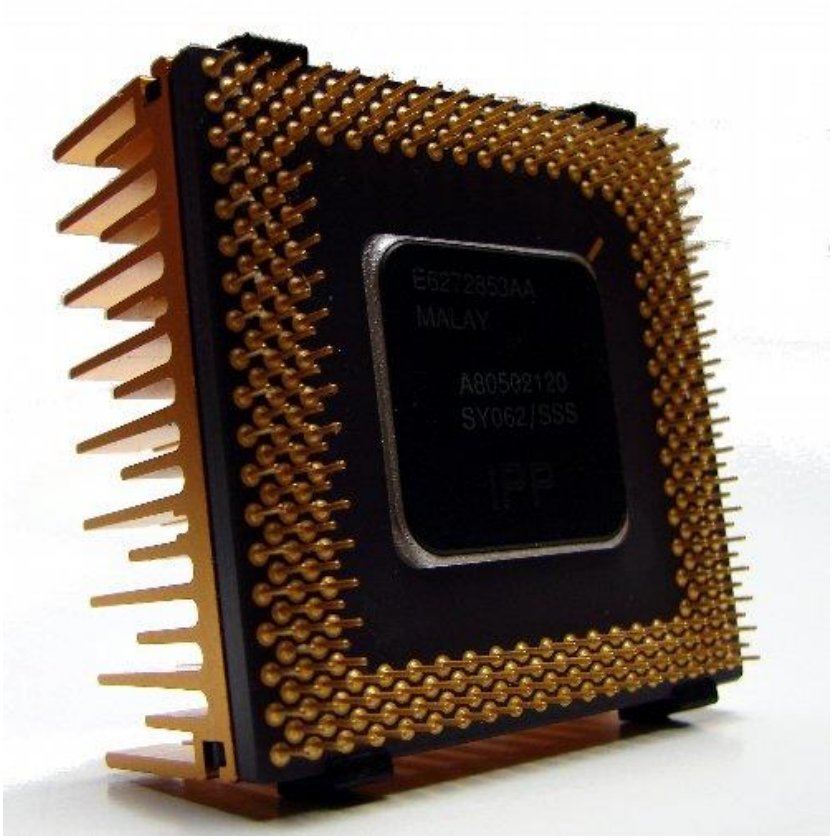
e

MATERIAL ADICIONAL F-IV

- **Ejercicios para desarrollar individualmente**
- **Ejercicio en grupo**

PRÁCTICA 1



2017

Proyecto de programación con GPGPU

F-IV. Implementación de una rutina con CUDA para la aceleración de algoritmos utilizando GPGPU

Arquitectura de los Computadores
Grado en Ingeniería Informática
Dpto. Tecnología Informática y Computación
Universidad de Alicante

MATERIAL ADICIONAL F-IV

PROYECTO DE ACELERACIÓN DE ALGORITMOS CON GPGPU

I. CONTENIDOS

En esta fase, se pretende comparar la arquitectura SIMD ya utilizada en la práctica anterior con las arquitecturas GPGPU. Se estudiarán los conceptos de GPGPU (General-Purpose Computing on Graphics Processing Units) y CUDA (Compute Unified Device Architecture). El objetivo es aprender la arquitectura y conceptos de programación de las nuevas GPUs para proporcionar al estudiante conocimientos y desarrollar sus aptitudes en sistemas actualmente utilizados en ordenadores personales y supercomputadoras, que permiten un alto rendimiento y eficiencia.

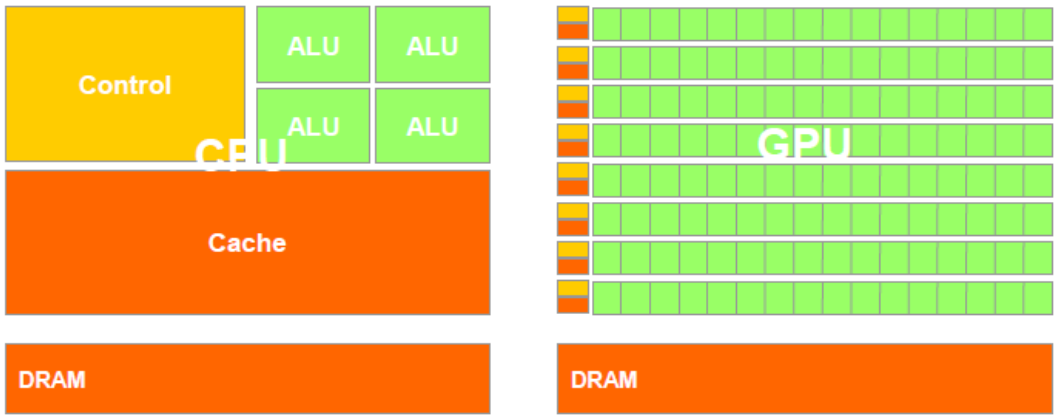
Las diferentes clases de prácticas relativas a esta fase se van a dividir dos partes principales: un primer seminario teórico en el que se presentará la arquitectura GPGPU y una práctica guiada para afianzar los conocimientos; y por otra parte una práctica en grupo en la que se utilicen los conocimientos adquiridos para resolver un problema con CUDA y la GPU.

II. BASE TEÓRICA

GPU

Existen muchos tipos distintos de procesadores, muchos de ellos dedicados a un propósito específicos y otro de propósito general. Tradicionalmente las tarjetas gráficas han sido empleadas para dar soporte al procesador general (CPU) en tareas de procesamiento gráfico. Actualmente estos procesadores gráficas (GPU) están siendo utilizados cada vez más para otro tipo de cálculos. Dentro de la taxonomía de Flynn corresponden con el modelo SIMD, y también son denominados procesadores masivamente paralelos.

Las CPUs y las GPUs tienen fundamentos de diseño diferentes:



	Cache	**Unidad de control**	**ALU**	

CPU	Grandes: Permiten bajar la latencia en los accesos a memoria	Compleja: Branch prediction Data forwarding	Compleja: Reducen latencia de las operaciones	
GPU	Pequeña Potenciar ancho banda memoria	Simple: No Branch prediction No Data forwarding	Muchas simples: Alta segmentadas para aumentar el ancho de banda de la memoria	Requiere un número muy elevado de hilos para ocultar las latencias

Los hilos que maneja una GPU son muy ligeros, por ello se necesita muy poco tiempo para crear y/o destruirlos. La GPU necesita más de 1000 hilos para ser eficiente. Una CPU multi-núcleo solo necesita unos pocos hilos.

Comparación entre CPU y GPU

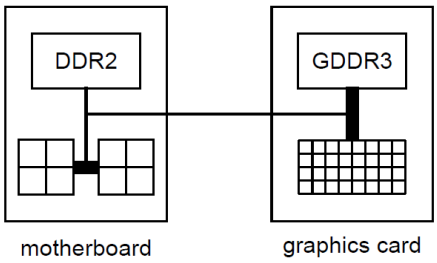
Intel Core 2 / Xeon / i7

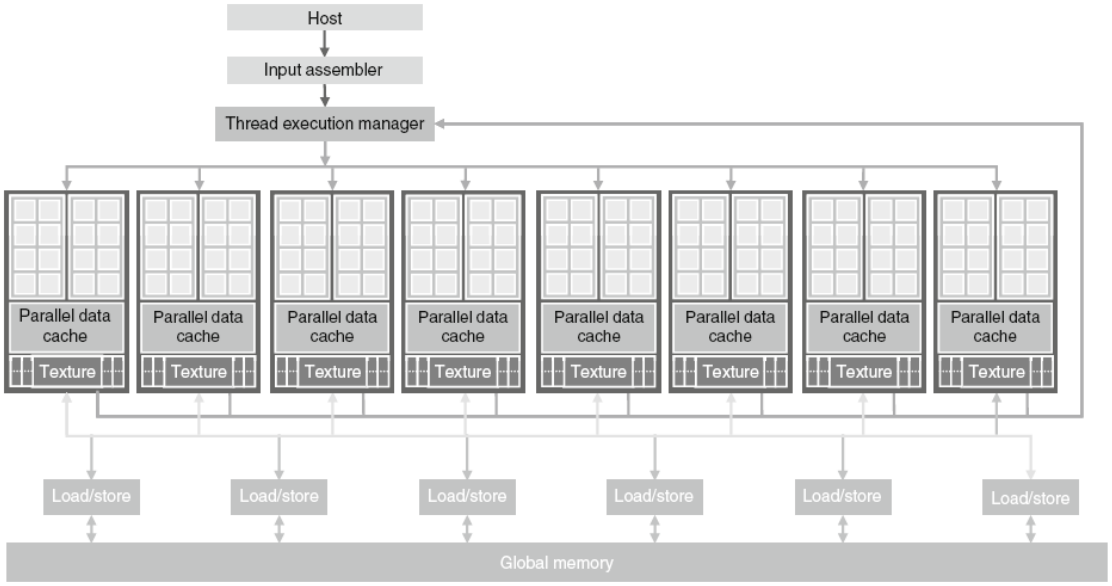
- 4-6 núcleos MIMD
- Pocos registros, cache multi-nivel
- 10-30 GB/s ancho de banda hacia la memoria principal

NVIDIA GTX480

- 512 núcleos, organizados en 16 unidades SM cada una con 32 núcleos.
- Muchos registros, inclusión cachés nivel 1 y 2.
- 5 GB/s ancho de banda hacia el procesador HOST.
- 180 GB/s ancho de banda memoria tarjeta gráfica.

La GPU es una un procesador que trabaja como procesador paralelo a la CPU, no de forma autónoma. Se sitúa sobr e una placa gráfica pci-e dentro de un computador con uno o varios núcleos.





Característica clave de los núcleos dentro de un Streaming Multiprocessor (SM).

- Todos los núcleos ejecutan la misma instrucción simultáneamente pero con distintos datos
- Similar a la computación en los supercomputadores CRAY
- Mínimo de 32 hilos realizando la misma tarea (casi) al mismo tiempo
- Técnica tradicional en el procesamiento gráfico y en muchas aplicaciones científicas

CUDA

CUDA (Compute Unified Device Architecture) CUDA es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento muy importante del rendimiento del sistema. Hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVIDIA.

La plataforma de cálculo paralelo CUDA® proporciona unas cuantas extensiones de C y C++ que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad. El programador puede expresar ese paralelismo mediante diferentes lenguajes de alto nivel como C, C++ y Fortran o mediante estándares abiertos como las directivas de OpenACC.

Los desarrolladores disponen de una gama completa de herramientas y soluciones pertenecientes al ecosistema de CUDA. Hay una gran cantidad de ejemplos y buena documentación, lo cual reduce la curva de aprendizaje para aquellos con experiencia en lenguajes como OpenMPI y MPI. Además existe una extensa comunidad de usuarios en los foros de NVIDIA.

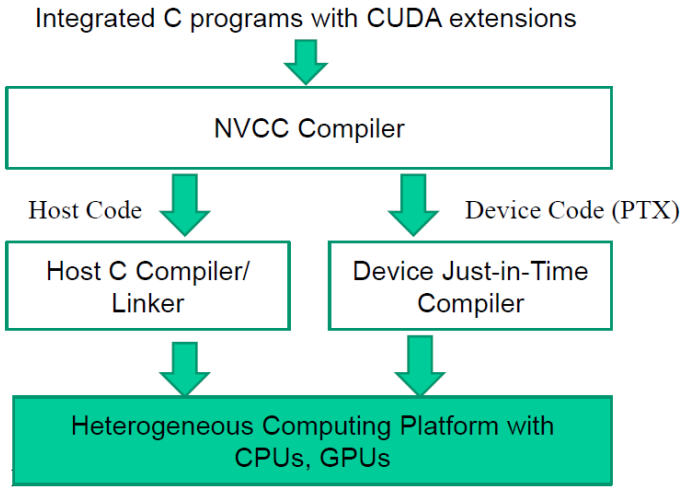
Un programa en CUDA tiene dos partes:

- Código Host en la CPU que hace interfaz con la GPU.
- Código Kernel que se ejecuta sobre la GPU.

