UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores
CUDA – Sesión 06 – Streams

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

CUDA Streams

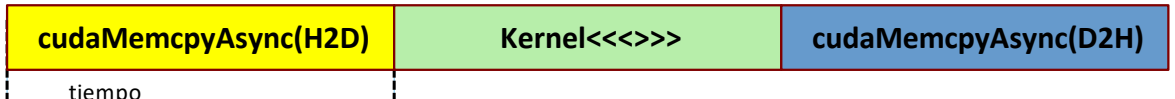
- Objetivo: ejecutar varias operaciones CUDA simultáneamente:
  - CUDA kernel<<<>>>
  - cudaMemcpyAsync(HostToDevice)
  - cudaMemcpyAsync(DeviceToHost)
  - Cálculos en la CPU

- Stream
  - Secuencia de operaciones que se ejecutan en la GPU en el orden en que se lanzan.

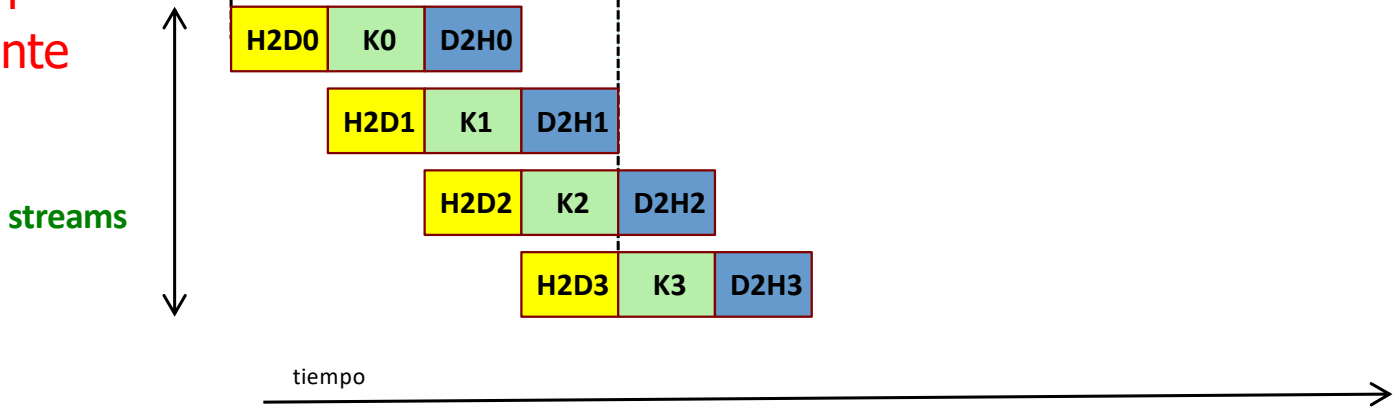
- Las operaciones CUDA de diferentes streams pueden ejecutarse concurrentemente.
- Las operaciones CUDA de diferentes streams pueden entrelazarse.

CUDA Streams

Ejecución secuencial



Ejecución concurrente



Punto de partida: stream00

```
__global__ void Kernel00(int N, float a, float b, float *A, float *B, float *C) {

  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j;
  if (i<N) {
      C[i] = a*A[i] + b*B[i];
      for (j=0; j<DUMMY; j++)
        C[i] += a*A[i] + b*B[i];
  }
}
```

No hacemos ningún cálculo útil.
DUMMY sirve para incrementar el peso del cálculo.
DUMMY == 100

Punto de partida: stream00

```
// Obtiene Memoria [pinned] en el host
// Inicializa las matrices
// Obtener Memoria en el device

// Copiar datos desde el host en el device
cudaMemcpy(d_A, h_A, numBytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, numBytes, cudaMemcpyHostToDevice);

// Ejecutar el kernel
Kernel00<<<nBlocks, nThreads>>>(N, a, b, d_A, d_B, d_C);

// Obtener el resultado desde el host
cudaMemcpy(h_C, d_C, numBytes, cudaMemcpyDeviceToHost);

 // Liberar Memoria del device
```

