UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores
CUDA – Sesión 05 – MultiGPU

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

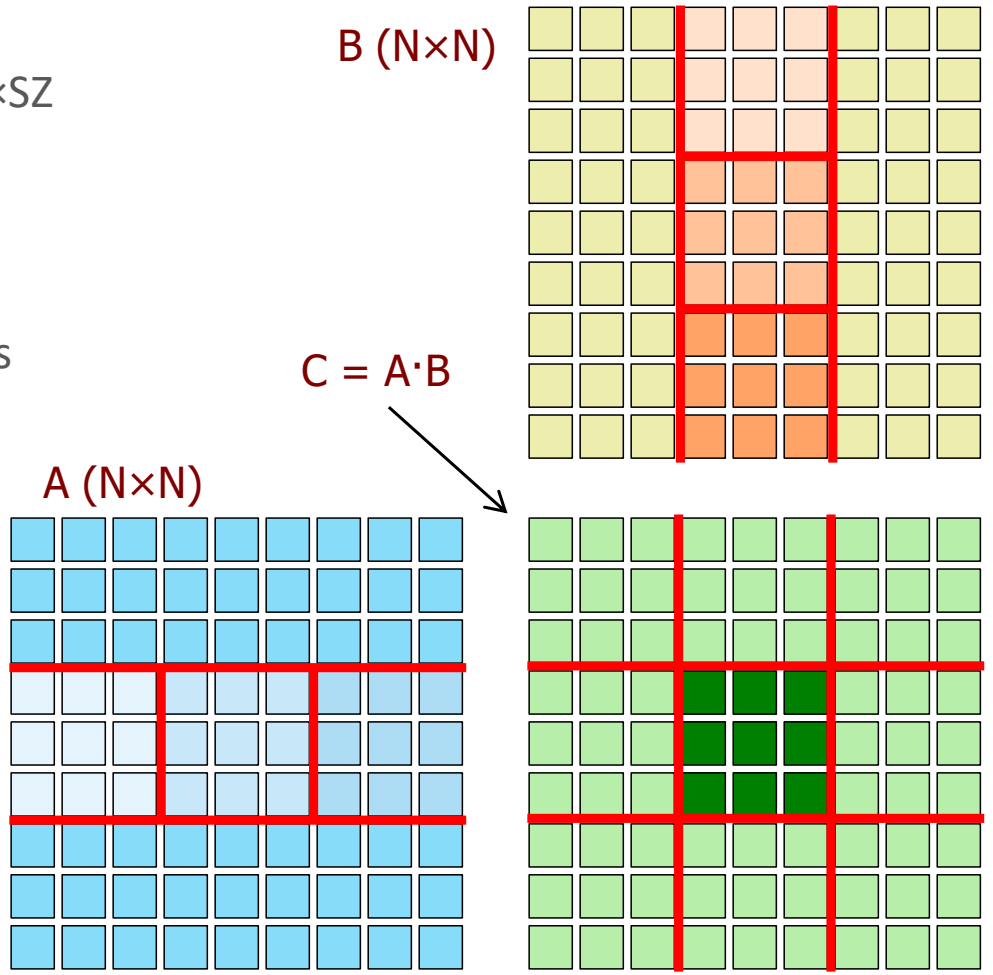
Universitat Politècnica de Catalunya

Producto de Matrices: kernel10 de la sesión 04

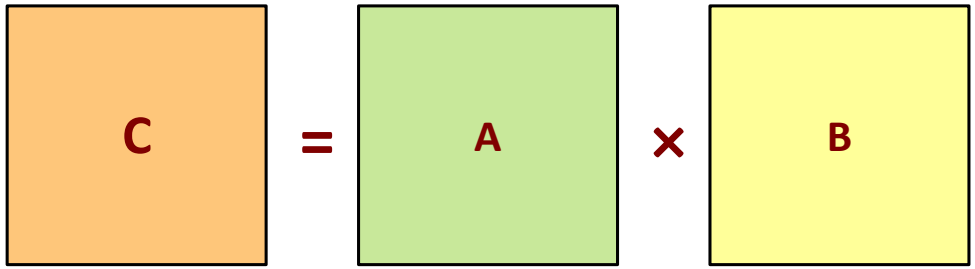
C = A · B (tamaño N × N)

