UNIVERSIDAD DE GRANADA

Arquitectura y computación de altas prestaciones (2022-2023)
Grado en Ingeniería Informática

Práctica 4

Realizado por:
Hlib Bukhtieiev

Estructura del paralelismo

Para empezar, definimos el problema y las estructuras de datos utilizadas en el código. Se pide encontrar niveles de gris en una imagen – se tiene un array que representa la imagen y otro array para almacenar el número de ocurrencias para cada nivel de gris. Es decir, lo que se busca paralelizar es una búsqueda secuencial en un array muy grande.

La versión del código propuesta se las arregla bien para recorrer correctamente el array de la imagen, sin embargo, falla en la escritura atómica siempre y cuando el número de hebras que constituyen un bloque es distinto al tamaño del array en el que contiene niveles de gris. Antes de empezar a solucionar el problema, tal vez sería oportuno mencionar que el programa inicial es fácilmente mejorable si en vez de hacerle a cada thread saltar por la memoria provocando fallos de caché, hacer que trabaje en un intervalo de tamaño SIZE / THREADS_totales, igual que hicimos en el resto de prácticas.

Solusione

Tras analizar bien el problema, la primera solución que se le puede ocurrir al estudiante es hacer un cambio muy simple pero que funcione para todo tipo de tamaño de bloque: hacer la suma atómica directamente sobre el array resultado de los niveles de gris. Resulta que para el mismo tamaño de la imagen de 100 Megas el rendimiento empeora casi 10 veces:

```
estudiante3@genmagic:~$ ./a.out
Tiempo de CPU (s): 0.1636
Check-sum CPU: 104857600
Check-sum GPU: 104857600
Calculo correcto!!
Tiempo medio de ejecucion del kernel<<<32, 256>>> sobre 104857600 bytes [s]: 0.0498
estudiante3@genmagic:~$ nvcc ej.cu
estudiante3@genmagic:~$ ./a.out
Tiempo de CPU (s): 0.1612
Check-sum CPU: 104857600
Check-sum GPU: 104857600
Calculo correcto!!
Tiempo medio de ejecucion del kernel<<<32, 256>>> sobre 104857600 bytes [s]: 0.0058
```

¿Qué ocurre? Esto se debe a que de esta forma desaprovechamos muchísimo el potencial paralelismo, ya que casi todo el tiempo se pierde en serialización de la concurrencia en memoria, es decir, todas las hebras de todos los bloques pueden escribir en el array resultante simultáneamente y existe mucha probabilidad de que dos hebras quieran sumar una ocurrencia al mismo color de gris a la misma vez y entonces una o más hebras tendrán que esperar. Introduciendo una variable temporal compartida exclusivamente entre las hebras de un solo bloque reducimos el volumen del problema. Por ejemplo, si el tamaño del grid es de 32 bloques, entonces, teóricamente mejorariamos en 32 veces la velocidad de cálculo, aunque en la práctica los números siempre van a ser inferiores.

La solución que se me ha ocurrido es la siguiente:

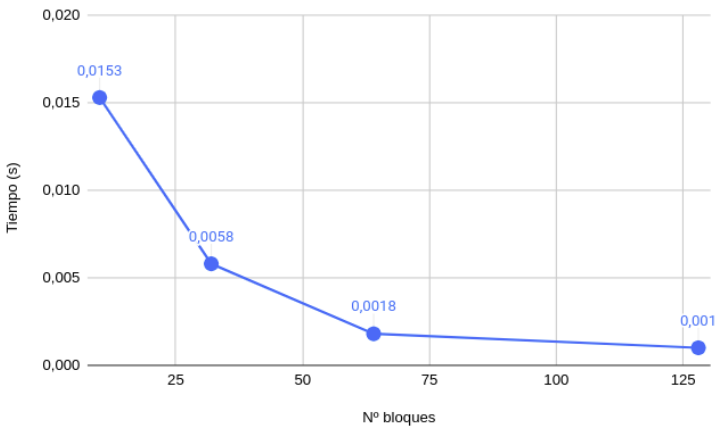
```
if (threadIdx.x < IMDEP){
        int thr_per_block = (THREADS_PER_BLOCK > IMDEP) ? IMDEP : THREADS_PER_BLOCK;
        int div = IMDEP / thr_per_block,
        remainder = IMDEP % thr_per_block,
        send_id;
        int num_sends = (threadIdx.x < remainder) ? div + 1 : div;

        for (int i = 0; i < num_sends; i++){
                send_id = (!remainder) ? i + threadIdx.x*div :
                        (threadIdx.x < remainder ? i + div*threadIdx.x + threadIdx.x
                                : i + div*threadIdx.x + remainder);

                atomicAdd( &(histo[send_id]), temp[send_id] );
        }
}
```

Hemos de tener en cuenta que el cuello de botella a la hora de escribir el resultado en el array “histo” es la longitud del array de los niveles de gris, así no tendría mucho sentido utilizar 257 hilos para escribir en un array con 256 posiciones. Entonces lo primero que se hace es acotar el número de hebras que se van a utilizar con el if, de esta forma, si en la búsqueda secuencial han participado más de IMDEP hebras, en la escritura de los resultados cada hebra hará solamente una suma atómica, en el caso contrario se calcula el intervalo en el que va a operar cada hebra de forma más equitativa, es decir, cuantos elementos del array “temp” va a recorrer para hacer la suma atómica en el array resultado.

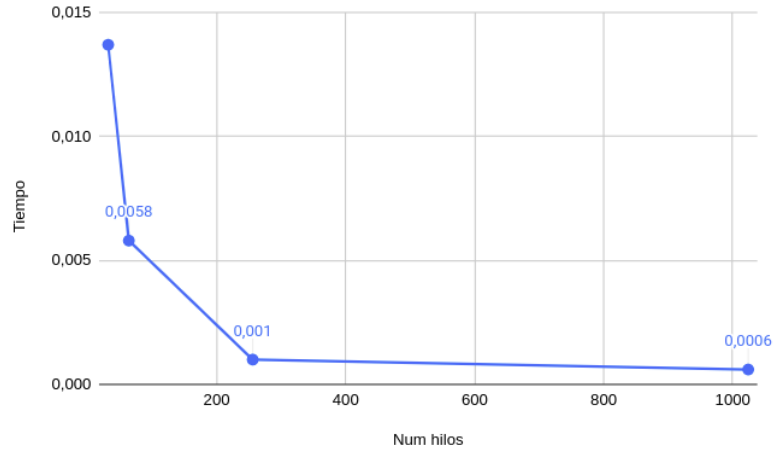
Experimento 1



Num bloques	Tiempo
10	0,0153
32	0,0058
64	0,0018
128	0,001

Se nos pide experimentar con el tamaño del grid. Al aumentar el tamaño del grid conseguimos más hilos paralelos, con lo que el tiempo también decrementa, aunque a partir de los 128 bloques por grid la mejora es prácticamente despreciable.

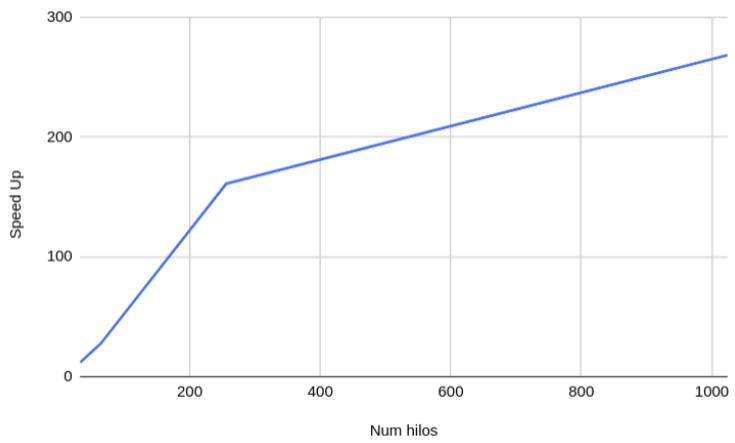
Experimento 2



Num hilos	Tiempo
32	0,0137
64	0,0058
256	0,001
1024	0,0006

En este experimento había que tantear con el número de hilos por bloque para el mejor valor de bloques del resultado anterior. La lógica es la misma en este caso: al añadir más hilos el tiempo decrementa. Sin embargo, hay que tener en cuenta que la unidad mínima para usar hilos en gpu es un “rollo” de 32 hilos, es decir, si yo quisiera poner que quiero usar 33 hilos por bloque, realmente se activarían 64. Además, hay que saber que el máximo de hilos por bloque es 1024.

Experimento 3



Num hilos	Speed Up
32	11,791
64	28
256	161,1
1024	268,5

He comparado los resultados de la versión secuencial de la CPU (0.1611 s para 100 MB) frente a versión paralela de la GPU con distinto número de hilos por un bloque. El bloque ha sido el mejor del experimento 1. Se aprecia como al principio se dispara la aceleración y con el máximo número de hilos por bloque ya que la sobrecarga se hace mayor también.