Sistemas complejos en máquinas paralelas.

2da entrega, GPU.

Alumno: Ezequiel Darío Gambaccini.

LU: 715/13

1. Dadas dos matrices A y B, hacer una función para calcular la suma y mostrar el resultado tanto en CPU como en GPU.
2. Variar los valores de bloques, cantidad de hilos por bloque y medir el impacto en la performance.
3. Analizar si tiene sentido utilizar memoria constante o de textura e implementarlo o indicar claramente por qué no aplica.
4. Comparar el desempeño de las implementaciones en CPU y GPU graficando los tiempos obtenidos (CPU en un core como base de la ejecución).

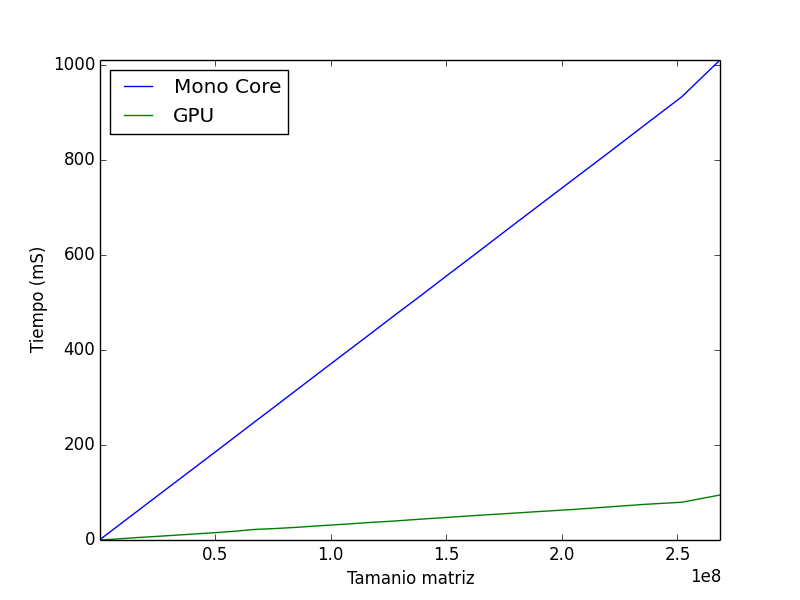
1) Ver código adjunto.

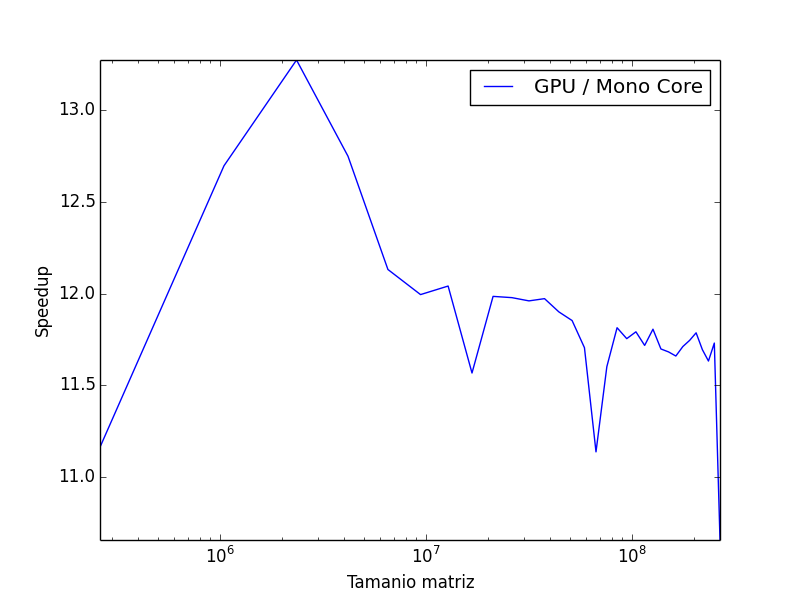
2) Variando la cantidad de bloques y threads por bloque para la suma de una matriz de 1024\*1024 elementos, desde 1024 bloques y 1024 threads, hasta 4096 bloques y 256 threads, se observo que la performance fluctuó entre 0,2 ms y 0,14 ms, consiguiendo su mejor valor con la combinación de 4096 bloques y 256 threads. Esto se debe a que al usar 256 threads por bloque, se logra ocupar al máximo cada stream multiprocessor de la gpu, cuya máxima cantidad de threads en ejecución es de 1536, y con esto, maximizar la performance del procesador.

|  |  |  |
| --- | --- | --- |
| Bloques | Threads | Tiempo |
| 1024 | 1024 | 0.1936 |
| 1156 | 961 | 0.255488 |
| 1296 | 900 | 0.266016 |
| 1444 | 841 | 0.280768 |
| 1600 | 784 | 0.282304 |
| 1764 | 729 | 0.207488 |
| 1936 | 676 | 0.214144 |
| 2116 | 625 | 0.223136 |
| 2304 | 576 | 0.20592 |
| 2500 | 529 | 0.24 |
| 2704 | 484 | 0.199072 |
| 2916 | 441 | 0.20496 |
| 3136 | 400 | 0.199712 |
| 3364 | 361 | 0.191168 |
| 3600 | 324 | 0.188896 |
| 3844 | 289 | 0.19456 |
| 4096 | 256 | 0.138688 |

3) No tiene sentido usar memoria constante o de textura ya que al ser una suma de matrices, cada posición de las matrices es accedida una única vez, por lo que no se aprovechan las ventajas de la memoria constante y de la memoria de textura, que son para varios accesos a un mismo valor. Este valor sería cacheado, por lo que la velocidad de acceso para los demás threads sería mucho menor, pero al ser accedido una única vez, no es aprovechado. Si esto fuera una multiplicación matricial, tendría sentido utilizar estos otros tipos de memoria, ya que cada posición de las matrices a operar sería accedida múltiples veces.

4) Para generar los graficos, se ejecutó el código 10 veces para cada tamaño de matriz y se sacó el promedio. El tamaño de las matrices va desde 5122 elementos hasta 51232 elementos. En el script g\_run.py se puede ver el código para correr las pruebas y con el script plot\_speed.py se puede generar los graficos.



Se usaron 256 threads por bloque para maximizar el uso de los stream multiprocesor, y se uso la minima cantidad de bloques necesaria en cada caso para que no haya threads sin hacer nada. Como se puede ver, la diferencia en performance es abismal, entre 11 y 13 veces. La velocidad de cálculo de la cpu disminuye linealmente al cuadruplicarse el tamaño de la matriz, mientras que el de la gpu apenas crece. A pesar de que el procesador de la cpu en 1 thread es mucho más potente que el de la gpu, y funciona a mayor frecuencia, en estas pruebas se demuestra que cantidad es mejor que calidad, ya que por cada instrucción que ejecuta la cpu, la gpu ejecuta la misma instrucción para 448 threads a la vez. (14 streaming multiprocessors \* 32 cuda cores por procesador) Por otro lado, la gpu también gana en velocidad de acceso a memoria, ya que hace levanta mayor cantidad de datos por lectura que la cpu.