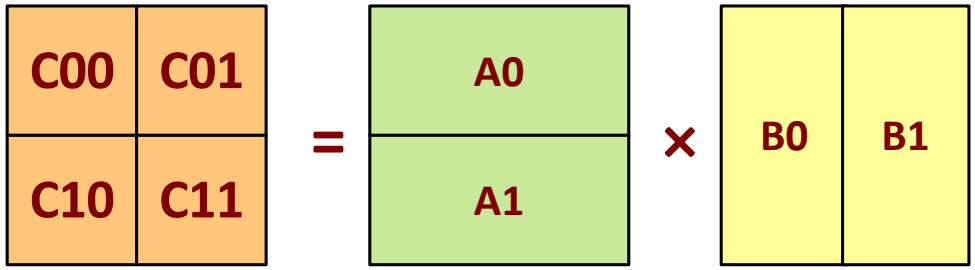
- Cada block thread calcula un bloque de datos de SZ×SZ elementos de la matriz C.
  - A es leída N/SZ veces desde la memoria global.
  - B es leída N/SZ veces desde la memoria global.
- El valor de SZ es importante, para asegurarse que las submatrices quepan en la memoria compartida.

Usaremos M×M para ilustrar el uso de varias GPUs



Producto de Matrices en varias GPUs





- 4 GPUs
- Cada GPU calcula una submatriz Cij = Ai · Bj
- En cada GPU se cargarán los datos necesarios.
- Supondremos que el coste de montar las submatrices es nulo.

Simplificaciones:
- Matrices cuadradas
- N×N
- $N = 2^p$

Usando varias GPUs

```
. . .
cudaGetDeviceCount(&count);
for (dev = 0; dev < count; dev++) {
  . . .
  cudaSetDevice(dev);
  kernel<<< . . . >>>(. . .);
  . . .
}
. . .
```

Averiguamos cuántas GPUs hay en el sistema

Lo que viene a continuación se ejecuta en la GPU "dev"

Usar varias GPUs es MUY SIMPLE

Producto de Matrices en 4 GPUs

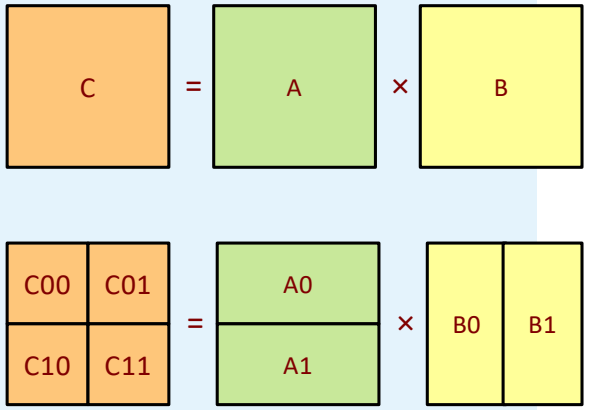
```
// Obtiene Memoria en el host: hA0, hA1, hB0, hB1, hC00, hC01, hC10 y hC11
// Obtener Memoria en cada device: dA0a, dB0a, dC00, dA0b, ...

// Código de cada device
cudaSetDevice(0);
cudaMemcpyAsync(dA0a, hA0, numBytesA, cudaMemcpyHostToDevice);
cudaMemcpyAsync(dB0a, hB0, numBytesB, cudaMemcpyHostToDevice);
KernelMM<<<dimGrid, dimBlock>>>(N/2, N/2, N, dA0a, dB0a, dC00);
cudaMemcpyAsync(hC00, dC00, numBytesC, cudaMemcpyDeviceToHost);

cudaSetDevice(1); // FALTA EL MOVIMIENTO DE DATOS
KernelMM<<<dimGrid, dimBlock>>>(N/2, N/2, N, dA0b, dB1a, dC01);

cudaSetDevice(2); // FALTA EL MOVIMIENTO DE DATOS
KernelMM<<<dimGrid, dimBlock>>>(N/2, N/2, N, dA1a, dB0b, dC10);

cudaSetDevice(3); // FALTA EL MOVIMIENTO DE DATOS
KernelMM<<<dimGrid, dimBlock>>>(N/2, N/2, N, dA1b, dB1b, dC11);
```



Tomando Tiempos

```
cudaEventCreate(&E0); cudaEventRecord(E0, 0);
. . .
//
// CÓDIGO a EVALUAR
//
. . .
cudaEventCreate(&E3); cudaEventRecord(E3, 0);
cudaEventSynchronize(E3);
cudaEventElapsedTime(&TiempoTotal, E0, E3);
```

- Cuando trabajamos con varias GPUs, obtener el tiempo de ejecución no es tan sencillo.
- Usaremos eventos. Pero,
  - los eventos son “propiedad” de la GPU dónde se crean
  - desde otras GPUs sólo podemos sincronizarnos con esos eventos.

