PAP

- Parámetros en los computadores paralelos
  - Tipo y número de procesadores
    - Procesador de grano fino (operaciones simples)
    - " " " grueso (operaciones complejas)
  - Presencia y ausencia de mecanismos de control
  - Funcionamiento síncrono y asíncrono
  - Formas de comunicación entre procesos

- Paralelismo interno:
  Queda oculto a la arquitectura del computador

- Paralelismo explícito:
  Aquel que queda visible al usuario

Clasificación de Hwang-Briggs

- Establecen una primera aproximación a las clases de computadores paralelos fijando 3 configuraciones básicas:
  - Computadores pipeline
  - " Matriciales
  - Sistemas multiprocesador
- De esta forma se pueden distinguir 2 tipos de paralelismo:
  - temporal (segmentación)
    - Aquellos que se les solapan varias operaciones en un instante
  - Espacial (Existencia de procesos replicados)
    - síncrono
    - Asíncrono

Clasificación Flynn

Single Data SD — Multiple Data MD

Single Instruction SI

Multiple Instruction MI

	SD	MD
SI	SISD	SIMD
MI	MISD	MIMD

SISD

- Modelo Von Neuman (secuencial)
- Instrucciones:
  De memoria a procesador
- Datos:
  Entre memoria y procesador

SIMD

- Una única unidad de control (vectorial)
- Misma instrucción se ejecuta síncronamente en todas las unidades de procesamiento

MISD

Utilizado por la NASA para comprobar los errores al lanzar un cohete

MIMD

Cada procesador ejecuta un programa diferente

- CPU
  - Dominadas por memorias Cache
  - La latencia a estas memorias es importante
  - Ejecutan Instrucciones complejas
    - Pocos Núcleos
- GPU
  - Poca importancia de la latencia
  - Carga de trabajo sobre pixel shader y Vertex shader es relativamente baja
    - Muchos Núcleos

CPU (- Cores + cache + Control)
GPU (+ Cores - Cache - Control)

String procesador // Escalar procesador

1 hilo = 1 escalar procesador

1 WARP = 32 hilos

LATENCIA +

- Global memory
- Local memory
- Constant memory
- shared memory (solo la puede leer su bloque)
- Registro

$$\text{Ancho de banda} = \frac{\text{Frecuencia memoria} \cdot \text{Ancho bus de memoria}}{8}$$

- Frecuencia memoria = 10 GB/s
- bus memoria = 256 bits

$$AB = \frac{10 \cdot 256}{8} = 320 \text{ GB/s}$$

- Frecuencia memoria = 1600 MHz = 1'6 GHz
- bus memoria = 512 bits

$$AB = \frac{1'6 \cdot 2 \cdot 512}{8} = 204'8 \text{ GB/s}$$

- Cada gráfico, proceso de renderización
  - A la GPU le llega una lista de primitivas (triángulos), que tienen la info asociada a peso, color, ...
  - Pasa al procesador de Vértices y aplica la geometría (puede hacer pequeños cambios)
  - Hace el rasterizado (pasar imagenes 3d a 2d) en 2 fases
    - Clipping (Marcar triángulos que puede eliminar porque están fuera de la cámara)
    - Culing (elimina los triángulos marcados previamente)
  - Transforma los triángulos en tramas de píxeles
  - Se pasa al procesador de píxeles
  - Se prepara la imagen para sacarla por pantalla (blending compone la imagen)

¿Que es un renderizado?

- Proceso de transformación de una imagen de 3 dimensiones a 2

¿Que es el rasterizado?

Conversión de triángulos a fragmentos

Anti-Aliasing

Difuminada los dientes de sierra de las imágenes

La GPU utiliza Single Program Multiple Data Stream

(un programa se ejecuta en múltiples hilos que acceden a diferentes flujos de datos

1 bloque = 512 hilos (Máximo 8 bloques por SM)

2 ó más Bloques = 768 hilos

ej:

- Si lanzo una matriz $32 \cdot 32 = 1024$ hilos necesito

⇓

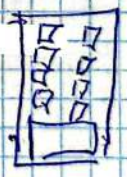
Necesitaré 2 bloques SM (1 de 512 y otro de 512)

¿ cuantas matrices de 256 hilos puedo ejecutar en 1 SM ?

↳ $\frac{768}{256} = 3$

ej:

$16 \cdot 16 = 256$ hilos, si tengo $\frac{768}{256} = 3$ matrices por SM



ej:

$\underbrace{8 \cdot 8}_{\text{Matriz}} \cdot \underbrace{8}_{\text{Bloque}} = 512$ hilos en un SM puedo ejecutar

Ocupación $\frac{1}{3}$ (lo máximo es 768)

16·16 = 256 hilos · 8 bloques (no entra)

⇓

$\frac{768}{256}$ = 3 Bloques Máximo puedo lanzar

$512 \cdot 512 = \frac{262\,144}{256}$ = 1024 Bloques de hilos de 256

cada SM puede 3 Bloques ⇒ $\frac{1024}{3}$ = 342 SM necesito

ej1

Matriz 40 · 40 (a máxima ocupación)

¿Cuantos SM necesito utilizar?

40 · 40 = 1600 hilos

Usaré una matriz 16 · 16 = 256 hilos

$\frac{768}{256}$ = 3 bloques por SM

⇓

$\frac{1600}{256}$ = 6'25 bloques de hilos necesito

⇓

$\left\lceil \frac{6'25}{3} \right\rceil$ = 3 SM necesitaré

Memoria

- Global 768MB - 1024MB
- Local 16KB
- Compartida 16KB
- Registros 8KB
- Constante 64KB
- Texturas 8KB

Compilación

- Pasamos el fichero .cu al compilador
  - Separa lo que va a ejecutar la CPU y la GPU

CPU { - Lo de la CPU pasa al compilador y devuelve un fichero .cpp

GPU { - Crea un fichero en ensamblador para la parte GPU .s
  - Este indica que procesos son paralelos
  - se lo pasa al OCG que unirá este fichero con las características de nuestra tarjeta
  - Devuelve un fichero .SASS que es lo que ejecutará sobre la GPU

- Para acceder a un hilo en concreto de un bloque

  Nº bloque · Nº hilos por bloque + thread

  blockIdx.x · blockDim.x + threadIdx.x

- bloques 2 Dim, hilos 3 dim

  grid(10,21,1) ✓ bloque(1,2,3)

  grid(10,21,2) ✗ bloque(1,2,3)

Ej 11

- 16 KB por SM de memoria compartida
- Matriz de 1024 hilos y consume 32 KB de memoria compartida

¿cuantos SM necesita?

Opciones

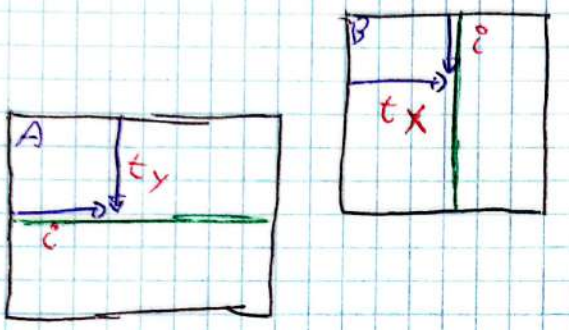
- 3 Bloques de 256 en 1 SM y 1 Bloque de 256 en otro bloque
- 1 y 3
- 2 y 2

$\frac{32\,KB}{4\,Bloques} = 8\,KB$ por Bloque (cada SM tiene 16 KB)

⇓

Solo puedo hacer 2 y 2

```
int valor = 0;
for (int i = 0; i < width; i++) {
    int primeraMatriz = a[threadIdx.y * width + i];
    int segundaMatriz = b[i * width + threadIdx.x];
    valor += primeraMatriz * segundaMatriz;
}
c[threadIdx.y * width + threadIdx.x] = valor;
```




```
int Row = BlockIdx.y * Tile_width + threadIdx.y;
int Col = BlockIdx.x * Tile_width + threadIdx.x;

int valor = 0
for (int i = 0; i < width; i++) {
    valor += a[Row * width + i] * b[i * width + Col];
}
c[Row * width + Col] = valor;

dim3 dimGrid (width / tile_width, width / tile_width);
dim3 dimBlock (tile_width, tile_width);

<<< dimGrid, dimBlock >>> ( . . . );
```