a futuro, consistiría en corregir posibles problemas de comportamiento existentes en los benchmarks de *OpenDwarfs*, y que no pudieron abordarse en este proyecto, debido a falta de tiempo y conocimientos. Ejemplos de estos problemas que hemos encontrado durante el proyecto serían problemas como el encontrado en el benchmark de *NEEDLE*. Este benchmark, como comentamos en capítulos anteriores, daba problemas a la hora de especificarle una entrada superior al tamaño 42000.

Se consideró inicialmente que podía deberse a falta de memoria RAM en la que poder llevar a cabo ejecuciones, pero, como hemos comentado en la especificación de nodo *Gorgon*, descartamos que fuese el caso, ya que el dicho nodo tiene a su disposición un TB de memoria RAM, aunque mensajes de errores en *Gorgon* a veces sugieran lo contrario. Debido a esto, no pudo ampliarse el tamaño de las entradas para los experimentos, ya que causaban este fallo, el cual no pudo llegar a corregirse. Aunque bajo nuestras pruebas, apenas encontramos este tipo de problemas en otros benchmarks, debería realizarse una revisión en mayor profundidad a cada uno de los benchmarks, con el propósito de tratar de detectar la existencia de errores más grandes y profundos, en cada uno de estos.

En resumen, consideramos que la refactorización de la nueva suite supone un importante paso adelante en su usabilidad y que su estado actual permite su uso en la medición de rendimientos. Acometer el trabajo futuro aquí descrito permitirá extender su uso a más contextos de evaluación.

Apéndices

Apéndice A

Glosario del documento

- **Benchmark suite**: Este término se refiere a una colección de programas presentes en un proyecto de benchmarking. El propósito de cada uno de los benchmarks, presentes en una benchmark suite, es el de evaluar cierta aspecto de computación, presentes en diversos sistemas, y proveer una valoración objetiva de la capacidad de un sistema a evaluar, juntando resultados de los distintos benchmarks. Cabe destacar que, en muchos casos, los benchmarks no son desarrollados todos por una misma, o grupo de personas, y suelen ser más adaptaciones de programas ya existentes, de modo que puedan funcionar de forma similar, dificultando así en muchos casos la tarea de mantenimiento de los diversos benchmarks.

- **Benchmark**: Con este término nos referimos uno o varios programas los cuales han sido diseñados con el propósito de analizar algunos aspectos específicos, o capacidades de computación, del sistema en el que se está ejecutando dicho programa.

- **Speedup:** Se trata de una comparación de tiempos de ejecución, entre 2 sistemas, dados un mismo programa a ejecutar con las mismas entradas. Este se calcula tomando el tiempo de ejecución de un programa en un sistema A, y dividiéndolo sobre un nuevo tiempo de ejecución, del mismo programa, sobre un sistema B.

- **CPU**: Es una abreviatura para referirse al procesador de un sistema informático (Central Processing Unit o Unidad Central de Procesamiento). Este es uno de los componentes vitales de cualquier sistema, ya que es el encargado de interpretar y realizar toda petición o instrucción demandada al sistema.

- **GPU**: Es una abreviatura, la cual se refiere a las tarjetas gráficas de un sistema informático (*Graphical Proccessing Unit* o *Unidad de Procesamiento Gráfico*). En su concepción original, la utilidad de estos componentes era la de poder generar salidas de imágenes y visualizaciones a través de periféricos externos, como los monitor. En los últimos años, se ha descubierto que, gracias a la capacidad de estas para llevar a cabo operaciones matemáticas, y su alto grado de paralelización, se les está dando usos como el entrenamiento de redes neuronales y diversidad de simulaciones.