Código a evaluar

Ejecución:
`./kernel00.exe 10000 Y`

Punto de partida: stream00

```
KERNEL 00: NO - Streams
Dimension Problema: 10240000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<10000, 1024>>> (10240000)
nKernels: 1
Usando Pinned Memory
Tiempo Global (00): 6.935104 milseg
Rendimiento Global (00): 595.05 GFLOPS
Tiempo Kernel (00): 2.111648 milseg
Tiempo HtoD (00): 3.257184 milseg
Tiempo DtoH (00): 1.566272 milseg
TEST PASS
```

Tiempos de cálculo y comunicación “equivalentes”.

1ª Versión: stream01

```
numB4 = numBytes/4; N4 = N/4; nB = (N4+nThr-1)/nThr;

cudaMemcpyAsync(&d_A[0], &h_A[0], numB4, cudaMemcpyHostToDevice, str1);
cudaMemcpyAsync(&d_B[0], &h_B[0], numB4, cudaMemcpyHostToDevice, str1);
Kernel01<<<nBlocks, nThreads, 0, str1>>>(N4, a, b, &d_A[0], &d_B[0], &d_C[0]);
cudaMemcpyAsync(&h_C[0], &d_C[0], numB4, cudaMemcpyDeviceToHost, str1);

cudaMemcpyAsync(&d_A[N4], &h_A[N4], numB4, cudaMemcpyHostToDevice, str2);
cudaMemcpyAsync(&d_B[N4], &h_B[N4], numB4, cudaMemcpyHostToDevice, str2);
Kernel01<<<nBlocks, nThreads, 0, str2>>>(N4, a, b, &d_A[N4], &d_B[N4], &d_C[N4]);
cudaMemcpyAsync(&h_C[N4], &d_C[N4], numB4, cudaMemcpyDeviceToHost, str2);

cudaMemcpyAsync(&d_A[2*N4], &h_A[2*N4], numB4, cudaMemcpyHostToDevice, str3);
cudaMemcpyAsync(&d_B[2*N4], &h_B[2*N4], numB4, cudaMemcpyHostToDevice, str3);
Kernel01<<<nBlocks, nThreads, 0, str3>>>(N4, a, b, &d_A[2*N4], &d_B[2*N4], &d_C[2*N4]);
cudaMemcpyAsync(&h_C[2*N4], &d_C[2*N4], numB4, cudaMemcpyDeviceToHost, str3);

cudaMemcpyAsync(&d_A[3*N4], &h_A[3*N4], numB4, cudaMemcpyHostToDevice, str4);
cudaMemcpyAsync(&d_B[3*N4], &h_B[3*N4], numB4, cudaMemcpyHostToDevice, str4);
Kernel01<<<nBlocks, nThreads, 0, str4>>>(N4, a, b, &d_A[3*N4], &d_B[3*N4], &d_C[3*N4]);
cudaMemcpyAsync(&h_C[3*N4], &d_C[3*N4], numB4, cudaMemcpyDeviceToHost, str4);
```

1ª Versión: stream01

```
KERNEL 01: 4 Streams
Dimension Problema: 10240000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<2500, 1024>>> (2560000)
nKernels: 4
Usando Pinned Memory
Tiempo Global (01): 4.298752 milseg
Rendimiento Global (01): 959.98 GFLOPS
TEST PASS
```

```
KERNEL 00: NO - Streams
...
Rendimiento Global (00): 595.05 GFLOPS
...
TEST PASS
```