Existen dos APIs para el desarrollo de la parte que será ejecutada en la GP. Se diferencian principalmente en el nivel de abstracción que permiten:

- Runtime: Simplificada, más sencilla de usar.
- Driver: más flexible, más compleja de usar. Esta versión no implica mayor rendimiento, sino más flexibilidad a la hora de trabajar con la GPU.

La siguiente figura muestra el proceso de compilación que se lleva a cabo sobre un software que va a ser ejecutado sobre una CPU y GPU.



Conceptos básicos de CUDA

En el nivel más alto encontramos un proceso sobre la CPU (Host) que realiza los siguientes pasos:

- Inicializa GPU.
- Reserva memoria en la parte host y device.
- Copia datos desde el host hacia la memoria device.
- Lanza la ejecución de múltiples copias del kernel.
- Copia datos desde la memoria device al host.
- Se repiten los pasos 3-5 tantas veces como sea necesario.
- Libera memoria y finaliza la ejecución proceso maestro.

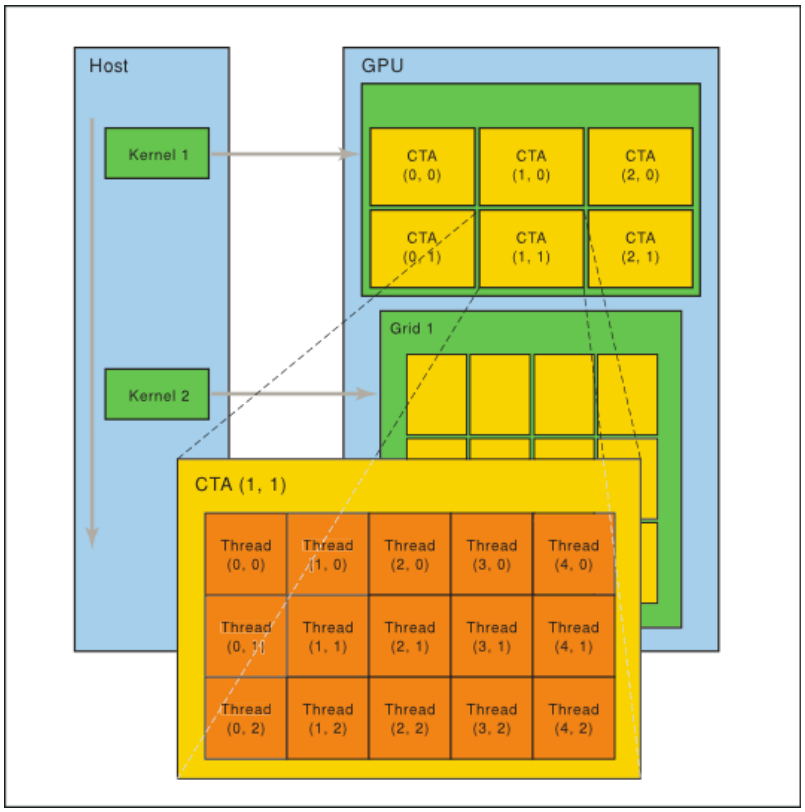
Un kernel CUDA se ejecuta sobre un grid de hilos:

- Todos los threads del grid ejecutan el mismo código.
- Cada hilo tiene sus índices y lo utiliza para direccionar la memoria y realizar su lógica.

Estructura interna

- **Kernel**: es la función que se ejecutará en N distintos hilos en vez de en secuencial.
- **Grid**: forma de estructurar los bloques en el kernel
    - Bloques: 1D, 2D
    - Hilos/Bloque: 1D, 2D o 3D

- **Bloque:**
    - Agrupación de hilos
    - Cada bloque se ejecuta sobre un solo SM
- Un SM puede tener asignados varios bloques



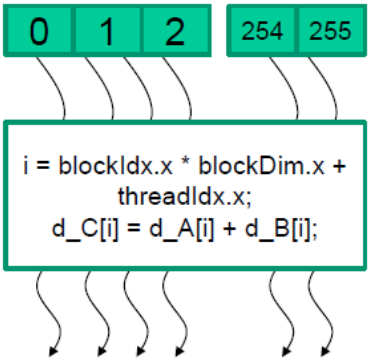
Invocación kernel:

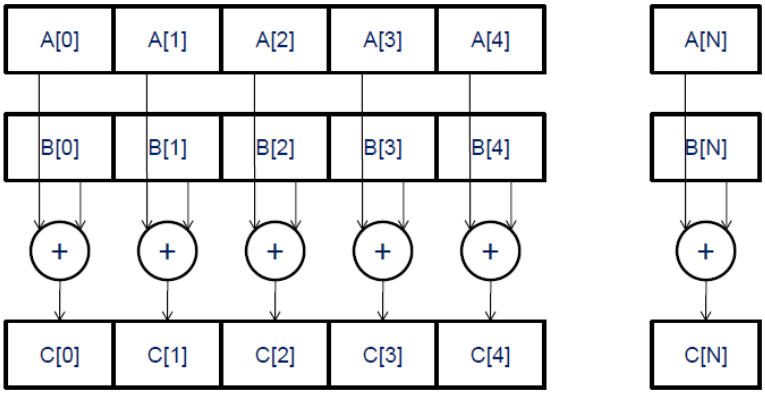
- kernel_routine<<<gridDim, blockDim>>>(args);
    - gridDim : número de bloques. Tamaño del grid.
    - blockDim: número de hilos que se ejecutan dentro de un bloque.
    - Args: número limitado de argumentos, normalmente punteros a memoria de la GPU.
- gridDim y blockDim permiten definirse con estructuras que representan 2 y 3 dimensiones para facilitar la programación de algunas aplicaciones.
    - dim3 dimBlock(256, 256, 1); 1024 hilos organizados matriz
    - dim3 dimGrid(4, 4, 1); 16 bloques organizados matriz

Una vez ejecutado el kernel, dentro de cada copia del mismo tenemos la siguiente información (propia de cada hilo):

- Variables pasadas por argumento
- Punteros a memoria de la GPU
- Constantes globales en memoria GPU
- Variables especiales:
    - gridDim: tamaño o dimensión de la malla de bloques
    - blockIdx: índice del bloque (propio de cada bloque)

- blockDim: tamaño o dimensión de cada bloque
- threadIdx: índice del hilo (propio de cada hilo)





Código tradicional en la CPU

```
// Compute vector sum C = A+B
void vecAdd(float* A, float* B, float* C, int n)
{
  for (i = 0, i < n, i++)
    C[i] = A[i] + B[i];
}
```

Código paralelizado en GPU

```
__global__
void vecAddkernel(float* d_A, float* d_B, float* d_C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) d_C[i] = d_A[i] + d_B[i];
}
```

III. DOCUMENTACIÓN GPGPU Y CUDA

A Continuación se lista una serie de documentos y enlaces sobre programación CUDA.

Uno de los primeros cursos sobre programación en CUDA impartido por la universidad de Illinois:

The first course talking about CUDA programming in the world:

https://courses.engr.illinois.edu/ece498al/Archive/Spring2007/

El manual official de programación CUDA de NVIDIA, el cual describe el modelo de programación, la sintaxis, una serie de consejos básicos para obtener más rendimiento, especificaciones técnicas, etcétera.

http://developer.nvidia.com/nvidia-gpu-computing-documentation

Libros de interés:

Programming Massively Parallel Processors: A Hands-on Approach. By David Kirk and Wen-mei Hwu

CUDA by Example: An Introduction to General-Purpose GPU Programming. By Jason Sanders, Edward Kandrot

GPU Computing GEMS. By Wen-mei W. Hwu

IV. EJERCICIOS PARA DESARROLLAR INDIVIDUALMENTE

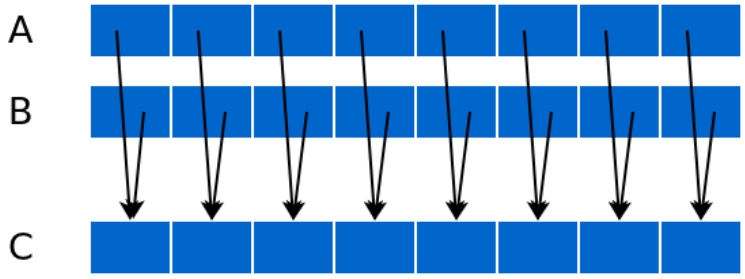
En esta práctica guiada se lleva a cabo una simple suma de vectores, la cual nos ayudará a entender la estructura de un programa en CUDA. Analiza el código que os proporcionamos para saber cómo utilizar los índices de hilos y bloques para escribir kernels en CUDA y como invocar a un kernel con estos parámetros previamente definidos.

Suma de vectores

Analiza el código detenidamente e intenta relacionar los conceptos que acabamos de ver con el código del programa. Sigue los siguientes pasos:

1. Analiza el "kernel" que se va a ejecutar en la GPU.
2. Observa en el código la estructura de funciones que son invocadas hasta completar una llamada a una función CUDA
    a. Reserva de memoria
    b. Transferencia de datos CPU → GPU
    c. Invocación kernel
    d. Transferencia resultados GPU → CPU
3. ¿Cuántas veces se ejecuta la función kernel?
4. Observa las constantes VECTOR_ELEMENTS y COMPUTE_N_ELEMENTS_PER_THREAD. ¿Qué función tienen en el código? Prueba a modificar sus valores y ejecutar el código proporcionado observando que ocurre. Razona correctamente los casos en los que la ejecución falle por algún motivo.
5. Tras ejecutar el código habrás observado que la GPU obtiene un tiempo de ejecución mayor que la CPU cuando las transferencias de datos se incluyen en el cómputo. Esto se debe a que las transferencias son el principal cuello de botella a la hora de portar un código a la GPU. La implementación paralela de la suma de 2 vectores no acarrea suficiente cómputo para ocultar las latencias producidas por las transferencias de memoria. Si la operación suma de vectores se realizase N veces, ¿se ocultaría entonces la latencia producida por las transferencias de datos entre GPU y CPU?

Nota: puedes probar esto último modificando ligeramente el código proporcionado y observar que ocurriría.



V. EJERCICIO EN GRUPO

En esta parte se pretende evaluar las bondades de la utilización de GPGPUs y CUDA a la hora de mejorar el rendimiento en el procesamiento de algoritmos de gran carga computacional. Para ello, cada grupo elegirá la implementación de uno de los siguientes algoritmos siempre que en una misma clase de prácticas, cada grupo elija un problema diferente. Si no hay consenso, se realizará por sorteo:

- Extracción de la región de interés en una secuencia de imágenes (background substraction)
- Operación de dilatación morfológica sobre una imagen
- Cálculo de las normales para una malla 3D
- Realización de un clasificador paralelo basado en SOM
- Aplicación de una rotación y posicionamiento a una malla 3D

Para el desarrollo del algoritmo, se proporcionará una función principal (main) que gestionará la lectura de datos y el cálculo del tiempo de una llamada a un procedimiento que será el que el grupo tenga que implementar.

VI. ENTREGA Y EVALUACIÓN

La evaluación se realizará sobre el desarrollo de la práctica en clase (individual y en grupo). Las entregas serán:

- **Individual**: Memoria explicativa de los ejercicios guiados vistos en clase.
- **Grupo**: Código del ejercicio propuesto y memoria explicativa de la rutina desarrollada incluyendo comparativa de rendimiento.

VII. COMPUTHON

Aquellos alumnos que deseen, podrán participar en un concurso tipo *hackathon* que se celebrará en el laboratorio de supercomputación L14 de la Escuela Politécnica Superior I la semana del 23 al 27 de mayo (día por concretar).

La competición consistirá en proporcionar una solución al problema elegido que proporcione el máximo rendimiento posible. Se establecerá una clasificación por tiempos para cada problema planteado. Para

ello, el profesorado de la asignatura compilará y lanzará el programa el día del concurso. El ganador de cada uno de los problemas conseguirá 1 punto que se le añadirá a la nota final de la asignatura. El resto de clasificados también obtendrán puntuación de la siguiente manera:

Se reparten así:

- 1º: 1 punto
- 2º: 0,6 puntos
- 3º: 0,3 puntos
- 4º: 0,2 puntos
- Resto: 0,1puntos