Tomando Tiempos

```
cudaSetDevice(0); cudaEventCreate(&E0); cudaEventRecord(E0, 0);
// MxM GPU 0
...
cudaSetDevice(1);
// MxM GPU 1
cudaEventCreate(&X1); cudaEventRecord(X1, 0);
...
cudaSetDevice(2);
// MxM GPU 2
cudaEventCreate(&X2); cudaEventRecord(X2, 0);
...
cudaSetDevice(3);
// MxM GPU 3
cudaEventCreate(&X3); cudaEventRecord(X3, 0);
...
cudaSetDevice(0);
cudaEventSynchronize(X1); cudaEventSynchronize(X2); cudaEventSynchronize(X3);
cudaEventCreate(&E3); cudaEventRecord(E3, 0); cudaEventSynchronize(E3);
cudaEventElapsedTime(&TiempoTotal,  E0, E3);
```

Resultados [RTX 3080]

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 16x16 (256)
nBlocks: 256x256 (65536)
Usando Pinned Memory.
Tiempo Global: 135.449432 milseg
Rendimiento Global: 8117.51 GFLOPs

NO TEST
```

Datos obtenidos de la ejecución directa de 1 sola GPU
Rendimiento Kernel: 2.427,85 GFLOPs

Comparando 1GPU – 4 GPUs [RTX 3080]

N	Kernel	Global	Global
	1 GPU		4 GPUs
512	1.777,3 GFLOPs	291,5 GFLOPs	1.299,6 GFLOPs
1.024	2.284,8 GFLOPs	825,9 GFLOPs	2.498,0 GFLOPs
2.048	2.411,7 GFLOPs	1.513,0 GFLOPs	4.304,8 GFLOPs
4.096	2.426,4 GFLOPs	1.842,9 GFLOPs	6.249,7 GFLOPs
8.192	2.589,0 GFLOPs	2.353,6 GFLOPs	8.136,6 GFLOPs

Kernel10 sesión 4
con pinned memory

Resultados [K40c]

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 32x32 (1024)
nBlocks: 128x128 (16384)
Usando Pinned Memory
Tiempo Global: 989.752380 milseg
Rendimiento Global: 1110.90 GFLOPs

NO TEST
```

1 Kernel: 372,59 GFLOPs
4 Kernels: 1.493,76 GFLOPs

```
==62083== NVPROF is profiling process 62083, command: ./kernel4GPUs.exe 8192 N
==62083== Profiling application: ./kernel4GPUs.exe 8192 N
==62083== Profiling result:Time(%)
            Time     Calls       Avg       Min       Max  Name
 88.31%  2.92582s         4  731.45ms  726.42ms  736.07ms  KernelMM(. . .)
  8.54%  282.91ms         4  70.728ms  6.3554ms  226.61ms  [CUDA memcpy DtoH]
  3.15%  104.41ms         8  13.051ms  12.712ms  13.670ms  [CUDA memcpy HtoD]
```

Comparando 1GPU – 4 GPUs [K40c]

N	Kernel	Global	Global
	1 GPU		4 GPUs
512	341,76 GFLOPs	60,21 GFLOPs	532,10 GFLOPs
1.024	370,24 GFLOPs	138,26 GFLOPs	936,35 GFLOPs
2.048	372,76 GFLOPs	209,53 GFLOPs	1.218,68 GFLOPs
4.096	378,14 GFLOPs	257,34 GFLOPs	1.140,60 GFLOPs
8.192	375,31 GFLOPs	311,38 GFLOPs	1.113,51 GFLOPs

Kernel10 sesión 4
con pinned memory

cudaMemcpyAsync vs cudaMemcpy

```
cudaSetDevice(0);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(1);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(2);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(3);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
cudaMemcpy(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(0); cudaMemcpy(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(1); cudaMemcpy(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(2); cudaMemcpy(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(3); cudaMemcpy(,,, cudaMemcpyDeviceToHost);
```

Transmisión datos: CPU → GPU

Invocación del kernel

Transmisión del resultado: GPU → CPU

cudaMemcpyAsync vs cudaMemcpy

```
cudaSetDevice(0);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(1);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(2);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(3);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
cudaMemcpyAsync(,,, cudaMemcpyHostToDevice);
KernelMM<<<,>>>(,,,,,);
cudaSetDevice(0); cudaMemcpyAsync(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(1); cudaMemcpyAsync(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(2); cudaMemcpyAsync(,,, cudaMemcpyDeviceToHost);
cudaSetDevice(3); cudaMemcpyAsync(,,, cudaMemcpyDeviceToHost);
```

Transmisión datos: CPU → GPU

Invocación del kernel

Transmisión del resultado: GPU → CPU

cudaMemcpyAsync vs cudaMemcpy

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 16x16 (256)
nBlocks: 256x256 (65536)
Usando Pinned Memory.
Tiempo Global: 136.161255 milseg
Rendimiento Global: 8075.07 GFLOPs
NO TEST
```

cudaMemcpyAsync

¿Por qué?

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 16x16 (256)
nBlocks: 256x256 (65536)
Usando Pinned Memory.
Tiempo Global: 164.139709 milseg
Rendimiento Global: 6698.63 GFLOPs
NO TEST
```

cudaMemcpy

cudaMemcpyAsync

```
nsys nvprof --print-gpu-trace ./kernel4GPUs.exe 8192 N
```

```
CUDA Kernel & Memory Operations Trace:

 Start(sec)  Duration(nsec)  CorrId      Grid         Block    R/T  StcSMem    Bytes      Thru(MB/s)  SrcMemKd  DstMemKd           Device          Ctx  Strm      Name
 ----------  --------------  ------  ------------  ---------  ----  ------  -----------  ----------  --------  -----------  --------------------  ---  ----  ------------------
   4.707090      12,047,894     444                                        134,217,728  11,140.348  Pinned    Device       GeForce RTX 3080 (0)    1     7  [CUDA memcpy HtoD]
   4.707178      11,332,882     448                                        134,217,728  11,843.212  Pinned    Device       GeForce RTX 3080 (1)    2    17  [CUDA memcpy HtoD]
   4.707221      13,129,833     452                                        134,217,728  10,222.348  Pinned    Device       GeForce RTX 3080 (2)    3    27  [CUDA memcpy HtoD]
   4.707263      13,117,153     456                                        134,217,728  10,232.230  Pinned    Device       GeForce RTX 3080 (3)    4    37  [CUDA memcpy HtoD]
   4.718514       5,466,712     449                                        134,217,728  24,551.820  Pinned    Device       GeForce RTX 3080 (1)    2    17  [CUDA memcpy HtoD]
   4.719140       5,313,627     445                                        134,217,728  25,259.155  Pinned    Device       GeForce RTX 3080 (0)    1     7  [CUDA memcpy HtoD]
   4.720352       8,761,190     453                                        134,217,728  15,319.577  Pinned    Device       GeForce RTX 3080 (2)    3    27  [CUDA memcpy HtoD]
   4.720383       8,749,324     457                                        134,217,728  15,340.354  Pinned    Device       GeForce RTX 3080 (3)    4    37  [CUDA memcpy HtoD]
   4.723990     110,823,731     450  (256,256,1)  (16,16,1)   40    2,048                                                GeForce RTX 3080 (1)    2    17  KernelMM(...)
   4.724459     113,234,172     446  (256,256,1)  (16,16,1)   40    2,048                                                GeForce RTX 3080 (0)    1     7  KernelMM(...)
   4.729123     113,203,596     454  (256,256,1)  (16,16,1)   40    2,048                                                GeForce RTX 3080 (2)    3    27  KernelMM(...)
   4.729138     109,324,164     458  (256,256,1)  (16,16,1)   40    2,048                                                GeForce RTX 3080 (3)    4    37  KernelMM(...)
   4.834815       2,543,723     462                                         67,108,864  26,382.143  Device    Pinned       GeForce RTX 3080 (1)    2    17  [CUDA memcpy DtoH]
   4.837696       2,544,318     460                                         67,108,864  26,375.973  Device    Pinned       GeForce RTX 3080 (0)    1     7  [CUDA memcpy DtoH]
   4.838470       2,544,237     468                                         67,108,864  26,376.813  Device    Pinned       GeForce RTX 3080 (3)    4    37  [CUDA memcpy DtoH]
   4.842329       2,544,322     465                                         67,108,864  26,375.932  Device    Pinned       GeForce RTX 3080 (2)    3    27  [CUDA memcpy DtoH]
```

cudaMemcpy

```
nsys nvprof --print-gpu-trace ./kernel4GPUs.exe 8192 N
```

```
CUDA Kernel & Memory Operations Trace:

 Start(sec)  Duration(nsec)  CorrId      Grid        Block    R/T  StcSMem    Bytes      Thru(MB/s)  SrcMemKd  DstMemKd          Device          Ctx  Strm      Name
 ----------  --------------  ------  ------------  ---------  ----  ------  -----------  ----------  -------  -----------  --------------------  ---  ----  ------------------
   4.640523       9,348,634     444                                        134,217,728  14,356.935  Pinned   Device       GeForce RTX 3080 (0)    1     7  [CUDA memcpy HtoD]
   4.649886       5,289,085     445                                        134,217,728  25,376.361  Pinned   Device       GeForce RTX 3080 (0)    1     7  [CUDA memcpy HtoD]
   4.655223     113,232,784     446   (256,256,1)  (16,16,1)   40    2,048                                              GeForce RTX 3080 (0)    1     7  KernelMM(...)
   4.655263       5,295,512     448                                        134,217,728  25,345.562  Pinned   Device       GeForce RTX 3080 (1)    2    17  [CUDA memcpy HtoD]
   4.660575       5,295,096     449                                        134,217,728  25,347.553  Pinned   Device       GeForce RTX 3080 (1)    2    17  [CUDA memcpy HtoD]
   4.665896     112,984,203     450   (256,256,1)  (16,16,1)   40    2,048                                              GeForce RTX 3080 (1)    2    17  KernelMM(...)
   4.665926       5,300,612     452                                        134,217,728  25,321.176  Pinned   Device       GeForce RTX 3080 (2)    3    27  [CUDA memcpy HtoD]
   4.671241       5,297,252     453                                        134,217,728  25,337.237  Pinned   Device       GeForce RTX 3080 (2)    3    27  [CUDA memcpy HtoD]
   4.676564     113,205,208     454   (256,256,1)  (16,16,1)   40    2,048                                              GeForce RTX 3080 (2)    3    27  KernelMM(...)
   4.676576       5,300,251     456                                        134,217,728  25,322.900  Pinned   Device       GeForce RTX 3080 (3)    4    37  [CUDA memcpy HtoD]
   4.681885       5,299,834     457                                        134,217,728  25,324.893  Pinned   Device       GeForce RTX 3080 (3)    4    37  [CUDA memcpy HtoD]
   4.687202     111,401,492     458   (256,256,1)  (16,16,1)   40    2,048                                              GeForce RTX 3080 (3)    4    37  KernelMM(...)
   4.768459       2,546,846     460                                         67,108,864  26,349.793  Device   Pinned       GeForce RTX 3080 (0)    1     7  [CUDA memcpy DtoH]
   4.778888       2,544,140     462                                         67,108,864  26,377.819  Device   Pinned       GeForce RTX 3080 (1)    2    17  [CUDA memcpy DtoH]
   4.789777       2,543,938     465                                         67,108,864  26,379.913  Device   Pinned       GeForce RTX 3080 (2)    3    27  [CUDA memcpy DtoH]
   4.798611       2,543,853     468                                         67,108,864  26,380.795  Device   Pinned       GeForce RTX 3080 (3)    4    37  [CUDA memcpy DtoH]
```

Rendimiento con diferentes kernels (kernel vs global)