Speedup = 1,61

1ª Versión: stream01 Reordenado

¡RENDIMIENTO EQUIVALENTE!

```
numB4 = numBytes/4; N4 = N/4; nB = (N4+nThr-1)/nThr;

cudaMemcpyAsync(&d_A[0], &h_A[0], numB4, cudaMemcpyHostToDevice, str1);
cudaMemcpyAsync(&d_A[N4], &h_A[N4], numB4, cudaMemcpyHostToDevice, str2);
cudaMemcpyAsync(&d_A[2*N4], &h_A[2*N4], numB4, cudaMemcpyHostToDevice, str3);
cudaMemcpyAsync(&d_A[3*N4], &h_A[3*N4], numB4, cudaMemcpyHostToDevice, str4);

cudaMemcpyAsync(&d_B[0], &h_B[0], numB4, cudaMemcpyHostToDevice, str1);
cudaMemcpyAsync(&d_B[N4], &h_B[N4], numB4, cudaMemcpyHostToDevice, str2);
cudaMemcpyAsync(&d_B[2*N4], &h_B[2*N4], numB4, cudaMemcpyHostToDevice, str3);
cudaMemcpyAsync(&d_B[3*N4], &h_B[3*N4], numB4, cudaMemcpyHostToDevice, str4);

Kernel01<<<nBlocks, nThreads, 0, str1>>>(N4, a, b, &d_A[0], &d_B[0], &d_C[0]);
Kernel01<<<nBlocks, nThreads, 0, str2>>>(N4, a, b, &d_A[N4], &d_B[N4], &d_C[N4]);
Kernel01<<<nBlocks, nThreads, 0, str3>>>(N4, a, b, &d_A[2*N4], &d_B[2*N4], &d_C[2*N4]);
Kernel01<<<nBlocks, nThreads, 0, str4>>>(N4, a, b, &d_A[3*N4], &d_B[3*N4], &d_C[3*N4]);

cudaMemcpyAsync(&h_C[0], &d_C[0], numB4, cudaMemcpyDeviceToHost, str1);
cudaMemcpyAsync(&h_C[N4], &d_C[N4], numB4, cudaMemcpyDeviceToHost, str2);
cudaMemcpyAsync(&h_C[2*N4], &d_C[2*N4], numB4, cudaMemcpyDeviceToHost, str3);
cudaMemcpyAsync(&h_C[3*N4], &d_C[3*N4], numB4, cudaMemcpyDeviceToHost, str4);
```

3ª Versión: stream02

```
nK = 2500*1024;  nB = nK * sizeof(float);
nTh = 1024;      // número de Threads
nBl = nK / nTh;  // número de Blocks

for (str=0, punt=0; punt < N; str = (str+1)%nStreams, punt += nK) {
  cudaMemcpyAsync(&d_A[punt], &h_A[punt], nB, cudaMemcpyHostToDevice, stream[str]);
  cudaMemcpyAsync(&d_B[punt], &h_B[punt], nB, cudaMemcpyHostToDevice, stream[str]);
  Kernel02<<<nBl, nTh, 0, stream[str]>>>(nK, a, b, &d_A[punt], &d_B[punt], &d_C[punt]);
  cudaMemcpyAsync(&h_C[punt], &d_C[punt], nB, cudaMemcpyDeviceToHost, stream[str]);
}
```

```
KERNEL 02: Streams 2
Dimension Problema: 10240000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<1000, 1024>>> (1024000)
nKernels: 4
Usando Pinned Memory
Tiempo Global (02): 4.301824 milseg
Rendimiento Global (02): 959.30 GFLOPS
TEST PASS
```

Speedup = 1,61

¡NO MEJORAMOS NADA!

