Proyecto Final
Histograma
Realizado Con Cuda

Diego Cruz Rodríguez
alu0101105802
Arquitectura Avanzada y de Propósito Específico
Grado de Ingeniería Informática
Universidad de La laguna
7/07/2021

Índice

Resumen

Durante el informe se intentará mostrar el trabajo realizado para la implementación en CUDA de un algoritmo e histogramas. Se comentarán las diferentes implantaciones realizadas y resultados de las pruebas de rendimiento realizadas a los mismos.

Enunciado

Problema a resolver

La tarea final es realizar un histograma de un vector V de un número elevado N de elementos enteros aleatorios. El histograma consiste en un vector H que tiene M elementos que representan "cajas". En cada caja se cuenta el número de veces que ha aparecido un elemento del vector V con el valor adecuado para asignarlo a esa caja (normalmente cada caja representa un rango o intervalo de valores). En nuestro caso, para simplificar la asignación del elemento de V a su caja correspondiente del histograma, vamos a realizar la operación ValorElementoV Modulo M, que directamente nos da el índice de la caja del histograma al que pertenecerá ese elemento y que deberemos incrementar. Se sugiere como N un valor del orden de millones de elementos y como M, 8 cajas.

Diferentes implementaciones

Implementación Base

Como implementación base (que podremos mejorar en tiempo o no) se pide crear tantos hilos como elementos de V para que cada uno se encargue de ir al elemento que le corresponda en V e incremente la caja que corresponda en el vector histograma H (posiblemente de forma atómica).

Segunda Implementación

Como segunda implementación, dividiremos la operación en dos fases. En la primera, en lugar de trabajar sobre un único histograma global, repartiremos el cálculo realizando un cierto número de histogramas (que llamaremos "locales"), cada uno calculado sobre una parte del vector de datos. La idea es reducir el número de hilos escribiendo sobre la misma posición del histograma, ya que dicha operación debe ser atómica y se serializan dichos accesos. La segunda fase realizará la suma de los histogramas locales en un único histograma global final. Se debe intentar llevar a cabo de la forma más paralela posible, utilizando el método de reducción.

Implementaciones realizadas

Implementación Base

Se crea un único histograma compartido por todos los hilos de ejecución del programa. A la hora de realizar los guardados del valor obtenido en el programa se emplea una suma atómica para evitar problemas de concurrencia. Archivo con nombre (histogramav1.cu)

Segunda Implementación

Cada hilo de ejecución del programa guarda sus valores en un vector local. Una vez finalizado el cálculo emplea una operación atómica para guardar los valores en el vector común del programa. Archivo con nombre (histogramav2.cu)

Mejoras incluidas

Debido a que cada hilo hiciera un único cálculo me parecía no aprovechar cada hilo, decidí crear una variable para que me permitiera controlar cuantos datos realizaba cada hilo. para ello use un DEFINE con el nombre SCALA que me permite hacer que un hilo realize mas cuentas. y también reducir el número de bloques necesarios para la ejecución del programa. Se realiza la inicialización del vector de datos iniciales en un kernel empleando números aleatorios.

Pruebas realizadas

Durante las pruebas se ejecutaron cada proceso 10000 veces para así obtener una media significativa del tiempo junto al máximo y mínimo de la ejecución. Y el uso de memoria de la aplicación durante la ejecución.
Se varió la SCALA (datos por hilo) y el tamaño de los datos N.

Media del tiempo de ejecución

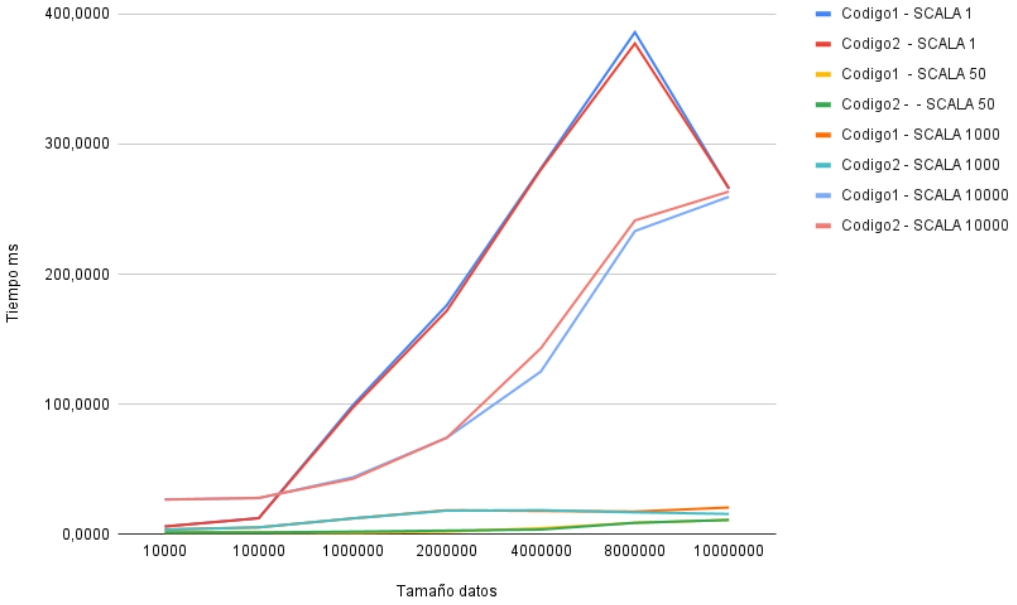

Tabla media de tiempo

Media ms	SCALA							
	1		50		1000		10000	
Tamaño datos	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2
10000	6,0493	6,0487	1,5387	1,5384	3,7399	3,5569	26,8417	26,6993
100000	12,4246	12,4223	1,4840	1,4740	5,3542	5,3568	27,7801	28,0462
1000000	99,2272	97,3135	1,2128	2,0725	12,2549	12,2803	43,8320	42,6488
2000000	176,1740	171,8820	2,1172	2,9020	18,5268	18,2360	74,2881	74,3170
4000000	281,2190	280,3560	4,7291	3,6632	17,9726	18,5807	125,1410	143,0470
8000000	385,8920	377,2590	8,8676	8,8633	17,5192	16,9017	233,0890	241,2470
10000000	265,5660	265,8680	11,0763	11,0658	20,6166	15,6720	259,4320	263,4700

Tabla minimo de tiempo

Min ms	SCALA							
	1		50		1000		10000	
Tamaño datos	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2
10000	0,6784	0,6802	0,5431	0,5414	0,7190	0,7228	12,8031	12,7946
100000	1,4214	1,4030	0,5428	0,5444	0,8887	0,9036	11,4249	13,6288
1000000	51,9117	48,1668	1,0361	0,8610	4,6639	5,0444	20,8124	27,0231
2000000	98,8556	91,5076	1,3437	1,1122	6,1098	5,2644	47,6937	54,2227
4000000	105,2740	106,3250	2,3451	2,2215	4,9910	6,8849	77,966	103,056
8000000	87,7627	87,3695	4,1535	3,8880	10,9774	5,4252	104,569	110,283
10000000	101,8030	134,6100	4,7543	4,6566	6,4064	8,1232	44,435	48,090

Tabla máximo de tiempo

Memoria Mib	SCALA							
	1		50		1000		10000	
Tamaño datos	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2
10000	155	155	155	155	155	155	155	155
100000	159	159	155	155	155	155	155	155
1000000	203	203	159	159	159	159	159	159
2000000	253	253	163	163	163	163	163	163
4000000	353	353	173	173	171	171	171	171
8000000	553	553	193	193	187	187	187	187
10000000	651	651	203	203	195	195	195	195

Uso de memoria durante la ejecución

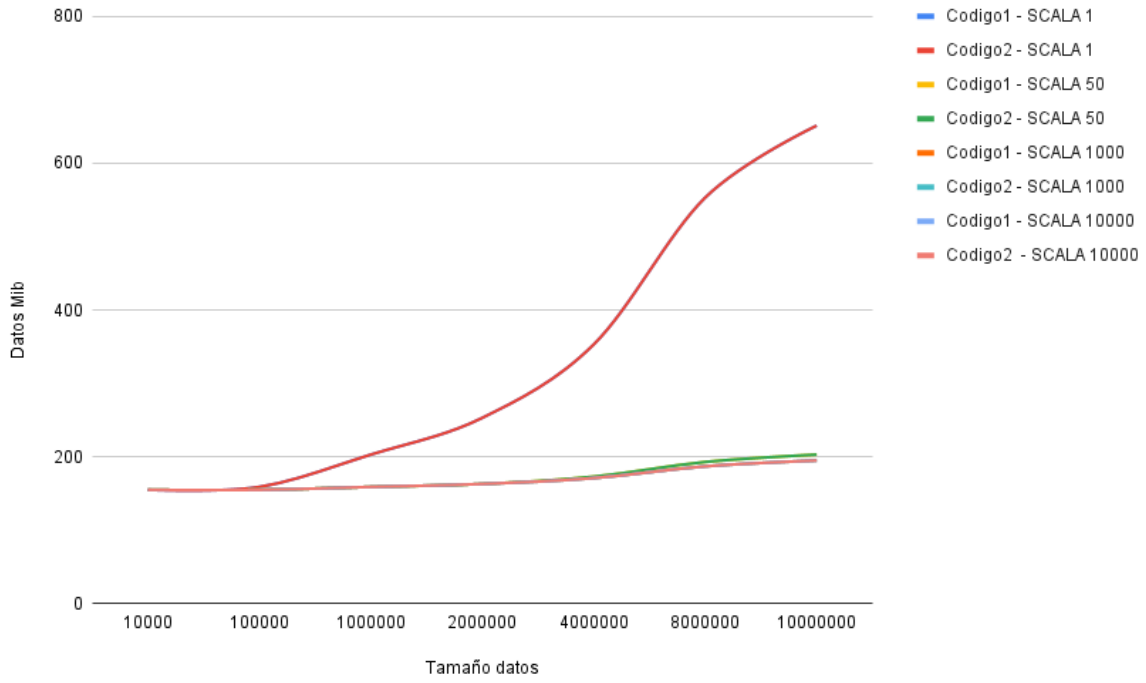

Tabla de uso de memoria durante la ejecución

Memoria Mib	SCALA							
	1		50		1000		10000	
Tamaño datos	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2	Codigo1	Codigo2
10000	155	155	155	155	155	155	155	155
100000	159	159	155	155	155	155	155	155
1000000	203	203	159	159	159	159	159	159
2000000	253	253	163	163	163	163	163	163
4000000	353	353	173	173	171	171	171	171
8000000	553	553	193	193	187	187	187	187
10000000	651	651	203	203	195	195	195	195

Conclusión

Mediante las pruebas hechas se puede deducir que la SCALA (datos a procesar por hilo) que mejor funciona es entre 50 y 1000 haciendo que el crecimiento del algoritmo sea lineal, respecto al tiempo. Si usamos una escala muy pequeña se tiene que crear tantos hilos que no compensa el proceso de creación de los mismos y si usamos muy pocos hilos no aprovechamos el potencial de la gráfica de paralelizar.

En cuanto a las diferencias de código, si parece que al realizarlo de forma local se consigue un aceleramiento del proceso pero detectamos que es mínimo ya que como podemos ver en pruebas con 100.000.000 datos de entrada usando el SCALA 1000 vemos diferencia de 5 ms. Aunque sí se puede apreciar en ese mismo banco de pruebas que a medida que crecen los datos se acentúa más la separación entre ambos códigos.

Respecto a la memoria no he encontrado diferencia alguna, pienso que quizás esto se deba a que el vector que se emplea como histograma es de únicamente 8 elementos. Y este tamaño no es realmente significativo. Otra cosa que he pensado es que el método empleado para tomar las mediciones no fuera el apropiado o no tuviera la suficiente velocidad de respuesta para detectar el uso del algoritmo. El método que usé fue el comando nvidia-smi durante la ejecución de las diferentes pruebas.