Primera parte: INTRODUCCIÓN:

En la actualidad la informática ha experimentado un desarrollo vertiginoso que la ha convertido en una herramienta esencial para muchas disciplinas. Su versatilidad, adaptabilidad y permanente evolución la ha hecho imprescindible tanto en el ámbito profesional, como el científico. De hecho la investigación científica se apoya cada vez más en la informática como herramienta para la simulación de sistemas y para la resolución de problemas.

Esta evolución se manifiesta claramente en el campo de la tecnología. Los sistemas modernos tienen una enorme capacidad de almacenamiento y los microprocesadores instalados les proporcionan una capacidad de cálculo impresionante. Además este fenómeno ha ido acompañada de una bajada de precios de los equipos y sus componentes haciendo que en la actualidad estén disponibles soluciones que hace diez o veinte años estaban al alcance únicamente de grandes instituciones y organismos oficiales. Naturalmente la arquitectura de los ordenadores también ha evolucionado para poder dar cabida a estas innovaciones y sobre todo aprovechar el potencial de los nuevos componentes electrónicos.

Esto fue muy patente, por ejemplo, en el aumento de la velocidad de los procesadores, sin embargo las limitaciones de la tecnología hicieron que los sistemas evolucionasen hacia un modelo multi-procesador para aprovechar y aumentar el potencial de estos componentes. En la actualidad los sistemas en venta son sistemas duales o cuádruples. Sin embargo, esta evolución no se circunscribe exclusivamente a la electrónica, sino también impone cambios en la filosofía de la programación en general; ya que al disponer de sistemas multiprocesadores nos permite desarrollar y ejecutar tareas o incluso programas enteros en paralelo.

Una estrategia novedosa, que se basa en el paralelismo de procesamiento y ejecución, lo representan las nuevas tecnologías que aprovechan la capacidad computacional de las tarjetas gráficas o GPU (Graphics Processor Unit) modernas que disponen de sistemas multiprocesador.

La función original de estos componentes era la de la representación gráfica, que cada vez fue ganando en calidad y realismo, sin embargo se descubrió que su arquitectura las hacía también aptas para otro tipo de aplicaciones.

En la actualidad se intenta aprovechar la gran potencia de cálculo de las GPU para aplicaciones no relacionadas con los gráficos, en lo que recientemente se denomina GPU de propósito general o GPGPU(General Purpose GPU).

Una de las disciplinas científicas que se beneficia de esta Tecnología es el Análisis Numérico que se ocupa de describir, analizar y crear algoritmos numéricos que nos permitan resolver problemas matemáticos, en los que estén involucradas cantidades numéricas, con una precisión determinada. El análisis numérico proporciona los métodos necesarios para llevar a cabo todos aquellos procedimientos matemáticos que permiten realizar simulación de sistemas o cálculo de procesos numéricamente. En general, estos métodos se aplican cuando se necesita un valor numérico como solución a un problema matemático, y los procedimientos "exactos" o "analíticos" (manipulaciones algebraicas, teoría de ecuaciones diferenciales, métodos de integración, etc.) son incapaces de dar una respuesta. Debido a ello, son procedimientos de uso frecuente en la ingeniería, y su desarrollo se ha visto favorecido por la necesidad de éstos de obtener soluciones, aunque la precisión no sea absoluta.

En este contexto, un algoritmo es un procedimiento que nos puede llevar a una solución aproximada de un problema mediante un número finito de pasos que pueden ejecutarse de manera lógica. En algunos casos, se les da el nombre de métodos constructivos a estos algoritmos numéricos.

Algunos de estos métodos se basan en la repetición de un número determinado de operaciones sencillas. Su exactitud aumenta con el aumento de datos de entrada. Por tanto la realización manual de estas tareas se vuelve muy tediosa o prácticamente imposible. Por ello el desarrollo de los ordenadores, gracias a su enorme potencia de cálculo ha convertido en posibles y en eficientes los algoritmos que hasta hace poco eran inaplicables.

Por tanto los ordenadores modernos son capaces de realizar gran número de operaciones en un tiempo cada vez más corto, su uso en el campo del Análisis Numérico es idóneo.

1. ALCANCE DEL PROYECTO:

A partir de este escenario han surgido tecnologías concretas para aprovechar estas oportunidades. En concreto la compañía NVIDIA ha creado la tecnología CUDA para aprovechar sus tarjetas gráficas para el cálculo numérico, mediante el uso de programación en paralelo. El beneficio previsto mediante el uso de esta tecnología es en general la aceleración de aplicaciones mediante la ejecución en paralelo de tareas y en concreto, para nuestro caso, la aceleración de las operaciones involucradas en el cálculo numérico.

La tecnología CUDA es una de entre varias existentes para el uso de la GPU, de hecho su más directa competidora, ATI ha desarrollado una tecnología con el mismo objetivo para el uso de sus tarjetas gráficas llamada AMD FireStream y basada sobre la API Close-To-Metal.

Además existe un entorno independiente de ambas; el llamado Lenguaje de Computación Abierto o OpenCL (Open Computing Language) que consta de una API y de un lenguaje de programación. Juntos permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en GPUs como CPUs. El lenguaje está basado en el C99, eliminando ciertas funcionalidades y extendiéndolo con operaciones vectoriales.

Apple creó la especificación original y la propuso al Grupo Khronos para convertirla en un estándar abierto y libre.

El Objetivo de este proyecto es Estudiar la tecnología CUDA y explorar las posibilidades de su aplicación en el cálculo científico para la resolución de problemas de Mecánica de Medios Continuos.

Se partirá del estudio de la tecnología y la disponibilidad del Hardware, luego se utilizará esta tecnología para resolver un caso concreto cuya solución ha sido obtenida previamente con otros métodos, aplicando los cambios y adaptaciones necesarias para la ejecución en la GPU y finalmente se establecerán comparaciones y extraerán conclusiones para posibles desarrollos posteriores y aplicaciones más sofisticadas. El alcance del proyecto desarrollará los apartados siguientes:

1. Analizar la tecnología y proponer soluciones Hardware: Se trata de estudiar la arquitectura de las GPU que soportan CUDA para conocer su funcionamiento y los parámetros que controlan la ejecución. Una vez hecho esto, se analizarán las soluciones disponibles en el mercado atendiendo a prestaciones, velocidad, precio y disponibilidad. Finalmente se propondrá una solución adecuada para la adquisición de un equipo para el Laboratorio.

2. Resolver en CUDA la ecuación del calor según un modelo de elementos finitos: Se partirá de un problema sencillo, resuelto previamente mediante un código MATLAB que se ejecuta en la CPU y donde la solución exacta es conocida. Se realizarán los cambios necesarias para adaptar este problema a la ejecución en GPU incluyendo cambios de programación secuencial a paralela.

3. Finalmente, se comparará el rendimiento de la solución CUDA desarrollada en base al modelo de ejecución CPU-GPU con el de MATLAB como modelo de aplicación CPU y se extraerán conclusiones: Una vez obtenido el código CUDA se comparará su velocidad con el código MATLAB que se ejecuta en la CPU para varios escenarios. También se comprobará su coincidencia con la solución exacta, previamente conocida. Esto permitirá extraer conclusiones sobre la tecnología, en especial se valorará su utilidad y eficiencia para el análisis numérico. Además se expondrán las experiencias realizadas durante el proyecto que permitirán un mejor aprovechamiento de la tecnología.

2. PROGRAMACIÓN EN PARALELO:

Un concepto esencial para entender la enorme capacidad de los sistemas multiprocesador en general y en especial los sistemas apoyados en GPU es la programación en paralelo.

a. Programación Secuencial:

En el ámbito de este proyecto se entenderá por programación secuencial aquella en la que el código se desarrolla para ser ejecutado de manera secuencial (una instrucción tras otra, manejando uno o dos datos a la vez) en un sistema CPU. Este tipo de programación se basa en la creación de programas a partir de un conjunto de sentencias escritas de forma secuencial y cuya ejecución sigue dicha secuencia.

El software se ha orientado tradicionalmente hacia la Programación secuencial. Para resolver un problema, se construye un algoritmo y se implementa en un flujo de instrucciones en serie. Estas instrucciones se ejecutan en la unidad central de procesamiento de un ordenador (CPU). En el momento en el que una instrucción se termina, se ejecuta la siguiente.

Se incluye el caso en que se ejecutan en un sistema multi-CPU ya que en estos códigos el programador no define ni controla los parámetros de ejecución sino que lo hace de manera transparente el Sistema Operativo.

b. Programación Paralela:

La Programación paralela es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que pueden resolverse de forma concurrente ("en paralelo").

Existen varios tipos de computación paralela: paralelismo a nivel de bit, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas. Durante muchos años, la programación paralela se ha aplicado en la computación de altas prestaciones, pero su interés ha aumentado en los últimos años debido a las restricciones físicas que impiden el escalado en frecuencia, o velocidad de los microprocesadores. La programación paralela se ha convertido en el paradigma dominante en la arquitectura de computadores, principalmente en los procesadores multinúcleo. Sin embargo, recientemente, el consumo de energía de los ordenadores paralelos se ha convertido en una preocupación.

La Programación paralela emplea elementos de procesamiento múltiple simultáneamente para resolver un problema. Esto se logra dividiendo el problema en partes independientes de tal manera que cada elemento de procesamiento pueda ejecutar su parte del algoritmo a la misma vez que los demás. Los elementos de procesamiento pueden ser diversos e incluir recursos tales como un único ordenador con muchos procesadores, varios ordenadores en red, hardware especializado o una combinación de los anteriores.

Los ordenadores paralelos se pueden clasificar según el nivel de paralelismo que admite su hardware: los ordenadores multinúcleo y multiproceso tienen varios elementos de procesamiento en una sola máquina, mientras que los clusters, los MPP y los grids emplean varios ordenadores para trabajar en la misma tarea.

Los programas de ordenador paralelos son más difíciles de escribir que los secuenciales porque la concurrencia introduce nuevos tipos de errores de software, siendo las condiciones de carrera los más comunes. La comunicación y la sincronización entre las diferentes subtareas son típicamente las grandes barreras para conseguir un buen rendimiento de los programas paralelos. El incremento de velocidad que se logra en un programa de este tipo obedece a la ley de Amdahl:

$$\frac{1}{(1-P)+\frac{P}{S}}.$$

Donde P es la proporción de código en paralelo y S la aceleración que se logra mediante la programación paralela.

3. CPU Vs. GPU.

Desde la introducción del primer microprocesador, el Intel 4004, en 1970, y del primer microprocesador ampliamente usado, el Intel 8080, en 1974, esta clase de CPUs ha desplazado casi totalmente el resto de los métodos de implementación de la Unidad Central de Proceso. Este hecho, junto con la aparición y popularización del PC, han hecho que el término "CPU" sea aplicado ahora casi exclusivamente a los microprocesadores.

Las generaciones previas de CPUs fueron implementadas como componentes discretos y numerosos circuitos integrados de pequeña escala de integración en una o más tarjetas de circuitos. Por otro lado, los microprocesadores son CPUs fabricadas con un número muy pequeño de Circuitos Integrados; usualmente solo uno. El tamaño más pequeño de la CPU, como resultado de estar implementado en una simple pastilla, significa tiempos de conmutación más rápidos debido a factores físicos como el decrecimiento de la capacitancia parásita de las puertas. Esto ha permitido que los microprocesadores síncronos tengan tiempos de reloj con un rango de decenas de megahercios a varios gigahercios. Además, como ha aumentado la capacidad de construir transistores extremadamente pequeños en un Circuito Integrado, la complejidad y el número de transistores en una simple CPU también se ha incrementado espectacularmente. Esta tendencia ampliamente observada es la expuesta por la famosa ley de Moore.

Mientras que, en los pasados sesenta años han cambiado drásticamente, la complejidad, el tamaño, la construcción, y la forma general de la CPU, es notable que el diseño y el funcionamiento básico no hayan cambiado demasiado. Casi todos las CPU comunes de hoy se pueden describir completamente como máquinas de von Neumann.

En la actualidad han surgido preocupaciones sobre los límites de la tecnología del transistor del circuito integrado. La miniaturización extrema de puertas electrónicas está causando que los efectos de algunos fenómenos, que antaño eran despreciables, se hayan vuelto mucho más significativos, y planteen limitaciones al aumento constante de frecuencia. Estos problemas están entre los muchos factores que han impulsado el estudio de la computadora cuántica, así como la extensión del uso del paralelismo, y otros métodos que amplían la utilidad del modelo clásico de von Neumann.

Como se ha descrito previamente una CPU ejecuta una instrucción simple sobre uno o dos datos a la vez. Debido a ello la CPU debe esperar que esa instrucción se complete antes de proceder a la siguiente instrucción. Como resultado, la CPU queda "paralizada" en instrucciones que toman más de un ciclo de reloj para completar su ejecución. Incluso la adición de una segunda unidad de ejecución no mejora mucho el desempeño. En lugar de que quede congelada una ruta, ahora se paralizan las dos y aumenta el número de transistores no usados.

Las tentativas de alcanzar un desempeño más eficiente, han resultado en una variedad de metodologías de diseño que hacen comportarse a la CPU menos de manera lineal y más en paralelo.

Cuando se habla de paralelismo en las CPUs, generalmente son usados dos términos para clasificar las técnicas de diseño.

* El paralelismo a nivel de instrucción, o ILP(Instruction Level Parallelism), busca aumentar la tasa en la cual las instrucciones son ejecutadas dentro de una CPU, es decir, aumentar la utilización de los recursos de ejecución en la pastilla

* El paralelismo a nivel de hilo de ejecución,o TLP(Thread level parallelism), que se propone incrementar el número de hilos (efectivamente programas individuales) que una CPU pueda ejecutar simultáneamente.

Las metodologías se diferencian tanto en la manera en la que son implementadas, como en su efectividad en el aumento del desempeño de la CPU para una aplicación.

Las modernas GPUs son descendientes de los chips gráficos monolíticos de finales de la década de 1970 y 1980. Estos chips tenían soporte BitBLT limitado en la forma de sprites, y usualmente no tenían soporte para dibujo de figuras. Algunas GPUs podían ejecutar varias operaciones en una lista de "display" y podían usar DMA para reducir la carga en el procesador anfitrión; un ejemplo temprano es el coprocesador ANTIC usado en el Atari 800 y el Atari 5200. Hacia finales de los 80 y principios de los 90, los microprocesadores de propósito general de alta velocidad fueron muy populares para implementar los GPUs más avanzados. Muchas (y muy caras) tarjetas gráficas para PCs y Estaciones de Trabajo usaban Procesadores Digitales de Señal (DSP) tales como la serie TMS340 de Texas Instruments, para implementar funciones de dibujo rápidas y muchas impresoras láser contenían un procesador de barrido de imágenes "PostScript" (un caso especial de GPU) corriendo en un procesador RISC como el AMD 29000.

Conforme la tecnología de proceso de semiconductores fue mejorando, fue posible transferir las funciones de dibujo y las BitBLT a la misma placa y posteriormente al mismo chip a manera de un controlador de buffer de "marcos"(frames), tal como VGA. Estos aceleradores gráficos de 2D "reducidos" no eran tan flexibles como los basados en microprocesadores, pero eran mucho más fáciles de hacer y vender. El Commodore AMIGA fue el primer ordenador producido en masa que incluía una unidad blitter y el sistema gráfico IBM 8514 fue una de las primeras tarjetas de vídeo para PC en implementar primitivas 2D en hardware.

a. ARQUITECTURAS:

Una GPU está altamente segmentada, lo que indica que posee gran cantidad de unidades funcionales. Estas unidades funcionales se pueden dividir principalmente en dos: aquéllas que procesan vértices, y aquéllas que procesan píxeles. Por tanto, se establecen el vértice y el píxel como las principales unidades que maneja la GPU.

Adicionalmente, y no con menos importancia, se encuentra la memoria. Ésta destaca por su rapidez, y va a jugar un papel relevante a la hora de almacenar los resultados intermedios de las operaciones y las texturas que se utilicen.

Inicialmente, a la GPU le llega la información de la CPU en forma de vértices. El primer tratamiento que reciben estos vértices se realiza en el vertex shader. Aquí se realizan transformaciones como la rotación o el movimiento de las figuras. Tras esto, se define la parte de estos vértices que se va a ver (clipping), y los vértices se transforman en píxeles mediante el proceso de rasterización. Estas etapas no poseen una carga relevante para la GPU.

Donde sí se encuentra el principal cuello de botella del chip gráfico es en el siguiente paso: el pixel shader. Aquí se realizan las transformaciones referentes a los píxeles, tales como la aplicación de texturas. Cuando se ha realizado todo esto, y antes de almacenar los píxeles en la caché, se aplican algunos efectos como el antialiasing, blending y el efecto niebla.

Otras unidades funcionales llamadas ROP toman la información guardada en la caché y preparan los píxeles para su visualización. También pueden encargarse de aplicar algunos efectos. Tras esto, se almacena la salida en el frame buffer. Ahora hay dos opciones: o tomar directamente estos píxeles para su representación en un monitor digital, o generar una señal analógica a partir de ellos, para monitores analógicos. Si es este último caso, han de pasar por un Convertidor Analógico-Digital o DAC,(Digital-Analog Converter), para ser finalmente mostrados en pantalla.

Al principio, la programación de la GPU se realizaba con llamadas a servicios de interrupción de la BIOS. Tras esto, la programación de la GPU se empezó a hacer en el lenguaje ensamblador específico para cada modelo. Posteriormente, se intercaló un nivel adicional entre el hardware y el software, diseñando las APIs (Application Program Interface), que proporcionaban un lenguaje más homogéneo para los modelos existentes en el mercado. la primera API usada ampliamente fue el estándar abierto OpenGL (Open Graphics Language), tras el cual Microsoft desarrolló DirectX.

Tras el desarrollo de APIs, se decidió crear un lenguaje más natural y cercano al programador, es decir, desarrollar un lenguaje de alto nivel para gráficos. Por ello, de OpenGL y DirectX surgieron otras propuestas como El lenguaje estándar de alto nivel, asociado a la biblioteca OpenGL que se llama el "OpenGL Shading Language", GLSL, implementado en principio por todos los fabricantes. La empresa NVIDIA creó un lenguaje propietario llamado Cg (del inglés, "C for graphics"), con mejores resultados que GLSL en las pruebas de eficiencia. En colaboración con NVIDIA, Microsoft desarrolló su "High Level Shading Language", HLSL, prácticamente idéntico a Cg, pero con ciertas incompatibilidades menores.

b. MODELO DE COMPUTACIÓN CPU_GPU.

La utilización de las GPUs para computación se basa fundamentalmente en el aprovechamiento del diseño de éstas para una programación paralela. En concreto el modelo de “instrucción única-múltiples datos”, lo que permite que una instrucción sea ejecutada por distintos hilos usando datos diferentes dependiendo del hilo.

En la figura 1 se ilustra este modelo. En general el modelo parte de un bloque principal que se ejecuta en la CPU. Este bloque llamado MAIN en el diagrama controla el inicio y final del programa. Cuando se alcanza una sección que permita ser ejecutada en paralelo, se invoca a las secciones HOST correspondientes.

Las secciones HOST establecen la transición entre la ejecución CPU y GPU. Se encargan de reservar memoria en la GPU, de copiar los datos necesarios para la operación de la memoria local a la memoria de la GPU, y de definir los parámetros de ejecución (Que en el caso de CUDA definen el número de hilos y la velocidad). Estas instrucciones son trasladadas a la GPU mediante la invocación de los Kernels.

Los Kernels son las secciones de ejecución en Paralelo que se ejecutan en la GPU. Sólo estas secciones son ejecutadas en la GPU. Se limitan a usar los datos previamente recibidos desde la memoria local y ejecutar las instrucciones paralelas definidas. Una vez concluidas estas operaciones se copian los resultados a la memoria local y se devuelve el control, primero, a la sección HOST y de ésta nuevamente a la sección principal MAIN.

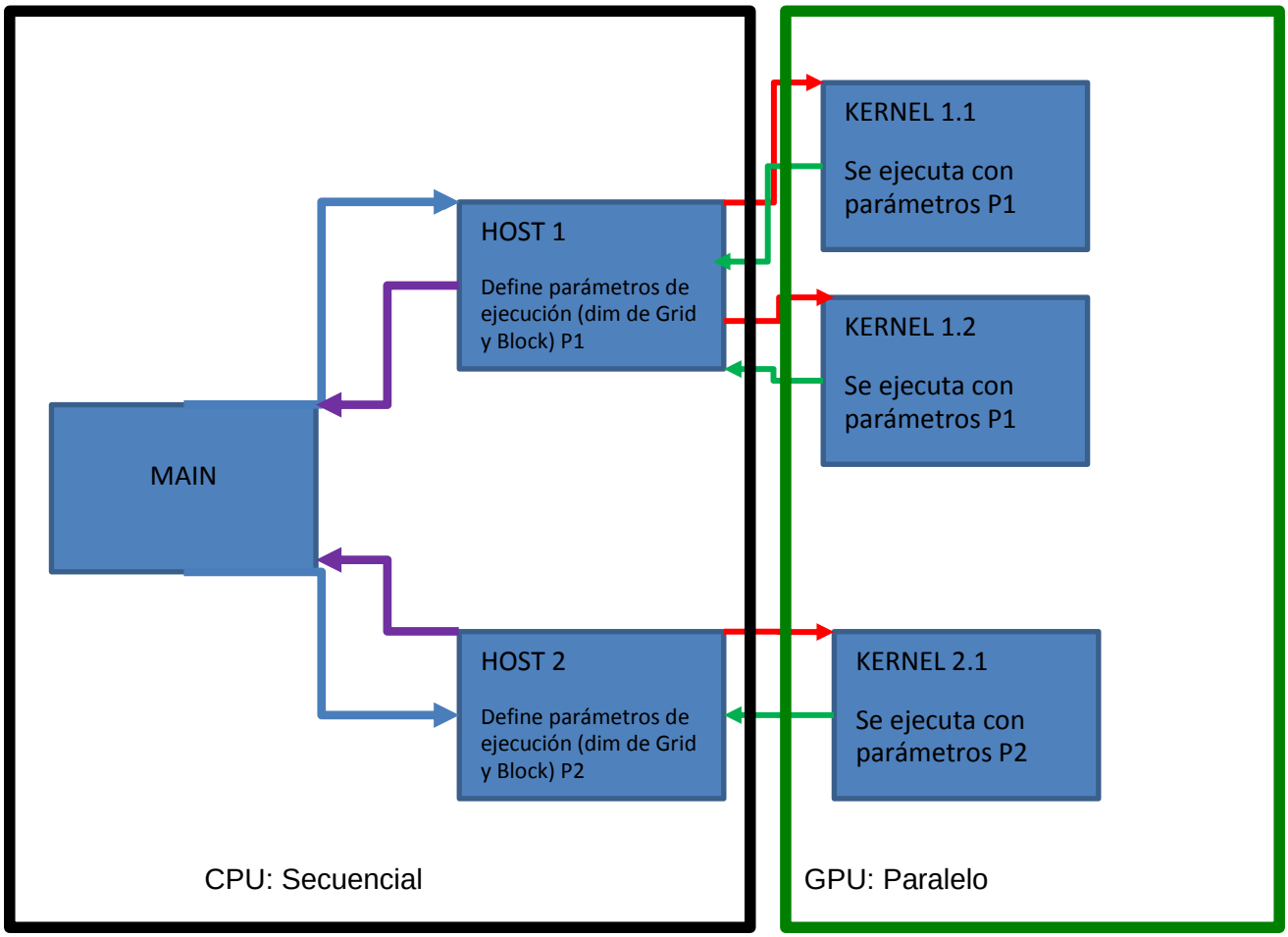

Fig. 1: Representación del modelo de computación CPU-GPU.