UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Sesion01

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

Entorno de Programación

- Usamos el servidor boada.ac.upc.edu (5 nodos)
- En boada-5 están instaladas las 4 GPUs (K40)
- Cuando entramos en boada, lo hacemos en boada-1
- Para editar / compilar / etc. usaremos boada-1
- Los programas tienen acceso exclusivo a las 4 GPUs.
- No se puede ejecutar de forma interactiva en boada-5
- Para acceder a las GPUS usaremos la cola cuda:
  - `sbatch job.sh`
- Algunos comandos útiles:
  - `scancel <job_id>`
  - `squeue`

Detalles de programación

```
// Obtener Memoria en el host
h_x = (float*) malloc(numBytes);
h_y = (float*) malloc(numBytes);
H_y = (float*) malloc(numBytes);

// Obtiene Memoria [pinned] en el host
//cudaMallocHost((float**)&h_x, numBytes);
//cudaMallocHost((float**)&h_y, numBytes);
//cudaMallocHost((float**)&H_y, numBytes);
```

Es de buen programador comprobar que los malloc han funcionado bien

```
 // Obtener Memoria en el host
h_x = (float*) malloc(numBytes);
if (h_x == NULL) ERROR&EXIT;
```

Gestión de errores en CUDA

```
// Obtener Memoria en el device
cudaMalloc((float**)&d_x, numBytes);
cudaMalloc((float**)&d_y, numBytes);
CheckCudaError((char *) "Obtener Memoria en el device", __LINE__);
```

Ponemos CheckCudaError después de cada llamada CUDA

```
void CheckCudaError(char sms[], int line) {
  cudaError_t error;
  error = cudaGetLastError();
  if (error) {
    printf("(ERROR) %s - %s in %s at line %d\n", sms,
           cudaGetErrorString(error), __FILE__, line);
    exit(EXIT_FAILURE);
  }
}
```

Estructura de un programa CUDA

```
// Obtener Memoria en el host

// Inicializar datos en el host

// Obtener memoria en el device

// Copiar datos del host en el device

// Ejecutar el kernel

// Copiar el resultado del device en el host

// Liberar espacio en el host y en el device
```

Código a Evaluar

Código a Evaluar

Cómo evaluamos un código: Tiempo de ejecución

```
. . .
  cudaEventRecord(start, 0);

// Copiar datos del host en el device

// Ejecutar el kernel

// Copiar el resultado del device en el host

  cudaEventRecord(stop, 0);
  cudaEventSynchronize(stop);
. . .
  cudaEventElapsedTime(&elapsedTime, start, stop);
```

Código a Evaluar

5 Ejecuciones:

```
Tiempo Global: 90.392578 milseg
Tiempo Kernel:  1.287584 milseg
Tiempo Global: 78.921089 milseg
Tiempo Kernel:  1.259680 milseg
Tiempo Global: 79.809441 milseg
Tiempo Kernel:  1.264640 milseg
Tiempo Global: 79.938660 milseg
Tiempo Kernel:  1.258464 milseg
Tiempo Global: 89.800385 milseg
Tiempo Kernel:  1.273408 milseg
```

elapsedTime (float) tiene el tiempo de ejecución medido en milisegundos

Otra forma de evaluar: nvprof

- Ejecutar el comando con:

```
nvprof ./SaxpyP.exe
```

```
==65349== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 56.95%  49.518ms         2  24.759ms  24.708ms  24.810ms  [CUDA memcpy HtoD]
 41.71%  36.273ms         1  36.273ms  36.273ms  36.273ms  [CUDA memcpy DtoH]
  1.34%  1.1635ms         1  1.1635ms  1.1635ms  1.1635ms  saxpyP(. . .)
```

- Tiempos obtenidos por programa:

```
Tiempo Global: 89.800385 milseg
```
(86.954 ms)

```
Tiempo Kernel:  1.273408 milseg
```
(1.163 ms)

Ancho de Banda entre CPU y GPU

- Nos centramos en las transferencias CPU – GPU (PCIe)

```
cudaMemcpy(d_x, h_x, numBytes, cudaMemcpyHostToDevice);
```

numBytes = 4*N = 2^{26} bytes = 67.108.864 bytes (64 MB)

N = 16.777.216

```
==65349== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 56.95%  49.518ms         2  24.759ms  24.708ms  24.810ms  [CUDA memcpy HtoD]
 41.71%  36.273ms         1  36.273ms  36.273ms  36.273ms  [CUDA memcpy DtoH]
  1.34%  1.1635ms         1  1.1635ms  1.1635ms  1.1635ms  saxpyP(. . .)
```

AnchoBanda (HtoD) = 2^{26} bytes/24.759ms = 2,71 GB/s

AnchoBanda (DtoH) = 2^{26} bytes/36.273ms = 1,85 GB/s

AnchoBanda = 3*2^{26} bytes/85.791ms = 2,35 GB/s

AB_{max} 8+8 GB/s

GFLOPS y Ancho de Banda con Memoria

- Nos centramos en el kernel

```
__global__ void saxpyP (int N, float a, float *x, float *y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  y[i] = a * x[i] + y[i];
}
```

N = 16.777.216

flops = 2*N = 2^{25} Ops CF = 33.554.432 Ops en CF

Accesos a memoria = 2*N reads + N writes = 3*4*N bytes = $12 \cdot 2^{24}$ bytes

Tiempo Kernel: 1,1635ms

Tiempo Total: 86,954ms

GFLOPS(Kernel) = 2^{25} flops/(1.1635ms*10^6) = 28,84 GFLOPS

GFLOPS(Global) = 2^{25} flops/(86.954ms*10^6) = 385,9 MFLOPS

AnchoBanda Memoria = $12 \cdot 2^{24}$ bytes /1.1635ms = 173,04 GB/s

Jugando con el número de Threads

```
nThreads = …;  // 32, 64, 128, 256, 512, 1024
nBlocks = N/nThreads;
saxpyP<<<nBlocks, nThreads>>>(N, 3.5, d_x, d_y);
```

nThreads	nBlocks	Tiempo Kernel
32	524288	2,24 – 2,31 ms
64	262144	1,5 – 1,59 ms
128	131072	1,34 – 142 ms
256	65536	1,36 – 1,44 ms
512	32768	1,34 – 1,45 ms
1024	16384	1,39 – 1,47 ms

Tiempo de Kernel, calculado con eventos

A partir de 128 los resultados no parecen concluyentes

¿Qué pasa cuando N no es múltiplo de nThreads?

Jugando con el número de Threads

```
nThreads = …;  // 96, 192, 384, 768, 1536
nBlocks = (N+nThreads-1)/nThreads;
saxpyP<<<nBlocks, nThreads>>>(N, 3.5, d_x, d_y);
```

nThreads	
96	(ERROR) Copiar Datos Device --> Host - an illegal memory access was encountered in main.cu at line 83
192	
384	
768	
1536	(ERROR) Invocar Kernel - invalid configuration argument in main.cu at line 75

```
__global__ void saxpyP (int N, …) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  y[i] = a * x[i] + y[i];
}
```

Estamos accediendo a posiciones i > N.

Jugando con el número de Threads

```
nThreads = …;  // 96, 192, 384, 768, 1536
nBlocks = (N+nThreads-1)/nThreads;
saxpyP<<<nBlocks, nThreads>>>(N, 3.5, d_x, d_y);
```

nThreads	
96	(ERROR) Copiar Datos Device --> Host - an illegal memory access was encountered in main.cu at line 83
192	
384	
768	
1536	(ERROR) Invocar Kernel - invalid configuration argument in main.cu at line 75

```
__global__ void saxpyP (int N, …) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i<N) y[i] = a * x[i] + y[i];
}
```

Ahora SI funciona correctamente

Jugando con el número de Blocks

```
N = 1024 * 1024 * 128;
nThreads = 8;
nBlocks = N/nThreads; // nBlocks = 16777216
saxpyP<<<nBlocks, nThreads>>>(N, 3.5, d_x, d_y);
```

Versión CUDA : 8.0.61

```
(ERROR) Invocar Kernel - invalid argument in main.cu at line 77
```

Versión CUDA: 9.0.176

```
nThreads: 8
nBlocks: 16777216
Tiempo Global: 1436.453735 milseg
Tiempo Kernel: 61.121151 milseg
TEST PASS
```

```
Maximum sizes of each dimension of a grid:     2147483647 x 65535 x 65535
```

Otra forma de hacer las cosas

```
N = 1024 * 1024 * 128;
nBlocks = 5000;
nThreads = 1024;
saxpyP<<<nBlocks, nThreads>>>(N, 3.5, d_x, d_y);
```

Fijamos el número de bloques y el número de threads.

```
__global__ void saxpyP (int N, …) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  while (i<N) {
    y[i] = a * x[i] + y[i];
    i = i + gridDim.x * blockDim.x;
  }
}
```

Hay que cambiar el kernel.

```
nThreads: 1024
nBlocks: 5000
Tiempo Global: 840.250488 milseg
Tiempo Kernel: 9.798336 milseg
TEST PASS
```

```
nThreads: 1024
nBlocks: 131072
Tiempo Global: 886.492981 milseg
Tiempo Kernel: 9.779456 milseg
TEST PASS
```

Liberar espacio

Una buena práctica es liberar todo el espacio que hemos pedido, cuando ya no sea necesario.

```
// Obtener Memoria en el host
h_x = (float*) malloc(numBytes);


// Obtiene Memoria [pinned] en el host
cudaMallocHost((float**)&h_y, numBytes);
```

```
// Liberar Memoria
free(h_x); h_x = NULL;


// Liberar Memoria [pinned]
cudaFreeHost(h_y); h_y = NULL;
```

```
// Obtener Memoria en el device
cudaMalloc((float**)&d_x, numBytes);
```

```
// Liberar Memoria en el device
cudaFree(d_x); d_x = NULL;
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Sesion01

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya