Alejandro López Correa – 1088313429 Richard Andrey Salazar Serna - 1088328019

Parcial I: Multiplicación de matrices

1 Descripción

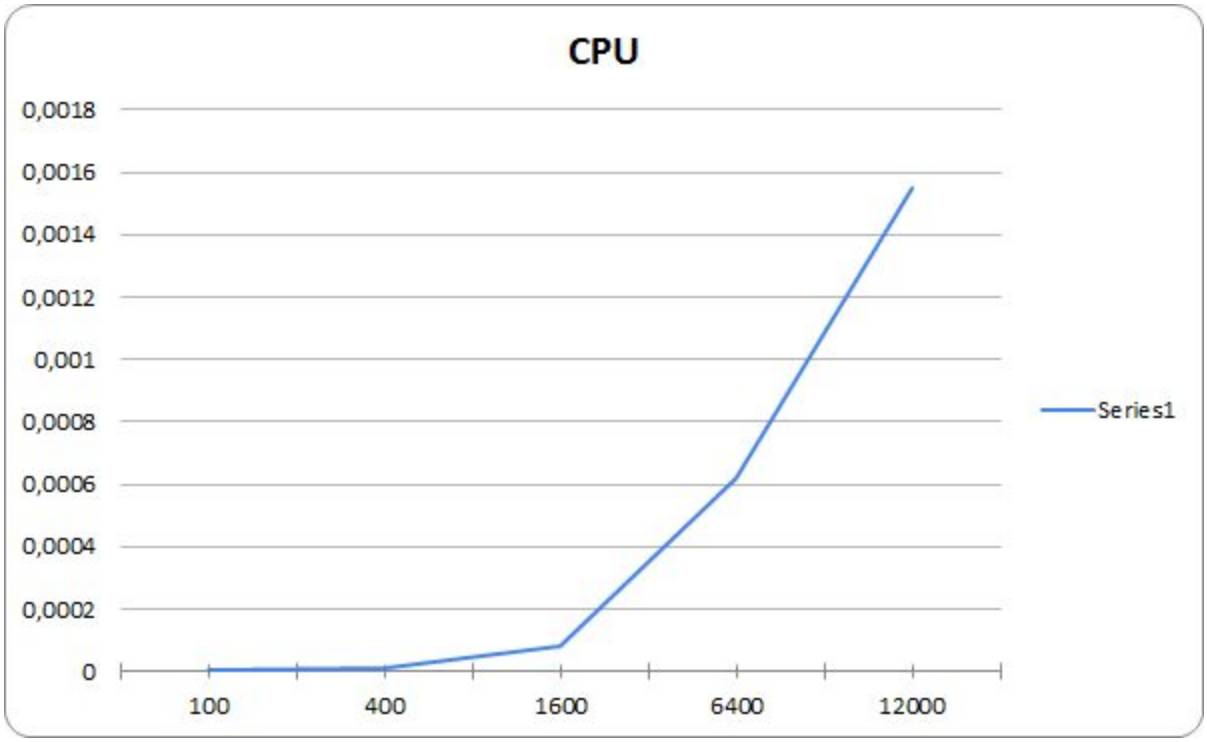
El parcial I consistió en el desarrollo programas que permitieran multiplicar matrices “n” por “m” multiplicadas por “m” por “n” o inclusive por “m” por “a”, más allá de realizar esto, el parcial consistió en realizar versiones de este programa en C de forma secuencial, en cuda C paralelizando el trato 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.

2 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”, en la toma de datos se ven dos columnas que tienen como nombre “Número de filas por columnas”, los tamaños de las matrices corresponden a: matriz A= NxM y matriz B= MxN.

CPU								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						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

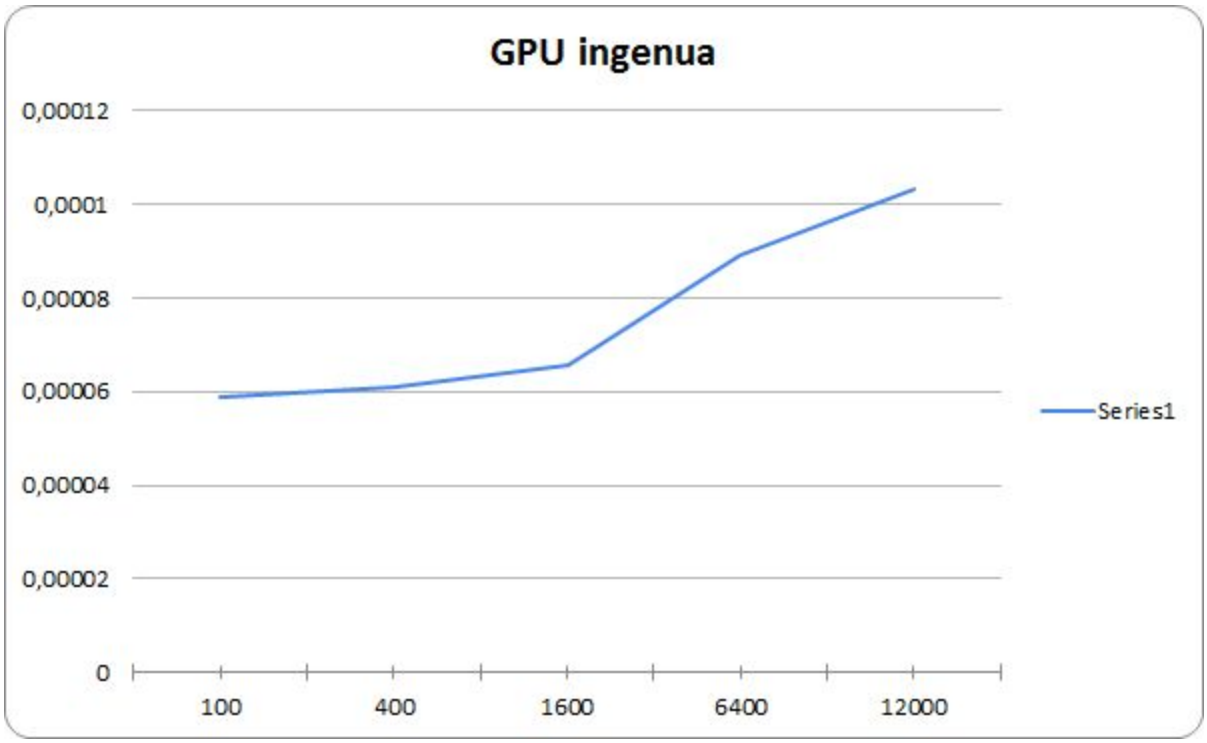
Produciendo la siguiente gráfica que **representa # de datos vs tiempo de ejecución.**



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 de filas por columnas(NxM)		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.

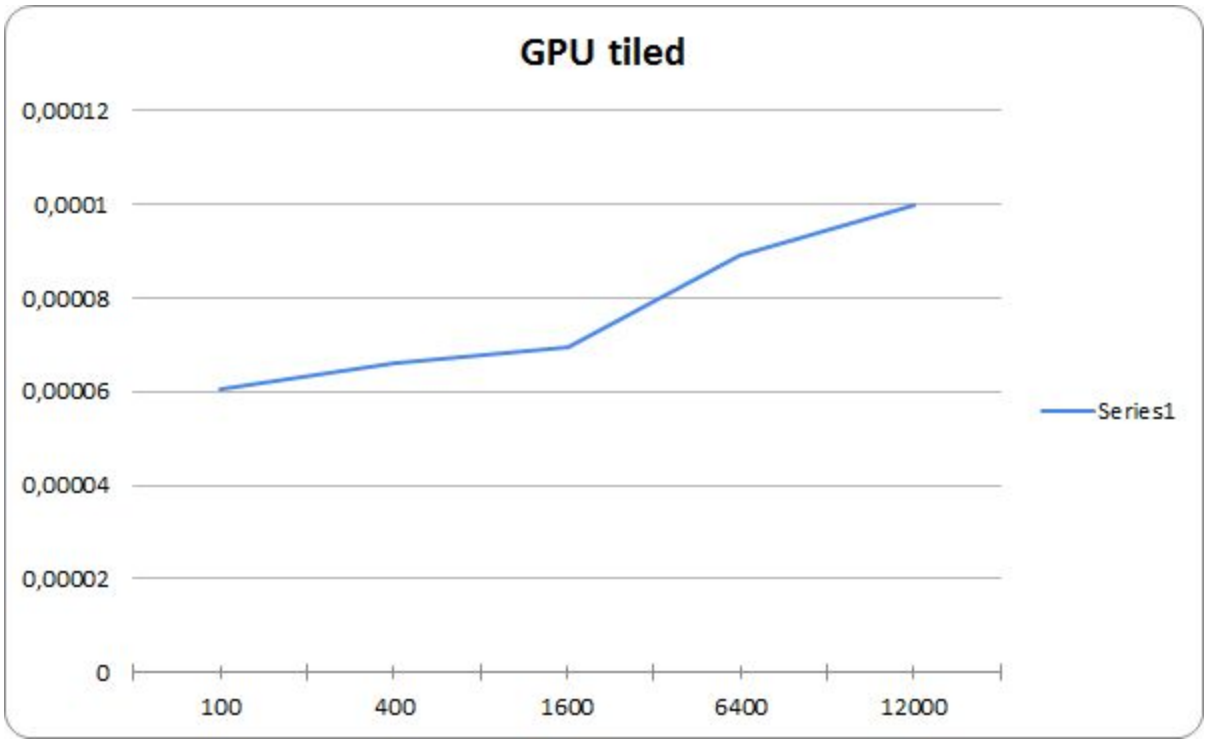


Y para finalizar también se realizaron pruebas de multiplicación de matrices pero en este caso se realizarón en el código de cuda C con Tile:

GPU tiling no flotante								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.00006	0.000076	0.000059	0.000062	0.000071	100	0.0000656
10	20	0.000062	0.000062	0.00006	0.000061	0.000063	400	0.0000616
20	40	0.000066	0.000064	0.000061	0.000063	0.000065	1600	0.0000638

40	80	0.000075	0.000074	0.000075	0.000081	0.000073	6400	0.0000756
60	100	0.000090	0.000085	0.000088	0.000087	0.000085	12000	0.000087

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



3 Diferentes tamaños de bloque.

Bloques de 4x4

GPU tiling								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000061	0.000056	0.00006	0.000057	0.000057	100	0.0000582

10	20	0.000077	0.000074	0.000076	0.000075	0.000071	400	0.0000746
20	40	0.000081	0.000080	0.000082	0.00010	0.00008	1600	0.0000846
40	80	0.000105	0.000109	0.000108	0.000108	0.000117	6400	0.0001094
60	100	0.000133	0.000132	0.000127	0.000129	0.000127	12000	0.0001296

GPU ingenuo								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000056	0.000058	0.000060	0.000075	0.000060	100	0.0000618
10	20	0.000064	0.000062	0.000060	0.000060	0.000059	400	0.000061
20	40	0.000071	0.000069	0.000067	0.000065	0.000066	1600	0.0000676
40	80	0.000097	0.00009	0.00008	0.00009	0.000081	6400	0.0000876
60	100	0.000116	0.000119	0.000111	0.000106	0.000119	12000	0.0001142

Bloques de 16x16

GPU tiling								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000055	0.000041	0.00004	0.000048	0.000041	100	0.000045
10	20	0.000044	0.000051	0.000043	0.000051	0.000048	400	0.0000474
20	40	0.000047	0.000046	0.000043	0.000057	0.000049	1600	0.0000484

40	80	0.000055	0.000057	0.000063	0.000089	0.000058	6400	0.0000644
60	100	0.000071	0.000066	0.000065	0.000063	0.000072	12000	0.0000674

GPU ingenuo								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000057	0.000057	0.000061	0.000058	0.000057	100	0.000058
10	20	0.000063	0.000060	0.000061	0.000057	0.000063	400	0.0000608
20	40	0.000065	0.000064	0.000064	0.000065	0.000064	1600	0.0000644
40	80	0.000077	0.000076	0.000075	0.000078	0.000076	6400	0.0000764
60	100	0.000095	0.000095	0.000098	0.000095	0.000097	12000	0.000096

Bloques de 32x32

GPU tiling								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000043	0.000053	0.000049	0.000048	0.000044	100	0.0000474
10	20	0.000043	0.000052	0.000047	0.000042	0.000042	400	0.0000452
20	40	0.000051	0.000055	0.000055	0.000066	0.000051	1600	0.0000556
40	80	0.000055	0.000067	0.000055	0.000055	0.000057	6400	0.0000578
60	100	0.000077	0.000069	0.000074	0.000063	0.00007	12000	0.0000706

GPU ingenuo								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						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

3 CON PUNTO FLOTANTE

Secuencial:

CPU flotante								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000003	0.000003	0.000001	0.000001	0.000002	100	0.000002
10	20	0.000005	0.000007	0.000006	0.000014	0.000007	400	0.0000078
20	40	0.000107	0.000044	0.000108	0.000051	0.000045	1600	0.000071
40	80	0.000332	0.000333	0.000343	0.000506	0.00038	6400	0.0003788
60	100	0.001096	0.002272	0.000927	0.001204	0.000991	12000	0.001298

Paralelo Ingenuo:

Alejandro López Correa – 1088313429 Richard Andrey Salazar Serna - 1088328019

GPU ingenuo flotante								
Número de filas por columnas(Nx M)		Iteraciones con tiempos de ejecución						promedio
5	10	0.000066	0.000061	0.000060	0.000057	0.000052	100	0.0000592
10	20	0.000063	0.000060	0.000058	0.000063	0.000069	400	0.0000626
20	40	0.000069	0.000065	0.000067	0.000066	0.000082	1600	0.0000698
40	80	0.000083	0.000084	0.000088	0.000089	0.000083	6400	0.0000854
60	100	0.000101	0.000106	0.000103	0.000107	0.000104	12000	0.0001042

Paralelo Tile:

GPU tiling flotante								
Número de filas por columnas(NxM)		Iteraciones con tiempos de ejecución						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

4 Conclusiones

- Se puede observar en las gráficas y en los datos tomados que al operar con una cantidad de datos pequeñas se tiene un mejor desempeño del Programa de multiplicación de matrices secuenciales, seguido del código ingenuo de multiplicación de matrices paralelo y por último la versión con Tile.
- Con una mayor cantidad de datos el orden desempeño se vuelve contrario es decir, se vuelve más rápido la versión paralelizada con Tile, la siguiente en desempeño sería la ingenua y finalmente la secuencial.
- No se ve una diferencia notable entre las versiones con datos enteros y con datos de punto flotante.

5 Servidor, código fuente.

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

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