	N	kernel00 (16×16)	kernel10 (16×16)	M×M Cut (8×8)	M×M Cut (16×16)
1 GPU	1.024	1.784,7 GFLOPs	2.660,7 GFLOPs	11.911,4 GFLOPs	10.887,2 GFLOPs
	2.048	1.817,6 GFLOPs	2.405,0 GFLOPs	12.721,1 GFLOPs	14.897,4 GFLOPs
	4.096	1.832,7 GFLOPs	2.424,9 GFLOPs	13.712,9 GFLOPs	15.638,3 GFLOPs
	8.192	1.935,8 GFLOPs	2.554,6 GFLOPs	13.865,7 GFLOPs	16.710,6 GFLOPs
4 GPUs	1.024	3.010,7 GFLOPs x1,69	3.382,2 GFLOPs x1,27	4.169,8 GFLOPs x0,35	3.814,3 GFLOPs x0,35
	2.048	4.597,6 GFLOPs x2,53	5.438,0 GFLOPs x2,26	9.955,9 GFLOPs x0,78	10.187,9 GFLOPs x0,68
	4.096	5.730,8 GFLOPs x3,13	7.091,5 GFLOPs x2,92	17.551,5 GFLOPs x1,28	18.620,7 GFLOPs x1,19
	8.192	6.455,7 GFLOPs x3,33	8.317,8 GFLOPs x3,26	27.087,1 GFLOPs x1,95	29.176,7 GFLOPs x1,75

Rendimiento con diferentes kernels (global vs global)

	N	kernel00 (16×16)	kernel10 (16×16)	M×M Cut (8×8)	M×M Cut (16×16)
1 GPU	1.024	1.238.6 GFLOPs	1.469,5 GFLOPs	2.904,5 GFLOPs	2.947,0 GFLOPs
	2.048	1.503.0 GFLOPs	1.879,4 GFLOPs	5.097,3 GFLOPs	5.437,7 GFLOPs
	4.096	1.658.9 GFLOPs	2.126,2 GFLOPs	7.616,4 GFLOPs	8.242,9 GFLOPs
	8.192	1.859,4 GFLOPs	2.412,6 GFLOPs	9.843,4 GFLOPs	11.295,6 GFLOPs
4 GPUs	1.024	3.010,7 GFLOPs x2,43	3.382,2 GFLOPs x2,30	4.169,8 GFLOPs x1,44	3.814,3 GFLOPs x1,29
	2.048	4.597,6 GFLOPs x3,06	5.438,0 GFLOPs x2,89	9.955,9 GFLOPs x1,95	10.187,9 GFLOPs x1,87
	4.096	5.730,8 GFLOPs x3,45	7.091,5 GFLOPs x3,34	17.551,5 GFLOPs x2,30	18.620,7 GFLOPs x2,26
	8.192	6.455,7 GFLOPs x3,47	8.317,8 GFLOPs x3,45	27.087,1 GFLOPs x2,75	29.176,7 GFLOPs x2,58

CUDA samples

- Ejemplos disponibles en cualquier instalación de CUDA.
  - Antes venían con la distribución de CUDA
  - Ahora están disponibles en: `https://github.com/NVIDIA/cuda-samples`
  - Hay que modificar los makefiles para que funcione
- Ejemplos de todo tipo. Información disponible en:
  http://docs.nvidia.com/cuda/cuda-samples/

simpleStreams

This sample uses CUDA streams to overlap kernel executions with memory copies between the host and a GPU device. This sample uses a new CUDA 4.0 feature that supports pinning of generic host memory. Requires Compute Capability 2.0 or higher.

Supported SM Architecture	SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
CUDA API	cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync
Key Concepts	Asynchronous Data Transfers, CUDA Streams and Events
Supported OSes	Linux, Windows, OS X

Documentación CUDA

- Documentación disponible en: https://docs.nvidia.com/cuda/

- Documentación disponible (¡ACTUALIZADA!):
  - CUDA C BEST PRACTICES GUIDE
  - CUDA C PROGRAMMING GUIDE
  - CUDA COMPILER DRIVER NVCC
  - NVIDIA CUDA GETTING STARTED GUIDE FOR LINUX
  - CUDA SAMPLES
  - PRECISION AND PERFORMANCE:FLOATING POINT AND IEEE 754 COMPLIANCE FOR NVIDIA GPUS
  - TUNING CUDA APPLICATIONS FOR KEPLER
  - TUNING CUDA APPLICATIONS FOR MAXWELL
  - . . .
- La documentación se actualiza para cada nueva versión del compilador.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores
CUDA – Sesión 05 – MultiGPU

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya