Cristina María Garrido López

Universidad de Granada

Ingeniería de Computadores

Arquitectura y Computación de Altas Prestaciones

Práctica 6. CUDA Toy

1. Introducción

Para esta práctica se nos proporciona un código en CUDA en el que debemos modificar algunos de los puntos especificados con el fin de obtener un rendimiento mayor.

2. Estudio realizado

En primer lugar se ha ejecutado el código utilizando el tipo float y un kernel por cada columna realizando una multiplicación. Estos son los datos obtenidos:

GFLOP/s	100000	10000	1000
200-200-27	36,2415	19,8998	3,1388
4000-200-540	200,2409	190,5191	71,2798
512000-200-69120	240,7787	237,651	215,9388
300-100-22	53,6385	30,4384	4,9158
32000-200-4320	234,1939	203,5805	150,8542
96000-100-7040	238,1346	238,5056	186,6442
6000-100-440	182,7037	168,6758	87,1174

Tabla 1. Float, multiplicación, kernel por columna.

A continuación se ha ejecutado lo mismo, solo que se ha cambiado el tipo float por int.

GFLOP/s	100000	10000	1000
200-200-27	41,2635	19,5345	3,1572
4000-200-540	107,8483	90,7193	52,3123
512000-200-69120	120,2403	119,7222	114,0148
300-100-22	50,851	34,5151	5,0474
32000-200-4320	118,8448	107,3983	105,1802
96000-100-7040	120,0437	112,7495	96,768
6000-100-440	106,3075	93,3292	63

Tabla 2. Enteros, multiplicación, kernel por columna.

El siguiente paso ha sido modificar la operación, esta vez realizamos una división con floats en las dos primeras matrices de la tabla, en las siguientes dos una resta y en las últimas una suma.

GFLOP/s	100000	10000	1000
200-200-27	6,7684	5,6048	2,5935
4000-200-540	200,7325	32,0145	22,9284
512000-200-69120	240,7775	238,5750	208,0908
300-100-22	53,6429	30,3272	6,2223
32000-200-4320	235,6047	218,5061	161,0977
96000-100-7040	237,7411	209,4545	187,9161
6000-100-440	188,1261	160,5494	90,8238

Tabla 3. Float, división, resta y suma, kernel por columna.

Por último, al ver que las anteriores operaciones no mejoraban el rendimiento, hemos vuelto a establecer la multiplicación como operación y hemos cambiado el kernel a uno global.

GFLOP/s	100000	10000	1000
32000-200-4320	181,55	180,9147	146,9996
96000-100-7040	234,1754	209,8938	194,5964
6000-100-440	85,0498	76,6598	70,7214

Tabla 4. Float, multiplicación, kernel global.

3. Razonamiento y gráficos

En primer lugar, en la Tabla 1 y la Tabla 2 se ha ejecutado el código con float con int y con double (aunque este era muy similar a la ejecución con enteros, por ello no aparece en las tablas). Vemos que la ejecución con float es mucho mejor que la de enteros, esto es debido a que la GPU está optimizada para operar en coma flotante, por lo tanto si le enviamos enteros o double lo hará de una manera más lenta. (Consultar el Gráfico 1).

Como vemos en la Tabla 3, la división empeora mucho el rendimiento y la suma y la resta lo igualan con respecto a la multiplicación. De nuevo, esto sucede porque la multiplicación está optimizada en la GPU, ya que es operación necesaria a la hora de generar gráficos. (Consultar los gráficos 2, 3 y 4).

Al aumentar el número de operaciones en todas las simulaciones realizadas, es rápido ver que el rendimiento aumenta con este, ya que hay más cálculos que hacer en menos tiempo.

Por último, al utilizar un kernel global en la Tabla 4 comprobamos que es mucho más

rápido utilizar uno por columna, esto es obvio, ya que la GPU está preparada para funcionar con un alto grado de paralelismo, lo cual le permite realizar estos cálculos mucho más rápido que si sólo consta de un kernel. (Consultar el gráfico 5).

