Reporte Técnico
Gualberto Casas Medina - A00942270
Septiembre 2018

Introducción:

Aunque los procesadores de hoy en día son multinúcleo y permiten el procesamiento de múltiples instrucciones al mismo tiempo, comúnmente no se explota esta característica de los procesadores. Al no utilizar los varios núcleos disponibles, el procesador tiene mucho potencial que no se está utilizando. Para poder utilizar los varios núcleos de un procesador ya sea cpu o gpu, es necesario programar instrucciones que puedan correr en paralelo.

En la siguiente actividad realizamos la multiplicación de 2 matrices. Para esto se desarrollan tres algoritmos diferentes:

1. CPU sin threads
2. CPU con threads (omp)
3. GPU con threads (cuda)

Desarrollo:

Hay un archivo principal para este proyecto y se llama *main.cu*, el cual compilamos con:

```
nvcc -o out -Wno-deprecated-gpu-targets main.cu -std=c++11
```

Este programa puede correr cada uno de los algoritmos dependiendo de los parámetros que se le pasen. El primer parámetro que toma, representa el tamaño de la matriz. El segundo parámetro elige el algoritmo con el que se va a resolver el problema. *0 = CPU No Threads, 1 = CPU OMP Threads, 2 = GPU CUDA Threads.* Se utiliza el archivo *common.h* para llamar de manera segura las funciones de CUDA.

```
G:\master\clean\Users\magnus\Projects\octavio\assignments\a-1-matrix-multiplication-gualcm>ls
LICENSE   README.md common.h  main.cu
```

CPU No Threads results:

N = 1000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 0
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithoutThreads elapsed 4239.621582 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 0
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithoutThreads elapsed 4105.771973 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 0
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithoutThreads elapsed 4112.796387 ms
```

N = 2000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 0
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithoutThreads elapsed 48834.519531 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 0
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithoutThreads elapsed 48681.007812 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 0
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithoutThreads elapsed 49175.226562 ms
```

N = 4000

El programa nunca logró correr para un tamaño de 4000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 4000 0
./out Starting...
Matrix size: nx 4000 ny 4000
Killed
```

CPU OMP Threads results:

N = 1000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 1
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithThreads elapsed 4210.567383 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 1
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithThreads elapsed 4211.019043 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 1
./out Starting...
Matrix size: nx 1000 ny 1000
inCPUWithThreads elapsed 4180.627441 ms
```

N = 2000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 1
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithThreads elapsed 49506.558594 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 1
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithThreads elapsed 49563.460938 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 1
./out Starting...
Matrix size: nx 2000 ny 2000
inCPUWithThreads elapsed 49389.812500 ms
```

N = 4000

El programa nunca logró correr para un tamaño de 4000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 4000 1
./out Starting...
Matrix size: nx 4000 ny 4000
Killed
```

GPU CUDA Threads

N = 1000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 2
./out Starting...
Matrix size: nx 1000 ny 1000
cudaWithBlocksAndThreads <<<(32,1), (32,1)>>> elapsed 1944.605957 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 2
./out Starting...
Matrix size: nx 1000 ny 1000
cudaWithBlocksAndThreads <<<(32,1), (32,1)>>> elapsed 1901.804199 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 1000 2
./out Starting...
Matrix size: nx 1000 ny 1000
cudaWithBlocksAndThreads <<<(32,1), (32,1)>>> elapsed 1864.738281 ms
```

N = 2000

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 2
./out Starting...
Matrix size: nx 2000 ny 2000
cudaWithBlocksAndThreads <<<(63,1), (32,1)>>> elapsed 11875.764648 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 2
./out Starting...
Matrix size: nx 2000 ny 2000
cudaWithBlocksAndThreads <<<(63,1), (32,1)>>> elapsed 11913.609375 ms
```

```
A00942270@alien1-lab:~/.../a-1-matrix-multiplication-gualcm$ ./out 2000 2
./out Starting...
Matrix size: nx 2000 ny 2000
cudaWithBlocksAndThreads <<<(63,1), (32,1)>>> elapsed 12185.253906 ms
```

N = 4000

El programa nunca logró correr para un tamaño de 4000

Resultados:

Los promedios de los experimentos fueron los siguientes:
N = 1000
CPU No Threads = 4152.73 ms
CPU Threads = 4200.33 ms
GPU Threads = 1903.72 ms

N = 2000
CPU No Threads = 48896.67 ms
CPU Threads = 49486.61 ms
GPU Threads = 11991.54 ms

N = 4000
Sin Resultados

Características del Sistema:
GPU:

- GeForce GTX 670
- Cuda Cores: 1344
- 915 MHz Graphics Clock
- 980 MHz Processor Clock
- Memory Capacity 2GB

CPU:

- Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
- Max Frequency: 3900 MHz
- Min Frequency: 800 MHz
- Cores: 4

Conclusión:

Como podemos ver en los resultados, aunque estemos hablando de tamaños de matrices relativamente pequeños (N = 1000), trabajar con Threads en el GPU nos da los resultados de una manera mucho más rápida que usando el CPU para realizar las operaciones. No hay mejora entre *CPU No Threads y CPU Threads* en N = 1000 y N = 2000, porque el CPU utiliza poder de procesamiento para poder llevar control de los Threads.

Si comparamos los resultados, podemos ver que en matrices de N = 1000, el GPU es aproximadamente 2x más rápido. Si estamos tratando con matrices de tamaño N = 2000, el GPU es aproximadamente 4x más rápido.