3ª Versión: stream02

```
nK = 1000*1024;  nB = nK * sizeof(float);
nTh = 1024;      // número de Threads
nBl = nK / nTh;  // número de Blocks

for (str=0, punt=0; punt < N; str = (str+1)%nStreams, punt += nK) {
  cudaMemcpyAsync(&d_A[punt], &h_A[punt], nB, cudaMemcpyHostToDevice, stream[str]);
  cudaMemcpyAsync(&d_B[punt], &h_B[punt], nB, cudaMemcpyHostToDevice, stream[str]);
  Kernel02<<<nBl, nTh, 0, stream[str]>>>(nK, a, b, &d_A[punt], &d_B[punt], &d_C[punt]);
  cudaMemcpyAsync(&h_C[punt], &d_C[punt], nB, cudaMemcpyDeviceToHost, stream[str]);
}
```

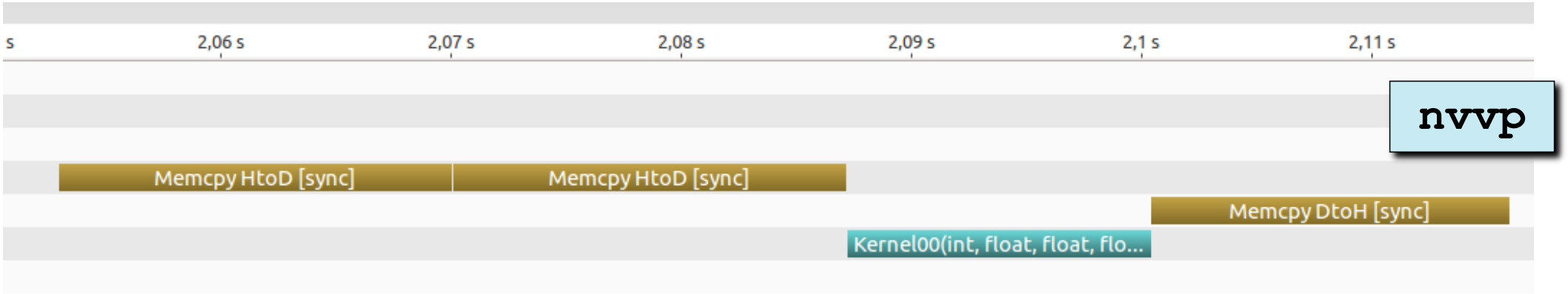
```
KERNEL 02: Streams 2
Dimension Problema: 10240000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<1000, 1024>>> (1024000)
nKernels: 10
Usando Pinned Memory
Tiempo Global (02): 3.836000 milseg
Rendimiento Global (02): 1075.79 GFLOPS
TEST PASS
```

Speedup = 1,81

Comparativa para N = 300.000 × 1024

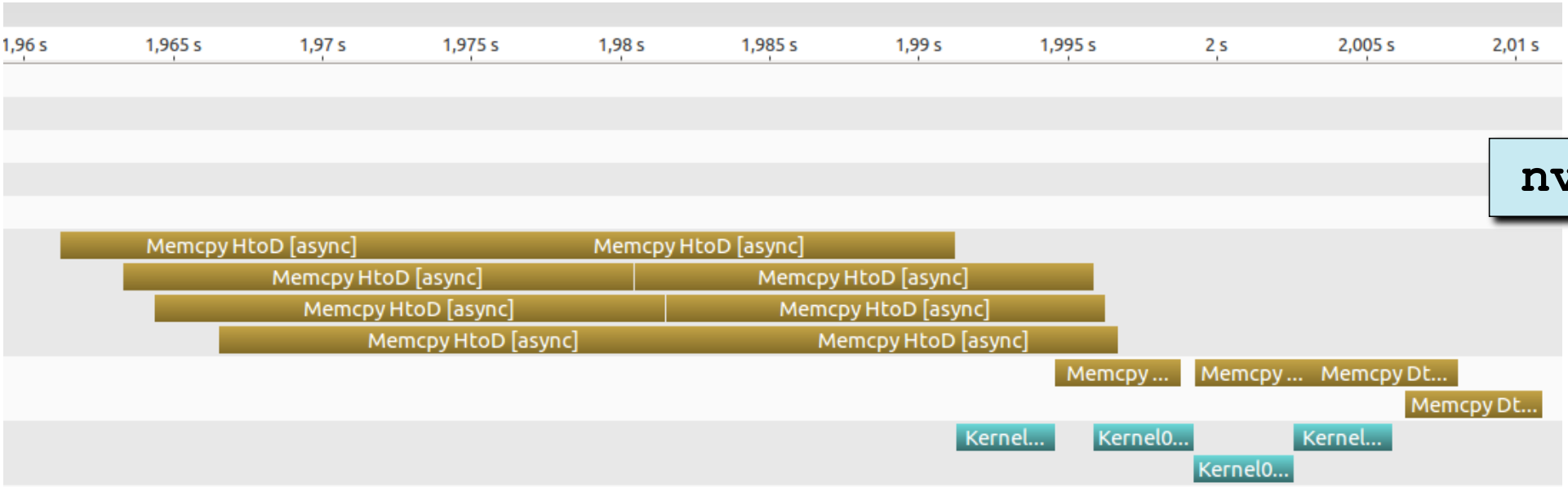
	STREAMS00	STREAMS01	STREAMS02
N	300.000×1024	300.000×1024	300.000×1024
#kernels ejecutados	1	4	120
nBlocks por kernel	300.000	75000	2500
nThreads por Block	1.024	1.024	1.024
Rendimiento	616,02 GFLOPs	984,12 GFLOPs	1.124,92 GFLOPs
Speedup	-	x1,60	x1,82

Sesion 06 en una GTX1080ti



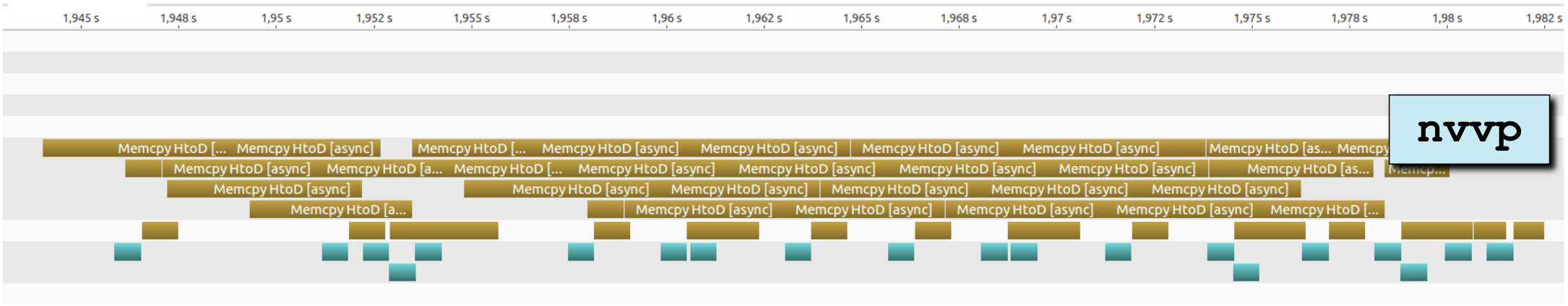

```
KERNEL 00: NO – Streams
Dimension Problema: 51200000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<50000, 1024>>> (51200000)
nKernels: 1
Usando Pinned Memory
Tiempo Global (00): 62.918270 milseg
Rendimiento Global (00): 100.09 GFLOPS
TEST PASS
```

Sesion 06 en una GTX1080ti




```
KERNEL 01: 4 Streams
Dimension Problema: 51200000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<12500, 1024>>> (12800000)
nKernels: 4
Usando Pinned Memory
Tiempo Global (01): 52.053600 milseg
Rendimiento Global (01): 120.98 GFLOPS
TEST PASS
```

Sesion 06 en una GTX1080ti




```
KERNEL 02: Streams 2
Dimension Problema: 51200000
Invocacion Kernel <<<nBlocks, nKernels>>> (N): <<<2500, 1024>>> (2560000)
nKernels: 20
Usando Pinned Memory
Tiempo Global (02): 38.490849 milseg
Rendimiento Global (02): 163.61 GFLOPS
TEST PASS
```