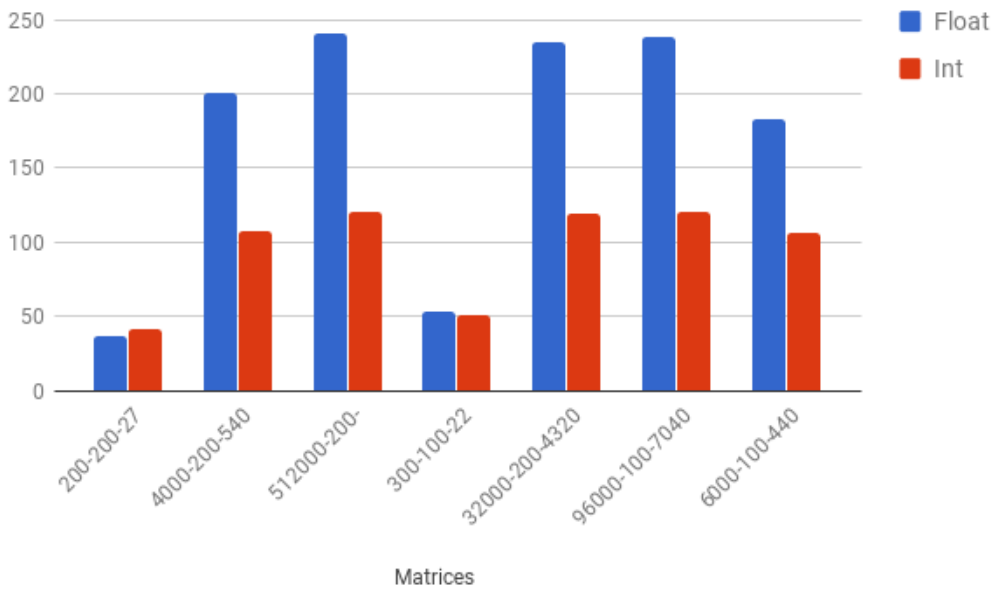

Gráfico 1. Comparativa entre float e int.

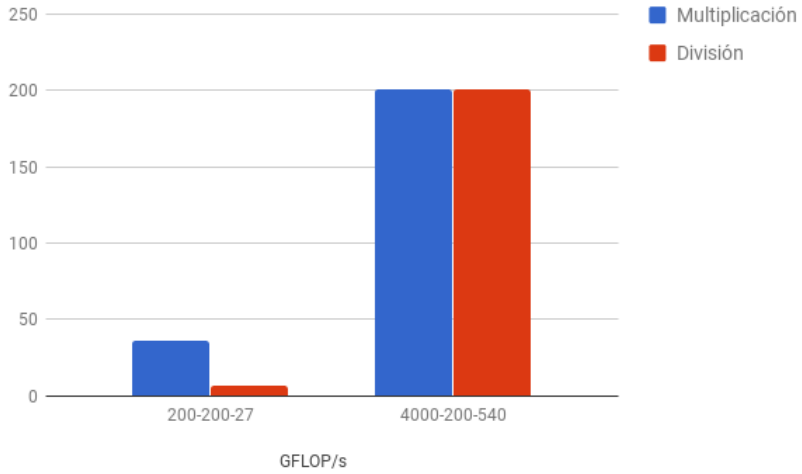

Gráfico 2. Comparativa entre multiplicación y división.

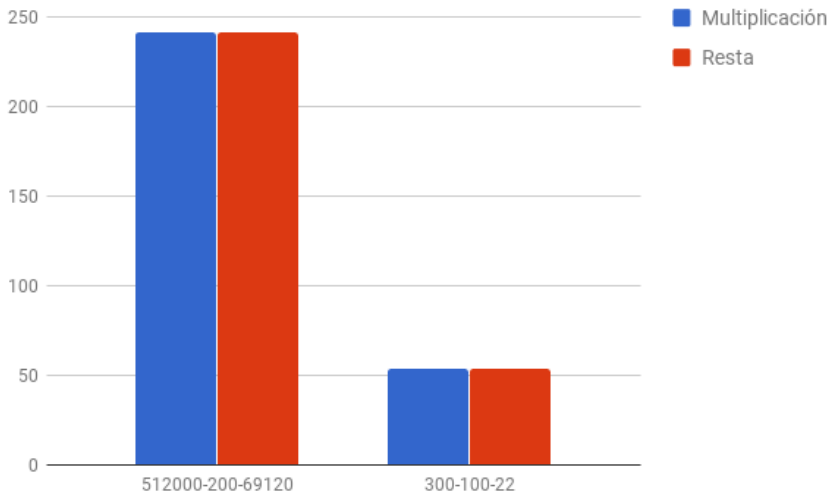

Gráfico 3. Comparativa entre multiplicación y resta.

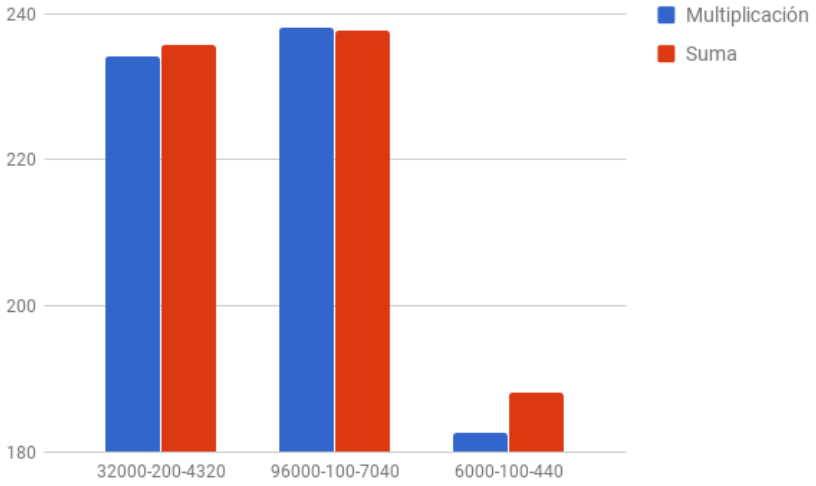

Gráfico 4. Comparativa entre multiplicación y suma.

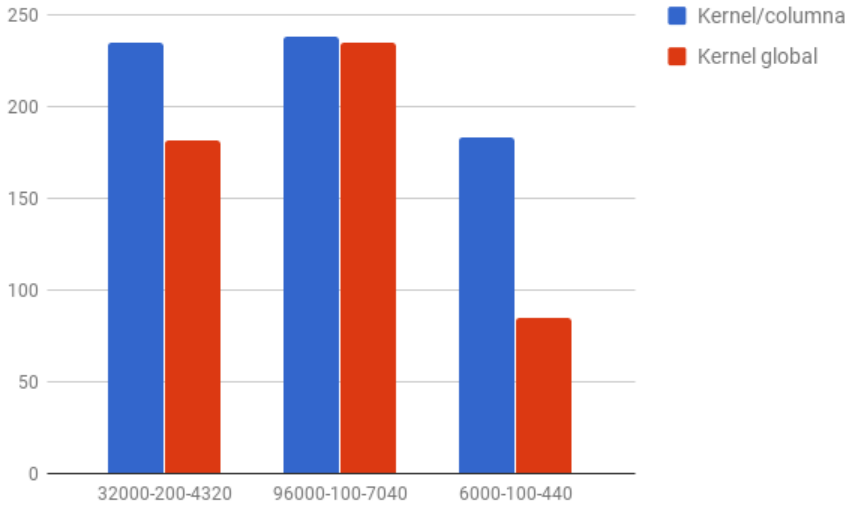

Gráfico 5. Comparativa entre kernel por columna y kernel global..