- **Sistemas FPGA**:Abreviado del término *Field Programmable Gate-Array*, las FPGAs son sistemas flexibles que puedan ser reconfigurados para adaptarlo a la realización de tareas específicas. Mientras que la lógica de funcionamiento de las CPUs y GPUs está definida por los fabricantes de estos sistemas *(Firmware)*, así como el repertorio de instrucciones que emplean para el uso de dicha lógica, los sistemas FPGAs carecen de lógica de funcionamiento. De modo que es responsabilidad del programador de implementar dicha lógica, necesaria para poder emplear y aprovechar las capacidades de este tipo de sistemas. Asimismo, estas también son sistemas físicos muy asequibles de fabricar y obtener.

- **Arquitectura MIC**: Abreviatura de los términos *Many Integrated Core Architecture*, estos son sistemas que se caracterizan por poseer una arquitectura única, desarrollada por la compañía de Intel, sistemas que realizasen gran cantidad de tareas, como supercomputadoras y servidores. Estas fueron, fundamentalmente, un elemento complementario a dichos sistemas, a través del uso de ranuras PCIe del sistema, actuando como una segunda unidad de procesamiento, y ayudando al procesador principal de la máquina en cuestión, a realizar tareas de mayor carga. Desafortunadamente, esta fue abandonada y descontinuada por Intel en 2017 [48] [49].

- **Hardware:** Esta término se emplea para referirse a cualquier componente físico contenido en un sistema informático, y que realiza una función necesaria para el uso de dicho sistema: ejemplos de hardware serían la CPU, placa base del sistema, periféricos, tarjeta gráfica, etc.

- **Software:** Este término se emplea para describir la lógica e interpretación de mensajes de comunicación entre los distintos componentes informática, así como los diversos programas que emplean el hardware para sus tareas. Ejemplos de software serían los sistemas operativos (Windows, iOS, Linux), así como cualquier programa que se use en un sistema informático.

- **Middleware:** Se una colección de software intermediario, o pila de software, cuyas funciones son: gestionar y distribuir de forma correcta y eficiente los recursos que posee el sistema; facilitar la escalabilidad del sistema, cuando a este se le decida añadir nuevo hardware; y una interfaz única de acceso y control de estas máquinas, entre otras tarea.

- **Máquina Clúster**. Estos son una colección de hardware informático, así como de software, con el fin de crear un sistema informático muy potente con el propósito de actuar como una solla máquina, capaz de realizar tareas de forma mucho más eficiente que un sistema informático común.

- **Clúster HTC *(High Throughput Computing)***. Estos clústeres se diseñan con el propósito de llevar a cabo múltiples tareas de forma concurrente y de la forma más eficiente posible. Ejemplos de estos serían los clústeres que se construyen para proveer servicio a una gran cantidad de personas, como los motores de búsquedas en internet, o la computación en la nube (Cloud Computing).

- **Clúster HPC *(High Performance Computing)***Estos son diseñados con la finalidad de llevar a cabo una única tarea, de la forma más rápida posible, empleando todos los recursos que el sistema dispone, para realizar dicha tarea demandada. Este tipo de clústeres se emplean, principalmente, en tareas de carácter científico, como son la realización de simulaciones, el análisis de un gran conjunto de datos (minería de Big Data), o el entrenamiento de sistemas de inteligencia artificial (Machine Learning, Deep Learning).
- **APIs**: Este es un término comúnmente empleado en el campo de la informática, el cual se refiere a aquellos métodos o funciones, que provee una librería, como es OpenCL, con los que permite al desarrollador abstraerse del funcionamiento interno de dicha librería, agilizando así el desarrollo de programas.

Bibliografía

[1] Desconocido. *Dwarf Mine*. Dic. de 2006. URL: http://view.eecs.berkeley.edu/wiki/Dwarf_Mine.

[2] Krste Asanović et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Inf. téc. UCB/EECS-2006-183. EECS Department, University of California, Berkeley, dic. de 2006. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf.

[3] W.Feng et al. «OpenCL and the 13 Dwarfs: A Work in Progress». En: *ACM Digital Library* (abr. de 2012). DOI: 10.1145/2188286.2188341. URL: https://dl.acm.org/doi/10.1145/2188286.2188341.

[4] Konstantinos Krommydas et al. «On the characterization of OpenCL dwarfs on fixed and reconfigurable platforms». En: *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. Jun. de 2014, págs. 153-160. ISBN: 978-1-4799-3609-0. DOI: 10.1109/ASAP.2014.6868650.

[5] Konstantinos Krommydas et al. «OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures». En: *Journal of Signal Processing Systems. Volume 85 Issue 3* (dic. de 2016). DOI: 10.1007/s11265-015-1051-z. URL: http://dx.doi.org/10.1007/s11265-015-1051-z.

[6] *GitHub: vtsynergy/OpenDwarfs*. URL: https://github.com/vtsynergy/OpenDwarfs.

[7] *The Khronos Group Inc.* URL: https://www.khronos.org.

[8] *The OpenCL™ Specification*. URL: https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html.

[9] *WebGL | Khronos Group*. URL: https://www.khronos.org/webgl/.

[10] *Vulkan | Khronos Group*. URL: https://www.vulkan.org.

[11] *SYCL | Khronos Group*. URL: https://www.khronos.org/sycl/.

[12] Shuai Che et al. «Rodinia: A benchmark suite for heterogeneous computing». En: *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*. IISWC '09. USA: IEEE Computer Society, 2009, págs. 44-54. ISBN: 9781424451562. DOI: 10.1109/IISWC.2009.5306797. URL: https://doi.org/10.1109/IISWC.2009.5306797.

[13] Shuai Che et al. «A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads». En: *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*. IISWC '10. USA: IEEE Computer Society, 2010, págs. 1-11. ISBN: 9781424492978. DOI: 10.1109/IISWC.2010.5650274. URL: https://doi.org/10.1109/IISWC.2010.5650274.

[14] Shuai Che et al. «Rodinia: A benchmark suite for heterogeneous computing». En: *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*. IISWC '09. USA: IEEE Computer Society, 2009, págs. 44-54. ISBN: 9781424451562. DOI: 10.1109/IISWC.2009.5306797. URL: https://doi.org/10.1109/IISWC.2009.5306797.

[15] *GitHub:yuhc/gpu-rodinia*. URL: https://github.com/yuhc/gpu-rodinia.

[16] *WayBack Machine*. URL: https://web.archive.org.

[17] *Virginia Technologies*. URL: https://www.vt.edu.

[18] Roland McGrath y Akim Demaille. *autoheader(1) - Linux man page*. URL: https://linux.die.net/man/1/autoheader.

[19] Free Software Foundation. *"GNU Software"*. URL: https://www.gnu.org/software/software.html.

[20] Pawan Harish y P. J. Narayanan. «Accelerating large graph algorithms on the GPU using CUDA». En: *Proceedings of the 14th International Conference on High Performance Computing*. HiPC'07. Goa, India: Springer-Verlag, 2007, págs. 197-208. ISBN: 3540772197.

[21] Andrew Corrigan et al. «Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware». En: *International Journal for Numerical Methods in Fluids* 66 (mayo de 2011), págs. 221-229. DOI: 10.1002/fld.2254.

[22] *Fast CRC32 | Stephan Brumme*. URL: https://create.stephan-brumme.com/crc32/.

[23] *Fast Fourier Transformation FFT - Basics*. URL: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft.

[24] Anthony Danalis et al. «The Scalable Heterogeneous Computing (SHOC) benchmark suite». En: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. GPGPU-3. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2010, págs. 63-74. ISBN: 9781605589350. DOI: 10.1145/1735688.1735702. URL: https://doi.org/10.1145/1735688.1735702.

[25] *Github: SHOC Suite*. URL: https://github.com/vetter/shoc.

[26] *Northwestern University*. URL: https://www.northwestern.edu.

[27] Cortés Rosas Jesús Javier et al. *Métodos de descomposición LU: Crout y Doolittles*. URL: https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema3/3-2_descomposicion_lu.pdf.

[28] *Google scholar - Liang Wang*. URL: https://scholar.google.com/citations?user=4a0-ZkoAAAAJ&hl=es.

[29] *LinkedIn - Liang Wang*. URL: https://www.linkedin.com/in/wangliang/.

[30] *Beyond3D*. URL: https://forum.beyond3d.com.

[31] Desconocido. *N-Queen Solver for OpenCL*. URL: http://forum.beyond3d.com/showthread.php?t=56105.

[32] *pcchen | Beyond3D*. URL: https://forum.beyond3d.com/members/pcchen.35/.

[33] Yongjian Yu y Scott Acton. «Speckle reducing anisotropic diffusion». En: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 11 (feb. de 2002), págs. 1260-70. DOI: 10.1109/TIP.2002.804276.

[34] *Heart Wall - Rodinia*. URL: https://www.cs.virginia.edu/rodinia/doku.php?id=heart_wall.

[35] Debprakash Patnaik et al. «Accelerator-Oriented Algorithm Transformation for Temporal Data Mining». En: *2009 Sixth IFIP International Conference on Network and Parallel Computing*. 2009, págs. 93-100. DOI: 10.1109/NPC.2009.26.

[36] Ponce Sean et al. «Towards Algorithm Transformation for Temporal Data Mining on GPU». Tesis doct. Virginia Tech, ene. de 2009.

[37] Yong Cao et al. «Towards chip-on-chip neuroscience: fast mining of neuronal spike streams using graphics hardware». En: *Proceedings of the 7th ACM International Conference on Computing Frontiers*. CF '10. Bertinoro, Italy: Association for Computing Machinery, 2010, págs. 1-10. ISBN: 9781450300445. DOI: 10.1145/1787275.1787277. URL: https://doi.org/10.1145/1787275.1787277.

[38] Wu-chun Feng. *VirginaTech: Wuchun Feng*. 2017. URL: https://people.cs.vt.edu/feng/.

[39] *Intel® FPGA SDK for OpenCL™ - Support Center*. URL: https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-guidance/opencl-software-support.html.

[40] Raymond Chen. *The sad history of the C++ throw(...) exception specifier*. URL: https://devblogs.microsoft.com/oldnewthing/20180928-00/?p=99855.

[41] *SPEC: Standard Performance Evaluation Corporation*. URL: https://www.spec.org/benchmarks.html.

[42] Neus Canal Díaz. «Distribuciones de probabilidad. El teorema central del límite». En: *Métodos estadísticos para enfermería nefrológica*. Ed. por Rodolfo Crespo Montero Antonia Guillén Serra. Sociedad Española De Enefermería Nefrológica SEDEN, jul. de 2006. Cap. 8, págs. 107-120. ISBN: 8468950106, 9788468950105. URL: https://formacion.seden.org/publicaciones_revistadet.asp?id=122.

[43] *La ley de Chargaff*. URL: https://www.biointeractive.org/es/classroom-resources/la-ley-de-chargaff.

[44] *Chargaff's Base-Pairing Rules*. URL: https://flexbooks.ck12.org/cbook/ck-12-advanced-biology/section/8.3/primary/lesson/chargaffs-base-pairing-rules-advanced-bio-adv/.

[45] *Chargaff's Rules: First and Second Rules, Applications*. URL: https://microbenotes.com/chargaffs-rules/.

[46] *What is middlewware*. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-middleware/.

[47] *Gestor de colas SLURM: Quickstart*. URL: https://slurm.schedmd.com/quickstart.html.

[48] Anton Shilov. *Intel Discontinues Xeon Phi 7200-Series 'Knights Landing' Coprocessor Cards*. URL: https://www.anandtech.com/show/11769/intel-discontinues-xeon-phi-7200-series-knights-landing-coprocessor-cards.

[49] Juan Diego de Usera. *Intel dice adiós a Xeon Phi, que serán reemplazados por sus tarjetas gráficas*. URL: https://hardzone.es/2018/07/25/intel-adios-xeon-phi-reemplazados-tarjetas-graficas/.