Parcial I: Multiplicación de matrices

# Descripción

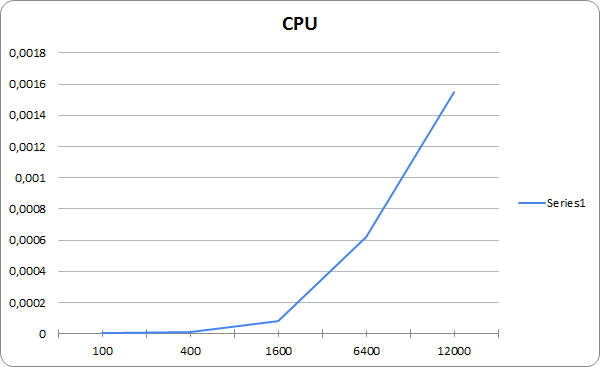
El parcial I consistió en el desarrollo programas que permitieran multiplicar matrices “n” por “m” multiplicadas por “m” por “n” o inclusive por “m” por “a”, más allá de realizar esto, el parcial consistió en realizar versiones de este programa en C de forma secuencial, en cuda C paralelizando el trato de las variables de las matrices optimizando el tiempo de ejecución, se hicieron dos versiones de código en cuda C, la primer versión consistió en pasar el codigo a cuda C sin realizar optimizaciones profundas llamándose así una versión ingenua, y en la segunda versión se hizo una profundización en la optimización usando técnicas cómo Tile, que sugiere pasar la máxima cantidad de datos a memoria compartida de la GPU para optimizar tiempo y no realizar varias veces innecesariamente un traslado de datos en los distintos tipos de memoria.

# Pruebas

Se probó el tiempo de ejecución de la multiplicación de matrices con diferentes tamaños, con diferente cantidad de filas y columnas, que a su vez cumplieran con la condición de que una multiplicación de matrices debe por lo menos ser así: “mxn” multiplicado “nxm”.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CPU | | | | | | | | |
| Número de filas por columnas | | Iteraciones con tiempos de ejcución | | | | |  | promedio |
| 5 | 10 | 0.000003 | 0.000001 | 0.000001 | 0.000001 | 0.000003 | 100 | 0.0000018 |
| 10 | 20 | 0.000006 | 0.000014 | 0.000006 | 0.000006 | 0.000007 | 400 | 0.0000078 |
| 20 | 40 | 0.000044 | 0.000044 | 0.000107 | 0.000108 | 0.000107 | 1600 | 0.000082 |
| 40 | 80 | 0.000807 | 0.000808 | 0.000808 | 0.000332 | 0.000333 | 6400 | 0.0006176 |
| 60 | 100 | 0.000929 | 0.002354 | 0.000926 | 0.00229 | 0.001243 | 12000 | 0.0015484 |

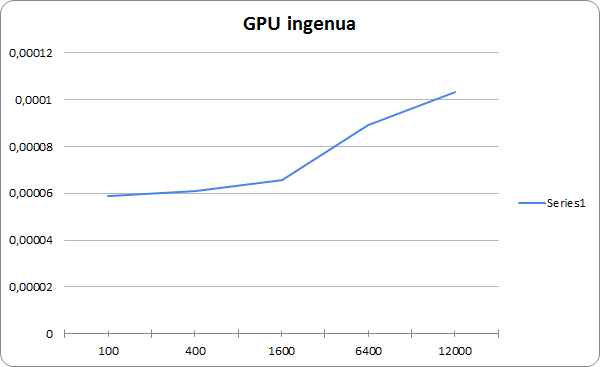
Produciendo la siguiente gráfica que **representa # de datos vs tiempo de ejecución.**



De la misma manera se probó el tiempo de ejecución de la multiplicación de matrices con las mismas condiciones anteriores pero en el código de cuda C ingenuo:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GPU ingenuo | | | | | | | | |
| Número filas por columnas | | Iteraciones con tiempos de ejcución | | | | |  | promedio |
| 5 | 10 | 0.000061 | 0.000058 | 0.000060 | 0.000057 | 0.000058 | 100 | 0.0000588 |
| 10 | 20 | 0.000064 | 0.000058 | 0.000059 | 0.000063 | 0.000060 | 400 | 0.0000608 |
| 20 | 40 | 0.000069 | 0.000064 | 0.000067 | 0.000063 | 0.000065 | 1600 | 0.0000656 |
| 40 | 80 | 0.000101 | 0.000087 | 0.000084 | 0.000089 | 0.000085 | 6400 | 0.0000892 |
| 60 | 100 | 0.000104 | 0.000101 | 0.000105 | 0.000105 | 0.000101 | 12000 | 0.0001032 |

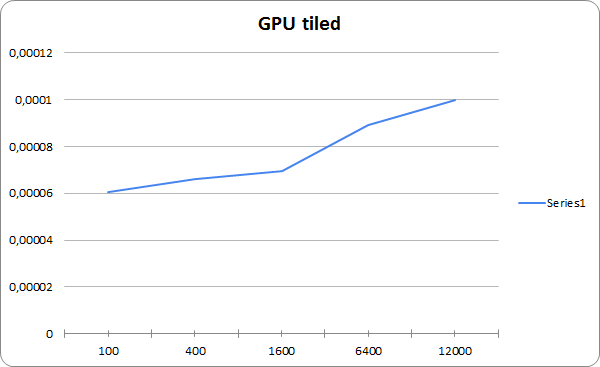
Produciendo la siguiente gráfica que representa # de datos vs tiempo de ejecución.



Y para finalizar también se realizaron pruebas de multiplicación de matrices pero en este caso se realizarón en el código de cuda C con Tile:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GPU tiling | | | | | | | | |
| Número filas por columnas | | Iteraciones con tiempos de ejcución | | | | |  | promedio |
| 5 | 10 | 0.000063 | 0.000061 | 0.000057 | 0.000061 | 0.000061 | 100 | 0.0000606 |
| 10 | 20 | 0.00008 | 0.000064 | 0.000063 | 0.000059 | 0.000065 | 400 | 0.0000662 |
| 20 | 40 | 0.000084 | 0.000065 | 0.000063 | 0.000063 | 0.000072 | 1600 | 0.0000694 |
| 40 | 80 | 0.000101 | 0.000087 | 0.000084 | 0.000089 | 0.000085 | 6400 | 0.0000892 |
| 60 | 100 | 0.000091 | 0.000143 | 0.000087 | 0.00009 | 0.000089 | 12000 | 0.0001 |

Produciendo la siguiente gráfica que **representa # de datos vs tiempo de ejecución.:**



# Diferentes tamaños de bloque.

Bloques de 4x4

Bloques de 16x16

Bloques de 32x32

# 4 Conclusiones

# 5 Servidor, código fuente.

<https://github.com/al-lo-co/HPC/tree/master>

<https://github.com/metalslayer95/HPC0120152/tree/master/Parcial%201>