Proyecto TGA / Resto del curso

- Para acabar el proyecto tenemos 3+3 sesiones de Laboratorio.
  - Las 3 sesiones de teoría (pasan a laboratorio)
    - ✓ 5, 12, y 19 de diciembre en vuestro aula de laboratorio habitual
  - Y las 3 sesiones de laboratorio
    - ✓ 1, 15, y 22 de diciembre en vuestro aula de laboratorio habitual

- Distribución del examen: **13 de diciembre**.
- Fecha límite para entregar examen y proyecto: **17 de enero a las 10:00**.
- La entrega del examen y del proyecto se realizará a través del racó.
  - Abriremos una ”práctica” para cada entrega

Proyecto TGA

El proyecto os ha de costar unas 30-40 horas de trabajo (incluyendo las horas de laboratorio).

La entrega la haremos a través del racó . Además del informe, en la entrega del proyecto hay que incluir las versiones implementadas, los makefiles y los job.sh. [TODO EN 1 ZIP.]

El informe [en general] debe contener los siguientes apartados:

1. Portada (1 pág.). Nombre y apellidos de los integrantes del grupo y título del proyecto.
2. Definición del problema a resolver (1-4 pág.). Documentación utilizada para entender el problema y realizar el código secuencial. Si el código secuencial no lo habéis implementado vosotros indicad la fuente. Incluid referencias bibliográficas.
3. Implementaciones CUDA realizadas (versiones del código) y descripción de los aspectos más relevantes de cada una (principales diferencias respecto al resto de versiones implementadas). Indicad para cada versión el nombre del fichero que tiene el fichero fuente (1-4 págs.)
4. Análisis de rendimiento obtenido para cada implementación CUDA realizada (1-2 págs.). Incluid alguna medida de rendimiento para evaluar las mejoras introducidas (GFLOPS, ancho de banda, tiempo, …). Si se observa alguna inconsistencia en los resultados indicad posible causas. Se recomienda realizar varias ejecuciones por cada prueba y obtener una media. Recordad que el código secuencial debe ejecutarse en el mismo servidor que el código CUDA.

Examen TGA

Examen Tarjetas Gráficas y Aceleradores 2022-23-Q2

Antes de empezar el examen, leed atentamente los siguientes comentarios:

- Esto es un examen y los exámenes se hacen de forma **individual y sin ayudas externas**.
- La fecha límite para entregar el examen (y el proyecto) es el **miércoles 21 de junio de 2023 a las 12:00**.
- He montado una "práctica" en el racó para hacer la entrega del examen.
- El examen que entreguéis ha de ser un **PDF**.
- Hemos calculado que el tiempo necesario para realizar el examen son unas 20-30 horas de trabajo. No lo dejéis para el final.
- Teniendo en cuenta que tenéis mucho tiempo para hacer el examen, una buena redacción y ortografía se tendrá en cuenta en la evaluación.
- No os limitéis a la información que hay en las transparencias del curso, hay que buscar en otras fuentes.
- Además del examen tenéis que entregar un anexo, en el que, para cada pregunta, indiquéis las referencias, páginas web, etc. que habéis consultado durante el examen.
- El examen consta de:
  - ✓ 6 preguntas estándar (2 hojas por pregunta, aprox. 1000 palabras por pregunta).
  - ✓ 1 pregunta doble, es cómo un pequeño trabajo (4 hojas, aprox. 2000 palabras)
  - ✓ 1 cuestionario, que equivale a 1 pregunta doble.
  - ✓ Los que hicisteis la actividad práctica el 27 de marzo (servidor de TGA) podéis dejar de hacer 1 de las 6 preguntas estándar.

- No os olvidéis de poner vuestro nombre al inicio del examen.

PREGUNTAS ESTÁNDAR

1) Describe el pipeline gráfico tradicional.

Comentarios previos del Examen del pasado curso

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Sesion06 - Streams

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya