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

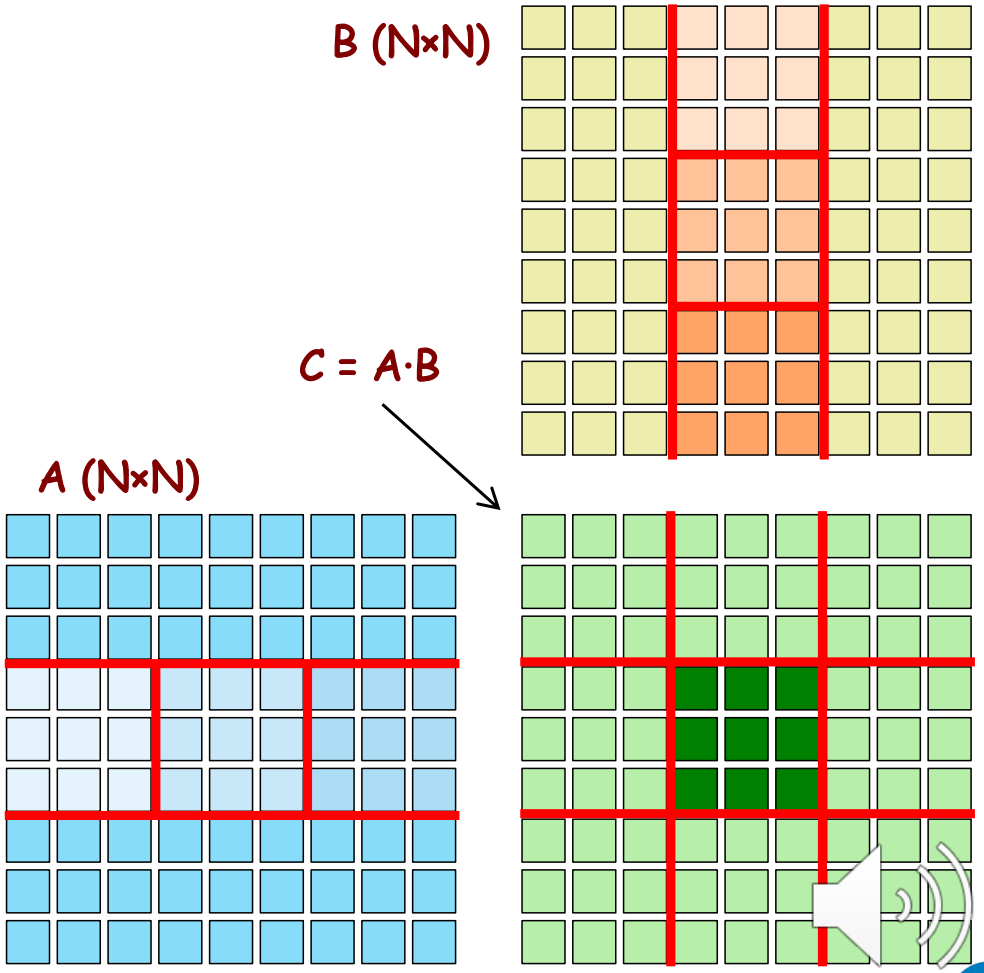
Universitat Politècnica de Catalunya

Producto de Matrices: kernel10 de la sesión 04

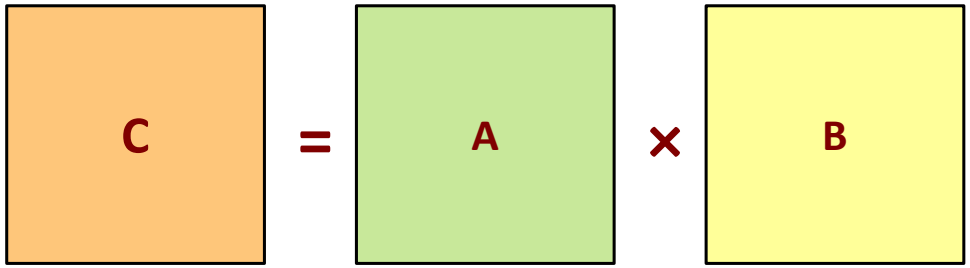
C = A · B (tamaño N × N)

- 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.

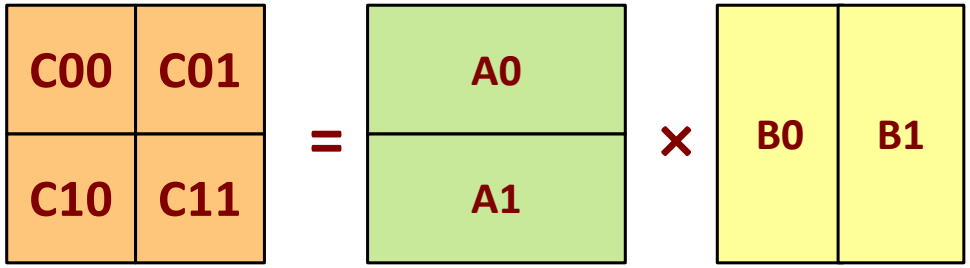
Usaremos M×M para ilustrar el uso de varias GPUs



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);
// Código GPU 0

// Código GPU 1
cudaSetDevice(1); cudaEventCreate(&X1); cudaEventRecord(X1, 0);

// Código GPU 2
cudaSetDevice(2); cudaEventCreate(&X2); cudaEventRecord(X2, 0);

// Código GPU 3
cudaSetDevice(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

```
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

N	1 GPUs		4 GPUs
	Kernel	Global	Global
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);
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 resultado: GPU → CPU

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 resultado: GPU → CPU

cudaMemcpyAsync vs cudaMemcpy

```
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
```

cudaMemcpyAsync

¿Por qué?

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 32x32 (1024)
nBlocks: 128x128 (16384)
Usando Pinned Memory.
Tiempo Global: 839.519287 milseg
Rendimiento Global: 1309.69 GFLOPS
NO TEST
```

cudaMemcpy

cudaMemcpyAsync

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

```
==63272== NVPROF is profiling process 63272, command: ./kernel4GPUs.exe 8192 N
==63272== Profiling application: ./kernel4GPUs.exe 8192 N
==63272== Profiling result:
   Start  Duration   Grid Size Block Size Regs SSMem DSMem   Size  Throughput         Device  Context  Stream  Name
5.73511s  12.712ms           -          -    -     -     -  128MB  9.8331GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.73522s  12.711ms           -          -    -     -     -  128MB  9.8338GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.73529s  13.643ms           -          -    -     -     -  128MB  9.1623GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.73534s  13.676ms           -          -    -     -     -  128MB  9.1402GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.74782s  12.731ms           -          -    -     -     -  128MB  9.8188GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.74793s  12.735ms           -          -    -     -     -  128MB  9.8158GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.74893s  13.132ms           -          -    -     -     -  128MB  9.5185GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.74902s  13.136ms           -          -    -     -     -  128MB  9.5160GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.76057s  731.82ms  (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (0)        1       7  KernelMM() [464]
5.76067s  726.38ms  (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (1)        2      17  KernelMM() [468]
5.76208s  735.87ms  (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (2)        3      27  KernelMM() [472]
5.76217s  731.34ms  (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (3)        4      37  KernelMM() [476]
6.48706s  6.3557ms           -          -    -     -     -   64MB  9.8337GB/s  Tesla K40c (1)        2      17  [CUDA memcpy DtoH]
6.49240s  6.3609ms           -          -    -     -     -   64MB  9.8256GB/s  Tesla K40c (0)        1       7  [CUDA memcpy DtoH]
6.49352s  39.964ms           -          -    -     -     -   64MB  1.5639GB/s  Tesla K40c (3)        4      37  [CUDA memcpy DtoH]
6.49795s  226.20ms           -          -    -     -     -   64MB  282.93MB/s  Tesla K40c (2)        3      27  [CUDA memcpy DtoH]
```

Aquí està el problema.

cudaMemcpy

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

```
==63746== NVPROF is profiling process 63746, command: ./kernel4GPUs.exe 8192 N
==63746== Profiling application: ./kernel4GPUs.exe 8192 N
==63746== Profiling result:
   Start  Duration   Grid Size Block Size Regs SSMem DSMem   Size  Throughput         Device  Context  Stream  Name
5.74225s  12.711ms           -          -    -     -     -  128MB  9.8343GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.75499s  12.709ms           -          -    -     -     -  128MB  9.8358GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.76777s  731.49ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (0)        1       7  KernelMM() [464]
5.76781s  12.710ms           -          -    -     -     -  128MB  9.8346GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.78055s  12.708ms           -          -    -     -     -  128MB  9.8361GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.79330s  738.50ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (1)        2      17  KernelMM() [468]
5.79333s  12.711ms           -          -    -     -     -  128MB  9.8341GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.80606s  12.709ms           -          -    -     -     -  128MB  9.8359GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.81882s  735.56ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (2)        3      27  KernelMM() [472]
5.81883s  12.711ms           -          -    -     -     -  128MB  9.8344GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.83157s  12.709ms           -          -    -     -     -  128MB  9.8356GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.84432s  731.43ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (3)        4      37  KernelMM() [476]
6.49926s  6.3560ms           -          -    -     -     -   64MB  9.8332GB/s  Tesla K40c (0)        1       7  [CUDA memcpy DtoH]
6.53180s  6.3561ms           -          -    -     -     -   64MB  9.8331GB/s  Tesla K40c (1)        2      17  [CUDA memcpy DtoH]
6.55438s  6.3655ms           -          -    -     -     -   64MB  9.8185GB/s  Tesla K40c (2)        3      27  [CUDA memcpy DtoH]
6.57575s  6.3674ms           -          -    -     -     -   64MB  9.8156GB/s  Tesla K40c (3)        4      37  [CUDA memcpy DtoH]
```

Aquí no hay ningún problema

cudaMemcpyAsync + cudaMemcpy

```
KERNEL MultiGPU - Producto Matrices
Dimensiones: 8192x8192
nThreads: 32x32 (1024)
nBlocks: 128x128 (16384)
Usando Pinned Memory
Tiempo Global: 783.705688 milseg
Rendimiento Global: 1402.96 GFLOPS

NO TEST
```

¿Cómo explicarlo?

cudaMemcpyAsync + cudaMemcpy

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

```
==63746== NVPROF is profiling process 63746, command: ./kernel4GPUs.exe 8192 N
==63746== Profiling application: ./kernel4GPUs.exe 8192 N
==63746== Profiling result:
   Start  Duration   Grid Size Block Size Regs SSMem DSMem   Size  Throughput          Device  Context  Stream  Name
5.73524s  12.712ms           -          -    -     -     -  128MB  9.8331GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.73536s  12.713ms           -          -    -     -     -  128MB  9.8327GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.73544s  13.674ms           -          -    -     -     -  128MB  9.1415GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.73550s  13.717ms           -          -    -     -     -  128MB  9.1125GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.74796s  12.721ms           -          -    -     -     -  128MB  9.8260GB/s  Tesla K40c (0)        1       7  [CUDA memcpy HtoD]
5.74808s  12.719ms           -          -    -     -     -  128MB  9.8281GB/s  Tesla K40c (1)        2      17  [CUDA memcpy HtoD]
5.74912s  13.119ms           -          -    -     -     -  128MB  9.5281GB/s  Tesla K40c (2)        3      27  [CUDA memcpy HtoD]
5.74922s  13.070ms           -          -    -     -     -  128MB  9.5639GB/s  Tesla K40c (3)        4      37  [CUDA memcpy HtoD]
5.76069s  731.73ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (0)        1       7  KernelMM() [464]
5.76080s  738.92ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (1)        2      17  KernelMM() [468]
5.76225s  735.95ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (2)        3      27  KernelMM() [472]
5.76230s  731.42ms (128 128 1)  (32 32 1)   26  8.0KB    0B     -           -  Tesla K40c (3)        4      37  KernelMM() [476]
6.49243s  6.3564ms           -          -    -     -     -   64MB  9.8325GB/s  Tesla K40c (0)        1       7  [CUDA memcpy DtoH]
6.49978s  6.3558ms           -          -    -     -     -   64MB  9.8336GB/s  Tesla K40c (1)        2      17  [CUDA memcpy DtoH]
6.50612s  6.3661ms           -          -    -     -     -   64MB  9.8177GB/s  Tesla K40c (2)        3      27  [CUDA memcpy DtoH]
6.51252s  6.3675ms           -          -    -     -     -   64MB  9.8155GB/s  Tesla K40c (3)        4      37  [CUDA memcpy DtoH]
```

El problema ha desaparecido

CUDA samples

- **Ejemplos disponibles en la instalación de CUDA.**
  - Accesibles en boada en el directorio: **`/Soft/cuda/11.2.1/samples/`**
  - Para que funcionen los makefiles hay que replicar la estructura de directorios
- **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

- **Toda la documentación de CUDA está disponible en el site de NVIDIA:**

  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
  - . . .

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