UNIVERSIDAD CARLOS III DE MADRID

ANÁLISIS DE TECNOLOGÍAS HPC EN ENTORNOS DE COMPUTACIÓN HETEROGÉNEOS

Trabajo Fin de Grado

Autor: Daniel García Stelzner

Tutor: Luis Miguel Sánchez García

Septiembre 2013

Agradecimientos

Ante todo, a mis padres por haberme mostrado siempre apoyo y brindado la oportunidad de estudiar una carrera de ingeniería.

A mis compañeros de clase por haber estado junto a ellos en los 4 años de carrera de ingeniería tanto en los malos y buenos momentos.

A Raúl Rivas por estar siempre allí para charlar de cualquier cosa aunque yo esté viviendo en Alemania (gracias *Frau* Merkel por ofrecerme trabajo).

A todos los profesores por todos los conocimientos que me han proporcionado tanto en el aspecto académico como en el personal, ¡en especial a Luis Miguel Sánchez García por haberme ofrecido este Trabajo Fin de Grado!

Finalmente a mi novia por depositar su confianza en mí, apoyarme y darme ánimos durante todo el periodo de este proyecto, tanto en los buenos como en malos momentos que hemos pasado juntos.

¡Gracias!

Índice general

Índice de ilustraciones

Índice de tablas

Índice de requisitos

Gráficas de evaluación

1. Introducción

En este capítulo inicial se expone la visión general de este Proyecto Fin de Grado, titulado *ANÁLISIS DE TECNOLOGÍAS HPC EN ENTORNOS DE COMPUTACIÓN HETEROGÉNEOS*. Para ello se parte de una somera descripción de la historia de los lenguajes de programación Java y C, explicando su origen y desarrollo. Lo mismo se hará con *OpenCL* y *OpenMP*. Además algunas breves nociones sobre las tecnologías *TBB* y *CUDA* y como último apartado, una explicación de los análisis realizados en los dispositivos de desarrollo finalizándolo con sus conclusiones y el presupuesto del trabajo realizado.

A continuación de este capítulo se detallan el contexto de trabajo, lo que me ha motivado a realizar este Proyecto, así como los objetivos a los que se pretende alcanzar con el proyecto y, como punto final, una pequeña explicación sobre la estructura del documento.

1.1 Contexto de trabajo

Gordon E. Moore, nacido en San Francisco el 3 de enero de 1929, es cofundador de Intel y autor de la Ley de Moore publicada el 19 de abril de 1965. Dicha ley expresa que cada dos años aproximadamente, el número de transistores en un circuito integrado se duplica. Más tarde modificó la ley al afirmar que el ritmo bajaría y que la densidad de los datos se doblaría aproximadamente cada 18 meses. La consecuencia directa de la Ley de Moore es que los precios bajan al mismo tiempo que las prestaciones suben: la computadora que hoy tiene un valor *X* costará la mitad al año siguiente y estará obsoleta en dos años.

Desde dicha publicación hasta hoy en día se ha cumplido con dicha predicción. En 26 años, el número de transistores en un chip ha incrementado ¡3200 veces!

Pero la Ley de Moore dejó ya de tener importancia, ya que hoy en día es mucho más aplicable la Ley de Amdahl (del creador Gene Amdahl, arquitecto computacional nacido el 16 de noviembre de 1922 en Dakota del Sur). Esta Ley sostiene que al añadir más procesadores para trabajar sobre un problema, no se logra la productividad global esperada calculada mediante la suma de la productividad individual de cada procesador. Lo que tendrá efecto en su productividad global será la capacidad de poder disgregar el problema original en subproblemas, que pueden distribuirse en diversos procesadores. Esta Ley es importante ya que la estrategia actual para desarrollar computadoras más veloces consiste en aumentar los nodos de procesamiento paralelos, ya sea colocando más núcleos en chip, más chips en un

servidor y más servidores en un clúster. Pero para ello, el programador ha de escribir su código específicamente para poder utilizar este paralelismo.

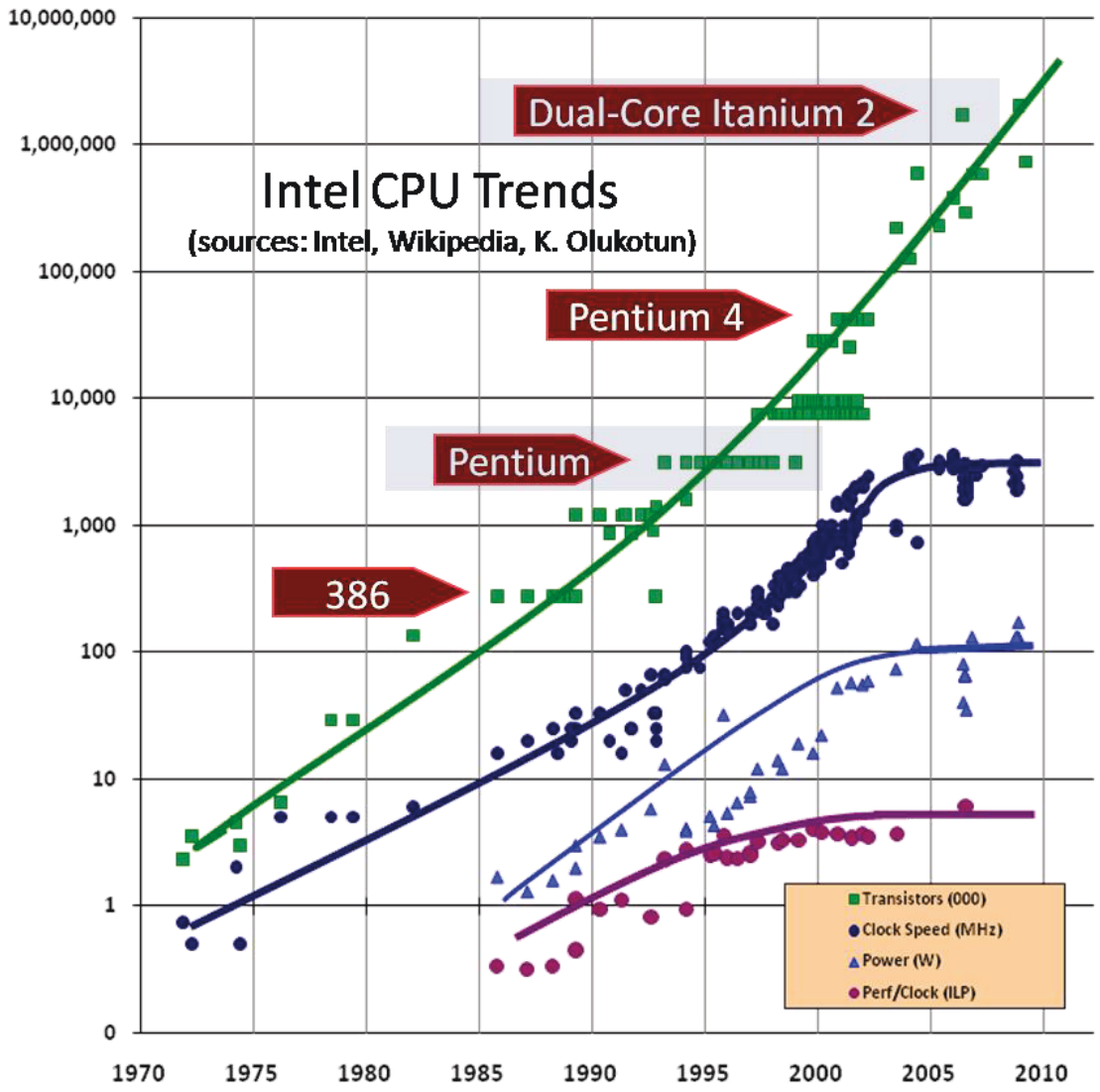


Figura 1: Ley de Moore

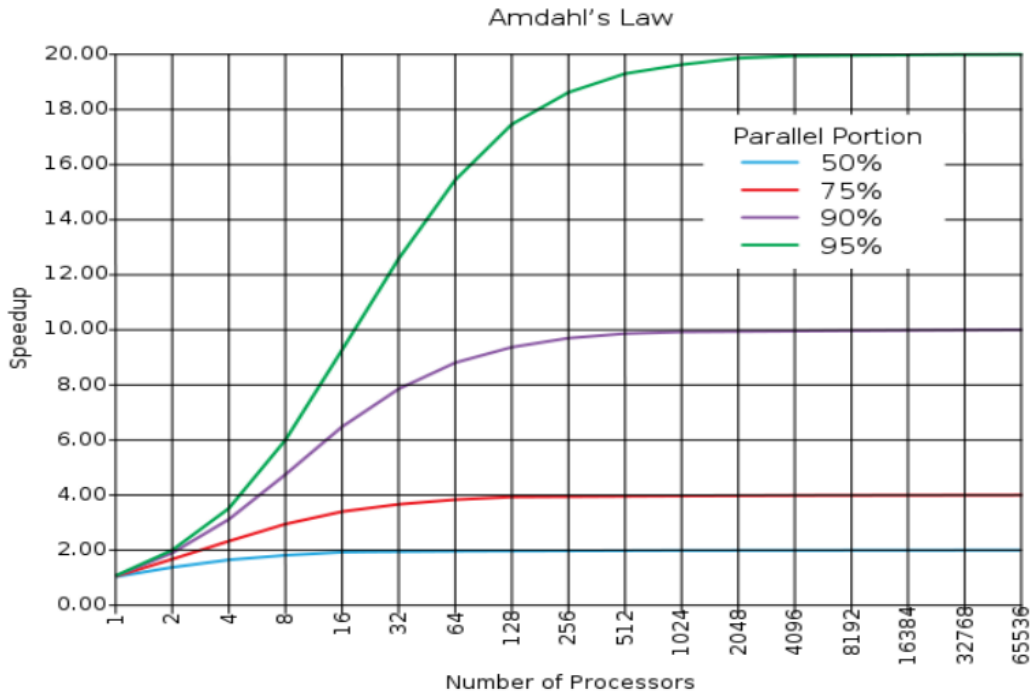


Figura 2: Ley de Amdahl

Herb Sutter es un experto en C++ que publicó un artículo denominado "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software" en el año 2005. Como bien dijo la Ley de Moore, cada 18 meses el número de transistores se duplica, pero ¿cuándo habrá un fin?

En dicha publicación el autor cita que la velocidad de los microprocesadores con procesamiento en serie está llegando a un límite físico (como la velocidad de la luz, no va a ir más rápido), el cual tiene dos consecuencias principales:

- Las fábricas de procesadores se centrarán en productos que soportan mejor *multithreading* (como los procesadores multi-cores)
- Los desarrolladores de software se verán obligados a desarrollar programas multiproceso para poder aprovechar mejor estos procesadores.

1.2 Objetivos

El objetivo principal de este proyecto Fin de Grado es realizar un estudio comparativo de *frameworks* de programación paralela de varios niveles (*thread level* y *SIMD*) usando diferentes sistemas *multicore* como lo son *OpenCL* y *OpenMP*. Estos *frameworks* serán evaluados usando distintos lenguajes de programación, como lo son C (cercano al lenguaje máquina) y Java (interpretado).

La motivación en realizar este Proyecto Fin de Grado consta en demostrar que los procesadores, tanto las CPUs como las GPUs, cuanto más actual, más rápido debido a la tecnología de hoy en día y que cada vez mejorada.

Los pasos realizados a lo largo del desarrollo del Proyecto han sido los siguientes:

- Estudio de la programación multiproceso *OpenMP*.
- Estudio del nuevo lenguaje de computación abierta *OpenCL*.
- Familiarización con el lenguaje.
- Implementación de los programas:
    - En lenguaje C:
        - En modo secuencial.
        - En la interfaz de aplicaciones *OpenMP*.
        - En la interfaz de aplicaciones *OpenCL*.
    - En lenguaje Java:
        - En modo secuencial.
        - En la interfaz de aplicaciones *OpenMP*.
        - En la interfaz de aplicaciones *OpenCL*.
- Ejecución de pruebas y análisis de los resultados.

1.3 Estructura del documento

En este apartado se detallan los diferentes capítulos que contiene este documento y en qué consisten:

- Apartado 1, Introducción: Aporta la idea general del proyecto, la motivación, los objetivos fundamentales que se pretende alcanzar y un glosario de términos.
- Apartado 2, Estado de la cuestión: Una pequeña visión general de la tecnología utilizada durante la realización de este proyecto.
- Apartado 3, Marco regulador.

- Apartado 4, Análisis de requisitos: Muestra los requisitos funcionales y no funcionales de los sistemas y de los programas.
- Apartado 5, Evaluación: Muestra de los resultados de las pruebas realizadas en las diferentes implementaciones y lenguajes.
- Apartado 6, Conclusiones: Las conclusiones generales sacadas del proyecto, las ideas extraídas de él y trabajos futuros que podría tener el utilizar los métodos realizados y utilizados.
- Apartado 7, Presupuesto: Detalle de los costes del proyecto.
- Apartado 8, Diagrama de Gantt: Muestra la evolución y los pasos que se ha realizado durante el proyecto.
- Apartado 9, Apéndices.
- Apartado 10, Anexos.
- Apartado 11, Bibliografía.

1.4 Glosario de términos

En la siguiente tabla se muestran los términos utilizados durante la memoria con su significado correspondiente:

Término	Significado
CPU	Central Processing Unit / Unidad Central de procesamiento
GPU	Graphics Processing Unit / Unidad de Procesamiento gráfico
OpenCL	Open Computing Language / Lenguaje de computación abierto
OpenMP	Open Multi-Processing / Programación multiproceso de memoria compartida
TBB	Threading Building Blocks / Biblioteca para la codificación de programas para sacar ventaja de los procesadores multinúcleo
CUDA	Compute Unified Device Architecture / Arquitectura Unificada de Dispositivos de Cómputo)
HPC	High Performance Computing / Computación de alto rendimiento
ISO	International Organization for Standarization / Organización Internacional de Normalización
ANSI	American National Standards Institute / Instituto Nacional Estadounidense de Estándares
API	Application Programming Interface / Interfaz de programación de aplicaciones
JVM	Java Virtual Machine / Máquina Virtual de Java
JRE	Java Runtime Environment / Entorno en Tiempo de Ejecución de Java
Framework	Estructura conceptual y tecnológica de

	soporte definido.
TLP	Thread Level Paralelism / Paralelismo a nivel de hilo

Tabla 1: Glosario de términos

2 Estado de la cuestión

En este capítulo se aportará una visión global de las tecnologías utilizadas durante la realización de este Proyecto Fin de Grado. Dado que se han utilizado dos lenguajes y varios métodos distintos en cada uno, se explicará durante los siguientes sub apartados las diferencias entre ellas.

2.1 Java vs C

El lenguaje de programación Java orientado a objetos fue desarrollado por *James Gosling*, *Mike Sheridan* y *Patrick Naughton* en junio de 1991. Originalmente fue diseñado para televisión interactiva, pero fue demasiado avanzado para aquél momento para la televisión por cable digital. El 23 de mayo de 1995 John Gage de *Sun Microsystems* y *Marc Andreessen*, cofundador y vicepresidente de *Netscape*, anunciaron la primera versión *alpha* en la conferencia *SunWorld '95*, que en aquel entonces solamente funcionaba en el sistema operativo Solaris, y que iba a ser incorporado en el navegador *Netscape*, el más utilizado de aquella época. En julio surgió la segunda versión *alpha* añadiendo así soporte para *Windows NT* y más tarde, en agosto, la tercera *alpha* con soporte para *Windows 95*. (1) (2)

El primer lanzamiento conocido como *Java 1.0* fue en enero de 1995 por la compañía *Sun Microsystems*, que a la vez fundó la empresa *Java Soft* para dedicarse al desarrollo de productos basados en Java y así trabajar con terceras partes para crear herramientas, aplicaciones, sistemas de plataforma y servicios para aumentar las capacidades del lenguaje. A lo largo del tiempo tuvo tres cambios de nombres: al principio se llamaba *Oak*, luego se cambió a *Green* y finalmente a lo que hoy en día se conoce con el nombre *Java*. En abril de 2009, la compañía *Oracle* adquirió *Sun Microsystems*, que hasta hoy en día sigue manteniéndolo, estando las versiones posteriores a la 6 bajo su control. (3)

Las versiones de Java que ha habido hasta ahora con sus fechas de lanzamiento son:

- JDK 1.0 (23 de enero de 1996)
- JDK 1.1 (19 de febrero de 1997)
- J2SE 1.2 (8 de diciembre de 1998)
- J2SE 1.3 (8 de mayo de 2000)
- J2SE 1.4 (6 de febrero de 2002)
- J2SE 5.0 (30 de septiembre de 2004)
- Java SE 6 (11 de diciembre de 2006)

- Java SE 7 (julio de 2011)

La parte positiva de todas estas versiones, es que siguen los mismos estándares de datos, eso significa que si alguien ha creado un programa con una versión antigua, puede ejecutarlo con una versión más nueva sin tener que cambiar nada.

El lenguaje Java se creó con cinco objetivos principales:

- Debe ser “simple, orientado a objetos y familiar”.
- Debe ser “robusto y seguro”.
- Debe ser “independiente de la plataforma y portable”.
- Debe poder ejecutarse con “gran rendimiento”.
- Debe ser “interpretado, que pueda crear hilos, y dinámico”

La palabra “orientado a objetos” es un paradigma de programación que representa conceptos como “objetos” que tienen campos de datos (atributos que describen el objeto) y los procedimientos asociados conocidos como métodos. Los objetos, que son en general las instancias de clases, se utilizan para interactuar el uno con el otro para aplicaciones de diseño y programas de ordenador.

Con “independiente de la plataforma” se refiere a que los programas escritos en este lenguaje pueden ejecutarse en cualquier tipo de hardware. De hecho, el axioma de Java es “*Write once, run anywhere*”.

Para poder utilizar Java en una máquina, es necesario tener instalado *Java virtual machine* (JVM), una máquina virtual de proceso nativo, que sea capaz de interpretar y ejecutar instrucciones expresadas en código *bytecode*, ya que las aplicaciones en Java están compilados en ese código (archivos con la extensión .*class* de Java). No sólo eso, sino que además hay que tener el JRE (*Java Runtime Environment*), que es el software necesario para poder ejecutar cualquier aplicación desarrollada para la plataforma Java.

```java
public class Hello_World {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }

}
```

Figura 1: Hello World en lenguaje de programación Java

En cambio, el lenguaje C nació mucho más antes que el lenguaje Java. Escrito por Ken Thompson, es la continuación y evolución del lenguaje de programación B, basado en BCPL del año 1960. El periodo más creativo fue en 1972 que es cuando tomaron el relevo *Brian Kernighan* y *Dennos Ritchie* en los laboratorios *Bell* de *AT&T* y escribieron todo UNIX en el lenguaje C. Se trata de un lenguaje de programación de medio nivel pero con muchas características de bajo nivel, con el que se pueden escribir programas con fines muy diversos.

Principalmente, las novedades que introdujeron fueron el diseño de tipos y estructura de datos. Los tipos básicos de datos eran *char, int, float* y *double*. Se añadieron más tarde los tipos short, long, unsigned y enumeraciones. Respecto a los tipos estructurados básicos son las estructuras, uniones y las matrices. Para controlar el flujo de ejecución de un programa, se utilizan las instrucciones if, for, while, switch-case. Además permite trabajar con direcciones de memoria, con funciones y soporta recursividad. (4)

Para que mientras tanto los usuarios y los desarrolladores puedan desarrollar en C antes de que se publicase un estándar oficial, *Dennos Ritchie* y *Brian Kernighan* escribieron un libro en 1978 denominado *The C Programming Language*, también conocido como “libro blanco” (*White book*) o como *K&R* (por los iniciales de los autores). En 1988 se publicó la segunda versión (5) (6) en el que se cubre el estándar ANSI C y en él se introducen varias características al lenguaje:

- El tipo de datos *struct*.
- El tipo de dato *unsigned int*.
- El tipo de dato *long int.*

- Los operadores *=+* y *=-* fueron sustituidos por *+=* y *-=* para eliminar la ambigüedad sintáctica de expresiones como *i=-5* ya que se podría interpretar como *i =- 5* o bien como *i = -5*.

En 1980, el lenguaje de programación C fue oficialmente estandarizado por el comité *ANSI X3J11*. Tras largos procesos, finalmente en 1989 se aprobó como *Lenguaje de Programación C ANSI X3 159-1989*, también conocido como el estándar *C89* o *ANSI C*. Un año más tarde el estándar *ANSI* pero adoptado por *ISO*, surgió el estándar *C90* o también *ISO/IEC 9899:1990*. En 1999 se llevó a cabo la publicación del estándar *ISO 9899:1999* con nuevas características como nuevas variables: long long int, un tipo de datos booleano y el tipo *complex* que representa los números complejos, además de *arrays* de longitudes variables, funciones nuevas como *snprintf*() y muchas más. 12 años más tarde, el 8 de diciembre de 2011 se publicó el último estándar *ISO/IEC 9899:2011* o también como *C11*. (7)

Una de las ventajas significativas de este lenguaje sobre otros, es que el código producido por el compilador C está muy optimizado en tamaño lo que redunda en una mayor velocidad de ejecución. Lo malo de C es que es independiente de la plataforma sólo en código fuente, lo cual significa que cada plataforma diferente debe proporcionar el compilador adecuado para obtener el código máquina que tiene que ejecutarse.

Las principales características de C son:

- Programación estructurada. Economía en las expresiones.
- Abundancia en operadores aritméticos lógicos como por ejemplo +, +=, ++, &, ~, etc.
- Tipos de datos que se pueden convertir implícitamente en otros.
- Codificación en alto y bajo nivel simultáneamente.
- Reemplaza ventajosamente la programación en ensamblador.
- Utilización natural de las funciones primitivas del sistema.
- No está orientado a ningún área en especial.
- Producción de código objeto altamente optimizado.
- Facilidad del aprendizaje.

En cambio, el lenguaje C no incluye algunas características nuevas que se encuentran en los nuevos y más modernos lenguajes de programación de alto nivel, como la orientación a objetos y el recolector de basura de Java.

```c
#include<stdio.h>

int main(void){

    printf("Hello World\n");

    return 0;

}
```

Figura 2: Hello World en lenguaje de programación C

2.2 Introducción del paralelismo

En tecnologías de la información, el paralelismo es una función que realiza el procesador para ejecutar varias tareas al mismo tiempo para así poder reducir el tiempo de ejecución. El funcionamiento es bastante sencillo, simplemente se divide el trabajo en tareas más pequeñas e independientes y cada hilo (*thread*) se encarga de ejecutar ese cacho de tarea.

Existen varios métodos de realizar paralelismo:

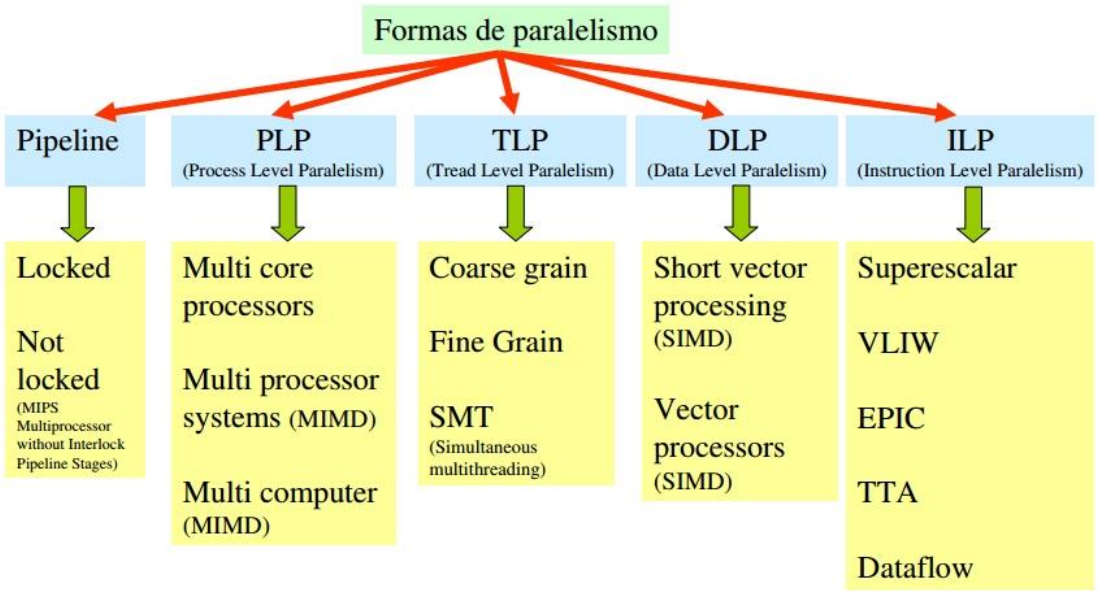


Figura 3: Formas de paralelismo

En este Proyecto Fin de Grado solamente se hablará del paralelismo a nivel de *thread* o hilo y sobre el SIMD.

2.2.1 Paralelismo a nivel de thread

Paralelismo a nivel de *thread*, también conocido como *TLP* (*Thread Level Parallelism*) permiten compartir las unidades de ejecución de un procesador entre los hilos independientes de un proceso. Para poder compartir, el procesador tiene que duplicar el estado de cada hilo: una copia del registro, del contador de programa y una tabla de páginas, todo por separado para cada hilo.

Como bien muestra la figura anterior, en *TLP* existen tres formas de realizar paralelismo: (8)

- Coarse Grain: En *coarse grain multi-threading* (grano grueso) los hilos son desalojados del procesador con baja frecuencia, usualmente cuando el hilo realiza alguna operación de entrada/salida o ante un fallo de caché.
- Fine Grain: En *fine grain multi-threading* (grano fino) el hilo en ejecución es cambiado (el denominado *thread swapping*) en cada ciclo de reloj.
- SMT: El *simultaneous multi-threading* es parecido al *fine grain*, pero permite ejecutar múltiples hilos en cada ciclo de reloj. Además permite concurrencia física, a diferencia de los anteriores que solamente manejan concurrencia virtual (multiplexado por división de tiempo).

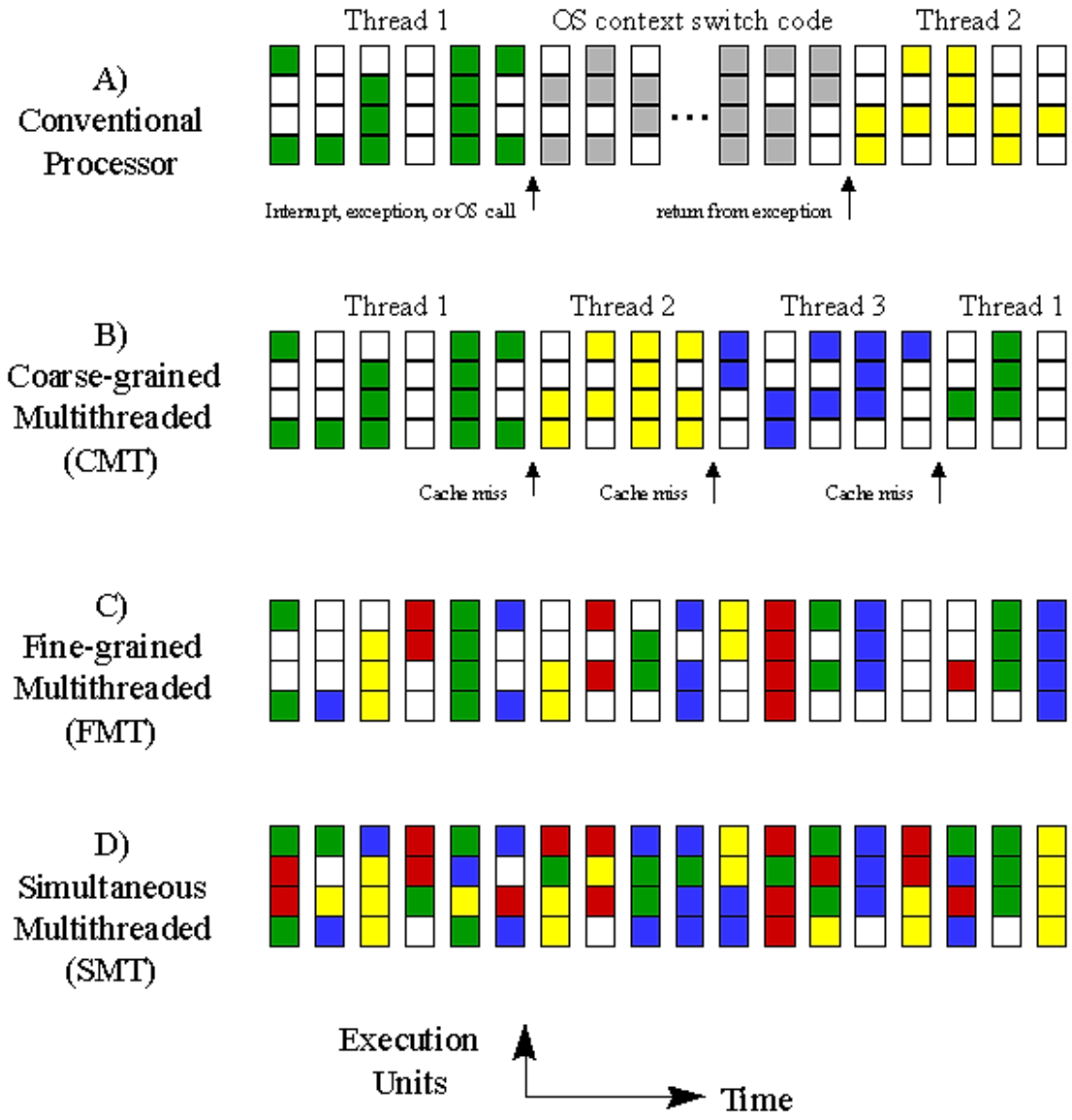


Figura 4: Formas de paralelizar en TLP

2.2.2 Paralelismo SIMD

Michael J. Flynn propuso en 1966 una clasificación de computadoras basada en la clasificación atendiendo al flujo de datos e instrucciones en un sistema. Por flujo de instrucciones se entiende como el conjunto de instrucciones secuenciales ejecutadas por un único procesador, mientras que por flujo de datos es el flujo secuencial de datos requeridos por el flujo de instrucciones. Con estos datos, *Flynn* clasifica los sistemas en 4 categorías:

- SISD (*Single Instruction, Single Data*)
- MISD (*Multiple Instruction, Single Data*)
- SIMD (*Single Instruction, Multiple Data*)

- MIMD (*Multiple Instruction, Multiple Data*)

En este Proyecto fin de Grado se estudiará solamente el *SIMD*.

El *SIMD*, flujo de instrucción simple y flujo de datos múltiple, consiste básicamente en instrucciones que aplican una misma operación sobre un conjunto de datos. Este tipo de arquitectura viene muy bien para la realización de operaciones de *array* o vectores ya que asignan cada elemento del vector a una unidad funcional diferente para procesamiento concurrente.

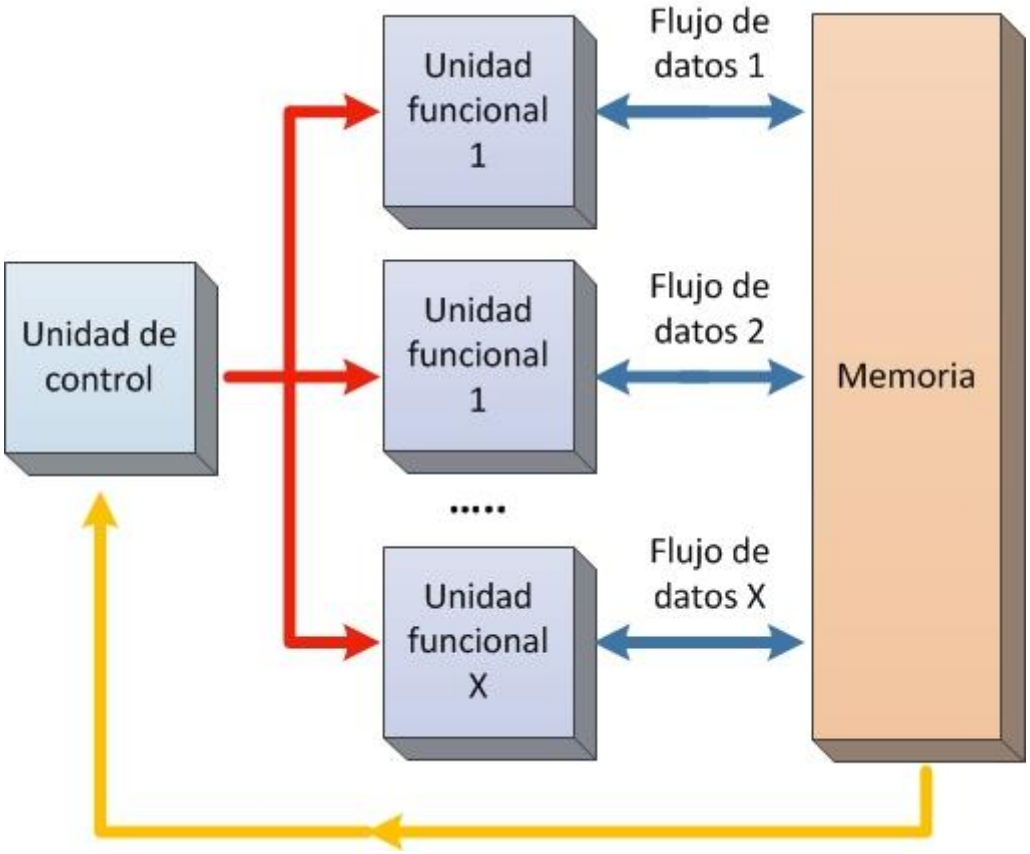


Figura 5: Funcionamiento de la arquitectura SIMD

Esta paralelización de vectores de datos se les llama también *procesadores matriciales.* Consiste en un conjunto de elementos de proceso y un procesador escalar que operan bajo una unidad de control. La unidad de control busca y decodifica las instrucciones de la memoria central y las manda, o bien al procesador escalar, o bien a los nodos procesadores dependiendo del tipo de instrucción. La instrucción que ejecutan los nodos procesadores es la misma simultáneamente y los datos son los que cada memoria de procesador tenga en ese momento, que son diferentes o también pueden ser iguales.

Todo este proceso funciona dividiendo los datos en fragmentos sobre las que se pueden realizar las mismas operaciones. En la siguiente figura por ejemplo, una matriz es dividida en 3 hilos:

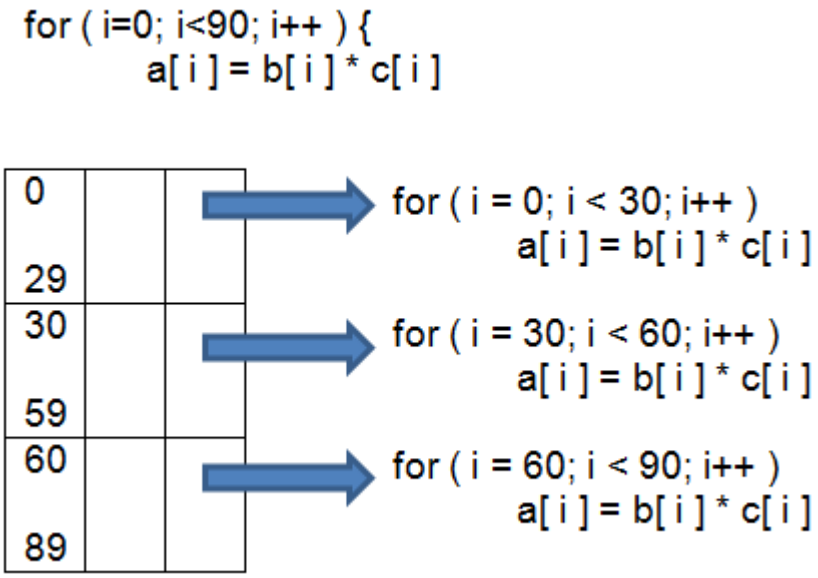


Figura 6: Ejemplo de paralelismo

Visto de otro modo:

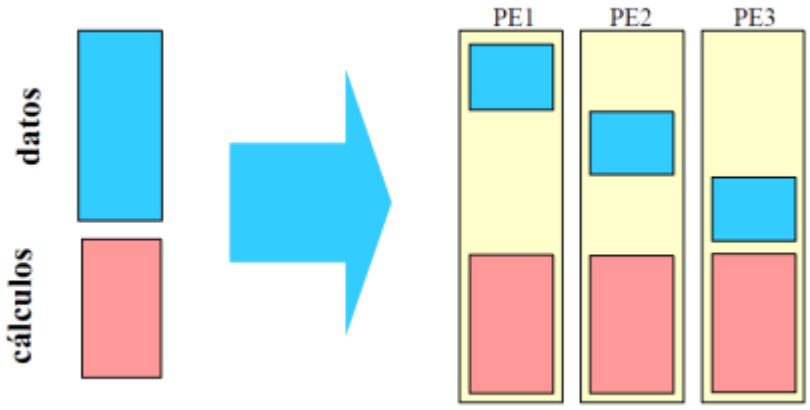


Figura 7: Ejemplo de paralelismo

2.3 OpenMP (threads)

OpenMP (*Open MultiProcessing*) es una interfaz de aplicaciones (*API*) para la paralelización sobre memoria compartida con multiprocesadores programando en C, C++ y Fortran,

soportado por todas las arquitecturas incluyendo plataformas como GNU/Linux, Windows, Solaris y Mac OS.

Las directivas de *OpenMP* facilitan al programador decidir qué parte del código tiene que ejecutar en paralelo y cómo distribuirlos entre los hilos. Para entender qué es una directiva de *OpenMP*, se trata de una instrucción en un formato especial que solamente es entendido por el compilador de *OpenMP*. (9)

Para la paralelización se utilizan *threads* o hilos de ejecuciones (tareas que se pueden ejecutar al mismo tiempo con otras tareas). Para ello, el sistema operativo crea un proceso para ejecutar el programa, que a la vez reparte algunos recursos para dicho proceso incluyendo el espacio de memoria y registros de los objetos. Un proceso puede consistir de múltiples hilos, cada uno de ellos teniendo su propio flujo de control, su propia pila y un contador de programa, pero compartiendo el mismo espacio de direcciones.

Está basado en el modelo *fork-join*. Un programa empieza con un hilo máster o único, que a la vez puede crear más en una sección o región paralela (parte del código que se va a replicar). Hay que tener en cuenta que el hecho de que los hilos compartan los recursos, los demás hilos pueden modificarlos también.

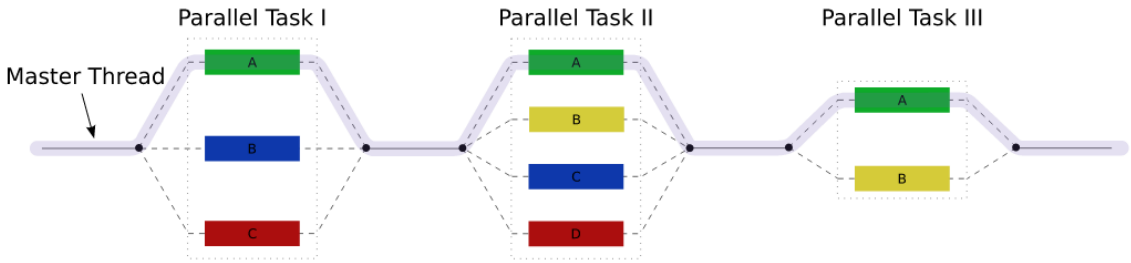


Figura 8: Funcionamiento de OpenMP

Cuando un hilo finaliza su bloque de código, existe una barrera implícita, lo cual quiere decir que cada hilo tiene que esperar al resto de los hilos a que terminen sus operaciones para que el hilo máster pueda seguir ejecutándose como se muestra en la figura anterior. Además permite la paralelización anidada, quiere decir cuando dentro de una región paralela se encuentra otra directiva que indica otra región paralela, se crean más hilos que ejecuten las tareas de esa región. La parte positiva de *OpenMP* es que es capaz de repartir de forma automática el trabajo entre los hilos creados.

Las versiones sacadas hasta hoy en día con sus respectivos años de lanzamiento son:

- OpenMP Fortran 1.0 (1997)
- OpenMP C/C++ 1.0 (1998)
- OpenMP Fortran 1.1. (1999)
- OpenMP Fortran 2.0 (2000)
- OpenMP C/C++ 2.0 (2002)
- OpenMP 2.5 (2005) Una única especificación para Fortran, C y C++
- OpenMP 3.0 (2008)
- OpenMP 3.1 (9 de julio 2011)
- OpenMP 4.0 (Julio de 2013)

Nota: todos los dispositivos de prueba para este Proyecto Fin de Grado utilizan el compilador GCC 4.6, por lo que se han seguido las especificaciones de la versión *OpenMP* v3.0 (10).

Existen varias directrices para poder paralelizar el código o parte del código. Para ello, se debe escribir una línea antes de la parte del código (en el lenguaje de programación C) *#pragma omp* (11) seguido de:

- parallel: en cada hilo se ejecuta lo mismo.
- for: parte el bucle de tal manera que cada hilo ejecute una parte del bucle.
- sections: indica qué parte del código se puede ejecutar en paralelo pero por un único hilo.
- single: indica qué parte del código se debe ejecutar únicamente en un hilo de todos los que se han lanzado y no tiene por qué ser el hilo padre.
- master: igual que el single pero éste es ejecutado por el hilo padre.

Condiciones de sincronización:

- critical: solamente puede haber un hilo ejecutándose en la parte del código.
- atomic: las operaciones deben ser realizadas en bloque como si de una única sentencia se tratara.
- ordered: el bloque es ejecutado en el orden en que las iteraciones serían ejecutadas siguiendo el orden secuencial.
- barrier: cada hilo espera hasta que el resto de los hilos que se están ejecutando de manera paralela, alcancen el mismo punto donde se encuentra éste.

- nowait: especifica que no hace falta esperar al resto de los hilos que finalicen su ejecución.
- flush: exporta a los hilos un valor modificado por otro hilo durante la ejecución del procesamiento en paralelo.

Condiciones para compartir datos:

- shared: los datos son compartidos y por lo tanto visibles y accesibles por todos los hilos.
- private: los datos son privados, lo que significa que cada hilo tiene una copia local de los datos.
- firstprivate: las copias de las variables privadas se inicializan con el valor original.
- lastprivate: al finalizar la ejecución paralela, la variable contiene el último valor

Condiciones para la planificación:

- schedule (static): cada hilo decide qué parte del código ejecutar antes de meterse en un bucle. Las iteraciones son divididas entre el número de hilos disponibles.
- schedule (dynamic): a cada hilo se le asigna un pequeño número de iteraciones totales. Cuando un hilo finaliza sus iteraciones, se le asigna más hasta que se termine la ejecución de todas.
- guided: a cada hilo se le asigna de manera dinámica un alto número de iteraciones y este número va decreciendo exponencialmente a medida que se realiza cada asignación del citado número de iteraciones.
- numthreads: especifica el número de hilos que serán creados para la paralelización.

Funciones que vienen en OpenMP:

- omp_set_num_threads: define el número de hilos que se ejecutan a la vez.
- omp_get_num_threads: obtiene el número de hilos en ejecución.
- omp_get_max_threads: obtiene el número máximo de hilos lanzados en la zona paralela.
- omp_get_thread_num: obtiene el número del hilo.
- omp_get_num_procs: obtiene el número de procesadores que tiene el ordenador.
- omp_set_dynamic: sirve para definir si los hilos tienen que crecer o decrecer dinámicamente.

Para poder paralelizar en el lenguaje de programación Java, existe una biblioteca llamada *JOMP*. Gracias a esta biblioteca, se pueden utilizar las mismas sintaxis anteriores descritas (12), la única diferencia es que no hay que escribir la sintaxis *#pragma*, si no que con doble barras y las sintaxis a utilizar de la lista anterior, por ejemplo para *#pragma omp parallel* de C, en Java se escribiría *//omp parallel* (13).

En el apéndice 10.3.1 Conversión de la función de multiplicación con la biblioteca JOMP se muestra el resultado tras haber aplicado la compilación utilizando la biblioteca *JOMP* del fragmento del código donde se paraleliza.

2.4 OpenCL (SIMD / threads)

OpenCL (*Open Computing Language* / Lenguaje de computación abierta) fue desarrollado principalmente por *Apple* y refinado, en una primera propuesta, con la colaboración de los equipos técnicos de *AMD, IBM, Intel* y Nvidia.

Más tarde, la propia *Apple* propuso al consorcio tecnológico sin ánimo de lucro *Khronos Group* convertir el lenguaje en un estándar abierto. El 16 de Junio de 2008, *Khronos Group* creó el *Compute Working Group* (Grupo de trabajo de computación) integrado por representantes de compañías de *CPUs* (Central Processing Unit), GPUs (Graphics Processing Unit), de procesadores integrados y de empresas de software para llevar a cabo el proceso de estandarización. Este grupo trabajó durante cinco meses para finalizar los detalles técnicos de las especificaciones para la primera versión *OpenCl 1.0*. Dichas especificaciones fueron revisadas por los miembros de *Khronos Group* y aprobadas para su difusión pública el 8 de diciembre de 2008.

La versión *OpenCL 1.2* publicada el 15 de noviembre de 2011, última versión disponible, mejora las funcionalidades de la versión 1.1 mejorando la flexibilidad de la programación paralela, la funcionalidad y el rendimiento.

OpenCL consta de una interfaz de programación de aplicaciones (*API*) y de un lenguaje de programación basado en *C99*, que es un estándar del lenguaje C. Juntos se pueden crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en *CPUs* como *GPUs*, como muestran las siguientes imágenes:

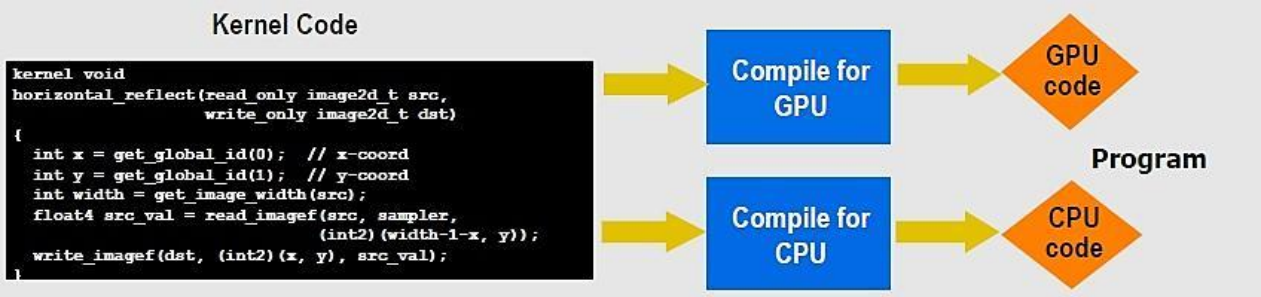


Figura 9: Compilación para ejecutar OpenCL en GPU o CPU

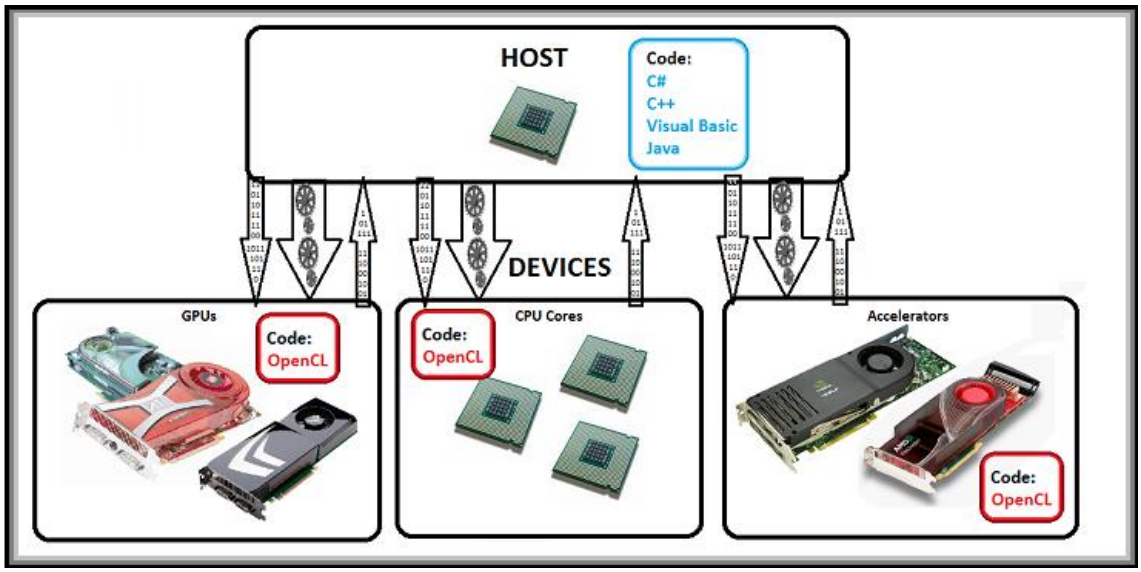


Figura 10: Compilación para ejecutar OpenCL en GPU o CPU

En los *drivers* de las tarjetas gráficas *ATI/AMD* como de *Nvidia* fueron integrados el soporte a *OpenCL*, siendo una alternativa al lenguaje *CUDA* y que soporta ejecuciones paralelas, homogéneas e híbridas.

Además consta de 4 espacios de memoria: (14)

- Memoria privada: accesible solamente por el elemento de trabajo en un elemento de proceso.
- Memoria local: accesible por cada elemento de trabajo en un grupo de trabajo.
- Memoria constante: accesible por cada grupo de trabajo en modo de sólo lectura.
- Memoria global: accesible por cada grupo de trabajo.

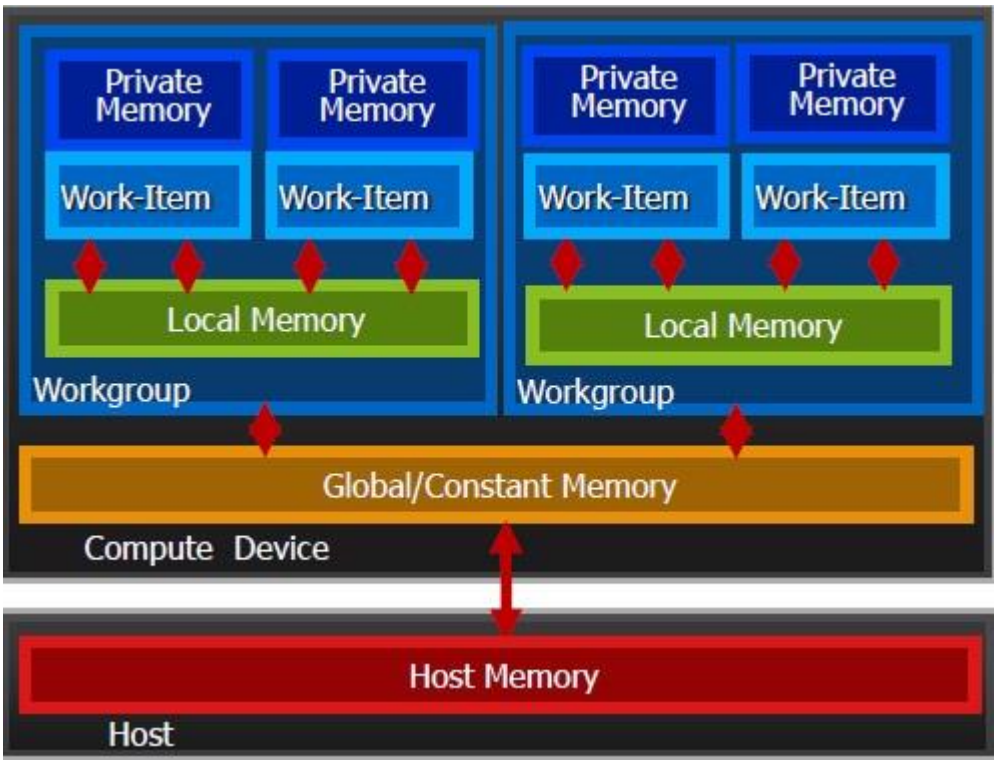


Figura 11: Arquitectura de la memoria

La tarjeta gráfica (en la siguiente figura como "dispositivo de computación") se subdivide en unidades de computamiento que a su vez se subdividen en elementos de procesado. *Host* se entiende como la CPU principal que configura la ejecución del *kernel*.

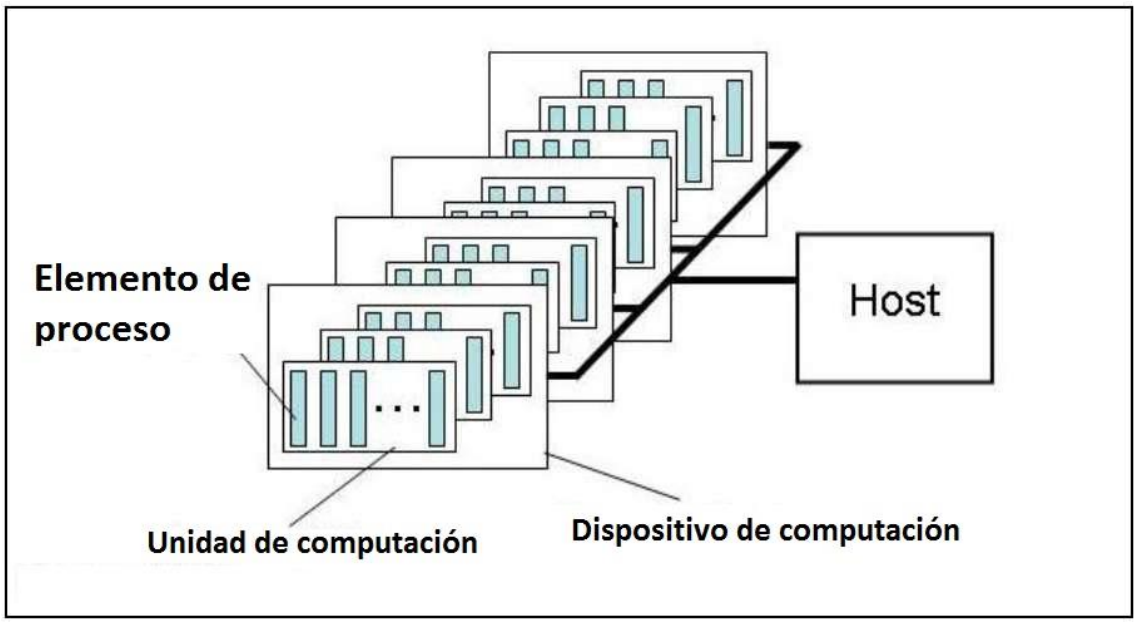


Figura 12: Estructura conceptual de las unidades de procesamiento

Cada una de las agrupaciones de unidades de procesamiento se sincroniza entre ellas para así dar a lugar a los grupos de trabajo, cuya finalidad es proporcionar una estructura jerárquica que permite un mejor manejo de todas esas unidades de procesamiento.

Para entender mejor estos grupos o agrupaciones de trabajo, se explican en este apartado:

- Work item (objeto de trabajo): Tratan solamente un elemento del problema.
- Work group (grupo de trabajo): Agrupan *work ítems* y paralelizan algunas operaciones más complejas coordinándolas cada uno de ellos.
- Compute device (dispositivo de computación): Agrupa a los *work groups* y tiene conocimiento en cada momento de qué está procesando cada *work ítem* y cada *work group*.

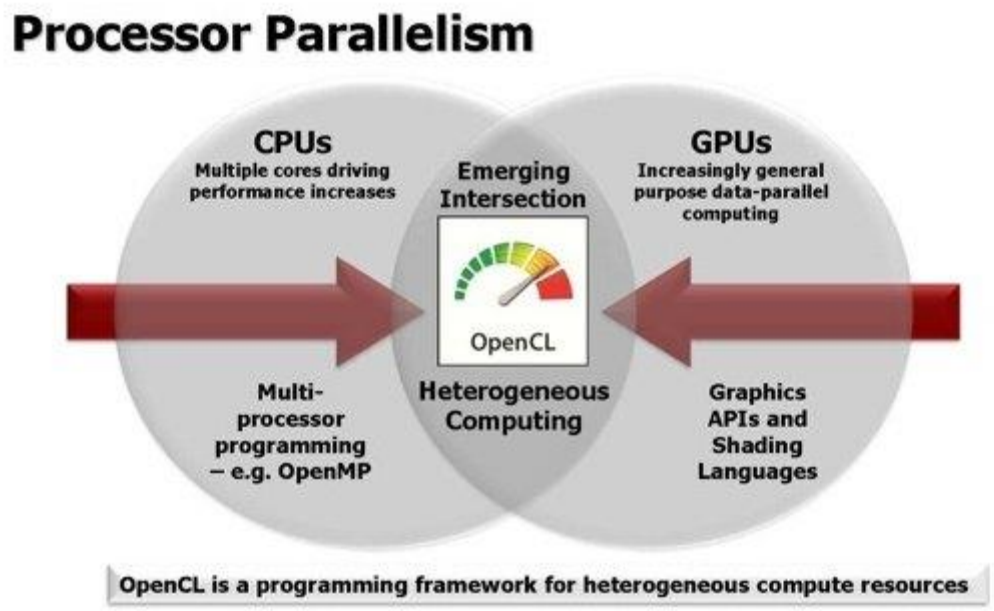


Figura 13: ¿Qué es OpenCL?

En el Apéndice 1: Comandos utilizados y funcionamiento de OpenCL se explica detalladamente las principales funciones de *OpenCL* que se ha utilizado en este Proyecto Fin de Grado para su funcionamiento.

2.5 Otras tecnologías

Además de *OpenCL* y *OpenMP*, existen otras tecnologías diferentes pero muy parecidas a éstas últimas, como son el caso de *TBB* de la compañía Intel y *CUDA* de la compañía *Nvidia*.

2.5.1 TBB

Intel Threading Building Blocks, desarrollada por la compañía *Intel*, se trata de una biblioteca para la codificación de programas para sacar ventaja de los procesadores multinúcleo. La biblioteca consiste en estructuras de datos y algoritmos que permiten al programador evitar complicaciones que aparecen al utilizar los paquetes nativos de hilos como lo son los hilos de POSIX o de Windows en los cuales cada hilo para la ejecución es creado, sincronizado y finalizado manualmente. Por ello, la biblioteca abstrae el acceso a los múltiples procesadores permitiendo que las operaciones sean tratadas como tareas, las cuales se reparten a los núcleos dinámicamente y automatizando el uso eficiente de la caché de las *CPU*. (15)

Para aprovechar los núcleos y la escalabilidad de los programas, *TBB* implementa el denominado "robo de tareas" (*task stealing*) (16) que balancea la carga de trabajo sobre los núcleos de procesamiento. Al principio la carga de trabajo se divide entre los núcleos disponibles. Si alguno de los núcleos termina su trabajo mientras los demás tienen aún trabajo que hacer, el gestor de tareas reasigna parte del trabajo al núcleo inactivo (17).

Las versiones publicadas hasta ahora con sus fechas de lanzamientos son:

- Versión 1.0: 29 de agosto de 2006.
- Versión 1.1: 10 de abril de 2007.
- Versión 2.0: 24 de julio de 2007.
- Versión 2.1: 22 de julio de 2008.
- Versión 3.0: 4 de marzo de 2010.
- Versión 4.0: 8 de septiembre de 2011.
- Versión 4.1: 5 de septiembre de 2012.

Las principales diferencias del *TBB* con *OpenMP* son las siguientes (18):

Challenges for parallel programming	Intel® Threading Build Blocks	OpenMP
Task level	x	x
Cross-platform support	x	x
Scalable runtime libraries	x	
Threads' Control		
Pre-tested and validated	x	x
C Development support		x
Intel® Threading Tools support	x	x
Maintenance for tomorrow	x	x
Scalable memory allocator	x	
"light" mutex	x	
Processor affinity	Thread affinity	

Tabla 2: Diferencias entre TBB y OpenMP

Resumiendo: (19)

- La mayoría de las bibliotecas de programación paralela requiere que sea el programador el que cree y gestione los hilos, pero en cambio, *TBB* trabaja con tareas lo cual es mucho menos pesado de construir que en los hilos y fáciles de planificar.
- *TBB* puede coexistir en el mismo programa con otras soluciones de paralelización. Esto está muy bien ya que no obliga al programador a utilizar solamente una única

tecnología. Con *TBB* se puede utilizar cualquier otra herramienta de paralelización como *OpenMP* o hilos nativos.

- *TBB* se centra en la paralelización de tareas de computación intensiva, desarrollando así soluciones de alto nivel.

He aquí un ejemplo de las bibliotecas que utiliza *TBB*:

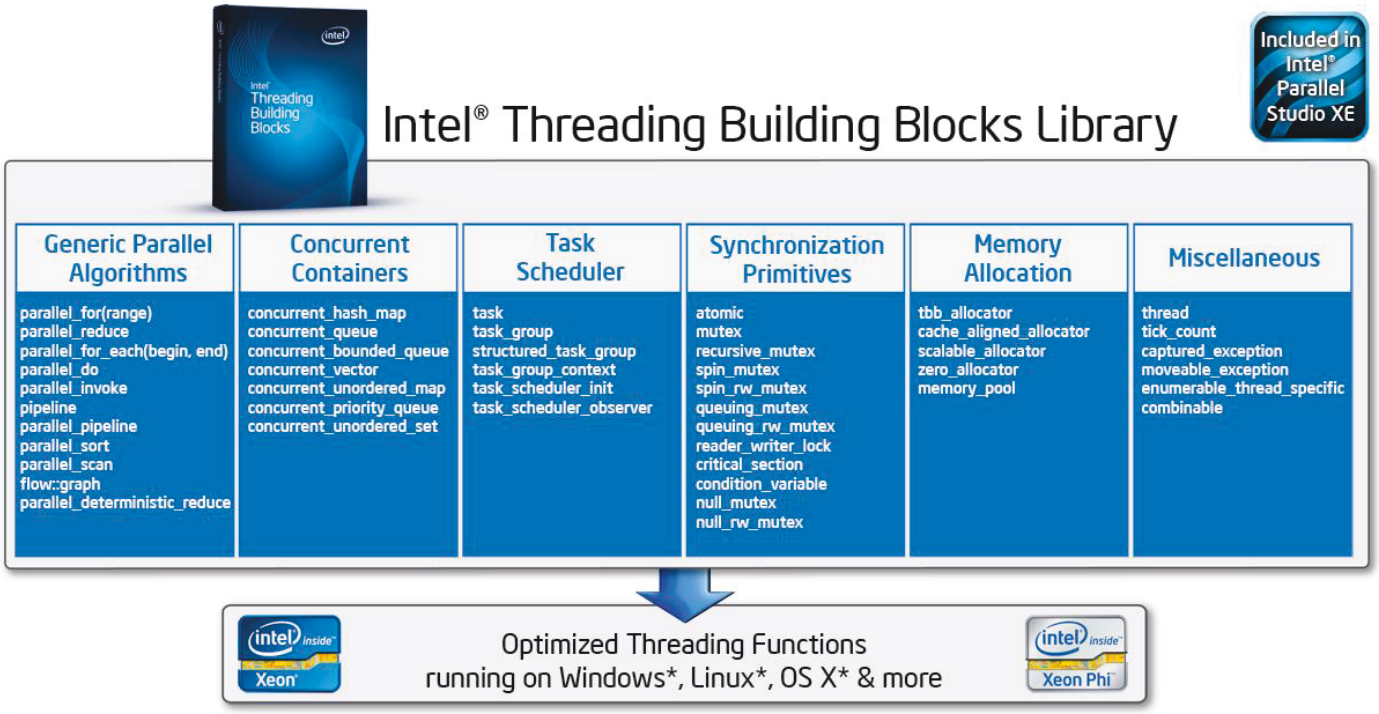


Figura 14: Bibliotecas que utiliza TBB

Si se quiere utilizar paralelización con lenguaje de programación C++, es mejor utilizar *TBB* que *OpenMP*.

2.5.2 CUDA

CUDA significa *Compute Unified Device Architecture*, arquitectura unificada de dispositivos de cómputo. *CUDA* es tanto un compilador como un conjunto de herramientas de desarrollo creadas por la compañía *Nvidia* que sirve para codificar algoritmos aprovechando la gran potencia de la GPU de *Nvidia*.

Los programas escritos en *CUDA*, igual que en *OpenCL*, se puede utilizar tanto la *CPU* como las *GPUs* disponibles del sistema. Para que el programa sepa en dónde ejecutar el programa, si por la CPU, la GPU o por ambas, se utiliza para ello identificadores adicionales. Lo mismo ocurre con los tipos de memoria, para especificar dónde guardar las variables.

El lenguaje *CUDA* está escrito en C/C++, pero también se pueden utilizar otros lenguajes para ello como Python, Fortran y Java, como ocurre también con *OpenCL*.

El flujo de procesamiento CUDA funciona de la siguiente manera:

- Paso 1: Se copian los datos de la memoria principal a la memoria de la GPU.
- Paso 2: La CPU manda el proceso a la GPU.
- Paso 3: La GPU ejecuta el proceso en paralelo en cada núcleo.
- Paso 4: Finalmente se copia el resultado de la memoria de la GPU a la memoria principal.

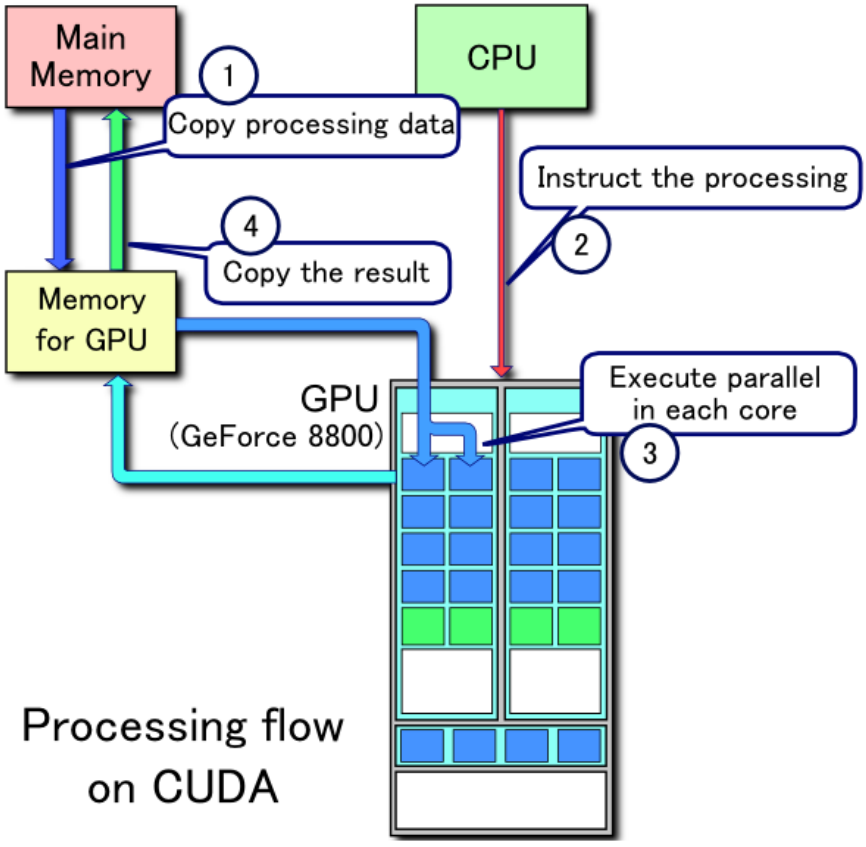


Figura 15: Flujo de procesamiento en CUDA

La jerarquía de memoria en CUDA consta de:

- Memoria privada: de cada hilo, solamente accesible desde él mismo.
- Memoria compartida: por los hilos del bloque.
- Memoria global: todos los hilos pueden acceder a él.

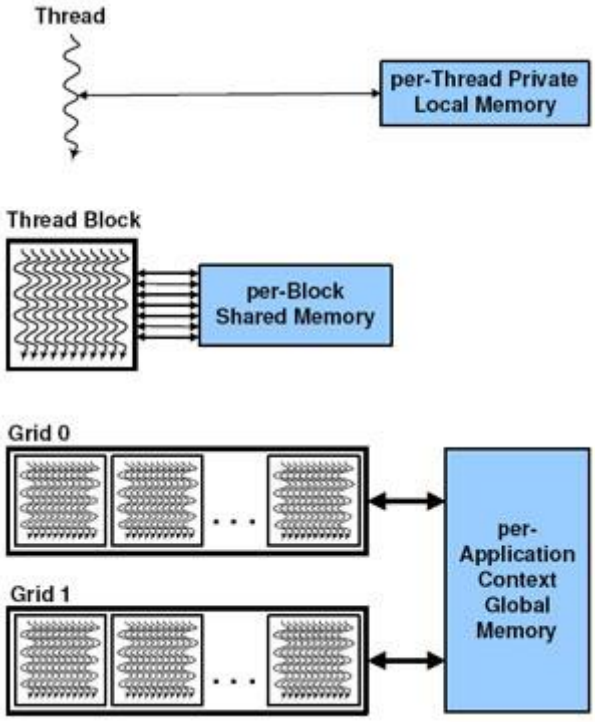


Figura 16: Jerarquía de memoria en CUDA

La unidad básica de operación en CUDA es el hilo. Dentro del dispositivo, los hilos se organizan en bloques y éstos a su vez en mallas. Cada malla de hilos solamente puede ejecutar una función (*kernel*) a su vez, como se puede ver en la siguiente imagen:

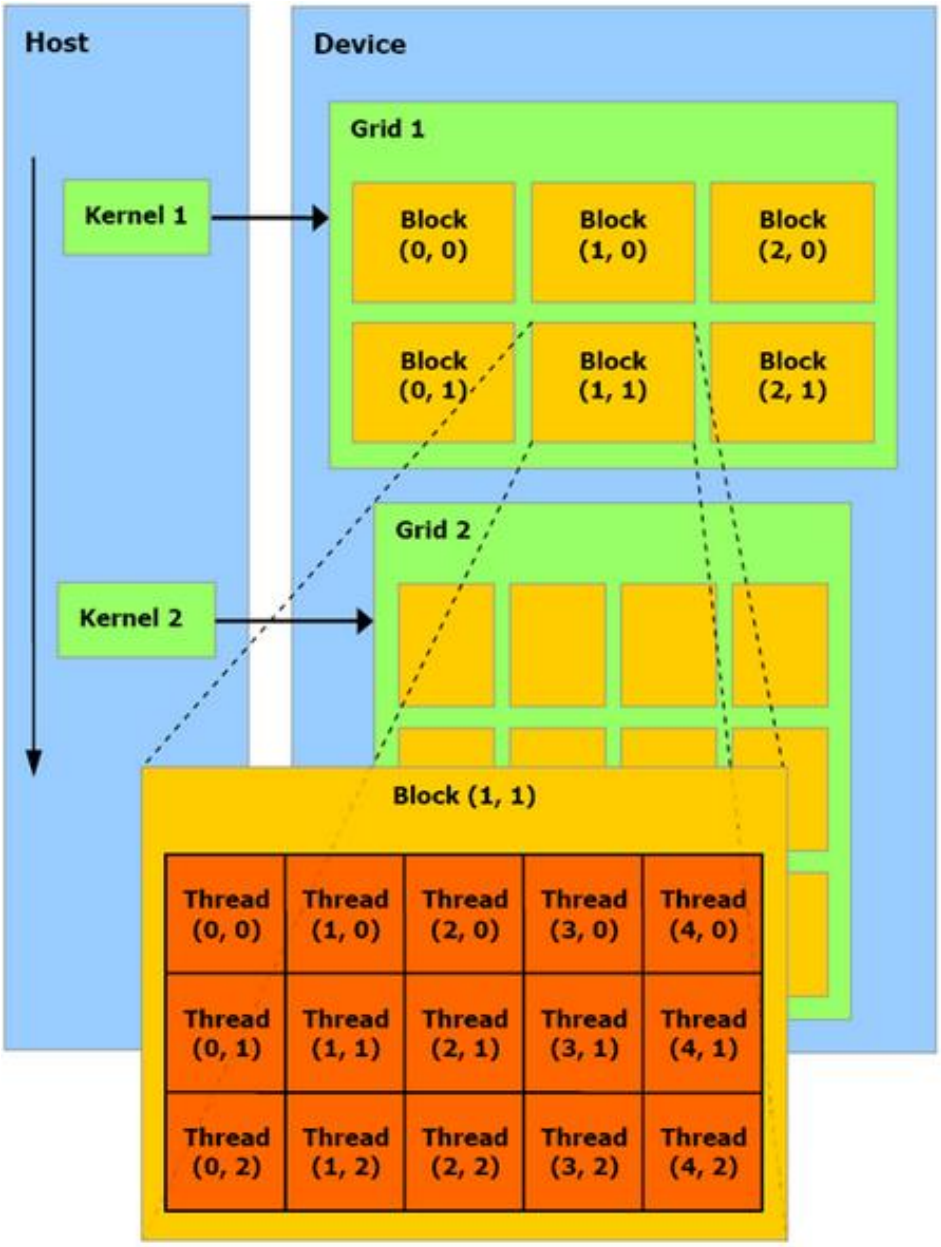


Figura 17: Hilos, bloques y mallas en CUDA

Las principales ventajas sobre *OpenCL* son: (20)

- Lecturas dispersas: se puede consultar cualquier posición de memoria.
- Memoria compartida que se comparte entre los hilos y además ser usada como caché.
- Lecturas más rápidas de y hacia la *GPU*.
- Soporte para enteros y operadores a nivel de bit.

Se podría decir que *CUDA* es bastante parecido a *OpenCL*. Las principales diferencias son:

- La terminología:

Terminología CUDA	Terminología OpenCL
Thread	Work-item
Block	Work-group
Global memory	Global memory
Shared memory	Local memory
Local memory	Private memory

Tabla 3: Terminología CUDA / OpenCL

- *CUDA* tiene mejor rendimiento que *OpenCL*, debido a que está optimizado para el hardware de la compañía *Nvidia*. La siguiente tabla (21) muestra una comparación del rendimiento entre *CUDA* y *OpenCL* realizando una simulación del método Monte Carlo (trata de un método no determinístico o estadístico numérico, usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud):

Qubits	GPU Operations Time				End-To-End Running Time			
	CUDA		OpenCL		CUDA		OpenCL	
	avg	stdev	avg	stdev	avg	stdev	avg	stdev
8	1.97	0.030	2.24	0.006	2.94	0.007	4.28	0.164
16	3.87	0.006	4.75	0.012	5.39	0.008	7.45	0.023
32	7.71	0.007	9.05	0.012	10.16	0.009	12.84	0.006
48	13.75	0.015	19.89	0.010	17.75	0.013	26.69	0.016
72	26.04	0.034	42.32	0.085	32.77	0.025	54.85	0.103
96	61.32	0.065	72.29	0.062	76.24	0.033	92.97	0.064
128	101.07	0.523	113.95	0.758	123.54	1.091	142.92	1.080

Tabla 4: Rendimiento CUDA vs. OpenCL

- La principal ventaja de *OpenCL* es que se trata de un estándar abierto independiente de plataforma, no como *CUDA* que es cerrado.

3 Marco regulador

Este capítulo recoge las normativas técnicas y legales que afectan al trabajo.

Esta normativa fue publicada en el Boletín Oficial del Estado número 298 el martes 14 de diciembre de 1999. Ley Orgánica 15/1999, de 13 de diciembre de 1999, de Protección de Datos de Carácter Personal: (22)

3.1 Artículo 1. Objeto.

La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar.

3.2 Artículo 2. Ámbito de aplicación

1. La presente Ley Orgánica será de aplicación a los datos de carácter personal registrados en soporte físico, que los haga susceptibles de tratamiento, y a toda modalidad de uso posterior de estos datos por los sectores público y privado.

Se regirá por la presente Ley Orgánica todo tratamiento de datos de carácter personal:

a. Cuando el tratamiento sea efectuado en territorio español en el marco de las actividades de un establecimiento del responsable del tratamiento.
b. Cuando al responsable del tratamiento no establecido en territorio español, le sea de aplicación la legislación española en aplicación de normas de Derecho Internacional público.
c. Cuando el responsable del tratamiento no esté establecido en territorio de la Unión Europea y utilice en el tratamiento de datos medios situados en territorio español, salvo que tales medios se utilicen únicamente con fines de tránsito.

2. El régimen de protección de los datos de carácter personal que se establece en la presente Ley Orgánica no será de aplicación:

a. A los ficheros mantenidos por personas físicas en el ejercicio de actividades exclusivamente personales o domésticas.
b. A los ficheros sometidos a la normativa sobre protección de materias clasificadas.

c. A los ficheros establecidos para la investigación del terrorismo y de formas graves de delincuencia organizada. No obstante, en estos supuestos el responsable del fichero comunicará previamente la existencia del mismo, sus características generales y su finalidad a la Agencia de Protección de Datos.

3. Se regirán por sus disposiciones específicas, y por lo especialmente previsto, en su caso, por esta Ley Orgánica los siguientes tratamientos de datos personales:

a. Los ficheros regulados por la legislación de régimen electoral.
b. Los que sirvan a fines exclusivamente estadísticos, y estén amparados por la legislación estatal o autonómica sobre la función estadística pública.
c. Los que tengan por objeto el almacenamiento de los datos contenidos en los informes personales de calificación a que se refiere la legislación del régimen del personal de las Fuerzas Armadas.
d. Los derivados del Registro Civil y del Registro Central de penados y rebeldes.
e. Los procedentes de imágenes y sonidos obtenidos mediante la utilización de videocámaras por las Fuerzas y Cuerpos de Seguridad, de conformidad con la legislación sobre la materia.

3.3 Artículo 3. Definiciones.

A los efectos de la presente Ley Orgánica se entenderá por:

a. Datos de carácter personal: cualquier información concerniente a personas físicas identificadas o identificables.
b. Fichero: todo conjunto organizado de datos de carácter personal, cualquiera que fuere la forma o modalidad de su creación, almacenamiento, organización y acceso.
c. Tratamiento de datos: operaciones y procedimientos técnicos de carácter automatizado o no, que permitan la recogida, grabación, conservación, elaboración, modificación, bloqueo y cancelación, así como las cesiones de datos que resulten de comunicaciones, consultas, interconexiones y transferencias.
d. Responsable del fichero o tratamiento: persona física o jurídica, de naturaleza pública o privada, u órgano administrativo, que decida sobre la finalidad, contenido y uso del tratamiento.
e. Afectado o interesado: persona física titular de los datos que sean objeto del tratamiento a que se refiere el apartado c) del presente artículo.

f. Procedimiento de disociación: todo tratamiento de datos personales de modo que la información que se obtenga no pueda asociarse a persona identificada o identificable.
g. Encargado del tratamiento: la persona física o jurídica, autoridad pública, servicio o cualquier otro organismo que, sólo o conjuntamente con otros, trate datos personales por cuenta del responsable del tratamiento.
h. Consentimiento del interesado: toda manifestación de voluntad, libre, inequívoca, específica e informada, mediante la que el interesado consienta el tratamiento de datos personales que le conciernen.
i. Cesión o comunicación de datos: toda revelación de datos realizada a una persona distinta del interesado.
j. Fuentes accesibles al público: aquellos ficheros cuya consulta puede ser realizada, por cualquier persona, no impedida por una norma limitativa o sin más exigencia que, en su caso, el abono de una contraprestación. Tienen la consideración de fuentes de acceso público, exclusivamente, el censo promocional, los repertorios telefónicos en los términos previstos por su normativa específica y las listas de personas pertenecientes a grupos de profesionales que contengan únicamente los datos de nombre, título, profesión, actividad, grado académico, dirección e indicación de su pertenencia al grupo. Asimismo, tienen el carácter de fuentes de acceso público los diarios y boletines oficiales y los medios de comunicación.

4 Análisis de requisitos

Para un correcto funcionamiento de los programas en modo secuencial, *OpenMP* y *OpenCL*, deben existir una serie de características indispensables que favorezcan el buen funcionamiento de los programas.

Las tablas se componen de los siguientes campos:

- ID requisito: identificador del requisito.
- Nombre: nombre del requisito.
- Descripción: una pequeña definición del cometido.
- Justificación: breve definición del por qué es importante este requisito.
- Criterio de cumplimiento: comprobante de una implementación del requisito satisfactoria.
- Prioridad: relevancia del requisito.
- Conflicto: el requisito no puede ser implementado o no funcionará si el requisito mostrado en esta celda no se implementa.
-

4.1 No funcionales

Los requisitos no funcionales son restricciones de los servicios o funcionales ofrecidos por el sistema. En las siguientes tablas se muestran los requisitos no funcionales sacados para este Proyecto Fin de Grado:

ID requisito	RNF-01
Nombre	Ubuntu
Descripción	El dispositivo de desarrollo ha de tener instalado Ubuntu 12.04.
Justificación	Sistema operativo base desde donde se van a realizar las pruebas.
Criterio de cumplimiento	
Prioridad	Muy alta
Conflicto	#

Requisito NF 1: Ubuntu

ID requisito	RNF-02
Nombre	Java
Descripción	Comprobar si hay Java instalado en Linux
Justificación	Sin Java en el dispositivo de desarrollo, no se puede ejecutar los programas realizados en el lenguaje de programación Java.
Criterio de cumplimiento	Ejecutar el comando *java -version* en la terminal y comprobar que está instalado.
Prioridad	Alta.
Conflicto	#

Requisito NF 2: Java

ID requisito	RNF-03
Nombre	Javac
Descripción	Comprobar si hay compilador Java instalado en Linux.
Justificación	Sin este compilador, no se puede compilar los programas realizados en el lenguaje de programación Java.
Criterio de cumplimiento	Compilar un código de prueba y obtener el resultado esperado.
Prioridad	Alta.
Conflicto	#

Requisito NF 3: Javac

ID requisito	RNF-04
Nombre	GCC
Descripción	Comprobar si hay compilador C instalado en Linux.
Justificación	Sin este compilador, no se puede compilar los programas realizados en el lenguaje de programación C.
Criterio de cumplimiento	Ejecutar el comando *gcc --version* en la terminal y comprobar que está instalado. Compilar un código de prueba y obtener el resultado esperado.
Prioridad	Alta.
Conflicto	#

Requisito NF 4: GCC

ID requisito	RNF-05
Nombre	Driver OpenCL GPU
Descripción	Comprobar si hay driver de OpenCL instalado en Linux.
Justificación	Sin OpenCL en el dispositivo de desarrollo, no se puede ejecutar los programas realizados en OpenCL en GPU.
Criterio de cumplimiento	Ejecutar el comando: *locate *opencl** Si se encuentra un fichero llamado *libOpenCL.so* o *libnvidia-opencl.so* en alguno de los directorios de biblioteca en */usr,* es que hay OpenCL para utilizarlo en GPU instalado en el sistema.
Prioridad	Alta.
Conflicto	#

Requisito NF 5: Driver OpenCL GPU

ID requisito	RNF-06
Nombre	Driver OpenCL CPU
Descripción	Comprobar si hay driver de OpenCL instalado en Linux.
Justificación	Sin OpenCL en el dispositivo de desarrollo, no se puede ejecutar los programas realizados en OpenCL en CPU.
Criterio de cumplimiento	Ejecutar el comando: *locate *opencl** Si se encuentra un fichero llamado *libintelocl.so* en alguno de los directorios de biblioteca en */usr,* es que hay OpenCL para utilizarlo en CPU instalado en el sistema.
Prioridad	Alta.
Conflicto	#

Requisito NF 6: Driver OpenCL CPU

ID requisito	RNF-07
Nombre	Múltiples hilos en la CPU
Descripción	Paralelismo con hilos.
Justificación	Sin ejecución en múltiples hilos, no se puede comprobar el funcionamiento de OpenMP.
Criterio de cumplimiento	Compilar un código de prueba con paralelismo y obtener el resultado esperado.
Prioridad	Alta.
Conflicto	#

Requisito NF 7: Múltiples hilos en la CPU

4.2 Funcionales

Los requisitos funcionales describen lo que el sistema o el programa debe de hacer. En las siguientes tablas se muestran los requisitos funcionales sacados para este Proyecto Fin de Grado:

ID requisito	RF-01
Nombre	Modo de ejecución
Descripción	El programa ha de mostrar en qué modo se está ejecutando: Secuencial, openMP u openCL.
Justificación	Para poder diferenciar los modos de ejecución.
Criterio de cumplimiento	Se ha de imprimir por pantalla de la siguiente manera: - Secuencial: SEC - OpenMP: OPENMP - OpenCL: OPENCL
Prioridad	Alta
Conflicto	#

Requisito F 1: Modo de ejecución

ID requisito	RF-02
Nombre	Tamaño de matriz
Descripción	El programa ha de mostrar el tamaño de la matriz en el cual está calculando en ese momento.
Justificación	Para saber el tamaño actual de la matriz que está calculando.
Criterio de cumplimiento	El formato a imprimir por pantalla ha de tener el siguiente formato: 10x10, 20x20, 30x30, …
Prioridad	Alta
Conflicto	#

Requisito F 2: Tamaño de matriz

ID requisito	RF-03
Nombre	Tiempo de ejecución
Descripción	El programa ha de mostrar el tiempo en milisegundos
Justificación	Para saber el tiempo que necesita el dispositivo de desarrollo en calcular la multiplicación.
Criterio de cumplimiento	El tiempo se ha de mostrar en el siguiente formato: *xxxxx.xxxxx*
Prioridad	Alta
Conflicto	#

Requisito F 3: Tiempo de ejecución

ID requisito	RF-04
Nombre	Coincidencia
Descripción	El programa ha de imprimir por pantalla (solamente en openMP u openCL) si coincide el resultado calculado en paralelo con la secuencial. Esta comprobación lo realiza si se le pasa un 1 en el tercer argumento.
Justificación	Para comprobar que el cálculo realizado sea el correcto
Criterio de cumplimiento	El formato a imprimir por pantalla ha de ser de la siguiente manera: - Si coincide: → MATCH!! - Si no coincide: → DOES NOT → MATCH!!
Prioridad	Alta
Conflicto	#

Requisito F 4: Coincidencia

ID requisito	RF-05
Nombre	Impresión por pantalla
Descripción	El programa ha de mostrar por pantalla el modo de ejecución, tamaño de la matriz, el tiempo de ejecución y si el resultado es el esperado (de manera opcional).
Justificación	Para tener un *feedback* por pantalla
Criterio de cumplimiento	Se ha de imprimir por pantalla con el siguiente formato: *modoEjecución tamMatriz tiempoMilisegundos [coincidencia]*
Prioridad	Alta
Conflicto	#

Requisito F 5: Impresión por pantalla

ID requisito	RF-06
Nombre	Ejecución del programa secuencial sin argumento
Descripción	Parámetros por argumento
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Si se ejecuta solamente el programa sin ningún argumento, el programa muestra qué argumentos se ha de utilizar para ejecutar el programa correctamente.
Prioridad	Alta
Conflicto	#

Requisito F 6: Ejecución del programa secuencial sin argumento

ID requisito	RF-07
Nombre	Ejecución del programa secuencial tamaño
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa únicamente el tamaño de la matriz como primer argumento, el programa ha de ejecutarse solamente con una iteración.
Prioridad	Alta
Conflicto	#

Requisito F 7: Ejecución del programa secuencial tamaño

ID requisito	RF-09
Nombre	Ejecución del programa OpenMP sin argumento
Descripción	Parámetros por argumento
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Si se ejecuta solamente el programa sin ningún argumento, el programa muestra qué argumentos se ha de usar para ejecutar el programa correctamente.
Prioridad	Alta
Conflicto	

Requisito F 8: Ejecución del programa OpenMP sin argumento

ID requisito	RF-10
Nombre	Ejecución del programa OpenMP tamaño
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa únicamente el tamaño de la matriz como primer argumento, el programa ha de ejecutarse solamente en un hilo.
Prioridad	Alta
Conflicto	#

Requisito F 9: Ejecución del programa OpenMP tamaño

ID requisito	RF-12
Nombre	Ejecución del programa OpenMP paralelización
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa el número de hilos a ejecutar como segundo argumento, el programa ha de paralelizar la multiplicación de matrices del tamaño pasado por el primer argumento en tantos hilos como se le haya pasado por segundo argumento.
Prioridad	Alta
Conflicto	#

Requisito F 10: Ejecución del programa OpenMP paralelización

ID requisito	RF-13
Nombre	Ejecución del programa OpenMP comprobación
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa como tercer argumento un 1, el programa ha de comprobar que el resultado calculado mediante paralelización sea el mismo que en modo secuencial.
Prioridad	Alta
Conflicto	#

Requisito F 11: Ejecución del programa OpenMP comprobación

ID requisito	RF-14
Nombre	Ejecución del programa OpenCL sin argumentos
Descripción	Parámetros por argumento
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Si se ejecuta solamente el programa sin ningún argumento, el programa muestra qué argumentos se ha de usar para ejecutar el programa correctamente.
Prioridad	Alta
Conflicto	#

Requisito F 12: Ejecución del programa OpenCL sin argumentos

ID requisito	RF-15
Nombre	Ejecución del programa OpenCL tamaño
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa únicamente el tamaño de la matriz como primer argumento, el programa ha de ejecutarse solamente en un hilo.
Prioridad	Alta
Conflicto	#

Requisito F 13: Ejecución del programa OpenCL tamaño

ID requisito	RF-17
Nombre	Ejecución del programa OpenCL paralelización
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa el número de hilos a ejecutar como segundo argumento, el programa ha de paralelizar la multiplicación de matrices del tamaño pasado por el primer argumento en tantos hilos como se le haya pasado por segundo argumento.
Prioridad	Alta
Conflicto	#

Requisito F 14: Ejecución del programa OpenCL paralelización

ID requisito	RF-18
Nombre	Ejecución del programa OpenCL comprobación
Descripción	Parámetros por argumentos
Justificación	Los parámetros serán pasados por argumentos.
Criterio de cumplimiento	Cuando se le pase al programa como tercer argumento un 1, el programa ha de comprobar que el resultado calculado mediante paralelización sea el mismo que en modo secuencial.
Prioridad	Alta
Conflicto	#

Requisito F 15: Ejecución del programa OpenCL comprobación

ID requisito	RF-19
Nombre	Comprobación plataforma OpenCL
Descripción	CPU y/o GPU
Justificación	El programa ha de saber en qué plataforma debe ejecutarse
Criterio de cumplimiento	Al principio de la ejecución comprueba qué plataformas disponibles hay en el dispositivo de desarrollo.
Prioridad	Alta
Conflicto	#

Requisito F 16: Comprobación plataforma OpenCL

ID requisito	RF-20
Nombre	Ejecución en la plataforma correspondiente OpenCL
Descripción	CPU y/o GPU
Justificación	El programa ha de ejecutarse en las plataformas correspondientes
Criterio de cumplimiento	A la hora de mostrar el tiempo de ejecución, ha de mostrar en qué plataforma se ha ejecutado: GPU o CPU.
Prioridad	Alta
Conflicto	#

Requisito F 17: Ejecución en la plataforma correspondiente OpenCL

5 Evaluación

En esta sección se realiza como primer punto, una comparación cualitativa de los programas realizados y como segundo punto, un análisis de los resultados obtenidos comparándolo con los distintos lenguajes utilizados y clases.

5.1 Comparación cualitativa (evaluación del código a nivel del texto)

En este capítulo se realizará un análisis estático del código escrito incluido la complejidad ciclomática.

La complejidad ciclomática es una métrica de software creada por *Thomas McCabe* en 1976 que refleja el número de caminos independientes que un programa puede tomar durante su ejecución. Para ello *McCabe* creó la siguiente tabla para determinar el riesgo que puede tener el código (cómo de difícil es probarlo, modificarlo y entenderlo):

Complejidad ciclomática	Evaluación de Riesgo
1 – 10	Programa simple, no hay riesgo.
11 – 20	Más complejo, riesgo moderado.
21 – 50	Complejo, riesgo alto.
+50	Programa no testeable, riesgo muy alto.

Tabla 5: Tabla de complejidad ciclomática

Cuanto más alto el valor, mayor es el número de defectos que puede tener el código.

El cálculo de la complejidad ciclomática se realiza de la siguiente manera:

$$M = E - N + 2*P$$

- M: Complejidad ciclomática.
- E: Número de aristas del grafo (conecta dos vértices si una sentencia puede ser ejecutada inmediatamente después de la primera).
- N: Número de nodos del grafo correspondiente a sentencias del programa.
- P: Número de componentes conexos, nodos de salida.

Si se quiere hacer manualmente el cálculo, se tendría que representar un gráfico de flujo de control del programa. Pongamos como ejemplo el siguiente código sencillo con su gráfico de flujo:

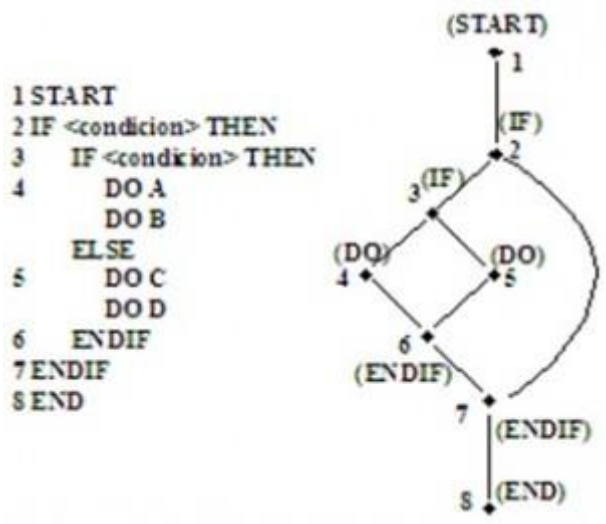


Figura 18: Ejemplo de gráfico de flujo de control del programa

Pero hoy en día existen programas o *plug-ins* que te facilitan la vida para realizar este cálculo además de muchas más operaciones como contar el número de líneas que tienen los códigos, el número de comentarios, número de métodos y muchos más. Para el código de programación Java se ha utilizado el *plug-in* llamado *CodePro AnalytiX* y para el código de programación C se ha utilizado el programa *CCCC (C and C++ Code Counter)*.

En la siguiente tabla se mostrará los resultados obtenidos con los plug-ins mencionados en el párrafo anterior:

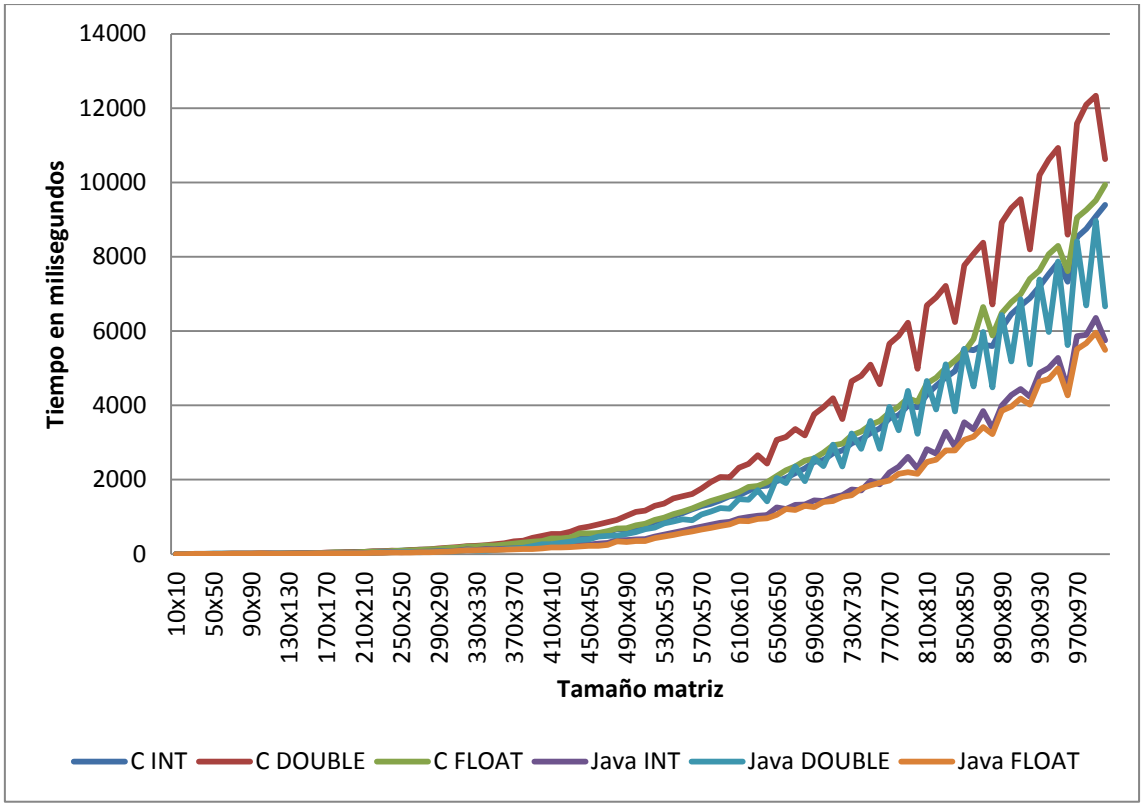
Lenguaje	Clase	Número de líneas de código	Número ciclomático
C	Secuencial	77	4,5
	OpenMP	110	7,75
	OpenCL	253	5
Java	Secuencial	75	5,33
	OpenMP	105	7,33
	OpenCL	316	6

Tabla 6: Tabla de complejidad ciclomática de los programas realizados

5.2 C vs Java (secuencial)

En este capítulo se mostrarán los resultados obtenidos de las pruebas realizadas y se explicará los motivos de las diferencias de tiempo.

Se mostrarán en gráficos cuyo eje vertical muestra el tiempo de ejecución del programa en milisegundos y en el eje horizontal el tamaño de la matriz utilizado para la operación. En este apartado se ha utilizado el modo secuencial de los lenguajes de programación C y Java utilizando los tipos INT, DOUBLE y FLOAT. Como bien se observa, el tiempo que tarda en calcular en DOUBLE es muy grande, tanto en C como en Java. El motivo se debe a que DOUBLE reserva 64 bits en memoria para guardar únicamente un valor. Mientras que tanto en C como en Java, los tipos INT y FLOAT tienen tiempos muy parecidos ya que ambos ocupan 32 bits.



Evaluación 1: Secuencial C-Java Máquina 1

En el tipo DOUBLE, se puede observar que la superación de 2 segundos de ejecución ya lo realiza a partir de un tamaño de 580x580 en C, mientras que en Java es a partir de 650x650. Lo mismo pasa con los tipos INT y FLOAT: en C a partir del tamaño 630x630 mientras que en Java 750x750.

La explicación a por qué Java es más rápido que C, se debe a la optimización del *bytecode* de la *JVM* (Java Virtual Machine) y al *Garbage collector*.

El *bytecode* de Java es el tipo de instrucciones que la *JVM* ejecuta. Las instrucciones caen en las siguientes categorías:

- Mover de memoria a registros y viceversa
- Aritmética y lógica
- Conversión de tipos
- Creación y manipulación de objetos
- Manipulación de la pila de operandos
- Control de flujo
- Invocación de métodos y retorno de los mismos

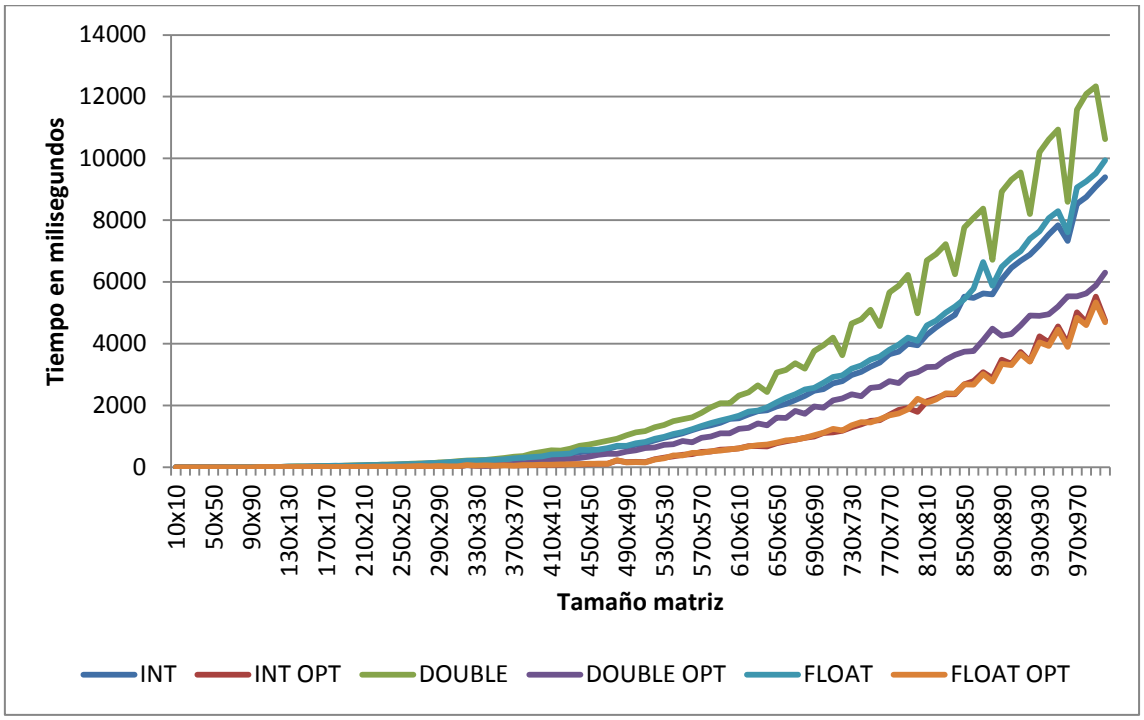
Las primeras versiones de Java (previas a la 1.2) no se realizaban optimizaciones de *bytecode* debido a que las diferentes versiones de *JVM* eran más bien genéricas y por lo tanto la ejecución tenía un desempeño pobre. Es a partir de la versión 1.2 (Diciembre de 1998) cuando Java incluyó un compilador denominado *Just-in-time* que optimiza el código *bytecode* en tiempo real de acuerdo a la carga de trabajo sobre el programa. Un modelo de compilación estática como lo es C "adivina" dónde se encuentran los cuellos de botella y se enfoca en ese fragmento de código para realizar la optimización.

El *Garbage collector* o recolector de basura en español, es el encargado de administrar la memoria de forma automática en Java. Es típico para la memoria crear un *stack* y un *heap*. *Stack* son las regiones de memoria donde los datos son añadidos o borrados de manera *LIFO* (*Last In First Out)*, mientras que *heap* es la zona dinámica donde se almacenan los objetos que se crean. Cuando el *Garbage collector* se ejecuta, que lo decide la *JVM*, su propósito es buscar y borrar los objetos que no pueden ser alcanzados.

En cambio, a la hora de compilar los programas de C, se puede utilizar un *flag* especial para que tome en cuenta que hay que optimizar todo lo posible para que dure menos la ejecución, como lo hace por ejemplo *JVM*. El *flag* utilizado para ellos es el *-Ofast*. Tras las ejecuciones con los programas optimizados, se puede observar que el tiempo de ejecución ha bajado hasta un 50%. Ello se debe a que el *flag* utilizado *-Ofast* optimiza todo lo que hace ya *-O3* y anteriores, que cuenta una lista muy grande de optimizaciones.

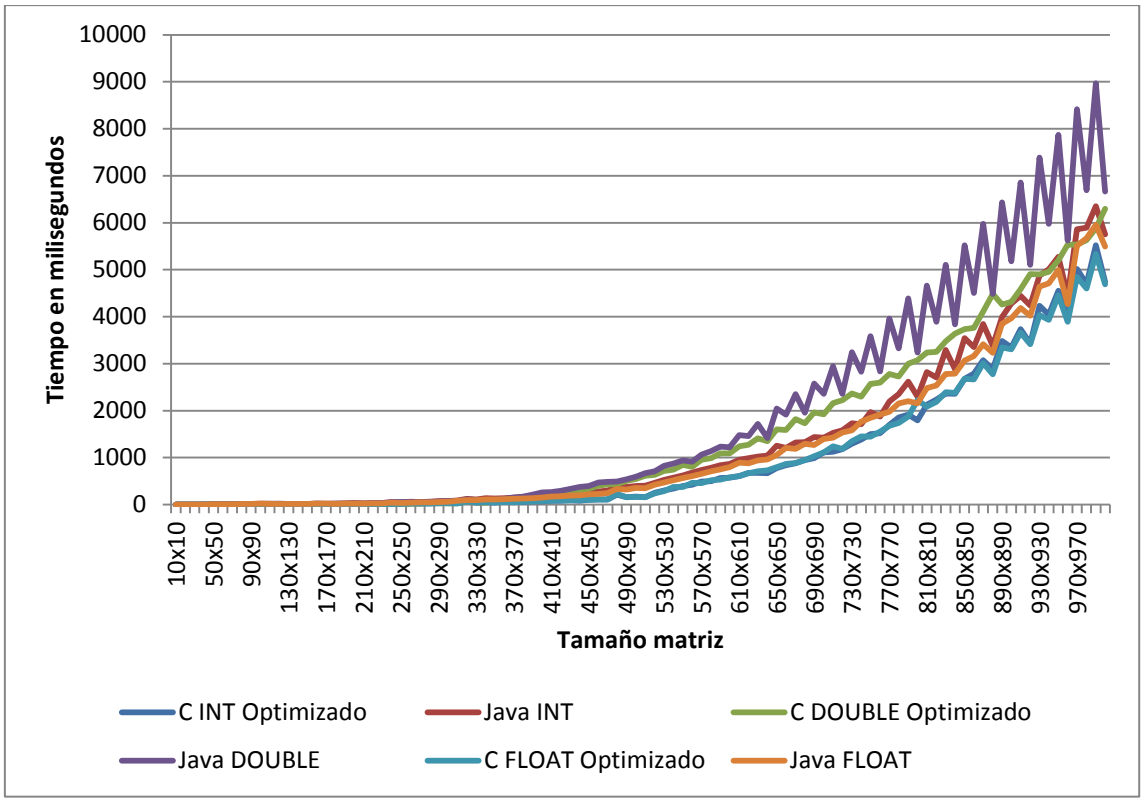
- O1: El compilador sigue permitiendo que se realicen todas las operaciones de depuración que sean necesarias en la aplicación y se hacen muy pocas optimizaciones locales. Esto es útil cuando se están desarrollando o depurando aplicaciones.
- O2: Aplica técnicas de optimización globales a nivel de archivos como también operaciones encauzamiento de software, predicación y otras optimizaciones. Además realiza aparte optimizaciones en los bucles, como la búsqueda de datos o desenrollado de bucles.
- O3: Se realizan optimizaciones de bucles de alto nivel, como transformaciones de bucles anidados, bloqueo de bucles o intercambio de bucles. Este es el nivel mínimo recomendado para las aplicaciones de coma flotante, que pasan la mayor parte del tiempo en bucles anidados y por tanto, la necesidad de optimizaciones de grupo de alto nivel que se consiguen con O3. También es muy efectivo para disminuir los fallos en la caché TLB (Translation Lookaside Buffer) que suelen ser frecuentes en aplicaciones que hace un uso intensivo de la memoria.
- Ofast: Contiene todas las optimizaciones de los puntos anteriores además de optimizaciones para programas que no son compatibles con el estándar. Este *flag* existe desde la versión 4.6.0 del compilador *GCC*.

La siguiente gráfica muestra la diferencia de tiempos en modo secuencial con los tipos INT, DOUBLE y FLOAT sin optimizar y optimizado a la hora de compilar.



Evaluación 2: Secuencial C Optimizado Máquina 1

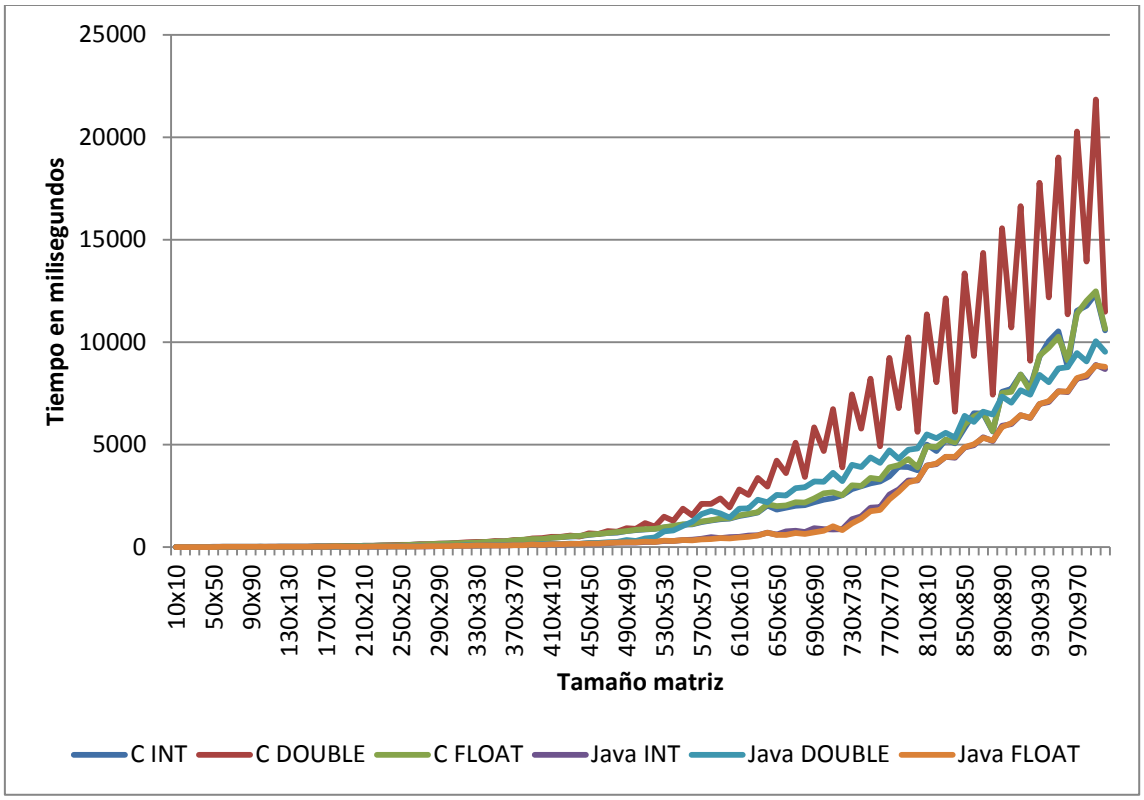
Tras observar que optimizado tarda hasta casi la mitad que sin optimizarlo, la siguiente gráfica muestra una comparación de los tipos INT, DOUBLE y FLOAT de Java (que optimiza automáticamente) con C optimizado.



Evaluación 3: Secuencial C optimizado-Java Máquina 1

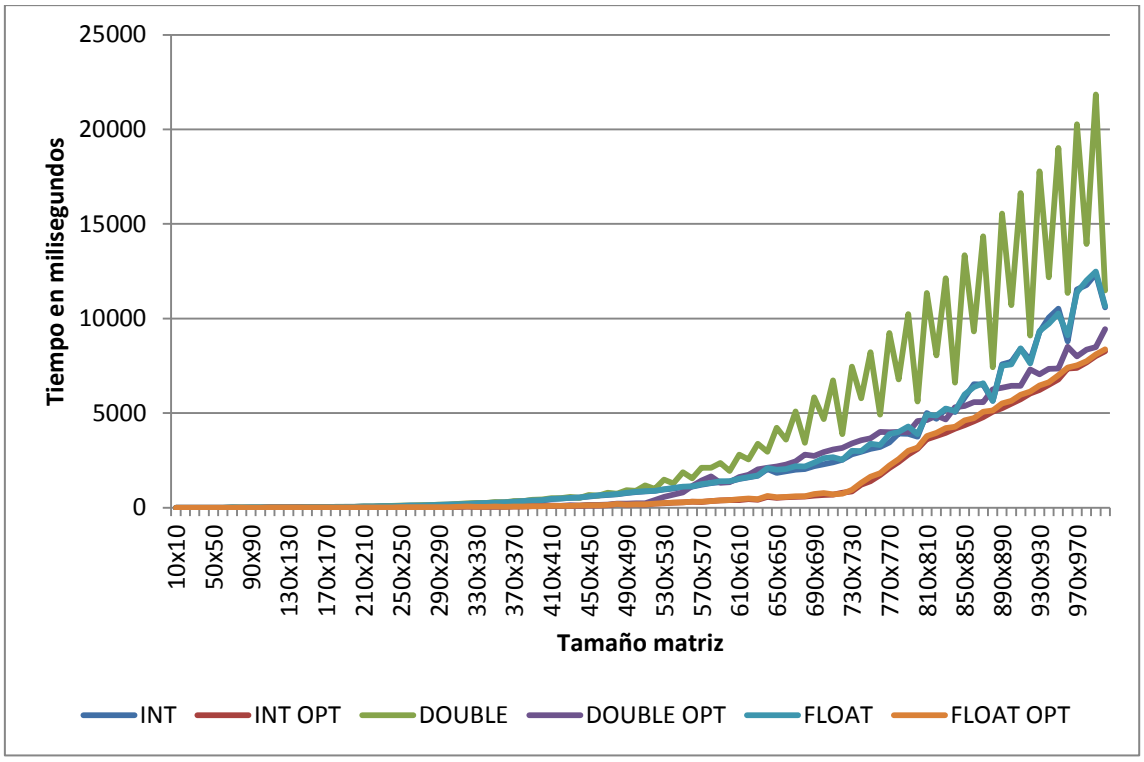
En cambio aquí se puede observar que el tipo DOUBLE en Java se comporta de forma extraña, haciendo que los tamaños impares tarden más que los tamaños pares, mientras que en C apenas se comportan de esa manera.

A continuación se muestran, igual que en la máquina 1, las gráficas con los resultados obtenidos en modo secuencial de la máquina 2 utilizando INT, DOUBLE y FLOAT, tanto en C, C optimizado y Java.



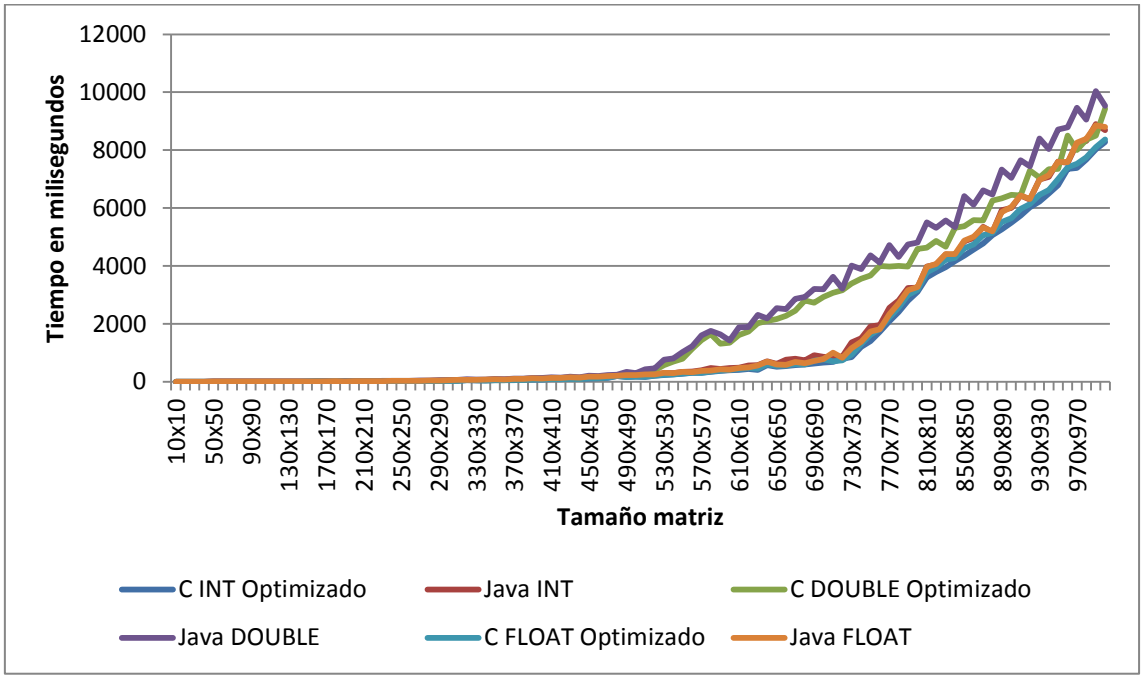
Evaluación 4: Secuencial C-Java Máquina 2

En la gráfica anterior se puede observar que en el tipo DOUBLE tiene el mismo efecto que en la máquina 1, pero esta vez se trata en el lenguaje de programación C en vez de en Java.



Evaluación 5: Secuencial C Optimizado Máquina 2

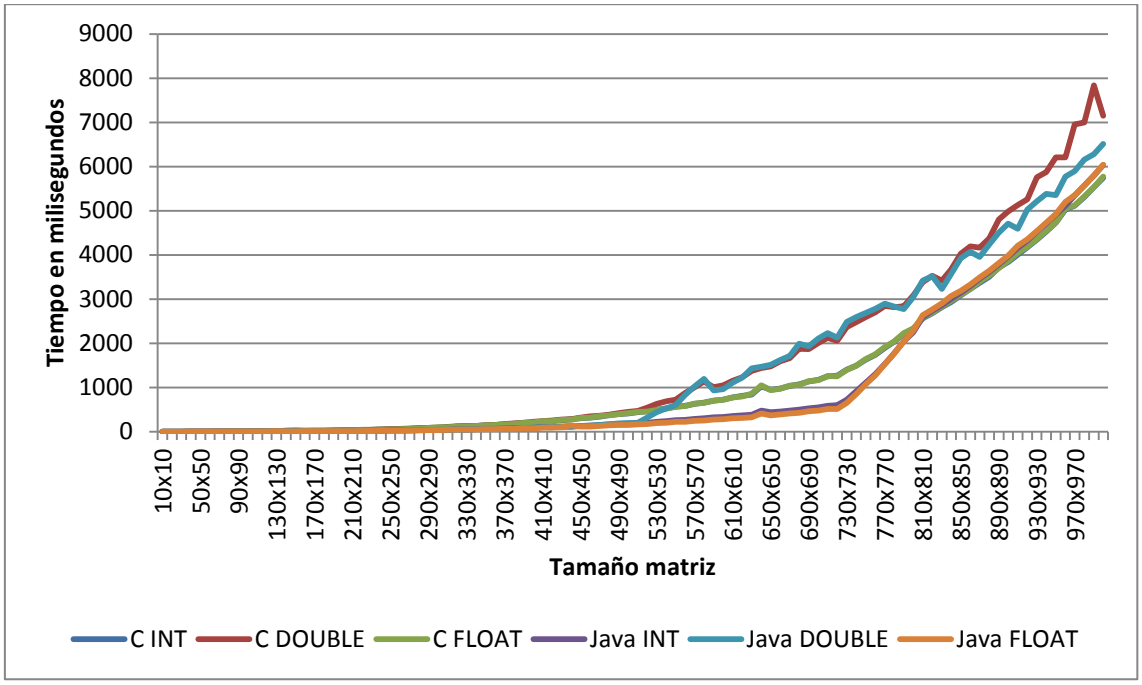
Queda demostrado que ejecutándolo de forma optimizada da mejores resultados, por lo tanto las gráficas se mostrarán con dicha optimización a la hora de hacer la comparación.



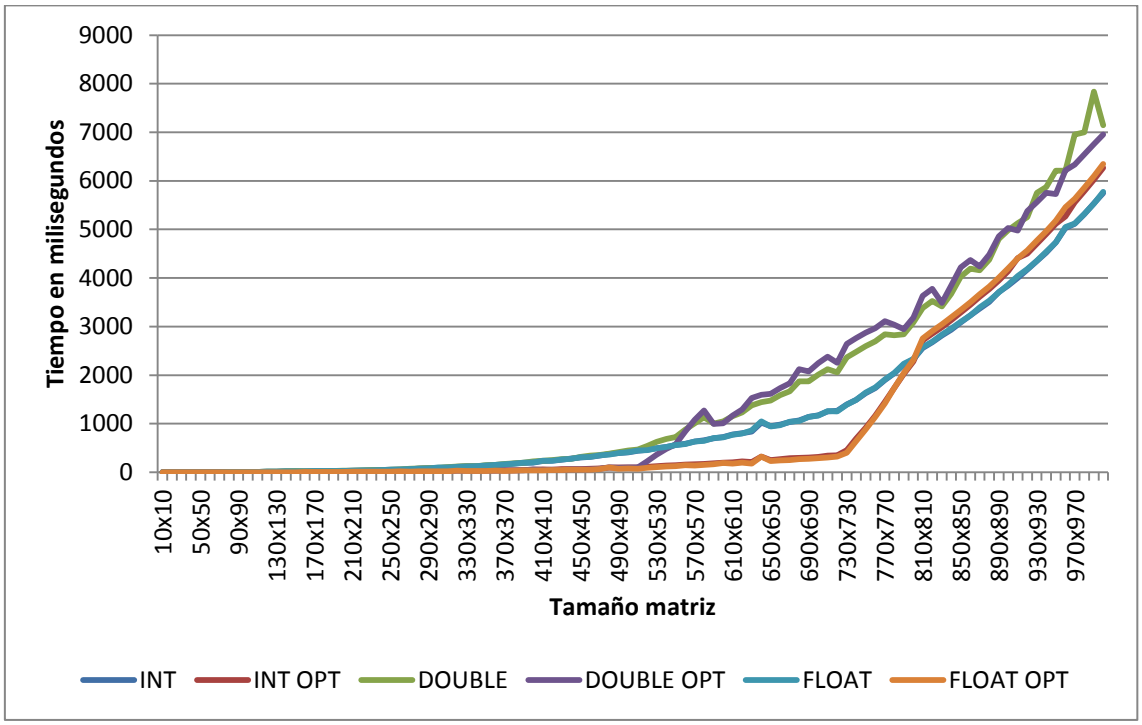
Evaluación 6: Secuencial C optimizado-Java Máquina 2

En esta última gráfica se puede observar muy bien que en la máquina 2, el C optimizado con el código Java tiene tiempos casi idénticos. Eso significa que la optimización a la hora de realizar la ejecución del programa en C es casi idéntica a la que realiza Java.

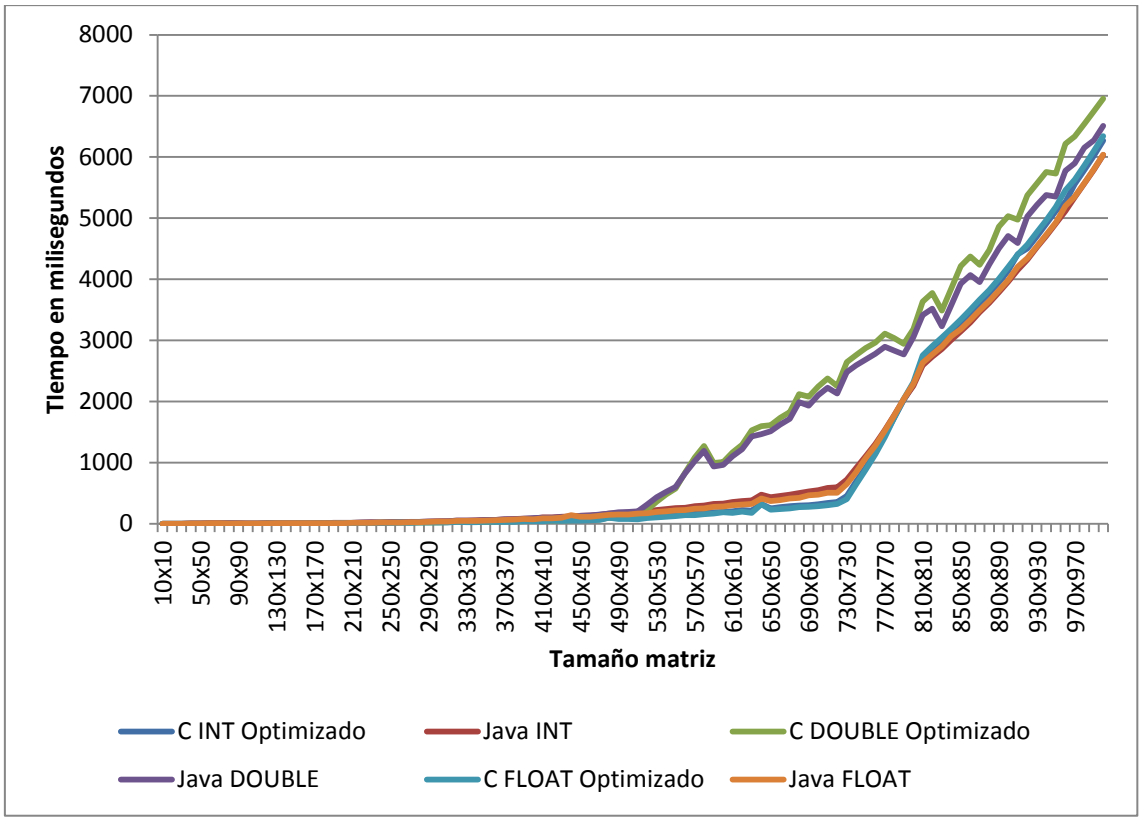
Mientras que en la máquina 3, se puede observar que en las siguientes tres gráficas que tanto en C, C optimizado y Java, los tiempos son casi idénticos.



Evaluación 7: Secuencial C-Java Máquina 3



Evaluación 8: Secuencial C optimizado Máquina 3



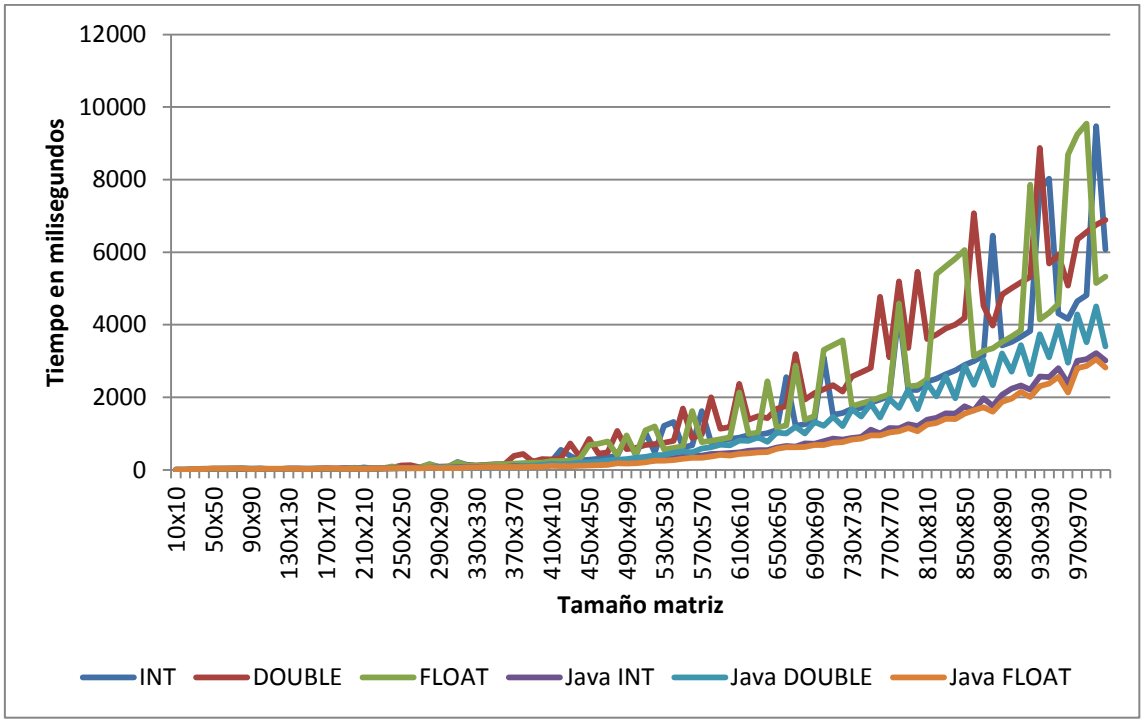
Evaluación 9: Secuencial C optimizado-Java Máquina 3

La máquina 3 dispone de un procesador *i7* por lo que los tiempos de ejecución son, como las gráficas lo han demostrado, mejores que en el resto de las máquinas.

5.3 C vs Java (OpenMP)

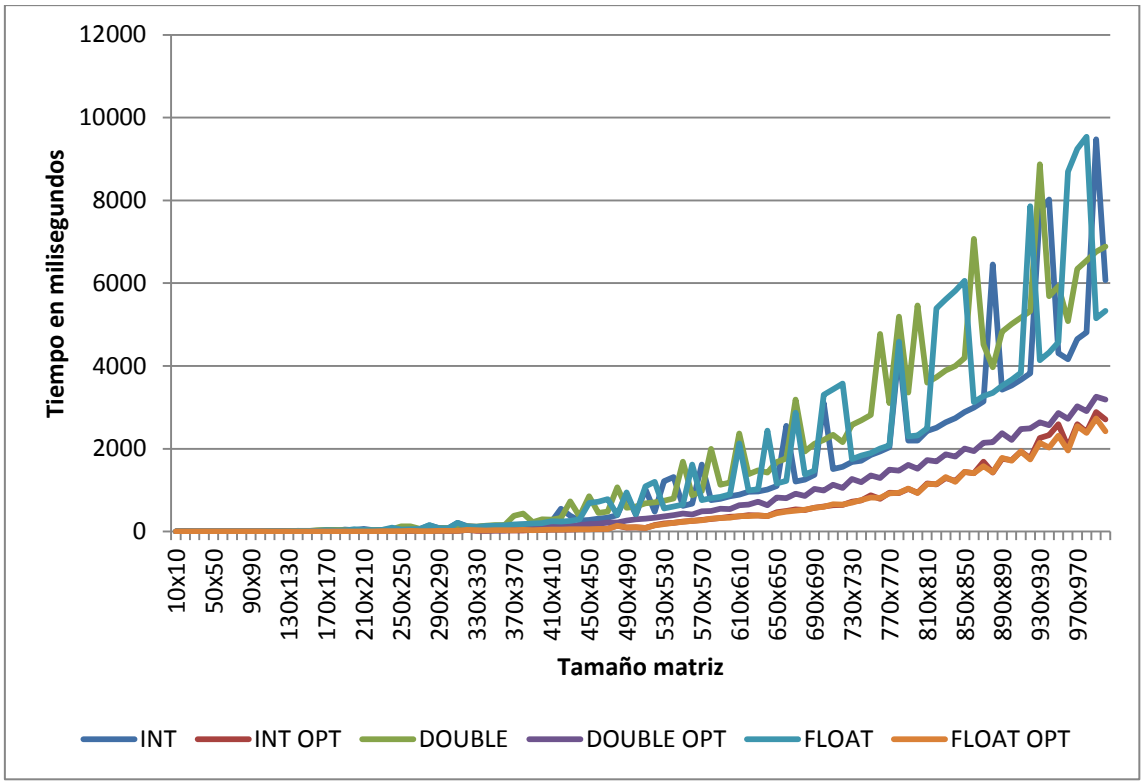
En este apartado se mostrará un conjunto de gráficas que resume los resultados obtenidos tras ejecutar los programas en modo *OpenMP* utilizando INT, DOUBLE y FLOAT en los lenguajes de programación Java y C.

Empezando con esta primera gráfica de la máquina 1, se puede observar que en C las mediciones de los tiempos están un poco descontroladas en comparación con los tiempos de Java debido a la mala gestión de memoria que ha realizado a la hora de la ejecución.



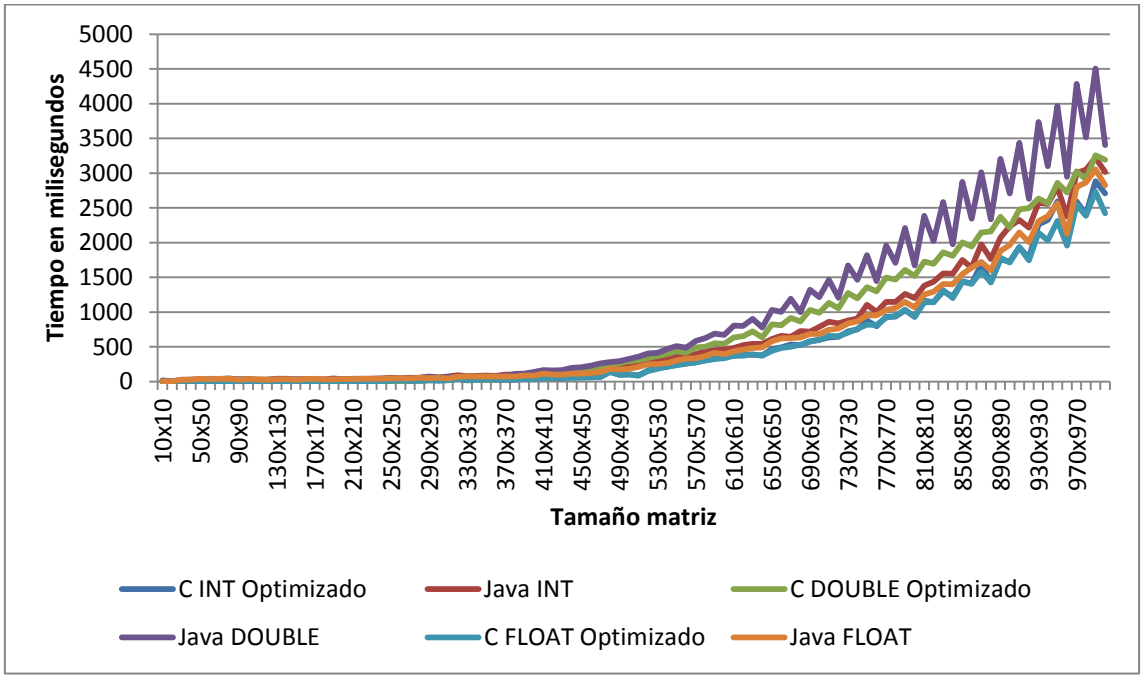
Evaluación 10: OpenMP C-Java Máquina 1

En cambio en la siguiente gráfica muestra la comparación de los tiempos resultantes en C y C optimizado, y se puede observar claramente, que la optimización funciona, como en el capítulo anterior se ha demostrado.



Evaluación 11: OpenMP C optimizado Máquina 1

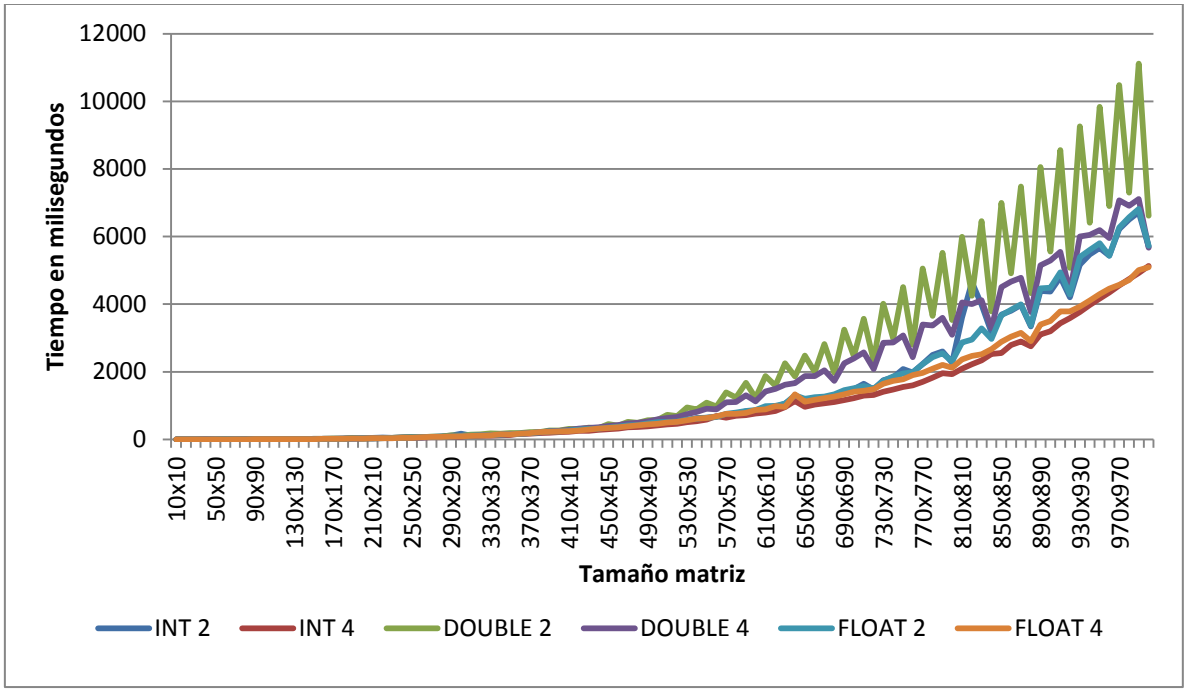
Por lo tanto, se ha decidido que a partir de aquí, las comparaciones se harán solamente con C optimizado y Java, como lo muestra la siguiente gráfica.



Evaluación 12: OpenMP C optimizado-Java Máquina 1

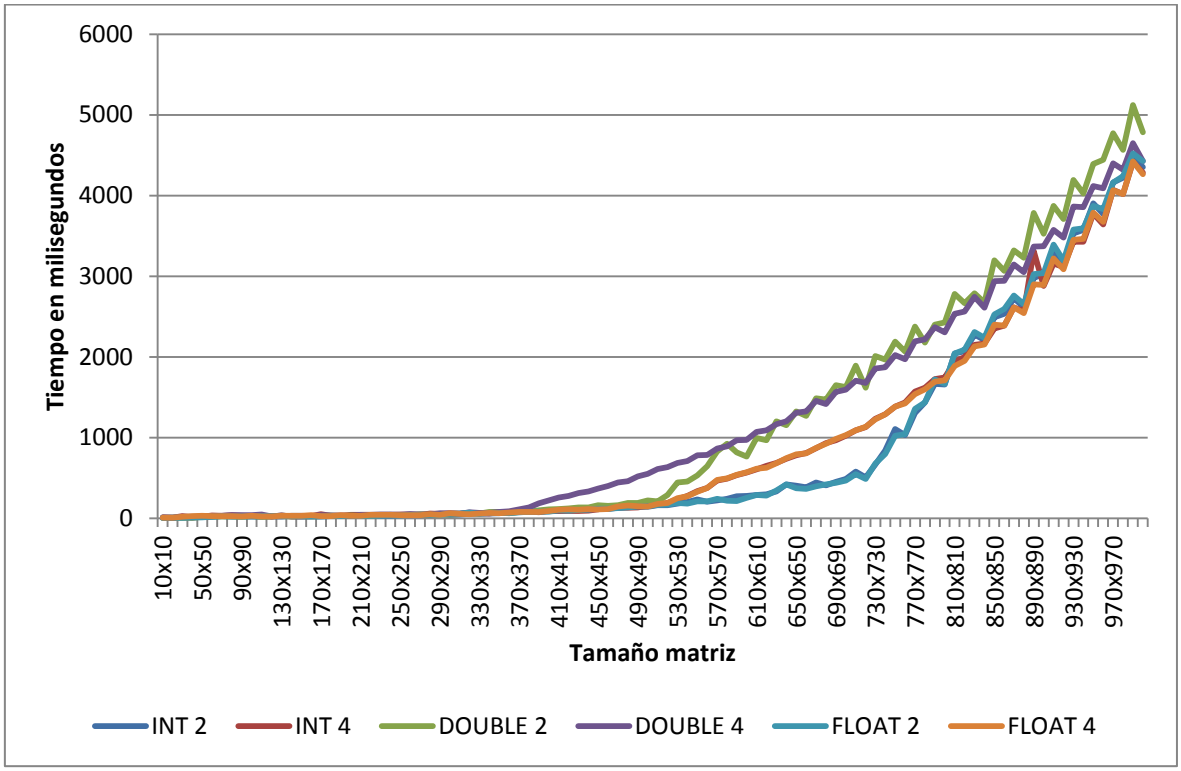
Tanto en *OpenMP* como en modo secuencial de la máquina 1, en Java tiene el mismo efecto zigzag con el tipo DOUBLE.

La siguiente gráfica muestra únicamente los tipos INT, DOUBLE y FLOAT de C, con 2 y 4 hilos ejecutados para la paralelización en la máquina 2, y como se observa, el DOUBLE hace el mismo efecto que en la máquina 1.



Evaluación 13: OpenMP C Máquina 2

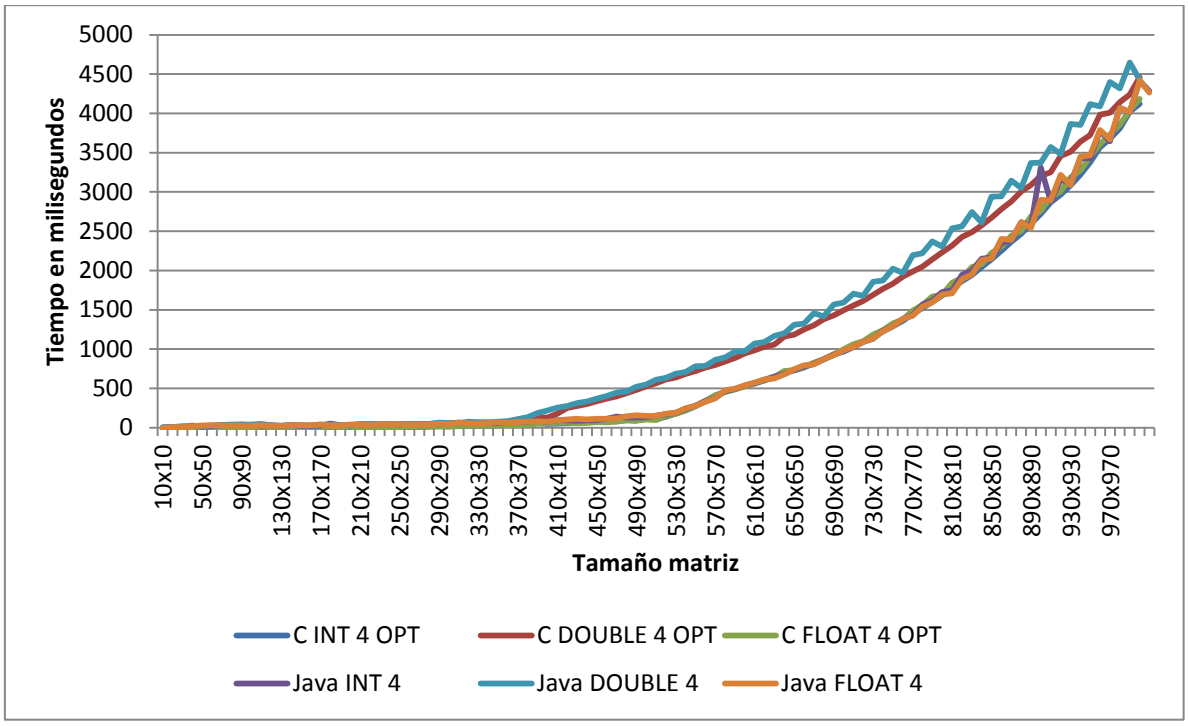
Esta gráfica muestra lo mismo que la anterior, sólo que ejecutado en Java.



Evaluación 14: OpenMP Java Máquina 2

Tras haber estudiado estas últimas gráficas se ha llegado a la conclusión de que se harán las comparaciones con los tipos INT, DOUBLE y FLOAT con 4 hilos, ya que son los más rápidos en calcular las multiplicaciones (en la máquina 2). Como bien se ha demostrado en el apartado anterior 5.2 C vs Java (secuencial) y en este también, que el código optimizado es mejor que sin optimizar, directamente se han cogido los resultados de los tiempos optimizados para las gráficas.

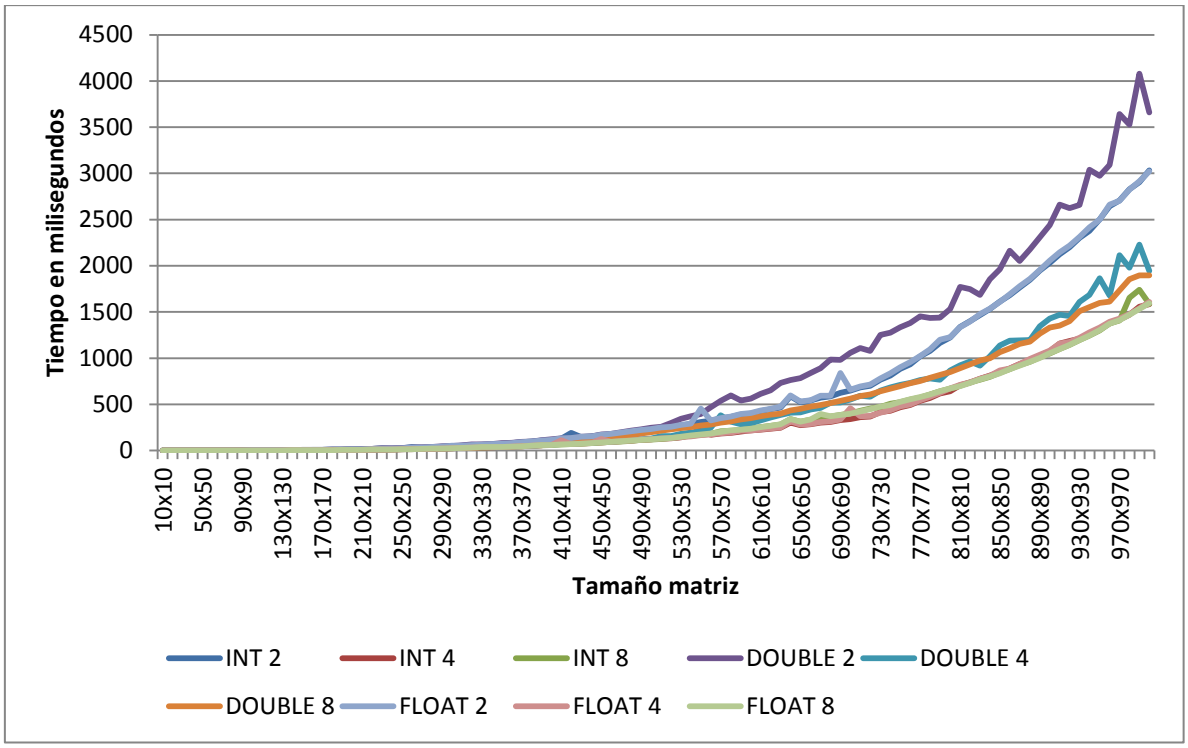
Por lo tanto, el resultado de la gráfica anterior con únicamente 4 hilos en cada tipo en una misma gráfica sería de la siguiente manera:



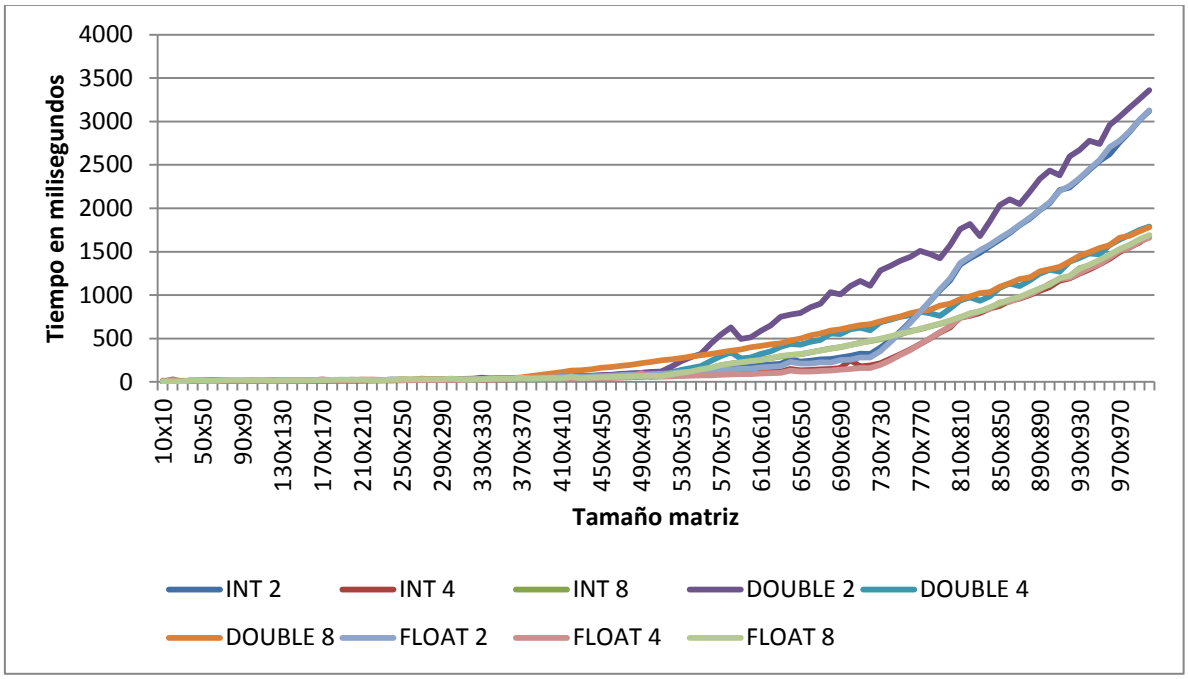
Evaluación 15: OpenMP C optimizado-Java Máquina 2

Como bien se observa, prácticamente el código ejecutado con optimización en C tarda casi el mismo tiempo que en Java.

Pasemos a los resultados de la máquina 3 (con 8 hilos). Tanto en Java como en C, ejecutando los programas con 2 hilos, no da buenos resultados de tiempo como se observa en las gráficas.

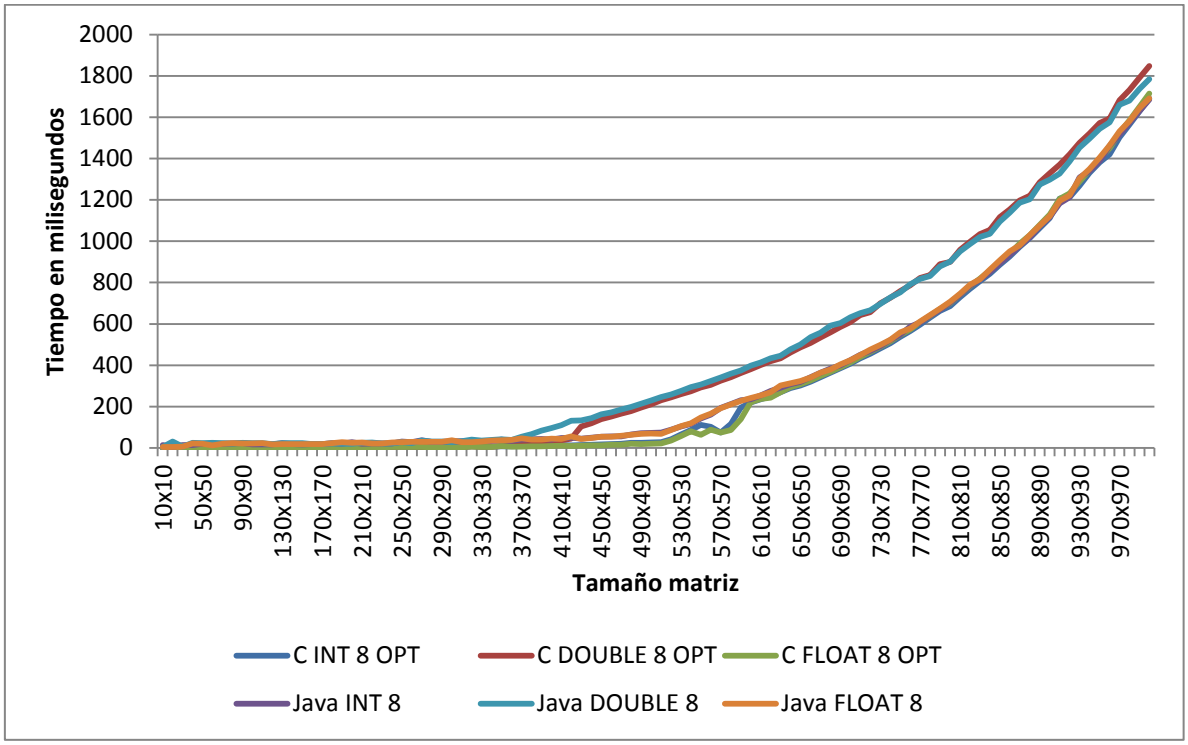


Evaluación 16: OpenMP C Máquina 3



Evaluación 17: OpenMP Java Máquina 3

Se ha procedido a ejecutar de forma optimizada el código C con 8 hilos, ya que tras haber estudiado los tiempos, es el que consume menos tiempo a la hora de la ejecución. Con los resultados obtenidos, comparándolo con el tiempo que ha tardado el programa en Java, se tiene la siguiente gráfica:

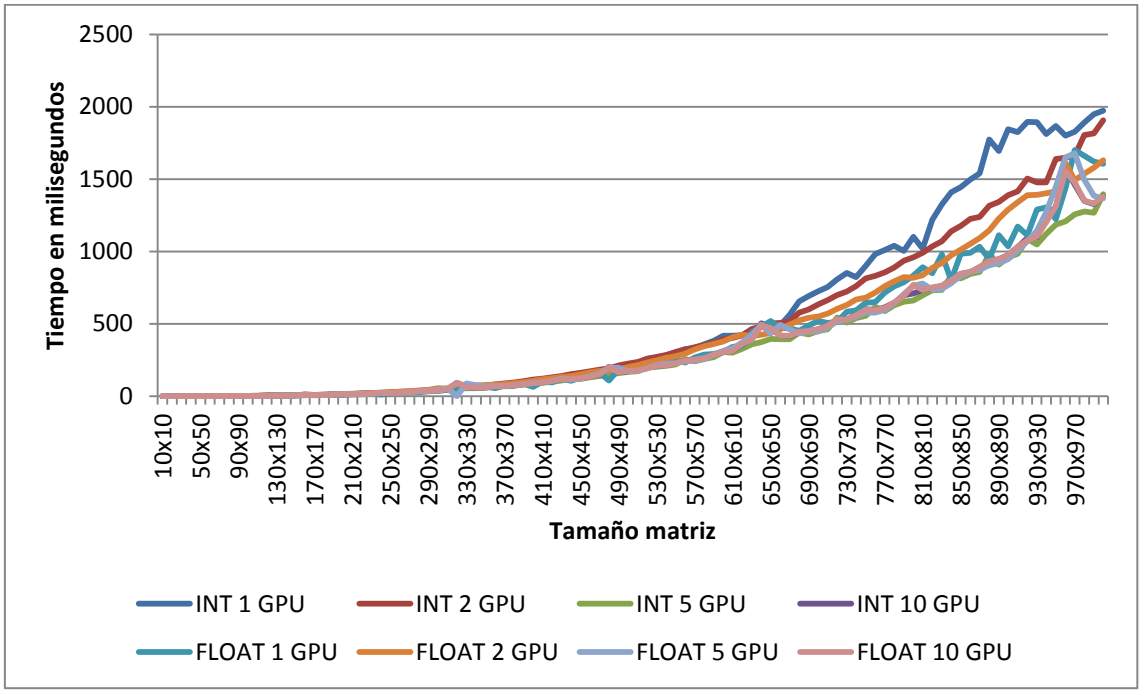


Evaluación 18: OpenMP C optimizado-Java Máquina 3

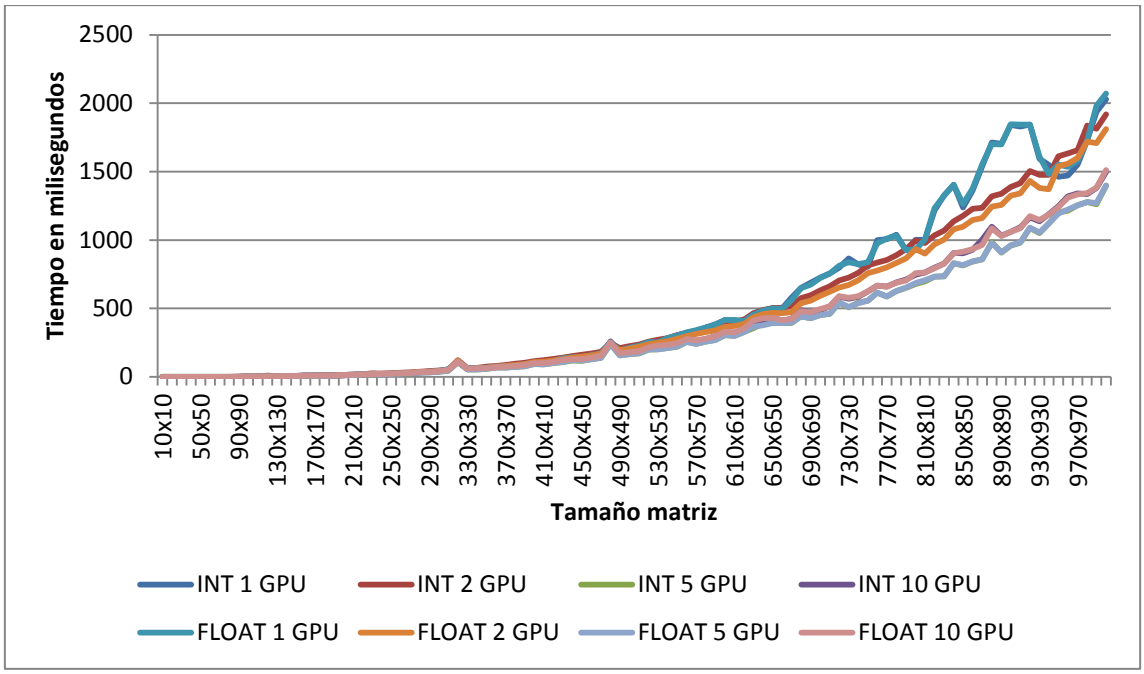
En este caso, se puede decir que en esta máquina, da igual lo que se ejecute, tanto en Java como en C optimizado, con 8 hilos se obtendrá el mismo tiempo.

5.4 C vs Java (OpenCL)

En esta sección se mostrarán los resultados de las ejecuciones de programas *OpenCL* con los datos INT, DOUBLE (únicamente en CPUs debido a que con las GPUs no está soportado) y FLOAT. Se han ejecutado utilizando 1, 2, 5 y 10 hilos (todos los tamaños de las matrices son divisibles por los números de hilos).

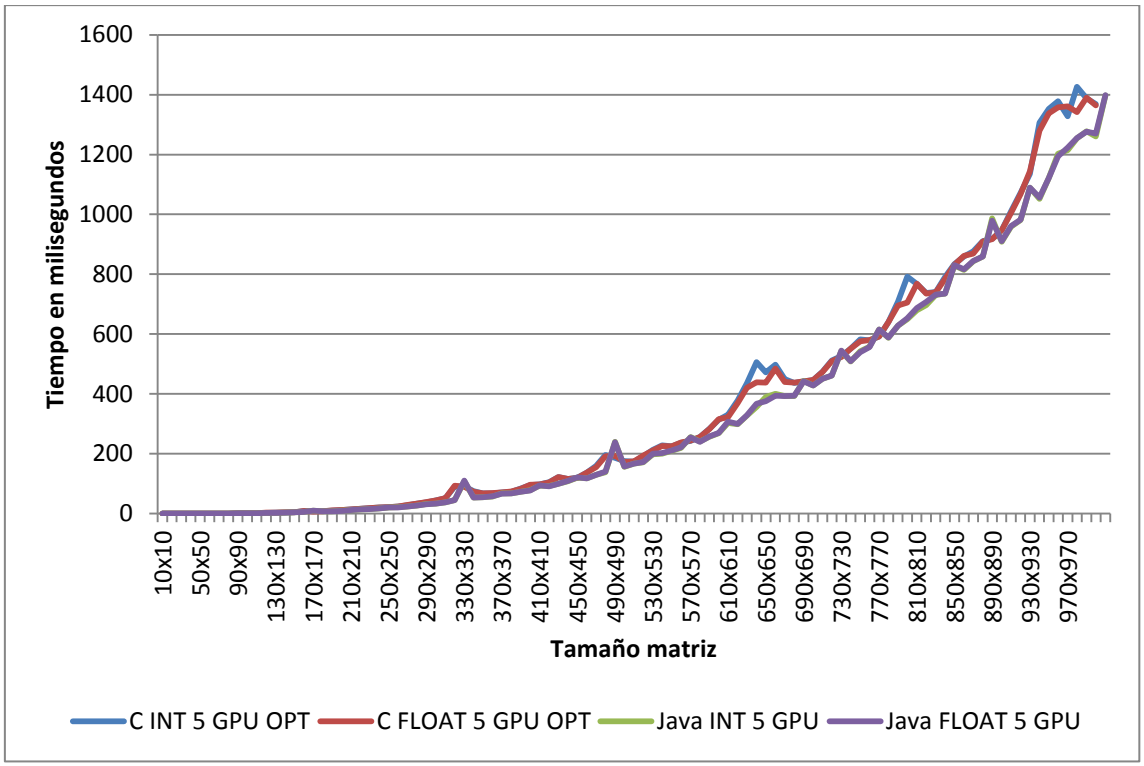


Evaluación 19: OpenCL C Máquina 1



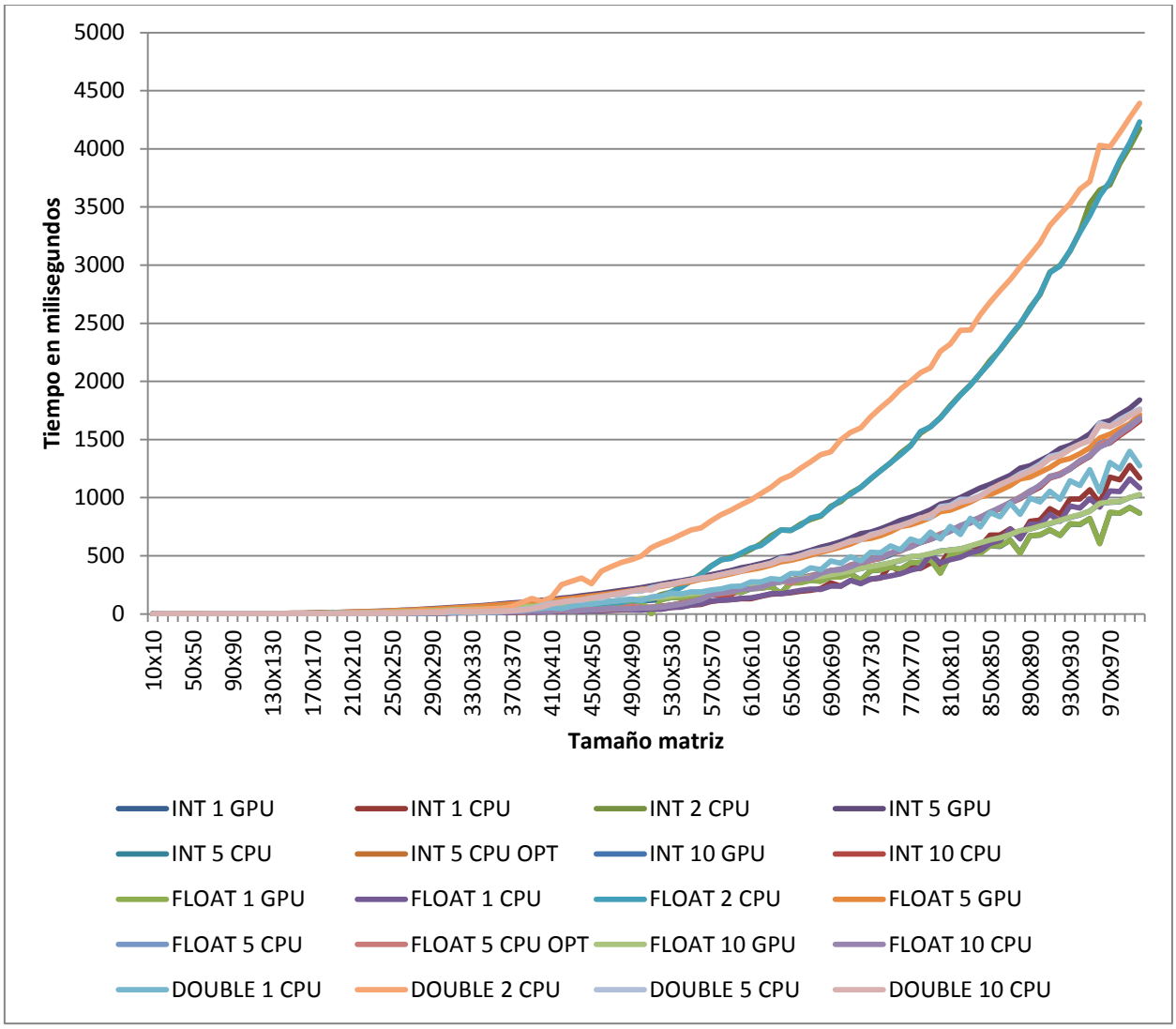
Evaluación 20: OpenCL Java Máquina 1

Tras un breve estudio de con cuántos hilos se ejecuta más rápido y eficiente, se ha llegado a la conclusión de que con 5 hilos, por lo tanto, la siguiente gráfica muestra el resultado de los tipos INT y FLOAT. El tipo DOUBLE no se muestra debido a que en esta máquina no tiene instalado OpenCL para CPU.

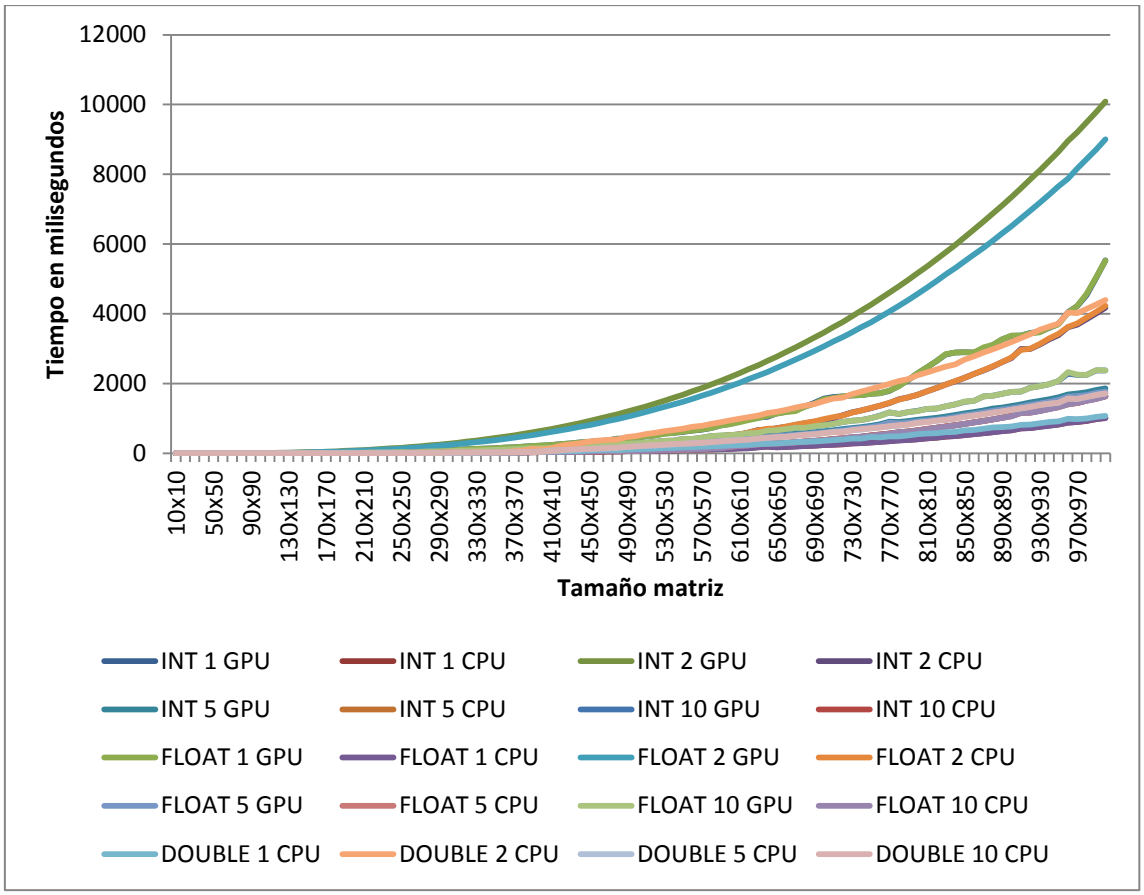


Evaluación 21: OpenCL C optimizado-Java Máquina 1

La siguiente gráfica en cambio, tiene muchos más datos ya que la máquina 2 sí tenía OpenCL para CPU. Se puede observar bien que con 2 hilos tanto en CPU como en GPU, los tiempos son demasiados altos.

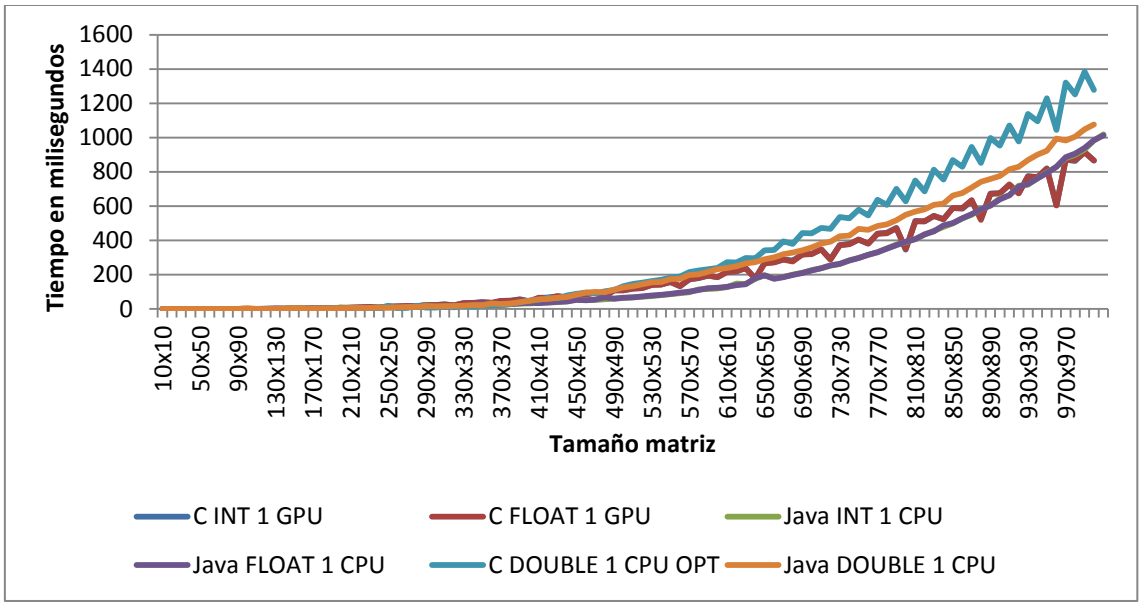


Evaluación 22: OpenCL C Máquina 2



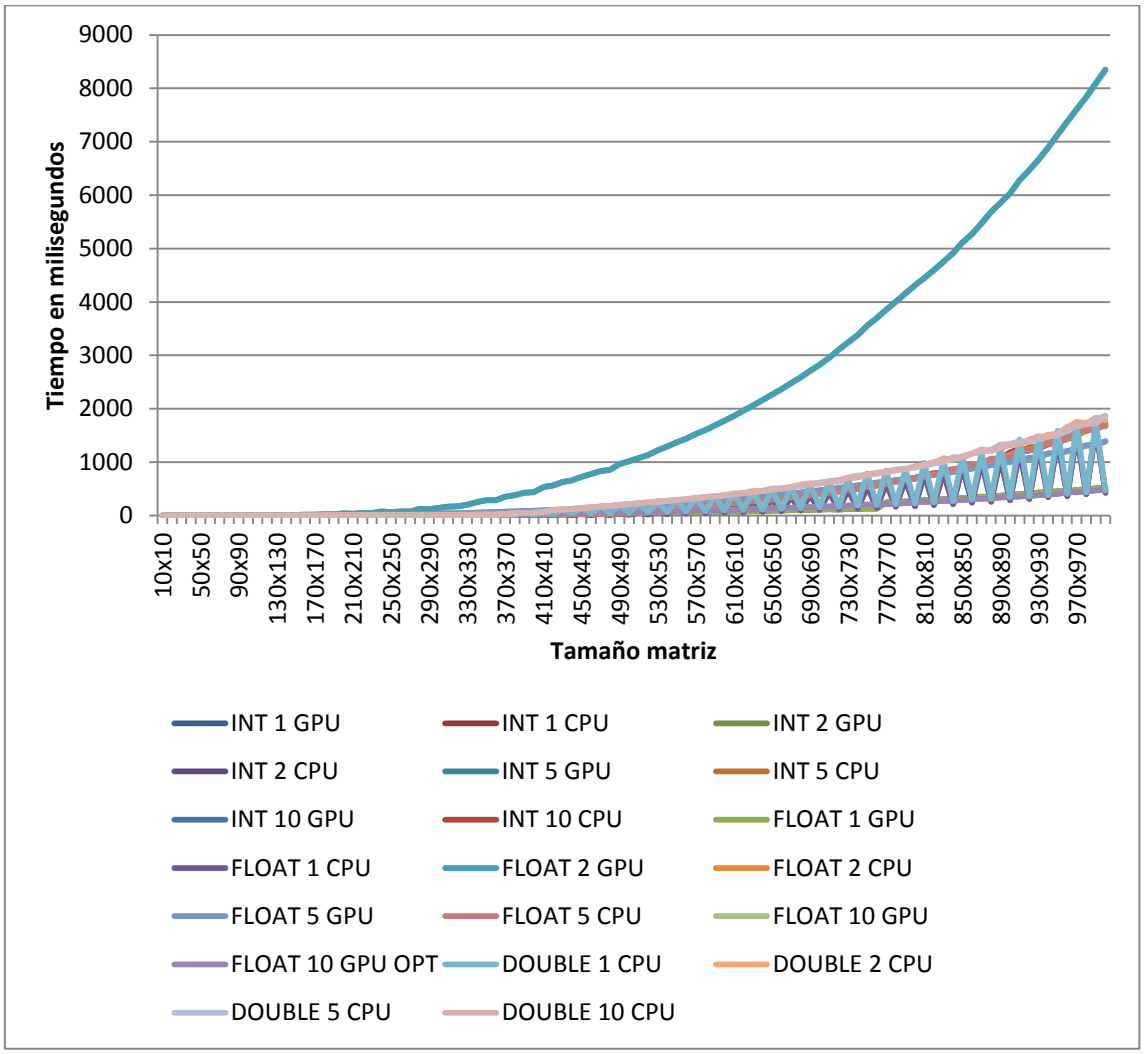
Evaluación 23: OpenCL Java Máquina 2

Tras haber estudiado los tiempos, se ha llegado a la conclusión de que con 1 hilo en esta máquina es la mejor solución, mientras que para la GPU es mejor utilizar el lenguaje C y para la CPU, el lenguaje Java.



Evaluación 24: OpenCL C optimizado-Java Máquina 2

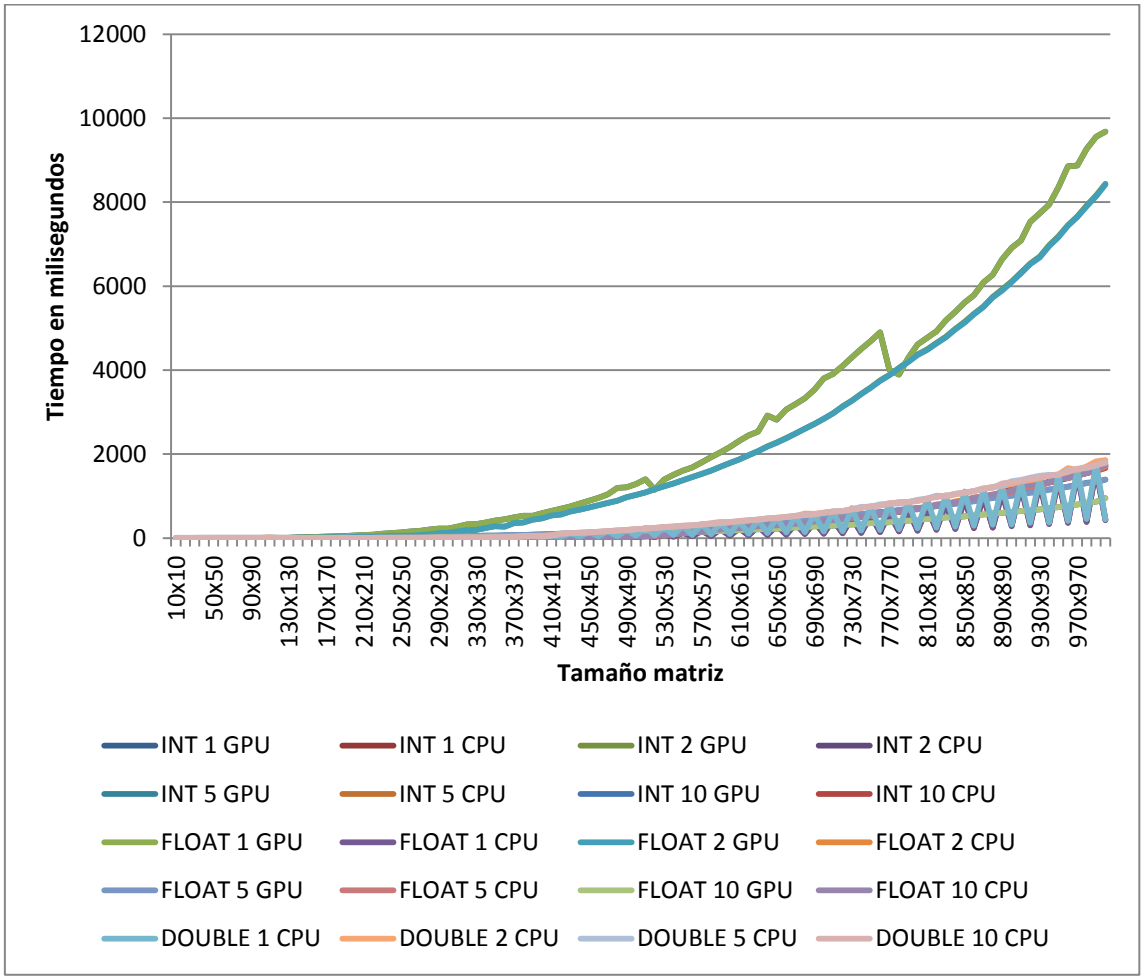
Tanto en la máquina 2 como en esta, utilizando 2 hilos no es la mejor solución pero solamente utilizando INT y FLOAT.



Evaluación 25: OpenCL C Máquina 3

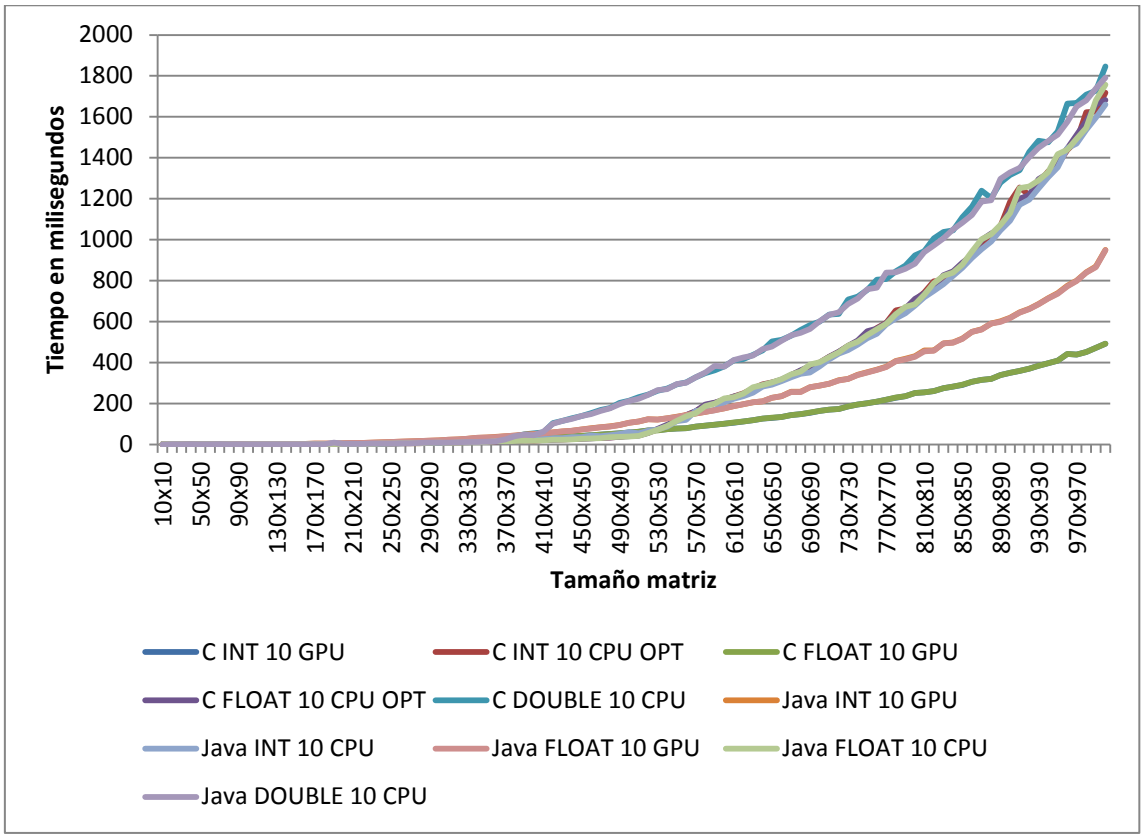
Los efectos rebote que se ven en la gráfica, se realiza únicamente cuando se utiliza 1 hilo, tanto en CPU como en GPU.

En la siguiente gráfica ocurre lo mismo pero utilizando Java.



Evaluación 26: OpenCL Java Máquina 3

Tras estudiar los tiempos, se ha llegado a la conclusión de utilizar los siguientes tipos para las ejecuciones mostradas en la gráfica siguiente.



Evaluación 27: OpenCL C optimizado-Java Máquina 3

Se han descartado los tiempos de 1 hilo en la anterior gráfica debido a que es mejor obtener un tiempo medianamente decente y no siempre tiempos muy diferentes porque los tamaños sean pares o impares. En este caso el procesador de la GPU gana utilizando 10 hilos.

6 Conclusiones

6.1 Conclusiones generales

Con la realización de este PFC se han cumplido con todos los objetivos que se propusieron, tanto los generales como los particulares, expuestos en la introducción de esta memoria.

Tras haber realizado todas las pruebas anteriores, se ha podido observar lo importante que es la paralelización a la hora de ejecutar programas que requieren muchas operaciones, y cuanto más grande los tamaños de matrices, más sentido tiene tener que paralelizar.

Cabe añadir que en promedio, el tiempo de ejecución utilizando los *flags* para optimizar la ejecución en C, ha sido de un aproximadamente 50%, y con ello igualando casi los tiempos que necesita Java.

Se ha llegado a la conclusión, que no todos los procesadores se comportan de la misma manera, como bien se ha podido en las gráficas de las tres máquinas de prueba utilizadas. Una de las razón es debido a la edad de los procesadores, ya que la máquina 1 tiene una edad de unos 6 años, la máquina 2 2 años y la máquina 3 es bastante actual. Y no sólo eso, sino también por el número de transistores que como bien dice la Ley de Moore, cada 18 meses se duplican y eso influye en el tiempo de ejecución.

Es muy muy importante tener que realizar un estudio previo de cada procesador, tanto en CPU como en GPU, de qué método es mejor utilizar (*OpenMP, OpenCL…)* y cuántos hilos utilizar, ya que puede dar tiempos de ejecuciones muy distintas (como se ha observado en las gráficas) antes de ponerse en serio con un proyecto o cualquier tarea.

6.2 Futuros trabajos

Respecto a los futuros trabajos al que se le puede aplicar el paralelismo o a las arquitecturas multicore, los siguientes proyectos podrían ser un ejemplo:

- Métodos espectrales, como en fluidos dinámicos, mecánica cuántica y predicciones de meteorología.
- Grandes operaciones matemáticas.
- Procesamiento de imágenes como también simulaciones en el campo de la física.
- Simulaciones de túneles de viento.
- Búsqueda distribuida como utiliza la compañía Google, simulaciones Monte Carlo.

- Computaciones con grandes datos a nivel de bit, como lo son en criptografía para cifrar y descifrar y el *hashing*.
- Programación dinámica como el mapeado paralelo o uniprocesador (técnica de gráficos computacionales 3D).
- Modelado gráfico.

Como se puede observar, son muchísimos campos en el que se puede aplicar la paralelización. Como hoy en día la tecnología va en aumento y en constante mejora, los programas necesitan cada vez más rendimiento, y por lo tanto la paralelización viene muy bien a la vez que los procesadores son bastantes potentes.

7 Presupuesto

Para realizar el cálculo de presupuesto de este proyecto fin de grado, se ha tomado en cuenta los siguientes aspectos:

7.1 Hardware

Nombre	Coste unitario	Cantidad	Total
Dispositivo A para desarrollo y pruebas	650 €	1	650 €
Dispositivo B para pruebas	750 €	1	750 €
Dispositivo C para pruebas	800 €	1	800 €

Tabla 7: Presupuesto Hardware

7.2 Software

Nombre	Coste unitario	Cantidad	Total
Microsoft Windows 7	120,64 €	1	120,64
Microsoft Office 2010	199 €	1	199 €
Ubuntu 12.04	0 €	3	0 €
Eclipse	0 €	1	0 €
OpenProj	0 €	1	0 €
Coste total			***319,64 €***

Tabla 8: Presupuesto Software

7.3 Amortización

Nombre	Coste	Amortización	Total
Dispositivo A para desarrollo y pruebas	650 €	20%	130 €
Dispositivo B para pruebas	750 €	20%	150 €
Dispositivo C para pruebas	800 €	100%	800 €
Microsoft Windows 7	120,64 €	20%	24,13 €
Microsoft Office 2010	199 €	20%	39,8 €
Coste total			***1143,93 €***

Tabla 9: Presupuesto amortización

7.4 Personal

Nombre	Coste / hora	Horas trabajadas	Total
Analista	14 €	192	2.688 €
Consultor / programador	19 €	464	8.816 €
Coste total			***11.504 €***

Tabla 10: Presupuesto personal

7.5 Gastos de transporte

Nombre	Coste unitario	Cantidad	Total
Vuelo Hamburgo a Madrid Barajas y vuelta	226,31 €	1	226,31 €
Coste total			***226,31 €***

Tabla 11: Presupuesto gastos de transporte

7.6 Total

Nombre	Total
Hardware	1143,93 €
Software	319,64 €
Personal	11.504,00 €
Gastos de transporte	226,31 €
Coste total	***13.193,88 €***

Tabla 12: Presupuesto total

8 Diagrama de Gantt

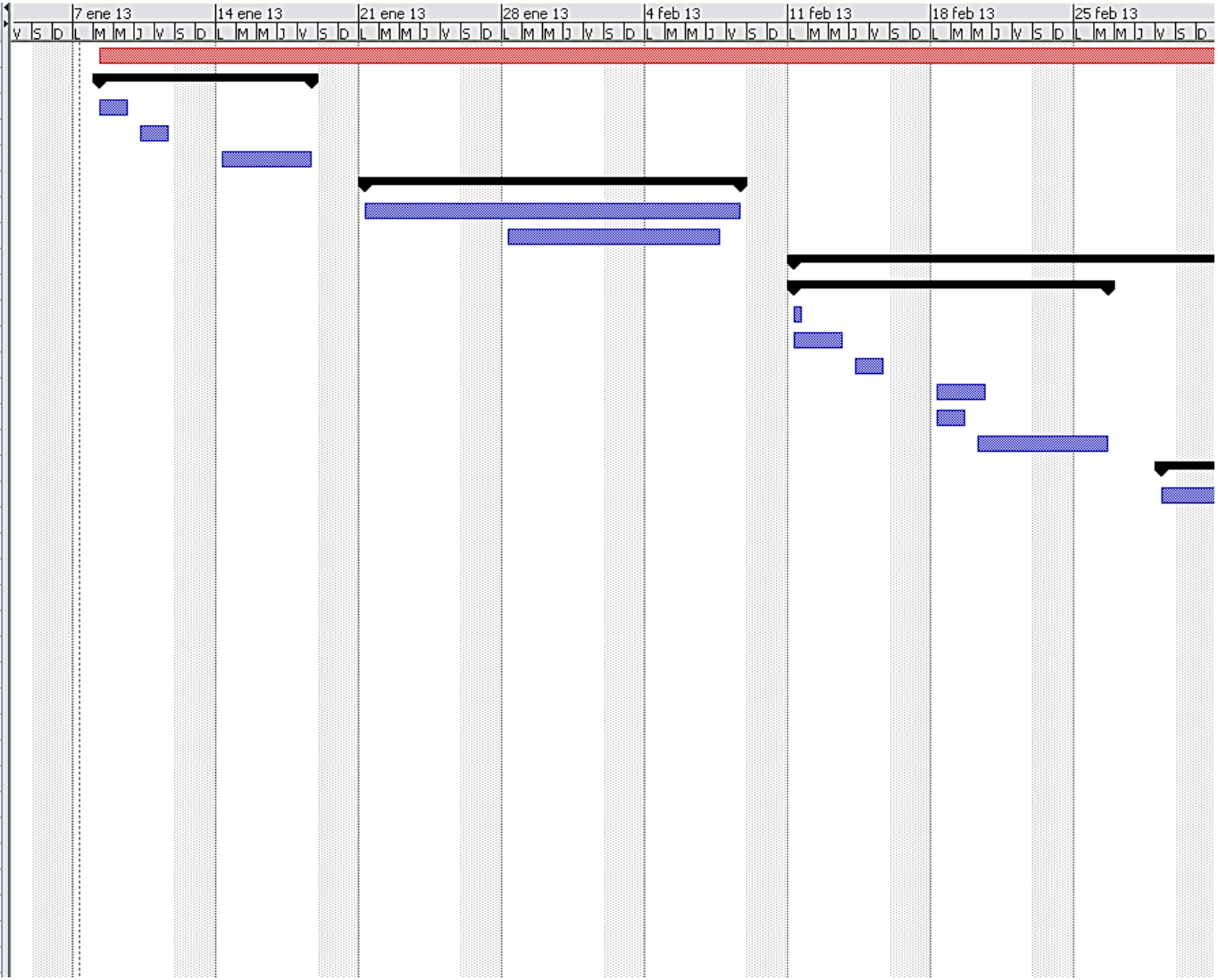
Henry Laurence Gantt, nacido en el Condado de Clavert en Estados Unidos en el año 1861, fue ingeniero industrial mecánico estadounidense. Estudió la forma de una mejor organización del trabajo industrial, en el cual Gantt se centró en el control y planificación de las operaciones productivas mediante el uso de técnicas gráficas, y de allí salió en denominado Diagrama de Gannt.

Este diagrama es una herramienta gráfica cuyo objetivo principal es mostrar el tiempo de dedicación previsto o real en diferentes tareas o en este caso, de este proyecto Fin de Grado. En el eje horizontal del diagrama muestra las unidades de tiempo, y en el vertical las distintas funciones o actividades.

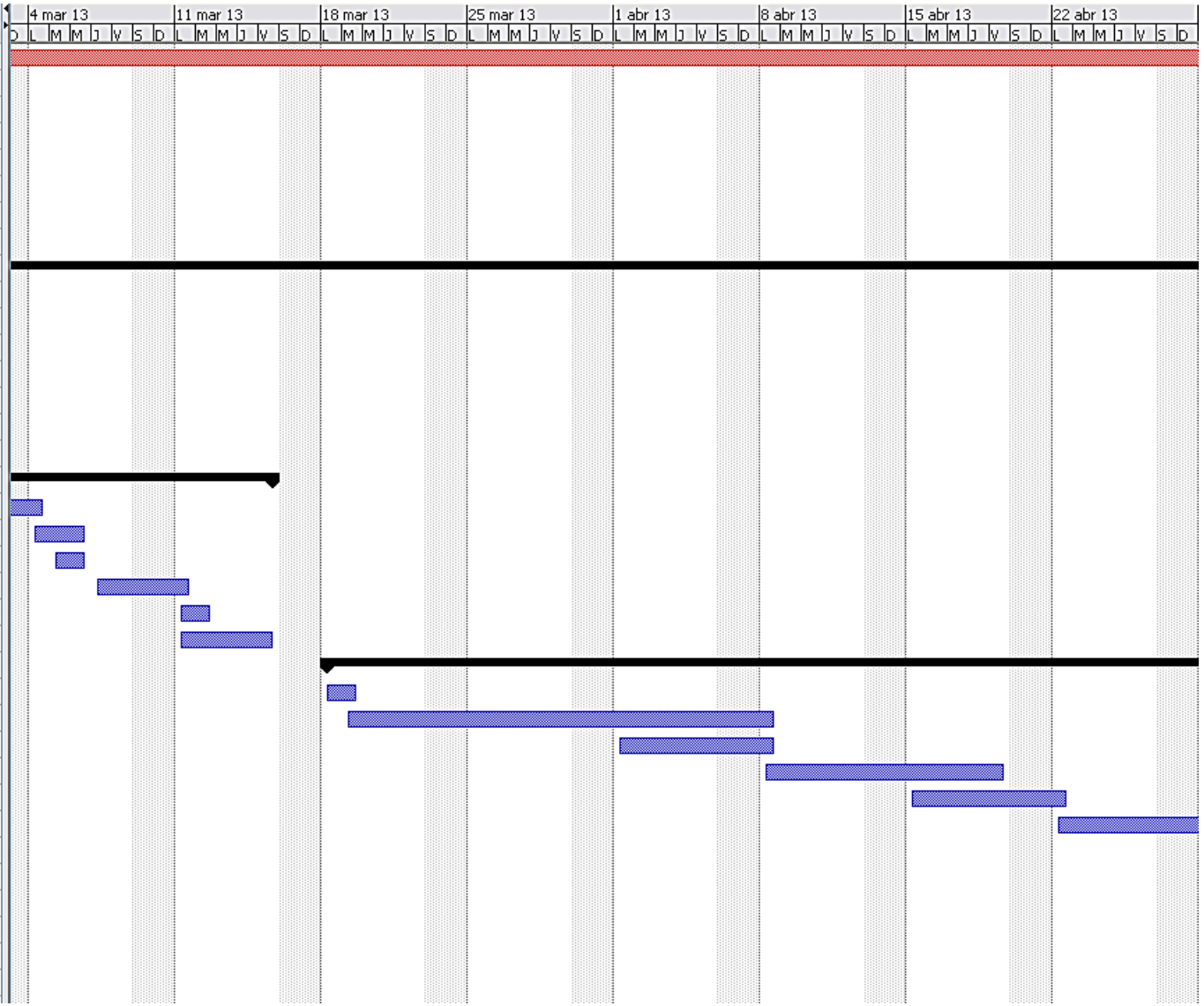
Para realizar dicha diagrama se ha utilizado un programa OpenSource denominado *OpenProj* [1].

[1] http://sourceforge.net/projects/openproj/

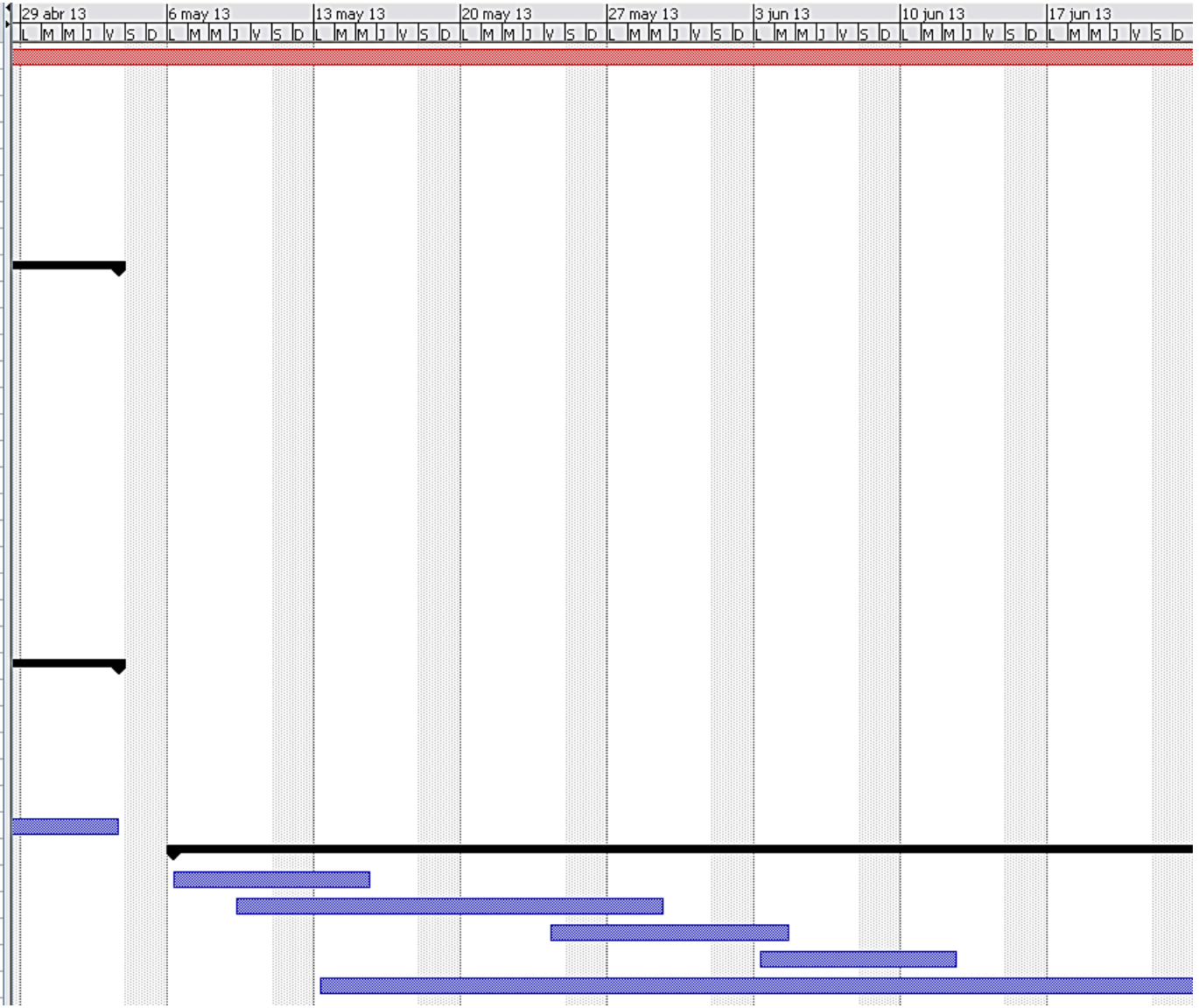
Nombre	Duración	Inicio	Terminado
PROYECTO	213 days	8/01/13 8:00	31/10/13 17:00
Especulación	**9 days**	**8/01/13 8:00**	**18/01/13 17:00**
Definición de objetivos	2 days	8/01/13 8:00	9/01/13 17:00
Planificación	2 days	10/01/13 8:00	11/01/13 17:00
Preparación del sistema	5 days	14/01/13 8:00	18/01/13 17:00
Análisis	**15 days**	**21/01/13 8:00**	**8/02/13 17:00**
Estado de la cuestión	15 days	21/01/13 8:00	8/02/13 17:00
Análisis de requisitos	9 days	28/01/13 8:00	7/02/13 17:00
Desarrollo	**60 days?**	**11/02/13 8:00**	**3/05/13 17:00**
Implementación secuencial	**12 days**	**11/02/13 8:00**	**26/02/13 17:00**
Algoritmo multiplicación matriz	1 day	11/02/13 8:00	11/02/13 17:00
Realización del programa en C	3 days	11/02/13 8:00	13/02/13 17:00
Pruebas programa en C	2 days	14/02/13 8:00	15/02/13 17:00
Realización del programa en Java	3 days	16/02/13 8:00	20/02/13 17:00
Pruebas programa en Java	2 days	18/02/13 8:00	19/02/13 17:00
Mejoras	5 days	20/02/13 8:00	26/02/13 17:00
Implementación OpenMP	**11 days**	**1/03/13 8:00**	**15/03/13 17:00**
Algoritmo multiplicación matriz	2 days	1/03/13 8:00	4/03/13 17:00
Realización del programa en C	3 days	3/03/13 8:00	6/03/13 17:00
Pruebas programa en C	2 days	5/03/13 8:00	6/03/13 17:00
Realización del programa en Java	3 days	7/03/13 8:00	11/03/13 17:00
Pruebas programa en Java	2 days	9/03/13 8:00	12/03/13 17:00
Mejoras	5 days	11/03/13 8:00	15/03/13 17:00
Implementación OpenCL	**35 days?**	**18/03/13 8:00**	**3/05/13 17:00**
Algoritmo multiplicación matriz	2 days	18/03/13 8:00	19/03/13 17:00
Realización del programa en C	15 days	19/03/13 8:00	8/04/13 17:00
Pruebas programa en C	6 days	1/04/13 7:00	8/04/13 17:00
Realización del programa en Java	10 days	8/04/13 7:00	19/04/13 17:00
Pruebas programa en Java	6 days	15/04/13 8:00	22/04/13 17:00
Mejoras	10 days?	22/04/13 8:00	3/05/13 17:00
Últimos pasos	**85 days**	**6/05/13 8:00**	**30/08/13 17:00**
Pruebas de todos los programas	8 days	6/05/13 8:00	15/05/13 17:00
Revisión y otras mejoras	15 days	9/05/13 7:00	29/05/13 17:00
Pruebas finales	8 days	24/05/13 7:00	4/06/13 17:00
Mejoras finales	8 days	3/06/13 7:00	12/06/13 17:00
Escritura memoria	80 days	13/05/13 7:00	30/08/13 17:00



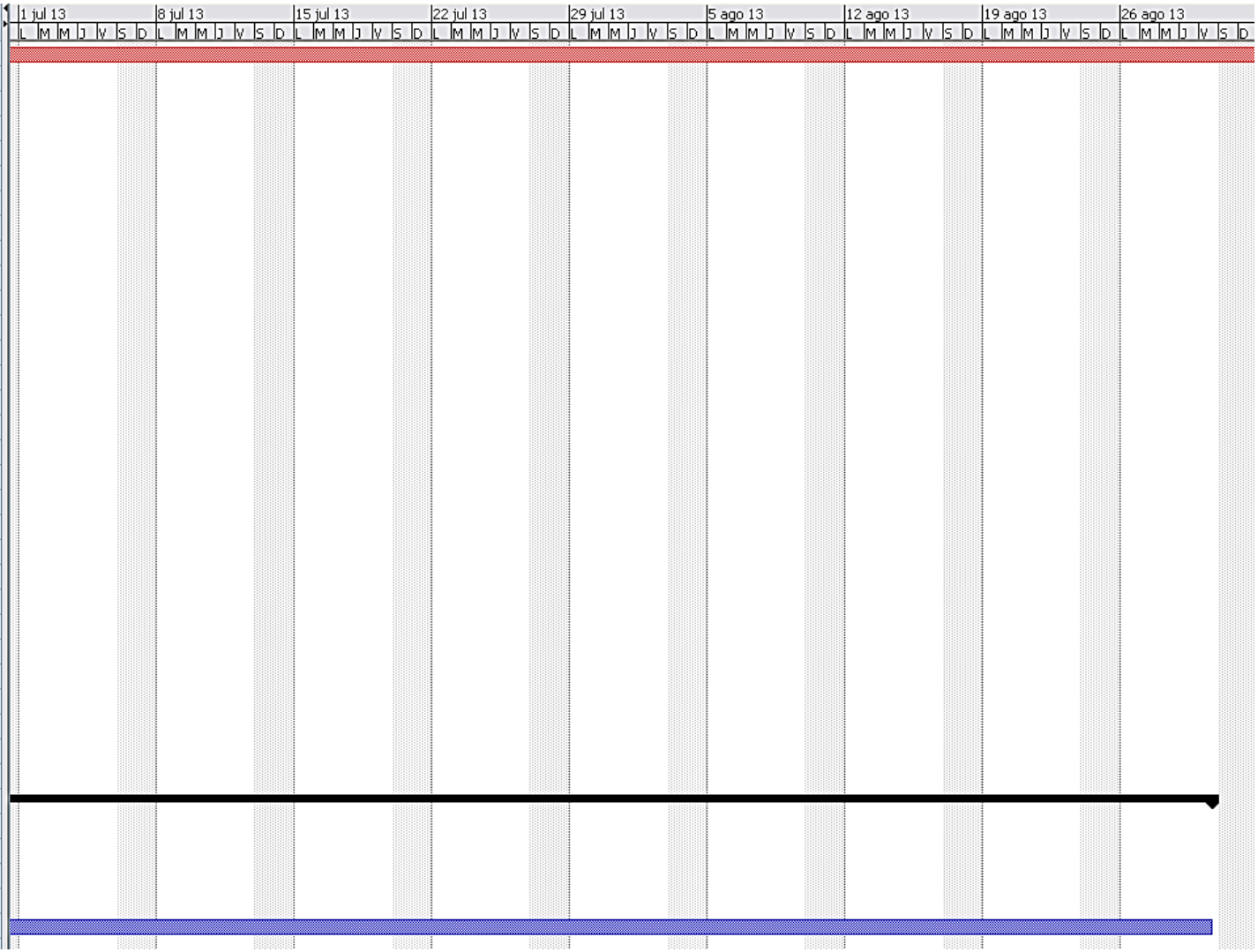
Nombre	Duración	Inicio	Terminado
PROYECTO	213 days	8/01/13 8:00	31/10/13 17:00
Especulación	**9 days**	**8/01/13 8:00**	**18/01/13 17:00**
Definición de objetivos	2 days	8/01/13 8:00	9/01/13 17:00
Planificación	2 days	10/01/13 8:00	11/01/13 17:00
Preparación del sistema	5 days	14/01/13 8:00	18/01/13 17:00
Análisis	**15 days**	**21/01/13 8:00**	**8/02/13 17:00**
Estado de la cuestión	15 days	21/01/13 8:00	8/02/13 17:00
Análisis de requisitos	9 days	28/01/13 8:00	7/02/13 17:00
Desarrollo	**60 days?**	**11/02/13 8:00**	**3/05/13 17:00**
Implementación secuencial	**12 days**	**11/02/13 8:00**	**26/02/13 17:00**
Algoritmo multiplicación matriz	1 day	11/02/13 8:00	11/02/13 17:00
Realización del programa en C	3 days	11/02/13 8:00	13/02/13 17:00
Pruebas programa en C	2 days	14/02/13 8:00	15/02/13 17:00
Realización del programa en Java	3 days	16/02/13 8:00	20/02/13 17:00
Pruebas programa en Java	2 days	18/02/13 8:00	19/02/13 17:00
Mejoras	5 days	20/02/13 8:00	26/02/13 17:00
Implementación OpenMP	**11 days**	**1/03/13 8:00**	**15/03/13 17:00**
Algoritmo multiplicación matriz	2 days	1/03/13 8:00	4/03/13 17:00
Realización del programa en C	3 days	3/03/13 8:00	6/03/13 17:00
Pruebas programa en C	2 days	5/03/13 8:00	6/03/13 17:00
Realización del programa en Java	3 days	7/03/13 8:00	11/03/13 17:00
Pruebas programa en Java	2 days	9/03/13 8:00	12/03/13 17:00
Mejoras	5 days	11/03/13 8:00	15/03/13 17:00
Implementación OpenCL	**35 days?**	**18/03/13 8:00**	**3/05/13 17:00**
Algoritmo multiplicación matriz	2 days	18/03/13 8:00	19/03/13 17:00
Realización del programa en C	15 days	19/03/13 8:00	8/04/13 17:00
Pruebas programa en C	6 days	1/04/13 7:00	8/04/13 17:00
Realización del programa en Java	10 days	8/04/13 7:00	19/04/13 17:00
Pruebas programa en Java	6 days	15/04/13 8:00	22/04/13 17:00
Mejoras	10 days?	22/04/13 8:00	3/05/13 17:00
Últimos pasos	**85 days**	**6/05/13 8:00**	**30/08/13 17:00**
Pruebas de todos los programas	8 days	6/05/13 8:00	15/05/13 17:00
Revisión y otras mejoras	15 days	9/05/13 7:00	29/05/13 17:00
Pruebas finales	8 days	24/05/13 7:00	4/06/13 17:00
Mejoras finales	8 days	3/06/13 7:00	12/06/13 17:00
Escritura memoria	80 days	13/05/13 7:00	30/08/13 17:00



Nombre	Duración	Inicio	Terminado
PROYECTO	213 days	8/01/13 8:00	31/10/13 17:00
Especulación	**9 days**	**8/01/13 8:00**	**18/01/13 17:00**
Definición de objetivos	2 days	8/01/13 8:00	9/01/13 17:00
Planificación	2 days	10/01/13 8:00	11/01/13 17:00
Preparación del sistema	5 days	14/01/13 8:00	18/01/13 17:00
Análisis	**15 days**	**21/01/13 8:00**	**8/02/13 17:00**
Estado de la cuestión	15 days	21/01/13 8:00	8/02/13 17:00
Análisis de requisitos	9 days	28/01/13 8:00	7/02/13 17:00
Desarrollo	**60 days?**	**11/02/13 8:00**	**3/05/13 17:00**
Implementación secuencial	**12 days**	**11/02/13 8:00**	**26/02/13 17:00**
Algoritmo multiplicación matriz	1 day	11/02/13 8:00	11/02/13 17:00
Realización del programa en C	3 days	11/02/13 8:00	13/02/13 17:00
Pruebas programa en C	2 days	14/02/13 8:00	15/02/13 17:00
Realización del programa en Java	3 days	16/02/13 8:00	20/02/13 17:00
Pruebas programa en Java	2 days	18/02/13 8:00	19/02/13 17:00
Mejoras	5 days	20/02/13 8:00	26/02/13 17:00
Implementación OpenMP	**11 days**	**1/03/13 8:00**	**15/03/13 17:00**
Algoritmo multiplicación matriz	2 days	1/03/13 8:00	4/03/13 17:00
Realización del programa en C	3 days	3/03/13 8:00	6/03/13 17:00
Pruebas programa en C	2 days	5/03/13 8:00	6/03/13 17:00
Realización del programa en Java	3 days	7/03/13 8:00	11/03/13 17:00
Pruebas programa en Java	2 days	9/03/13 8:00	12/03/13 17:00
Mejoras	5 days	11/03/13 8:00	15/03/13 17:00
Implementación OpenCL	**35 days?**	**18/03/13 8:00**	**3/05/13 17:00**
Algoritmo multiplicación matriz	2 days	18/03/13 8:00	19/03/13 17:00
Realización del programa en C	15 days	19/03/13 8:00	8/04/13 17:00
Pruebas programa en C	6 days	1/04/13 7:00	8/04/13 17:00
Realización del programa en Java	10 days	8/04/13 7:00	19/04/13 17:00
Pruebas programa en Java	6 days	15/04/13 8:00	22/04/13 17:00
Mejoras	10 days?	22/04/13 8:00	3/05/13 17:00
Últimos pasos	**85 days**	**6/05/13 8:00**	**30/08/13 17:00**
Pruebas de todos los programas	8 days	6/05/13 8:00	15/05/13 17:00
Revisión y otras mejoras	15 days	9/05/13 7:00	29/05/13 17:00
Pruebas finales	8 days	24/05/13 7:00	4/06/13 17:00
Mejoras finales	8 days	3/06/13 7:00	12/06/13 17:00
Escritura memoria	80 days	13/05/13 7:00	30/08/13 17:00



Nombre	Duración	Inicio	Terminado
PROYECTO	213 days	8/01/13 8:00	31/10/13 17:00
⊟Especulación	**9 days**	**8/01/13 8:00**	**18/01/13 17:00**
Definición de objetivos	2 days	8/01/13 8:00	9/01/13 17:00
Planificación	2 days	10/01/13 8:00	11/01/13 17:00
Preparación del sistema	5 days	14/01/13 8:00	18/01/13 17:00
⊟Análisis	**15 days**	**21/01/13 8:00**	**8/02/13 17:00**
Estado de la cuestión	15 days	21/01/13 8:00	8/02/13 17:00
Análisis de requisitos	9 days	28/01/13 8:00	7/02/13 17:00
⊟Desarrollo	**60 days?**	**11/02/13 8:00**	**3/05/13 17:00**
⊟Implementación secuencial	**12 days**	**11/02/13 8:00**	**26/02/13 17:00**
Algoritmo multiplicación matriz	1 day	11/02/13 8:00	11/02/13 17:00
Realización del programa en C	3 days	11/02/13 8:00	13/02/13 17:00
Pruebas programa en C	2 days	14/02/13 8:00	15/02/13 17:00
Realización del programa en Java	3 days	16/02/13 8:00	20/02/13 17:00
Pruebas programa en Java	2 days	18/02/13 8:00	19/02/13 17:00
Mejoras	5 days	20/02/13 8:00	26/02/13 17:00
⊟Implementación OpenMP	**11 days**	**1/03/13 8:00**	**15/03/13 17:00**
Algoritmo multiplicación matriz	2 days	1/03/13 8:00	4/03/13 17:00
Realización del programa en C	3 days	3/03/13 8:00	6/03/13 17:00
Pruebas programa en C	2 days	5/03/13 8:00	6/03/13 17:00
Realización del programa en Java	3 days	7/03/13 8:00	11/03/13 17:00
Pruebas programa en Java	2 days	9/03/13 8:00	12/03/13 17:00
Mejoras	5 days	11/03/13 8:00	15/03/13 17:00
⊟Implementación OpenCL	**35 days?**	**18/03/13 8:00**	**3/05/13 17:00**
Algoritmo multiplicación matriz	2 days	18/03/13 8:00	19/03/13 17:00
Realización del programa en C	15 days	19/03/13 8:00	8/04/13 17:00
Pruebas programa en C	6 days	1/04/13 7:00	8/04/13 17:00
Realización del programa en Java	10 days	8/04/13 7:00	19/04/13 17:00
Pruebas programa en Java	6 days	15/04/13 8:00	22/04/13 17:00
Mejoras	10 days?	22/04/13 8:00	3/05/13 17:00
⊟Últimos pasos	**85 days**	**6/05/13 8:00**	**30/08/13 17:00**
Pruebas de todos los programas	8 days	6/05/13 8:00	15/05/13 17:00
Revisión y otras mejoras	15 days	9/05/13 7:00	29/05/13 17:00
Pruebas finales	8 days	24/05/13 7:00	4/06/13 17:00
Mejoras finales	8 days	3/06/13 7:00	12/06/13 17:00
Escritura memoria	80 days	13/05/13 7:00	30/08/13 17:00



9 Apéndices

En esta sección se detallarán todos los apéndices que contiene este Proyecto Fin de Grado.

9.1 Apéndice 1: Comandos utilizados y funcionamiento de OpenCL

Los pasos que realiza OpenCL para ejecutar un programa son de la siguiente manera:

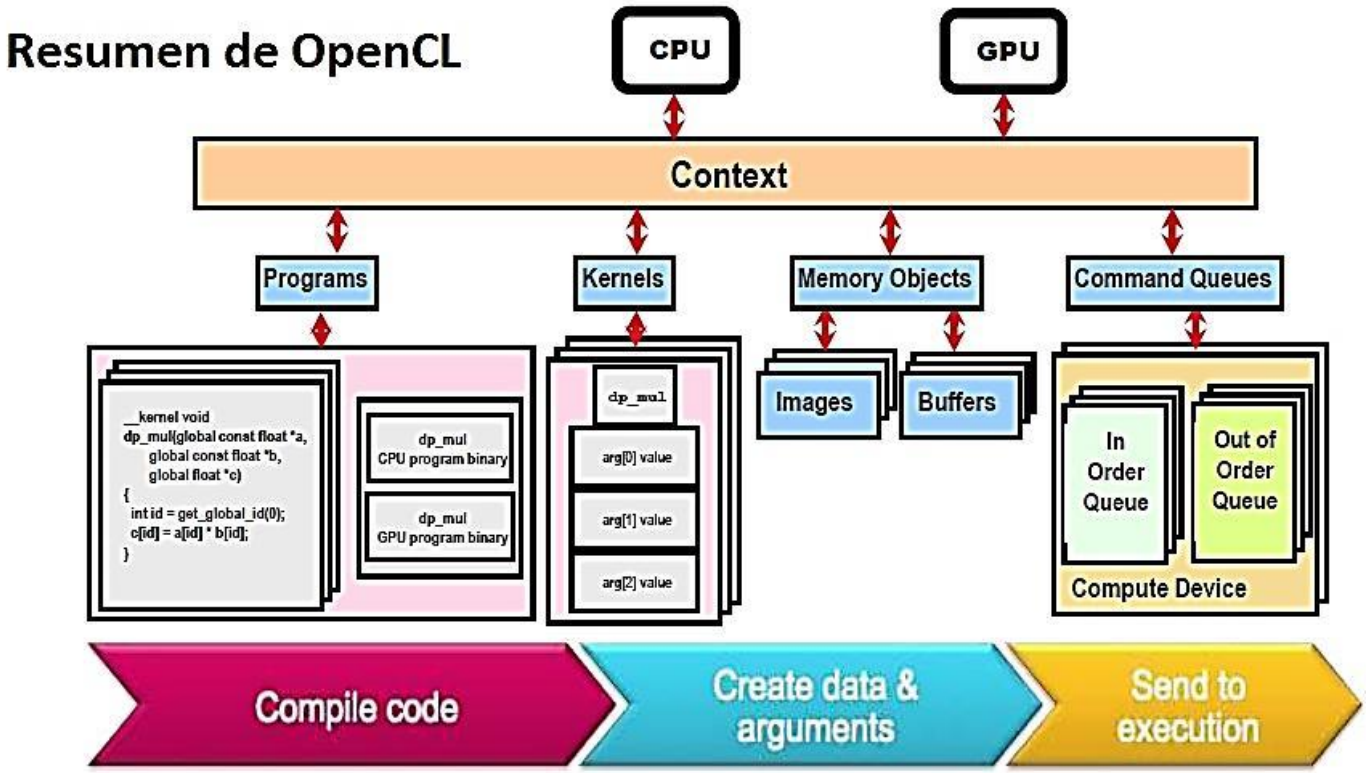


Figura 19: Resumen del funcionamiento OpenCL

En comandos viene a ser:

1. Enumerar las plataformas

```
clGetPlatformIDs (cl_uint num_entries, cl_platform_id *platforms,
cl_uint *num_platforms)
```

Se utiliza para obtener el número de plataformas disponibles en la implementación. Esta función se ha de llamar dos veces, debido a que la primera llamada se usa para ver la cantidad de plataformas disponibles en la implementación. En la segunda llamada ya se obtiene los objetos de la plataforma.

2. Enumerar los dispositivos

```
clGetDeviceIDs (cl_platform_id platform, cl_device_type device_type,
cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices)
```

Una vez obtenido el número de plataformas, se consulta los dispositivos que hay en la máquina. *Device_type* puede ser de tres tipos: *CL_DEVICE_TYPE_CPU, CL_DEVICE_TYPE_GPU, CL_DEVICE_TYPE_ALL*, con ello especificamos el tipo de dispositivo que se quiere utilizar (CPU, GPU o todos).

3. Crear el contexto

```
clCreateContext (const cl_context_properties *properties, cl_uint
num_devices, const cl_device_id *devices, void (*pfn_notify)(const char
*errinfo, const void *private_info, size_t cb, void *user data), void
*user_data, cl_int *errcode_ret)
```

El contexto es un espacio de para manejar los objetos y recursos de OpenCL, donde están incluidos los kernels (las funciones que se van a ejecutar), dispositivos, objetos de memoria (los datos de los hilos) y la cola de comandos (los datos que se van a transferir, ejecución de kernels y la sincronización), etc.

4. Crear la cola de comandos

```
clCreateCommandQueue (cl_context context, cl_device_id device,
cl_command_queue_properties, cl_int *errcode_ret)
```

La cola de comandos es un objeto donde los comandos se van apilando o encolando para ser ejecutado por el dispositivo o dicho de otra manera, el procesador pide acciones al dispositivo para ser ejecutado. En el momento de la llamada de la función se realizan transferencias de datos a la memoria además de la ejecución de la tarea.

5. Crear el programa y el kernel

```
clCreateProgramWithSource (cl_context context, cl_uint count, const char
**strings, const size_t *lenghts, cl_int *errcode_ret)
```

Esta función crea un objeto programa, pudiendo pasarle lo que tiene que ejecutar mediante un *string* directamente o teniendo que leerlo desde un archivo.

```
clBuildProgram (cl_program program, cl_uint num_devices, const
cl_device_id *device_list, const char *options, void
(*pfn_notify)(cl_program, void *user_data), void *user_data)
```

Dicha función realiza la compilación y enlaza un ejecutable a cada dispositivo.

```
clCreateKernel (cl_program program, const char *kernel_name, cl_int
*errcode_ret)
```

En este apartado, la función crea el kernel a partir del objeto programa creado anteriormente. En el atributo *kernel_name* se tiene que escribir el nombre de la función del kernel a ejecutar del programa.

6. Reservar e inicializar la memoria

```
clCreateBuffer (cl_context context, cl_mem_flags flags, size_t size,
void *host_ptr, cl_int *errcode_ret)
```

Este apartado describe la forma de manejar los datos. Se pueden clasificar en buffers o en imágenes, pero en este Proyecto Fin de Grado se utilizará solamente los buffers. Los buffers son trozos de memoria contiguos y se pueden leer y escribir en ellos, que se realizan de la siguiente manera, pudiendo ser *flags*:

CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY, CL_MEM_USE_HOST_PTR, CL_MEM_ALLOC_HOST_PTR, CL_MEM_COPY_HOST_PTR.

7. Poner los argumentos y poner el kernel en la cola

```
clSetKernelArg (cl_kernel kernel, cl_uint arg_index, size_t arg_size)
```

En esta función se añaden los argumentos que tiene el programa.

```
clEnqueueNDRangeKernel (cl_command_queue command_queue, cl_kernel
kernel, cl_uint work_dim, const size_t *global_work_offset, const size_t
*global_work_size, const size_t *local_work_size, cl_uint
num_events_in_wait_list, const cl_event *event_wait_list, cl_event
*event)
```

Una vez finalizado el pasar los argumentos al programa, se agrega a la cola de ejecución con esta última función.

Un dato importante en esta función; se trata de las variables *global_work_size* y *local_work_size*. En la variable *global_work_size* se ha de especificar el tamaño total de la matriz en forma de array de dos posiciones, debido al número de filas y al número de columnas. Lo mismo ocurre con la variable *local_work_size*, allí se especifica el número de hilos que se utilizará para calcular los bloques. Esta variable siempre ha de ser menor o igual que la global además de ser múltiplo de este.

8. Leer los resultados

```
clEnqueueReadBuffer (cl_command_queue command_queue cl_mem buffer,
cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_uint
num_events_in_wait_list, const cl_event *event_wait_list, cl_event
*event)
```

Una vez que el programa haya finalizado sus operaciones, hay que copiar los datos del dispositivo a la CPU que se realiza con la anterior función.

9. “Limpiar”

```
clReleaseMMM
```

Una vez terminado todo, hay que liberar todo y se realiza con la función anterior, donde *MMM* se debe reemplazar por *Kernel, CommandQueue, Context* y *MemObject*, por lo tanto un mínimo de 4 funciones hay que ejecutar para liberar todo. El número

de funciones para liberar puede aumentar dependiendo del número de reservas en memoria se haya realizado.

Para poder compilar un programa en OpenCL, he utilizado los comandos que se muestran en el fichero *make* en el Anexo 5: Makefile.

9.2 Apéndice 2: Utilización de OpenMP

Para poder compilar en el lenguaje de programación C, se ha de ejecutar el siguiente comando:

gcc -fopenmp nombreFichero.*c -o* nombreEjecutable

En cambio, para poder compilarlo en el lenguaje de programación Java (23), antes se ha de exportar la biblioteca correspondiente de JOMP al *CLASSPATH* con el siguiente comando:

export CLASSPATH=$CLASSPATH:~/workspace/Java/jar/jomp1.0b.jar

Una vez guardada la ruta en donde se encuentra la biblioteca JOMP en la *CLASSPATH*, ejecutamos el siguiente comando para poder compilar con la biblioteca:

java jomp.compiler.Jomp openMP

El archivo *openMP* tiene que tener explícitamente la extensión *.jomp*, si no, no compila.

El compilador *JOMP* crea un archivo Java correspondiente con la "traducción" del código que se va a paralelizar en formato "entendible" para el PC. En el apartado Conversión de la función de multiplicación con la biblioteca JOMP muestra el código de Anexo: Código de multiplicación de matrices en OpenMP en lenguaje de programación Java tras compilarlo con la biblioteca *JOMP*.

Ahora con el archivo *.java* creado, se compila con el comando:

javac openMP.java

Y finalmente se ejecuta el programa con el comando:

java openMP

Al no pasarle ningún parámetro al programa, este mismo mostrará un ejemplo de qué parámetros se puede utilizar en el programa.

9.3 Apéndice 3: Instalación y utilización de OpenCL en Ubuntu 12.04

En esta sección se explicará el cómo poder instalar OpenCL para poder utilizarlo.

9.3.1 GPU

Para poder utilizar la GPU en OpenCL, basta con instalar el controlador correspondiente de la tarjeta gráfica del sistema. En este caso bastó con utilizar el siguiente comando en la terminal: *apt-get install nvidia-current*. Para saber si el sistema tiene *OpenCL* hay que ejecutar el siguiente comando en la terminal: *locate *opencl**. Si encuentra alguna biblioteca en la carpeta */usr/lib*, */usr/lib32* o */usr/lib64* con el nombre *libOpenCL.so* o *libnvidia-opencl.so*, es que se puede ejecutar *OpenCL* con la tarjeta gráfica.

9.3.2 CPU

Para poder utilizar la CPU en OpenCL en los procesadores Intel, se ha instalado el SDK oficial de *Intel*, versión XE 2013. (24)

Los instaladores que ofrece el paquete de *Intel* son los siguientes:

- Opencl-1.2-base-[versión]: contiene la biblioteca *libOpenCL.so*.
- Opencl-1.2-devel-[versión]: contiene las cabeceras de *OpenCL*, que también se puede instalar con el comando en la terminal *apt-get install opencl-headers*.
- Opencl-1.2-intel-cpu-[versión]: contiene la biblioteca *OpenCL* para la utilización de la CPU del sistema.
- Opencl-1.2-intel-devel-[versión]: contiene herramientas para poder desarrollar en *OpenCL*.
- Opencl-1.2-intel-mic-[versión]: contiene bibliotecas adicionales para soportar los procesadores *Xeon Phi* además de los *Core*.

Para este Proyecto Fin de Grado se ha utilizado la “base”, “devel” y “cpu”. Como no son ejecutables *.deb* para Ubuntu, el primer paso fue crearlo en ese formato. Para ello se ha instalado *alien* con el siguiente comando en la terminal *sudo apt-get install rpm alien*.

Una vez finalizada la instalación, convertimos los archivos en *.deb* de la siguiente manera en la terminal *fakeroot alien --to-deb <paquete>* ignorando los mensajes que aparecen. El siguiente paso es instalar los paquetes *.deb* con el comando *sudo dpkg –i <paquete.deb>*.

9.4 Apéndice 4: Likwid

Likwid (Lightweight performance tools) se trata de un programa para Linux que calcula muchos datos importantes durante las ejecuciones de los programas como lo son el CPI, tasa de error, etc.

El programa ofrece muchas herramientas, pero se mostrarán solamente los utilizados en este proyecto:

- Likwid-topology: muestra la topología de hilo y caché en máquinas multicore/multisocket.
- Likwid-perfctr: herramienta que mide el rendimiento del hardware durante la ejecución de un programa.

A continuación se mostrarán los resultados obtenidos tras las ejecuciones de los programas en modo secuencial, *OpenMP* y *OpenCL*.

Esta tabla muestra los resultados en modo secuencial. La primera columna muestra los tamaños de las matrices utilizados (200, 400, 600, 800, 1000), la segunda columna muestra los resultados obtenidos en los dos núcleos de la CPU en la máquina 1 y en la tercera muestra los mismos resultados sólo que utilizando los programas en C optimizados.

SECUENCIAL	C		C optimizado	
	Core 0	Core 1	Core 0	Core 1
200				
CPI	0.88986	1.5669	0.438934	1.21334
Data cache misses	544767	64765	526513	4438
Data cache miss rate	0.00356752	0.00608171	0.00817258	0.00662028
400				
CPI	1.40561	0.894663	1.415	0.432751
Data cache misses	90528	6.52792e+06	83939	4.97895e+06
Data cache miss rate	0.007252	0.00550626	0.00623823	0.0103604
600				
CPI	1.15773	1.04863	1.10242	1.02576
Data cache misses	2.45022e+08	607408	2.48004e+08	256797
Data cache miss rate	0.0618175	0.00497911	0.156405	0.00427554
800				
CPI	1.26293	1.20494	1.41985	1.76603
Data cache misses	5.5626e+08	1.09372e+06	5.59446e+08	669467
Data cache miss rate	0.0594885	0.00586879	0.150658	0.0081832
1000				
CPI	1.0895	1.57011	1.9449	1.24669
Data cache misses	6.48728e+06	1.18955e+09	1.19759e+09	1.46268e+06
Data cache miss rate	0.00801723	0.0653228	0.166262	0.0053182

Tabla 13: Likwid C secuencial Máquina 1

Se puede observar que la versión optimizada obtiene mejores resultados que ejecutándolo sin optimización, por lo tanto, tras haber hecho unas pruebas más con otros métodos, se procederá únicamente a mostrar los resultados con los programas optimizados.

La siguiente tabla muestra el resultado tras la ejecución en modo *OpenMP*. La estructura de la tabla es la misma que la anterior.

OPENMP	C	
	Core 0	Core 1
200		
CPI	0.578392	0.566448
Data cache misses	277699	322426
Data cache miss rate	0.00696555	0.00867478
400		
CPI	0.444241	0.504149
Data cache misses	2.57665e+06	2.58101e+06
Data cache miss rate	0.0111653	0.00964273
600		
CPI	1.30883	1.25134
Data cache misses	1.24769e+08	1.2424e+08
Data cache miss rate	0.161045	0.15003
800		
CPI	1.45812	1.48747
Data cache misses	2.79845e+08	2.8042e+08
Data cache miss rate	0.145771	0.15016
1000		
CPI	2.00045	1.96522
Data cache misses	6.01036e+08	5.99848e+08
Data cache miss rate	0.165739	0.161819

Tabla 14: Likwid C OpenMP Máquina 1

En esta siguiente tabla se muestra lo mismo que anteriormente, sólo que ahora utilizando Java:

SECUENCIAL / OPENMP	Java SEC		Java OpenMP	
	Core 0	Core 1	Core 0	Core 1
200				
CPI	0.755238	0.970018	1.02047	1.36598
Data cache misses	1.54276e+06	761524	812579	99231
Data cache miss rate	0.00443625	0.0063681	0.00679737	0.00704846
400				
CPI	1.09941	0.637737	0.915636	1.04378
Data cache misses	887731	7.61863e+06	218122	846719
Data cache miss rate	0.00853505	0.00677771	0.00423724	0.00692402
600				
CPI	1.12823	0.900876	1.64248	1.0233
Data cache misses	1.27253e+06	2.47198e+08	153795	810910
Data cache miss rate	0.00771837	0.0775469	0.00989012	0.00678753
800				
CPI	1.02461	1.13556	1.01071	1.40855
Data cache misses	5.58171e+08	1.95244e+06	684538	247972
Data cache miss rate	0.0777072	0.00711014	0.00636625	0.0091376
1000				
CPI	1.18198	1.29354	1.62921	1.0338
Data cache misses	2.98403e+06	1.19435e+09	122042	815194
Data cache miss rate	0.00677821	0.0869126	0.00825608	0.00682109

Tabla 15: Likwid Java Secuencial-OpenMP Máquina 1

Esta tabla muestra la ejecución secuencial en C de la máquina 2:

SECUENCIAL	C optimizado			
	Core 0	Core 1	Core 2	Core 3
200				
CPI	12.9345	7.27009	3.25256	0.457887
Data cache misses	1078	758	3523	516959
Data cache miss rate	0.0496225	0.0334245	0.0287093	0.00837792
400				
CPI	7.45184	8.05499	18.7916	0.474757
Data cache misses	434	3572	253	4.6678e+06
Data cache miss rate	0.0201954	0.0452828	0.0860252	0.00993616
600				
CPI	1.7994	6.10471	1.28625	0.755567
Data cache misses	160028	27603	49580	2.2573e+08
Data cache miss rate	0.0136252	0.0689491	0.0101586	0.144646
800				
CPI	2.45086	2.71811	2.01979	2.45077
Data cache misses	1.60942e+08	746036	1.08724e+06	3.96535e+08
Data cache miss rate	0.144743	0.0289194	0.0152533	0.150395
1000				
CPI	3.54707	1.70199	3.4176	2.7175
Data cache misses	5.53512e+07	1.1254le+06	1.08173e+09	5.999e+07
Data cache miss rate	0.153927	0.0124317	0.169381	0.121356

Tabla 16: Likwid C Secuencial Máquina 2

En versión OpenMP en C de la máquina 2:

OPENMP	C optimizado			
	Core 0	Core 1	Core 2	Core 3
200				
CPI	1.31649	1.33017	1.30709	0.602937
Data cache misses	217084	275835	115622	834368
Data cache miss rate	0.0134188	0.0170505	0.00713515	0.0109302
400				
CPI	1.46131	1.2396	1.43806	0.623528
Data cache misses	1.51199e+07	7.38415e+06	1.52555e+07	1.22632e+07
Data cache miss rate	0.128132	0.0624666	0.125163	0.0209394
600				
CPI	3.13041	3.80324	3.27607	1.37094
Data cache misses	6.15626e+07	5.78823e+07	6.02845e+07	2.85662e+08
Data cache miss rate	0.140889	0.147032	0.152129	0.146729
800				
CPI	4.84667	5.12832	4.96768	3.02
Data cache misses	1.33543e+08	1.42177e+08	1.361e+08	6.97697e+08
Data cache miss rate	0.128972	0.145496	0.13561	0.151719
1000				
CPI	4.08775	6.58052	6.49453	5.92497
Data cache misses	1.43097e+09	2.82859e+08	2.93538e+08	3.3745e+08
Data cache miss rate	0.167965	0.153565	0.157516	0.150838

Tabla 17: Likwid C OpenMP Máquina 2

Versión secuencial en Java en la máquina 2:

SECUENCIAL	Java			
	Core 0	Core 1	Core 2	Core 3
200				
CPI	1.88618	1.17719	0.828369	1.55961
Data cache misses	504410	435400	4.43095e+06	239179
Data cache miss rate	0.0205381	0.0171287	0.0157467	0.0191439
400				
CPI	1.39672	0.526648	1.48399	1.96626
Data cache misses	605244	9.27389e+06	640940	118022
Data cache miss rate	0.0181765	0.00991368	0.0171969	0.0158726
600				
CPI	2.03762	1.22587	0.541454	1.64466
Data cache misses	396369	1.94837e+06	2.30169e+08	1.08485e+06
Data cache miss rate	0.0208503	0.0135069	0.0907525	0.0214975
800				
CPI	1.96921	1.7275	1.63711	1.88996
Data cache misses	3.34376e+06	2.66374e+06	5.60494e+08	2.79803e+06
Data cache miss rate	0.0292882	0.0176026	0.0945366	0.0176429
1000				
CPI	1.80463	2.25459	1.34201	1.84256
Data cache misses	2.66272e+06	1.19484e+09	5.08892e+06	2.03033e+06
Data cache miss rate	0.0149553	0.106808	0.0187422	0.018653

Tabla 18: Likwid Java Secuencial Máquina 2

Versión OpenMP en Java en la máquina 2:

OPENMP	Java			
	Core 0	Core 1	Core 2	Core 3
200				
CPI	1.87449	4.54274	1.09747	1.89975
Data cache misses	18214	11654	1.46831e+06	53380
Data cache miss rate	0.024691	0.023689	0.0127393	0.0265135
400				
CPI	1.80753	1.11478	1.61922	1.83662
Data cache misses	63588	1.45845e+06	66228	44033
Data cache miss rate	0.015556	0.0128224	0.0187956	0.011285
600				
CPI	4.81016	1.09565	1.63279	1.62438
Data cache misses	6890	1.45512e+06	30279	43286
Data cache miss rate	0.0448597	0.0127985	0.0119059	0.0285072
800				
CPI	3.65457	1.30481	1.08409	2.15297
Data cache misses	25055	423320	1.09569e+06	72912
Data cache miss rate	0.0316833	0.0203379	0.0110379	0.0227222
1000				
CPI	3.03055	1.09313	1.58298	1.62247
Data cache misses	8992	1.44609e+06	27098	42641
Data cache miss rate	0.0210112	0.0127296	0.0129282	0.0280582

Tabla 19: Likwid Java OpenMP Máquina 2

Debido a que la máquina 3 no es propia, si no que externo, el programa *likwid* no estaba instalado y por eso no se ha podido realizar las pruebas.

10 Anexos

En esta sección se detallarán los anexos para este Proyecto Fin de Grado.

10.1 Anexo 1: Características de las máquinas de prueba

10.1.1 Máquina 1

Se trata del dispositivo donde se ha realizado todo el desarrollo además de las pruebas.

- Procesador:
  - producto: Intel(R) Core(TM)2 Duo CPU E7600 @ 3.06GHz
  - fabricante: Intel Corp.
  - ranura: Socket 775
  - tamaño: 3059MHz
  - capacidad: 3800MHz
  - anchura: 64 bits
  - reloj: 266MHz
- Memoria:
  - Principal (3 veces, por lo tanto 3GB en total):
    - descripción: DIMM SDRAM Síncrono 667 MHz (1,5 ns)
    - tamaño: 1GiB
    - anchura: 64 bits
    - reloj: 667MHz (1.5ns)
  - Caché (por procesador):

    L1 caché
    - tamaño: 64KiB

    L2 caché
    - tamaño: 3MiB
- Gráfica:
  - producto: G80 [GeForce 8800 GTS]
  - fabricante: NVIDIA Corporation
  - tamaño:

10.1.2 Máquina 2

Se trata del segundo dispositivo de pruebas del fabricante ASUSTek Computer Inc. U36UJC con los siguientes datos:

- Procesador:
  - producto: Intel(R) Core(TM) i5 CPU M 480 @ 2.67GHz
  - fabricante: Intel Corp.
  - información del bus: cpu@0
  - ranura: Socket 989
  - tamaño: 1199MHz
  - capacidad: 4GHz
  - anchura: 64 bits
  - reloj: 133MHz
  - configuración: cores=2 enabledcores=1 threads=2
- Memoria:
  - Principal (2 veces, por lo tanto 4GB en total):
    - descripción: SODIMM DDR3 Síncrono 1067 MHz (0,9 ns)
    - tamaño: 2GiB
    - anchura: 64 bits
    - reloj: 1067MHz (0.9ns)
  - Caché (por procesador):

    L1 caché
    - tamaño: 32KiB

    L2 caché
    - tamaño: 256KiB

    L3 caché
    - tamaño: 3MiB
- Gráfica:
  - producto: GT218 [GeForce 310M]
  - fabricante: NVIDIA Corporation
  - tamaño:

10.1.3 Máquina 3

Se trata del tercer dispositivo de pruebas con los siguientes datos:

- Procesador:
  - producto: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
  - fabricante: Intel Corp.
  - ranura: LGA1155
  - tamaño: 1600MHz
  - capacidad: 3800MHz
  - anchura: 64 bits
  - reloj: 100MHz
- Memoria:
  - Principal (2 veces, por lo tanto 4GB en total):
    - descripción: SODIMM DDR3 Síncrono 1067 MHz (0,9 ns)
    - producto: SSZ3128M8-EDJEF
    - fabricante: 06C1
    - id físico: 2
    - serie: 00000700
    - ranura: DIMM2
    - tamaño: 2GiB
    - anchura: 64 bits
    - reloj: 1067MHz (0.9ns)
    - configuración: cores=4 enabledcores=1 threads=2
  - Caché (por procesador):

    L1 caché
    - tamaño: 32KiB

    L2 caché
    - tamaño: 256KiB

    L3 caché
    - tamaño: 3MiB
- Gráfica:
  - producto: G119 [GeForce GT 520]
  - fabricante: NVIDIA Corporation
  - tamaño:

10.2 Anexo 2: Código kernel de multiplicación de matrices en OpenCL

```
__kernel void sampleKernel (__global const int *a, __global const int
*b, __global int *c, const int cM1, const int cM2){

        int k,aa,bb,cc;
        int gid = get_global_id(0);
        int gid2 = get_global_id(1);
        cc=0;
        for(k=0;k<cM1;k++){
              aa=a[gid*cM1+k];
              bb=b[k*cM2+gid2];
              cc+=aa*bb;}
        c[gid*cM2+gid2]=cc;
  };
```

10.3 Anexo 3: Código de multiplicación de matrices en OpenMP

10.3.1 Lenguaje de programación C

```
#pragma omp parallel for shared(m1,m2,result) private(j,k,suma)
int j, i, suma, k;
for (i = 0; i < fM1; i++) {
      for (j = 0; j < cM2; j++) {
            suma = 0;
            for (k = 0; k < cM1; k++) {
                  suma += m1[i*cM1 + k] * m2[k * cM2 + j];
            }
            result[i * cM2 + j] = suma;
      }
}
```

10.3.1 Lenguaje de programación Java

```
//omp parallel for shared(m1,m2,result) private(j,k,suma)
int j, i, suma, k;
for (i = 0; i < fM1; i++) {
      for (j = 0; j < cM2; j++) {
            suma = 0;
            for (k = 0; k < cM1; k++) {
                  suma += m1[i*cM1 + k] * m2[k * cM2 + j];
            }
            result[i * cM2 + j] = suma;
      }
}
```

10.3.2 Conversión de la función de multiplicación con la biblioteca JOMP

```
// OMP PARALLEL BLOCK BEGINS
{
  __omp_Class0 __omp_Object0 = new __omp_Class0();
  // shared variables
    omp Object0.matrizA = matrizA;
  __omp_Object0.matrizB = matrizB;
  __omp_Object0.result = result;
  // firstprivate variables
  try {
    jomp.runtime.OMP.doParallel(__omp_Object0);
  } catch(Throwable __omp_exception) {
    System.err.println("OMP Warning: Illegal thread exception
ignored!");
    System.err.println(__omp_exception);
  }
  // reduction variables
  // shared variables
  matrizA =   omp Object0.matrizA;
  matrizB = __omp_Object0.matrizB;
  result = __omp_Object0.result;
}
// OMP PARALLEL BLOCK ENDS

            return result;
      }

// OMP PARALLEL REGION INNER CLASS DEFINITION BEGINS
private static class __omp_Class0 extends jomp.runtime.BusyTask {
  // shared variables
  int [ ] matrizA;
  int [ ] matrizB;
  int [ ] result;
  // firstprivate variables
  // variables to hold results of reduction

  public void go(int __omp_me) throws Throwable {
  // firstprivate variables + init
  // private variables
  int j;
  int k;
  int suma;
  // reduction variables, init to default
    // OMP USER CODE BEGINS

    { // OMP FOR BLOCK BEGINS
    // copy of firstprivate variables, initialized
    // copy of lastprivate variables
    // variables to hold result of reduction
    boolean amLast=false;
    {
        // firstprivate variables + init
        // [last]private variables
        // reduction variables + init to default
        // -------------------------------------
        jomp.runtime.LoopData __omp_WholeData2 = new
jomp.runtime.LoopData();
        jomp.runtime.LoopData __omp_ChunkData1 = new
jomp.runtime.LoopData();
```

```
        __omp_WholeData2.start = (long)( 0);
        __omp_WholeData2.stop = (long)( fM1);
          omp WholeData2.step = (long)(1);
        jomp.runtime.OMP.setChunkStatic(__omp_WholeData2);
        while(!__omp_ChunkData1.isLast &&
jomp.runtime.OMP.getLoopStatic(__omp_me, __omp_WholeData2,
__omp_ChunkData1)) {
             for(;;) {
                if(__omp_WholeData2.step > 0) {
                  if(__omp_ChunkData1.stop > __omp_WholeData2.stop)
__omp_ChunkData1.stop = __omp_WholeData2.stop;
                  if(__omp_ChunkData1.start >= __omp_WholeData2.stop)
break;
                } else {
                  if(__omp_ChunkData1.stop < __omp_WholeData2.stop)
  omp ChunkData1.stop =   omp WholeData2.stop;
                  if(__omp_ChunkData1.start > __omp_WholeData2.stop)
break;
                }
                for(int i = (int)__omp_ChunkData1.start; i <
__omp_ChunkData1.stop; i += __omp_ChunkData1.step) {
                                  // OMP USER CODE BEGINS
           {
                 for (j = 0; j < cM2; j++) {
                       suma = 0;
                       for (k = 0; k < cM1; k++) {
                             suma += matrizA[i*cM1 + k] * matrizB[k *
cM2 + j];
                       }
                       result[i * cM2 + j] = suma;
                 }
           }
                                  // OMP USER CODE ENDS
                                  if (i == (__omp_WholeData2.stop-1))
amLast = true;
                               } // of for
                               if(__omp_ChunkData1.startStep == 0)
                                 break;
                               __omp_ChunkData1.start +=
__omp_ChunkData1.startStep;
                               __omp_ChunkData1.stop +=
__omp_ChunkData1.startStep;
                             } // of for(;;)
                             } // of while
                             // call reducer
                             jomp.runtime.OMP.doBarrier(__omp_me);
                             // copy lastprivate variables out
                             if (amLast) {
                             }
                           }
                           // set global from lastprivate variables
                           if (amLast) {
                           }
                           // set global from reduction variables
                           if (jomp.runtime.OMP.getThreadNum(__omp_me)
== 0) {
                           }
                           } // OMP FOR BLOCK ENDS

    // OMP USER CODE ENDS
  // call reducer
```

```
  // output to _rd_ copy
  if (jomp.runtime.OMP.getThreadNum(__omp_me) == 0) {
  }
  }
}
// OMP PARALLEL REGION INNER CLASS DEFINITION ENDS
```

10.4 Anexo 4: Código secuencial de multiplicación de matrices en C y Java

```
int j, i, suma, k;
for (i = 0; i < fM1; i++) {
	for (j = 0; j < cM2; j++) {
		suma = 0;
		for (k = 0; k < cM1; k++) {
			suma += m1[i*cM1 + k] * m2[k * cM2 + j];
		}
		result[i * cM2 + j] = suma;
	}
}
```

10.5 Anexo 5: Makefile

```
CC=gcc
CFLAGS=-fopenmp

all: compilar

compilar: C Java

C: secuencialC paraleloC openCLC

Java: secuencialJ paraleloJ openCLJ

secuencialC: C/secuencial.c
	$(CC) C/secuencial.c -o C/secuencial

paraleloC: C/openMP.c
	$(CC) $(CFLAGS) C/openMP.c -o C/openMP

openCLC: C/openCL.c
	$(CC) -c -I C/ C/openCL.c -o C/openCL.o
	$(CC) C/openCL.o -o C/openCL -L /usr/lib -l OpenCL

getInfoC: C/getInfo.c
	$(CC) -c -I C/ C/getInfo.c -o C/getInfo.o
	$(CC) C/getInfo.o -o C/getInfo -L /usr/lib -l OpenCL

secuencialJ: Java/secuencial.java
	javac Java/secuencial.java

paraleloJ: Java/openMP.jomp
	java jomp.compiler.Jomp Java/openMP
	javac Java/openMP.java

openCLJ: Java/openCL.java
	javac Java/openCL.java -cp Java/jar/JOCL-0.1.9.jar
	cp Java/jar/JOCL-0.1.9.jar .
	mv JOCL-0.1.9.jar openCL.jar
	mv Java/openCL.class .
	jar uf openCL.jar openCL.class
	mv openCL.class Java/
	mv openCL.jar Java/

getInfoJ: Java/getInfo.java
	javac Java/getInfo.java -cp Java/jar/JOCL_getInfo.jar
	cp Java/jar/JOCL_getInfo.jar .
	mv JOCL_getInfo.jar getInfo.jar
	mv Java/getInfo.class .
	jar uf getInfo.jar getInfo.class
	mv getInfo.class Java/
	mv getInfo.jar Java/

clean:
	rm -rf C/secuencial C/openMP C/openCL C/*.o C/getInfo
Java/secuencial Java/openMP.java Java/openMP.class Java/*.class
Java/*.jar
```

10.6 Anexo 6: Estructura del proyecto

```
workspace/
├── 1_scpToServer
├── 2_scpToServer
├── C
│   ├── CL
│   │   ├── cl_egl.h
│   │   ├── cl_ext.h
│   │   ├── cl.h
│   │   ├── cl_platform.h
│   │   └── opencl.h
│   ├── double
│   │   ├── ejecutaC
│   │   ├── kernel.cl
│   │   ├── makefile
│   │   ├── openCL.c
│   │   ├── openMP.c
│   │   └── secuencial.c
│   ├── float
│   │   ├── ejecutaC
│   │   ├── kernel.cl
│   │   ├── makefile
│   │   ├── openCL.c
│   │   ├── openMP.c
│   │   └── secuencial.c
│   ├── getInfo.c
│   ├── kernel.cl
│   ├── openCL.c
│   ├── openMP.c
│   └── secuencial.c
├── ejecutaC
├── ejecutaJava
├── exportCLASSPATH
├── firstTimeUni
├── info
│   ├── infoPC.txt
│   ├── infoPort.txt
│   ├── infoUni.txt
```

```
│   ├── openCLpcC.txt
│   ├── openCLpcJava.txt
│   ├── openCLportC.txt
│   ├── openCLportJava.txt
│   ├── openCLuniC.txt
│   └── openCLuniJava.txt
├── Java
│   ├── double
│   │   ├── ejecutaJava
│   │   ├── makefile
│   │   ├── openCL.java
│   │   ├── openMP.jomp
│   │   └── secuencial.java
│   ├── float
│   │   ├── ejecutaJava
│   │   ├── makefile
│   │   ├── openCL.java
│   │   ├── openMP.jomp
│   │   └── secuencial.java
│   ├── getInfo.java
│   ├── jar
│   │   ├── JOCL-0.1.9.jar
│   │   ├── JOCL_getInfo.jar
│   │   └── jomp1.0b.jar
│   ├── openCL.java
│   ├── openMP.jomp
│   └── secuencial.java
└── makefile

11 directories, 55 files
```

10.7 Anexo 7: Fichero bash para la ejecución automática de los programas

10.7.1 C

```
#secuencial
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           ./C/secuencial $i
     done
done
echo ""

#paralelo
export OMP_NUM_THREADS=$2
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           ./C/openMP $i
     done
done
echo ""

#openCL
cd C
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           ./openCL $i $3
     done
done
echo ""
```

10.7.2 Java

```
#secuencial
cd Java
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           java secuencial $i
     done
done
cd ..
echo ""

#paralelo
cd Java
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           java openMP $i $2
     done
done
cd ..
echo ""

#openCL
cd Java
for i in {10..1000..10}
do
     for ((j = 1; j <= $1; j++))
     do
           java -jar openCL.jar $i $3
     done
done
cd ..
echo ""
```

10.8 Anexo 8: Información OpenCL de la Máquina 1

10.8.1 C

```
platform[0x9e85020]: Found 1 device(s).
        device[0x9e85088]: NAME: GeForce 8800 GTS
        device[0x9e85088]: VENDOR: NVIDIA Corporation
        device[0x9e85088]: PROFILE: FULL_PROFILE
        device[0x9e85088]: VERSION: OpenCL 1.0 CUDA
        device[0x9e85088]: EXTENSIONS: cl_khr_byte_addressable_store cl_khr_icd
cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll
        device[0x9e85088]: DRIVER_VERSION: 319.37

        device[0x9e85088]: Type: GPU
        device[0x9e85088]: EXECUTION_CAPABILITIES: Kernel
        device[0x9e85088]: GLOBAL_MEM_CACHE_TYPE: None (0)
        device[0x9e85088]: CL_DEVICE_LOCAL_MEM_TYPE: Local (1)
        device[0x9e85088]: SINGLE_FP_CONFIG: 0x3e
        device[0x9e85088]: QUEUE_PROPERTIES: 0x3

        device[0x9e85088]: VENDOR_ID: 4318
        device[0x9e85088]: MAX_COMPUTE_UNITS: 12
        device[0x9e85088]: MAX_WORK_ITEM_DIMENSIONS: 3
        device[0x9e85088]: MAX_WORK_GROUP_SIZE: 512
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_CHAR: 1
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_SHORT: 1
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_INT: 1
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_LONG: 1
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_FLOAT: 1
        device[0x9e85088]: PREFERRED_VECTOR_WIDTH_DOUBLE: 0
        device[0x9e85088]: MAX_CLOCK_FREQUENCY: 1188
        device[0x9e85088]: ADDRESS_BITS: 32
        device[0x9e85088]: MAX_MEM_ALLOC_SIZE: 134217728
        device[0x9e85088]: IMAGE_SUPPORT: 1
        device[0x9e85088]: MAX_READ_IMAGE_ARGS: 128
        device[0x9e85088]: MAX_WRITE_IMAGE_ARGS: 8
        device[0x9e85088]: IMAGE2D_MAX_WIDTH: 4096
        device[0x9e85088]: IMAGE2D_MAX_HEIGHT: 16383
        device[0x9e85088]: IMAGE3D_MAX_WIDTH: 2048
        device[0x9e85088]: IMAGE3D_MAX_HEIGHT: 2048
        device[0x9e85088]: IMAGE3D_MAX_DEPTH: 2048
        device[0x9e85088]: MAX_SAMPLERS: 16
        device[0x9e85088]: MAX_PARAMETER_SIZE: 4352
        device[0x9e85088]: MEM_BASE_ADDR_ALIGN: 2048
        device[0x9e85088]: MIN_DATA_TYPE_ALIGN_SIZE: 128
        device[0x9e85088]: GLOBAL_MEM_CACHELINE_SIZE: 0
        device[0x9e85088]: GLOBAL_MEM_CACHE_SIZE: 0
        device[0x9e85088]: GLOBAL_MEM_SIZE: 334823424
        device[0x9e85088]: MAX_CONSTANT_BUFFER_SIZE: 65536
        device[0x9e85088]: MAX_CONSTANT_ARGS: 9
        device[0x9e85088]: LOCAL_MEM_SIZE: 16384
        device[0x9e85088]: ERROR_CORRECTION_SUPPORT: 0
```

```
device[0x9e85088]: PROFILING_TIMER_RESOLUTION: 1000
device[0x9e85088]: ENDIAN_LITTLE: 1
device[0x9e85088]: AVAILABLE: 1
device[0x9e85088]: COMPILER_AVAILABLE: 1
```

10.8.2 Java

```
Number of platforms: 1
PLATFORM NUMBER: 0
Number of devices in platform NVIDIA CUDA: 1

--- Info for device GeForce 8800 GTS: ---
CL_DEVICE_NAME:                         GeForce 8800 GTS
CL_DEVICE_VENDOR:                       NVIDIA Corporation
CL_DRIVER_VERSION:                      319.37
CL_DEVICE_TYPE:                                 CL_DEVICE_TYPE_GPU -->  0
CL_DEVICE_MAX_COMPUTE_UNITS:            12
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:     3
CL_DEVICE_MAX_WORK_ITEM_SIZES:          512 / 512 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE:          512
CL_DEVICE_MAX_CLOCK_FREQUENCY:          1188 MHz
CL_DEVICE_ADDRESS_BITS:                 32
CL_DEVICE_MAX_MEM_ALLOC_SIZE:           128 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:              319 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:     no
CL_DEVICE_LOCAL_MEM_TYPE:               local
CL_DEVICE_LOCAL_MEM_SIZE:           16 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:     64 KByte
CL_DEVICE_QUEUE_PROPERTIES:
        CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES:             CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:            1
CL_DEVICE_MAX_READ_IMAGE_ARGS:          128
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                 8
CL_DEVICE_SINGLE_FP_CONFIG:             CL_FP_INF_NAN CL_FP_ROUND_TO_NEAREST
CL_FP_ROUND_TO_ZERO CL_FP_ROUND_TO_INF CL_FP_FMA
CL_DEVICE_2D_MAX_WIDTH                  4096
CL_DEVICE_2D_MAX_HEIGHT                 16383
CL_DEVICE_3D_MAX_WIDTH                  2048
CL_DEVICE_3D_MAX_HEIGHT                 2048
CL_DEVICE_3D_MAX_DEPTH                  2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>    CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 1,
DOUBLE 0
```

10.9 Anexo 9: Información OpenCL de la Máquina 2

Tras ejecutar el código que muestra toda la información de la GPU/CPU, tanto en C[2] como en Java[3], se ha obtenido la siguiente información:

10.9.1 C

Found 2 platform(s).
platform[0x2570bc0]: profile: FULL_PROFILE
platform[0x2570bc0]: version: OpenCL 1.1 CUDA 4.2.1
platform[0x2570bc0]: name: NVIDIA CUDA
platform[0x2570bc0]: vendor: NVIDIA Corporation
platform[0x2570bc0]: extensions: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll
platform[0x2570bc0]: Found 1 device(s).
device[0x2570c60]: NAME: GeForce 310M
device[0x2570c60]: VENDOR: NVIDIA Corporation
device[0x2570c60]: PROFILE: FULL_PROFILE
device[0x2570c60]: VERSION: OpenCL 1.0 CUDA
device[0x2570c60]: EXTENSIONS: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics
device[0x2570c60]: DRIVER_VERSION: 319.49

device[0x2570c60]: Type: GPU
device[0x2570c60]: EXECUTION_CAPABILITIES: Kernel
device[0x2570c60]: GLOBAL_MEM_CACHE_TYPE: None (0)
device[0x2570c60]: CL_DEVICE_LOCAL_MEM_TYPE: Local (1)
device[0x2570c60]: SINGLE_FP_CONFIG: 0x3e
device[0x2570c60]: QUEUE_PROPERTIES: 0x3

device[0x2570c60]: VENDOR_ID: 4318
device[0x2570c60]: MAX_COMPUTE_UNITS: 2
device[0x2570c60]: MAX_WORK_ITEM_DIMENSIONS: 3
device[0x2570c60]: MAX_WORK_GROUP_SIZE: 512
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_CHAR: 1
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_SHORT: 1
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_INT: 1
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_LONG: 1
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_FLOAT: 1
device[0x2570c60]: PREFERRED_VECTOR_WIDTH_DOUBLE: 0
device[0x2570c60]: MAX_CLOCK_FREQUENCY: 1468
device[0x2570c60]: ADDRESS_BITS: 32
device[0x2570c60]: MAX_MEM_ALLOC_SIZE: 268369920
device[0x2570c60]: IMAGE_SUPPORT: 1
device[0x2570c60]: MAX_READ_IMAGE_ARGS: 128
device[0x2570c60]: MAX_WRITE_IMAGE_ARGS: 8
device[0x2570c60]: IMAGE2D_MAX_WIDTH: 4096
device[0x2570c60]: IMAGE2D_MAX_HEIGHT: 16383

[2] © Jeremy Sugerman, 13 August 2009
[3] © Marco Hutter, 2010

```
        device[0x2570c60]: IMAGE3D_MAX_WIDTH: 2048
        device[0x2570c60]: IMAGE3D_MAX_HEIGHT: 2048
        device[0x2570c60]: IMAGE3D_MAX_DEPTH: 2048
        device[0x2570c60]: MAX_SAMPLERS: 16
        device[0x2570c60]: MAX_PARAMETER_SIZE: 4352
        device[0x2570c60]: MEM_BASE_ADDR_ALIGN: 2048
        device[0x2570c60]: MIN_DATA_TYPE_ALIGN_SIZE: 128
        device[0x2570c60]: GLOBAL_MEM_CACHELINE_SIZE: 0
        device[0x2570c60]: GLOBAL_MEM_CACHE_SIZE: 0
        device[0x2570c60]: GLOBAL_MEM_SIZE: 1073479680
        device[0x2570c60]: MAX_CONSTANT_BUFFER_SIZE: 65536
        device[0x2570c60]: MAX_CONSTANT_ARGS: 9
        device[0x2570c60]: LOCAL_MEM_SIZE: 16384
        device[0x2570c60]: ERROR_CORRECTION_SUPPORT: 0
        device[0x2570c60]: PROFILING_TIMER_RESOLUTION: 1000
        device[0x2570c60]: ENDIAN_LITTLE: 1
        device[0x2570c60]: AVAILABLE: 1
        device[0x2570c60]: COMPILER_AVAILABLE: 1

platform[0x2577440]: profile: FULL_PROFILE
platform[0x2577440]: version: OpenCL 1.2 LINUX
platform[0x2577440]: name: Intel(R) OpenCL
platform[0x2577440]: vendor: Intel(R) Corporation
platform[0x2577440]: extensions: cl_khr_fp64 cl_khr_icd cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store cl_intel_printf
cl_ext_device_fission cl_intel_exec_by_local_thread
platform[0x2577440]: Found 1 device(s).
        device[0x2578568]: NAME: Intel(R) Core(TM) i5 CPU       M 480  @ 2.67GHz
        device[0x2578568]: VENDOR: Intel(R) Corporation
        device[0x2578568]: PROFILE: FULL_PROFILE
        device[0x2578568]: VERSION: OpenCL 1.2 (Build 67279)
        device[0x2578568]: EXTENSIONS: cl_khr_fp64 cl_khr_icd
cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics
cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics
cl_khr_byte_addressable_store cl_intel_printf cl_ext_device_fission
cl_intel_exec_by_local_thread
        device[0x2578568]: DRIVER_VERSION: 1.2

        device[0x2578568]: Type: CPU
        device[0x2578568]: EXECUTION_CAPABILITIES: Kernel Native
        device[0x2578568]: GLOBAL_MEM_CACHE_TYPE: Read-Write (2)
        device[0x2578568]: CL_DEVICE_LOCAL_MEM_TYPE: Global (2)
        device[0x2578568]: SINGLE_FP_CONFIG: 0x7
        device[0x2578568]: QUEUE_PROPERTIES: 0x3

        device[0x2578568]: VENDOR_ID: 32902
        device[0x2578568]: MAX_COMPUTE_UNITS: 4
        device[0x2578568]: MAX_WORK_ITEM_DIMENSIONS: 3
        device[0x2578568]: MAX_WORK_GROUP_SIZE: 1024
        device[0x2578568]: PREFERRED_VECTOR_WIDTH_CHAR: 1
        device[0x2578568]: PREFERRED_VECTOR_WIDTH_SHORT: 1
```

```
device[0x2578568]: PREFERRED_VECTOR_WIDTH_INT: 1
device[0x2578568]: PREFERRED_VECTOR_WIDTH_LONG: 1
device[0x2578568]: PREFERRED_VECTOR_WIDTH_FLOAT: 1
device[0x2578568]: PREFERRED_VECTOR_WIDTH_DOUBLE: 1
device[0x2578568]: MAX_CLOCK_FREQUENCY: 2670
device[0x2578568]: ADDRESS_BITS: 64
device[0x2578568]: MAX_MEM_ALLOC_SIZE: 981630976
device[0x2578568]: IMAGE_SUPPORT: 1
device[0x2578568]: MAX_READ_IMAGE_ARGS: 480
device[0x2578568]: MAX_WRITE_IMAGE_ARGS: 480
device[0x2578568]: IMAGE2D_MAX_WIDTH: 16384
device[0x2578568]: IMAGE2D_MAX_HEIGHT: 16384
device[0x2578568]: IMAGE3D_MAX_WIDTH: 2048
device[0x2578568]: IMAGE3D_MAX_HEIGHT: 2048
device[0x2578568]: IMAGE3D_MAX_DEPTH: 2048
device[0x2578568]: MAX_SAMPLERS: 480
device[0x2578568]: MAX_PARAMETER_SIZE: 3840
device[0x2578568]: MEM_BASE_ADDR_ALIGN: 1024
device[0x2578568]: MIN_DATA_TYPE_ALIGN_SIZE: 128
device[0x2578568]: GLOBAL_MEM_CACHELINE_SIZE: 64
device[0x2578568]: GLOBAL_MEM_CACHE_SIZE: 262144
device[0x2578568]: GLOBAL_MEM_SIZE: 3926523904
device[0x2578568]: MAX_CONSTANT_BUFFER_SIZE: 131072
device[0x2578568]: MAX_CONSTANT_ARGS: 480
device[0x2578568]: LOCAL_MEM_SIZE: 32768
device[0x2578568]: ERROR_CORRECTION_SUPPORT: 0
device[0x2578568]: PROFILING_TIMER_RESOLUTION: 1
device[0x2578568]: ENDIAN_LITTLE: 1
device[0x2578568]: AVAILABLE: 1
device[0x2578568]: COMPILER_AVAILABLE: 1
```

10.9.2 Java

```
Number of platforms: 2
PLATFORM NUMBER: 0
Number of devices in platform NVIDIA CUDA: 1
PLATFORM NUMBER: 1
Number of devices in platform Intel(R) OpenCL: 1

--- Info for device GeForce 310M: ---
CL_DEVICE_NAME:                         GeForce 310M
CL_DEVICE_VENDOR:                       NVIDIA Corporation
CL_DRIVER_VERSION:                      319.49
CL_DEVICE_TYPE:                                 CL_DEVICE_TYPE_GPU -->  0
CL_DEVICE_MAX_COMPUTE_UNITS:            2
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:     3
CL_DEVICE_MAX_WORK_ITEM_SIZES:          512 / 512 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE:          512
CL_DEVICE_MAX_CLOCK_FREQUENCY:          1468 MHz
CL_DEVICE_ADDRESS_BITS:                 32
CL_DEVICE_MAX_MEM_ALLOC_SIZE:           255 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:              1023 MByte
```

```
CL_DEVICE_ERROR_CORRECTION_SUPPORT:    no
CL_DEVICE_LOCAL_MEM_TYPE:              local
CL_DEVICE_LOCAL_MEM_SIZE:          16 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:    64 KByte
CL_DEVICE_QUEUE_PROPERTIES:
        CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES:            CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:           1
CL_DEVICE_MAX_READ_IMAGE_ARGS:         128
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                8
CL_DEVICE_SINGLE_FP_CONFIG:            CL_FP_INF_NAN CL_FP_ROUND_TO_NEAREST
CL_FP_ROUND_TO_ZERO CL_FP_ROUND_TO_INF CL_FP_FMA
CL_DEVICE_2D_MAX_WIDTH                 4096
CL_DEVICE_2D_MAX_HEIGHT                16383
CL_DEVICE_3D_MAX_WIDTH                 2048
CL_DEVICE_3D_MAX_HEIGHT                2048
CL_DEVICE_3D_MAX_DEPTH                 2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>   CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 1,
DOUBLE 0


--- Info for device Intel(R) Core(TM) i5 CPU      M 480  @ 2.67GHz: ---
CL_DEVICE_NAME:                        Intel(R) Core(TM) i5 CPU      M 480  @ 2.67GHz
CL_DEVICE_VENDOR:                      Intel(R) Corporation
CL_DRIVER_VERSION:                     1.2
CL_DEVICE_TYPE:                        CL_DEVICE_TYPE_CPU -->  1
CL_DEVICE_MAX_COMPUTE_UNITS:           4
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:    3
CL_DEVICE_MAX_WORK_ITEM_SIZES:         1024 / 1024 / 1024
CL_DEVICE_MAX_WORK_GROUP_SIZE:         1024
CL_DEVICE_MAX_CLOCK_FREQUENCY:         2000 MHz
CL_DEVICE_ADDRESS_BITS:                64
CL_DEVICE_MAX_MEM_ALLOC_SIZE:          936 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:             3744 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:    no
CL_DEVICE_LOCAL_MEM_TYPE:              global
CL_DEVICE_LOCAL_MEM_SIZE:          32 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:    128 KByte
CL_DEVICE_QUEUE_PROPERTIES:
        CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES:            CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:           1
CL_DEVICE_MAX_READ_IMAGE_ARGS:         480
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                480
CL_DEVICE_SINGLE_FP_CONFIG:            CL_FP_DENORM CL_FP_INF_NAN
CL_FP_ROUND_TO_NEAREST
CL_DEVICE_2D_MAX_WIDTH                 16384
CL_DEVICE_2D_MAX_HEIGHT                16384
CL_DEVICE_3D_MAX_WIDTH                 2048
CL_DEVICE_3D_MAX_HEIGHT                2048
CL_DEVICE_3D_MAX_DEPTH                 2048
```

CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t> CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 1, DOUBLE 1

10.10 Anexo 10: Información OpenCL de la Máquina 3

10.10.1 C

Found 2 platform(s).
platform[0x145d610]: profile: FULL_PROFILE
platform[0x145d610]: version: OpenCL 1.1 CUDA 4.2.1
platform[0x145d610]: name: NVIDIA CUDA
platform[0x145d610]: vendor: NVIDIA Corporation
platform[0x145d610]: extensions: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll
platform[0x145d610]: Found 1 device(s).
device[0x145d6d0]: NAME: GeForce GT 520
device[0x145d6d0]: VENDOR: NVIDIA Corporation
device[0x145d6d0]: PROFILE: FULL_PROFILE
device[0x145d6d0]: VERSION: OpenCL 1.1 CUDA
device[0x145d6d0]: EXTENSIONS: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_device_attribute_query cl_nv_pragma_unroll cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_fp64
device[0x145d6d0]: DRIVER_VERSION: 304.88

device[0x145d6d0]: Type: GPU
device[0x145d6d0]: EXECUTION_CAPABILITIES: Kernel
device[0x145d6d0]: GLOBAL_MEM_CACHE_TYPE: Read-Write (2)
device[0x145d6d0]: CL_DEVICE_LOCAL_MEM_TYPE: Local (1)
device[0x145d6d0]: SINGLE_FP_CONFIG: 0x3f
device[0x145d6d0]: QUEUE_PROPERTIES: 0x3

device[0x145d6d0]: VENDOR_ID: 4318
device[0x145d6d0]: MAX_COMPUTE_UNITS: 1
device[0x145d6d0]: MAX_WORK_ITEM_DIMENSIONS: 3
device[0x145d6d0]: MAX_WORK_GROUP_SIZE: 1024
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_CHAR: 1
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_SHORT: 1
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_INT: 1
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_LONG: 1
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_FLOAT: 1
device[0x145d6d0]: PREFERRED_VECTOR_WIDTH_DOUBLE: 0
device[0x145d6d0]: MAX_CLOCK_FREQUENCY: 1620
device[0x145d6d0]: ADDRESS_BITS: 32
device[0x145d6d0]: MAX_MEM_ALLOC_SIZE: 268222464
device[0x145d6d0]: IMAGE_SUPPORT: 1
device[0x145d6d0]: MAX_READ_IMAGE_ARGS: 128
device[0x145d6d0]: MAX_WRITE_IMAGE_ARGS: 8
device[0x145d6d0]: IMAGE2D_MAX_WIDTH: 32768
device[0x145d6d0]: IMAGE2D_MAX_HEIGHT: 32768
device[0x145d6d0]: IMAGE3D_MAX_WIDTH: 2048
device[0x145d6d0]: IMAGE3D_MAX_HEIGHT: 2048
device[0x145d6d0]: IMAGE3D_MAX_DEPTH: 2048
device[0x145d6d0]: MAX_SAMPLERS: 16
device[0x145d6d0]: MAX_PARAMETER_SIZE: 4352

device[0x145d6d0]: MEM_BASE_ADDR_ALIGN: 4096
device[0x145d6d0]: MIN_DATA_TYPE_ALIGN_SIZE: 128
device[0x145d6d0]: GLOBAL_MEM_CACHELINE_SIZE: 128
device[0x145d6d0]: GLOBAL_MEM_CACHE_SIZE: 16384
device[0x145d6d0]: GLOBAL_MEM_SIZE: 1072889856
device[0x145d6d0]: MAX_CONSTANT_BUFFER_SIZE: 65536
device[0x145d6d0]: MAX_CONSTANT_ARGS: 9
device[0x145d6d0]: LOCAL_MEM_SIZE: 49152
device[0x145d6d0]: ERROR_CORRECTION_SUPPORT: 0
device[0x145d6d0]: PROFILING_TIMER_RESOLUTION: 1000
device[0x145d6d0]: ENDIAN_LITTLE: 1
device[0x145d6d0]: AVAILABLE: 1
device[0x145d6d0]: COMPILER_AVAILABLE: 1

platform[0x1463580]: profile: FULL_PROFILE
platform[0x1463580]: version: OpenCL 1.1 LINUX
platform[0x1463580]: name: Intel(R) OpenCL
platform[0x1463580]: vendor: Intel(R) Corporation
platform[0x1463580]: extensions: cl_khr_fp64 cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store cl_intel_printf cl_ext_device_fission cl_intel_immediate_execution cl_khr_icd
platform[0x1463580]: Found 1 device(s).
device[0x146f380]: NAME: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
device[0x146f380]: VENDOR: Intel(R) Corporation
device[0x146f380]: PROFILE: FULL_PROFILE
device[0x146f380]: VERSION: OpenCL 1.1 (Build 15293.6649)
device[0x146f380]: EXTENSIONS: cl_khr_fp64 cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store cl_intel_printf cl_ext_device_fission cl_intel_immediate_execution
device[0x146f380]: DRIVER_VERSION: 1.1

device[0x146f380]: Type: CPU
device[0x146f380]: EXECUTION_CAPABILITIES: Kernel Native Unknown (0x4)
device[0x146f380]: GLOBAL_MEM_CACHE_TYPE: Read-Write (2)
device[0x146f380]: CL_DEVICE_LOCAL_MEM_TYPE: Global (2)
device[0x146f380]: SINGLE_FP_CONFIG: 0x7
device[0x146f380]: QUEUE_PROPERTIES: 0x3

device[0x146f380]: VENDOR_ID: 32902
device[0x146f380]: MAX_COMPUTE_UNITS: 8
device[0x146f380]: MAX_WORK_ITEM_DIMENSIONS: 3
device[0x146f380]: MAX_WORK_GROUP_SIZE: 1024
device[0x146f380]: PREFERRED_VECTOR_WIDTH_CHAR: 16
device[0x146f380]: PREFERRED_VECTOR_WIDTH_SHORT: 8
device[0x146f380]: PREFERRED_VECTOR_WIDTH_INT: 4
device[0x146f380]: PREFERRED_VECTOR_WIDTH_LONG: 2
device[0x146f380]: PREFERRED_VECTOR_WIDTH_FLOAT: 4
device[0x146f380]: PREFERRED_VECTOR_WIDTH_DOUBLE: 2
device[0x146f380]: MAX_CLOCK_FREQUENCY: 3400

```
device[0x146f380]: ADDRESS_BITS: 64
device[0x146f380]: MAX_MEM_ALLOC_SIZE: 1030622208
device[0x146f380]: IMAGE_SUPPORT: 1
device[0x146f380]: MAX_READ_IMAGE_ARGS: 128
device[0x146f380]: MAX_WRITE_IMAGE_ARGS: 128
device[0x146f380]: IMAGE2D_MAX_WIDTH: 8192
device[0x146f380]: IMAGE2D_MAX_HEIGHT: 8192
device[0x146f380]: IMAGE3D_MAX_WIDTH: 2048
device[0x146f380]: IMAGE3D_MAX_HEIGHT: 2048
device[0x146f380]: IMAGE3D_MAX_DEPTH: 2048
device[0x146f380]: MAX_SAMPLERS: 128
device[0x146f380]: MAX_PARAMETER_SIZE: 1024
device[0x146f380]: MEM_BASE_ADDR_ALIGN: 1024
device[0x146f380]: MIN_DATA_TYPE_ALIGN_SIZE: 128
device[0x146f380]: GLOBAL_MEM_CACHELINE_SIZE: 64
device[0x146f380]: GLOBAL_MEM_CACHE_SIZE: 262144
device[0x146f380]: GLOBAL_MEM_SIZE: 4122488832
device[0x146f380]: MAX_CONSTANT_BUFFER_SIZE: 131072
device[0x146f380]: MAX_CONSTANT_ARGS: 128
device[0x146f380]: LOCAL_MEM_SIZE: 32768
device[0x146f380]: ERROR_CORRECTION_SUPPORT: 0
device[0x146f380]: PROFILING_TIMER_RESOLUTION: 1
device[0x146f380]: ENDIAN_LITTLE: 1
device[0x146f380]: AVAILABLE: 1
device[0x146f380]: COMPILER_AVAILABLE: 1
```

10.10.2 Java

```
Number of platforms: 2
PLATFORM NUMBER: 0
Number of devices in platform NVIDIA CUDA: 1
PLATFORM NUMBER: 1
Number of devices in platform Intel(R) OpenCL: 1

--- Info for device GeForce GT 520: ---
CL_DEVICE_NAME:                         GeForce GT 520
CL_DEVICE_VENDOR:                       NVIDIA Corporation
CL_DRIVER_VERSION:                      304.88
CL_DEVICE_TYPE:                                 CL_DEVICE_TYPE_GPU -->  0
CL_DEVICE_MAX_COMPUTE_UNITS:            1
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:     3
CL_DEVICE_MAX_WORK_ITEM_SIZES:          1024 / 1024 / 64
CL_DEVICE_MAX_WORK_GROUP_SIZE:          1024
CL_DEVICE_MAX_CLOCK_FREQUENCY:          1620 MHz
CL_DEVICE_ADDRESS_BITS:                 32
CL_DEVICE_MAX_MEM_ALLOC_SIZE:           255 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:              1023 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:     no
CL_DEVICE_LOCAL_MEM_TYPE:               local
CL_DEVICE_LOCAL_MEM_SIZE:               48 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:     64 KByte
```

```
CL_DEVICE_QUEUE_PROPERTIES:
        CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES:             CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:        1
CL_DEVICE_MAX_READ_IMAGE_ARGS:          128
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                 8
CL_DEVICE_SINGLE_FP_CONFIG:             CL_FP_DENORM CL_FP_INF_NAN
CL_FP_ROUND_TO_NEAREST CL_FP_ROUND_TO_ZERO CL_FP_ROUND_TO_INF CL_FP_FMA
CL_DEVICE_2D_MAX_WIDTH                  32768
CL_DEVICE_2D_MAX_HEIGHT                 32768
CL_DEVICE_3D_MAX_WIDTH                  2048
CL_DEVICE_3D_MAX_HEIGHT                 2048
CL_DEVICE_3D_MAX_DEPTH                  2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>    CHAR 1, SHORT 1, INT 1, LONG 1, FLOAT 0,
DOUBLE 1


--- Info for device     Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz: ---
CL_DEVICE_NAME:                         Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
CL_DEVICE_VENDOR:               Intel(R) Corporation
CL_DRIVER_VERSION:              1.1
CL_DEVICE_TYPE:                         CL_DEVICE_TYPE_CPU -->  1
CL_DEVICE_MAX_COMPUTE_UNITS:            8
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:     3
CL_DEVICE_MAX_WORK_ITEM_SIZES:          1024 / 1024 / 1024
CL_DEVICE_MAX_WORK_GROUP_SIZE:          1024
CL_DEVICE_MAX_CLOCK_FREQUENCY:          3400 MHz
CL_DEVICE_ADDRESS_BITS:                 64
CL_DEVICE_MAX_MEM_ALLOC_SIZE:           982 MByte
CL_DEVICE_GLOBAL_MEM_SIZE:              3931 MByte
CL_DEVICE_ERROR_CORRECTION_SUPPORT:     no
CL_DEVICE_LOCAL_MEM_TYPE:               global
CL_DEVICE_LOCAL_MEM_SIZE:       32 KByte
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE:     128 KByte
CL_DEVICE_QUEUE_PROPERTIES:
        CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE
CL_DEVICE_QUEUE_PROPERTIES:             CL_QUEUE_PROFILING_ENABLE
CL_DEVICE_IMAGE_SUPPORT:        1
CL_DEVICE_MAX_READ_IMAGE_ARGS:          128
CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                 128
CL_DEVICE_SINGLE_FP_CONFIG:             CL_FP_DENORM CL_FP_INF_NAN
CL_FP_ROUND_TO_NEAREST
CL_DEVICE_2D_MAX_WIDTH                  8192
CL_DEVICE_2D_MAX_HEIGHT                 8192
CL_DEVICE_3D_MAX_WIDTH                  2048
CL_DEVICE_3D_MAX_HEIGHT                 2048
CL_DEVICE_3D_MAX_DEPTH                  2048
CL_DEVICE_PREFERRED_VECTOR_WIDTH_<t>    CHAR 16, SHORT 8, INT 4, LONG 2, FLOAT 4,
DOUBLE 2
```

11 Bibliografía

1. **Garcia, F.** Historia de Java. [En línea] http://zarza.usal.es/~fgarcia/doc/tuto2/I_2.htm.

2. Historia del lenguaje Java. [En línea] http://www.cad.com.mx/historia_del_lenguaje_java.htm.

3. **Luis, Mario de.** *Programación en Java.* Madrid : Prensa Técnica, 2002. 84-95956-08-X.

4. **Ceballos, Fco. Javier.** *C/C++ Curso de programación.* s.l. : Ra-Ma, 2002. 84-7897-480-6.

5. **Brian W. Kernighan and Dennis M. Ritchie.** The C Programming Language, Second Edition. [En línea] http://cm.bell-labs.com/cm/cs/cbook/.

6. **Brian W. Kernighan and Dennis M. Ritchie.** *El Lenguaje de Programación C.* s.l. : Prentice-Hall Hispanoamericana. 968-880-205-0.

7. **Ritchie, Dennis M.** The Development of the C Language. [En línea] http://cm.bell-labs.com/cm/cs/who/dmr/chist.html.

8. Multiprocessors and Thread-Level Parallelism. [En línea] http://csis.bits-pilani.ac.in/faculty/sundarb/courses/old/fall09/parcom/readings/multiproc-hennessey.pdf.

9. **Barbara Chapman, Gabriele Jost and Ruud van der Pas.** *Using OpenMP.* October 2007. 9780262533027.

10. **Free Software Foundation.** The GNU OpenMP Implementation. [En línea] http://gcc.gnu.org/onlinedocs/gcc-4.6.4/libgomp.pdf.

11. **Board, OpenMP Architecture Review.** GCC Wiki. [En línea] http://gcc.gnu.org/wiki/openmp.

12. Parallel programming in Java with OpenMP-like directives. [En línea] http://www2.epcc.ed.ac.uk/computing/research_activities/jomp/about.html.

13. **J. M. Bull and M.E. Kambites.** JOMP - an OpenMP-like interface for Java. [En línea] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.116.9029&rep=rep1&type=pdf.

14. Arquitectura OpenCL. [En línea] http://sabia.tic.udc.es/gc/trabajos%202011-12/ATIvsCUDA/arquitectura.html.

15. **Intel.** Threading Building Blocks. [En línea] https://www.threadingbuildingblocks.org/.

16. **Farnham, Kevin.** Threading Building Blocks Scheduling and Task Stealing. [En línea] http://software.intel.com/en-us/blogs/2007/08/13/threading-building-blocks-scheduling-and-task-stealing-introduction.

17. —. Threading Building Blocks Scheduling and Task Stealing: Introduction. [En línea] 13 de 08 de 2007. http://software.intel.com/en-us/blogs/2007/08/13/threading-building-blocks-scheduling-and-task-stealing-introduction.

18. **Wang, Peter.** Compare OpenMP & Intel Threading Building Blocks for parallel programmin. [En línea] 16 de 12 de 2008. http://software.intel.com/en-us/blogs/2008/12/16/compare-windows-threads-openmp-intel-threading-building-blocks-for-parallel-programming/.

19. **Intel.** Intel Threading Building. [En línea] http://software.intel.com/en-us/intel-tbb.

20. Ventajas de CUDA. [En línea] http://sabia.tic.udc.es/gc/trabajos%202011-12/ATIvsCUDA/ventajas.html.

21. **Kamran Karimi, Neil G. Dickson, Firas Hamze.** Cornell University Library. *A Performance Comparison of CUDA and OpenCL.* [En línea] http://arxiv.org/abs/1005.2581v3.

22. Boletin del Estado. *Ley Orgánica 15/1999.* [En línea] http://www.boe.es/boe/dias/1999/12/14/pdfs/A43088-43099.pdf.

23. **Klauserc.** Parallel Programming OpenMP und JOMP. [En línea] http://n.ethz.ch/~klauserc/FS10/PP/PP10.pdf.

24. Intel® SDK for OpenCL Applications Products Matrix. [En línea] http://software.intel.com/en-us/vcsource/tools/opencl.