Proyecto final Histograma realizado con CUDA

Arquitecturas Avanzadas y de Propósito Específico

Cheuk Kelly Ng Pante (alu0101364544@ull.edu.es)

14 de junio de 2024

Índice general

1. Introducción

Consiste en realizar un histograma de un vector V, de un número elevado de N 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 Módulo M, que nos da directamente el índice de la caja del histograma a la que pertenecerá ese elemento, y cuyo contenido deberemos incrementar. Se sugiere como N un valor del orden de millones de elementos y como M 8 cajas.

2. Desarrollo del proyecto

2.1. Implementación base

Como implementación base, se pide crear tantos hilos como elementos V para que cada uno se encargue de ir al elemento que le corresponda a V, e incremente así la caja correcta en en el vector histograma H, de forma atómica.

El código base es el siguiente:

```
__global__ void histogram() {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < N) {
    atomicAdd(&vector_H[vector_V[i] % M], 1);
  }
}
```

La implementación base lo que hace es crear un único histogram compartido por todos los hilos. Cada hilo se encarga de incrementar la caja correspondiente al elemento V que le corresponda. Para ello, se utiliza la función atomicAdd, para incrementar de forma atómica el valor de la caja correspondiente al elemento V. La implementación se encuentra en el archivo *histogram_1.cu*.

2.2. Segunda implementación

Como segunda implementación, se va a dividir el cálculo del histograma en dos fases. En la primera fase, se va a usar un array en memoria compartida local que se usará para almacenar los histogramas calculados por cada hilo. Tras esto, cada hilo inicializa una parte del histograma local a 0. Por último, cada hilo procesa un elemento del vector V, incrementando el contador correspondiente en el histograma local. En la segunda fase, se realiza la suma de los histogramas locales en un único histograma global final. La implementación se encuentra en el archivo *histogram_2.cu*.

El código es el siguiente:

```
__global__ void histogram() {

  extern __shared__ int local_histogram[];

  int tid = threadIdx.x;
  for (int i = tid; i < M; i += blockDim.x) {
    local_histogram[i] = 0;
```

```
  }
  __syncthreads();

  int i = blockIdx.x * blockDim.x + tid;
  if (i < N) {
    atomicAdd(&local_histogram[vector_V[i] % M], 1);
  }
  __syncthreads();

  for (int j = tid; j < M; j += blockDim.x) {
    atomicAdd(&vector_H[j], local_histogram[j]);
  }
}
```

3. Pruebas realizadas

Durante las pruebas realizadas, se he ejecutado cada implementación 10000 veces para obtener un tiempo promedio significativo de tiempo. De igual forma, se ha obtenido el tiempo máximo y mínimo de la ejecución.

3.1. Tiempos promedio de ejecución

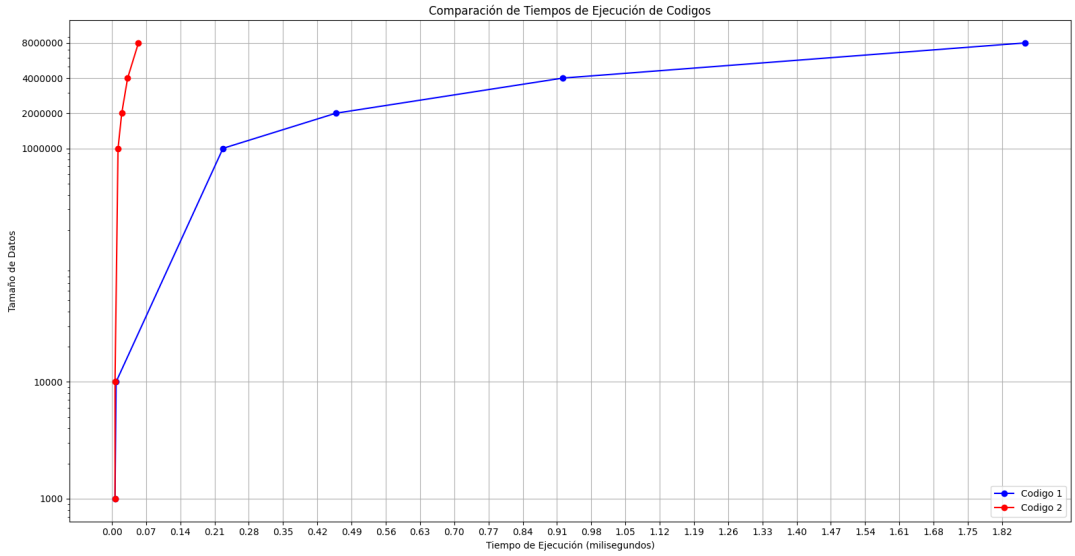

Figura 3.1: Tiempo promedio de ejecución

Tamaño de datos	Tiempo Código 1	Tiempo Código 2
1000	0.00642909	0.00644173
10000	0.00865969	0.00644241
1000000	0.227177	0.0126207
2000000	0.458186	0.0201534
4000000	0.922377	0.0322431
8000000	1.86746	0.0543328

Cuadro 3.1: Tiempo promedio de ejecución

3.2. Tiempos máximo de ejecución

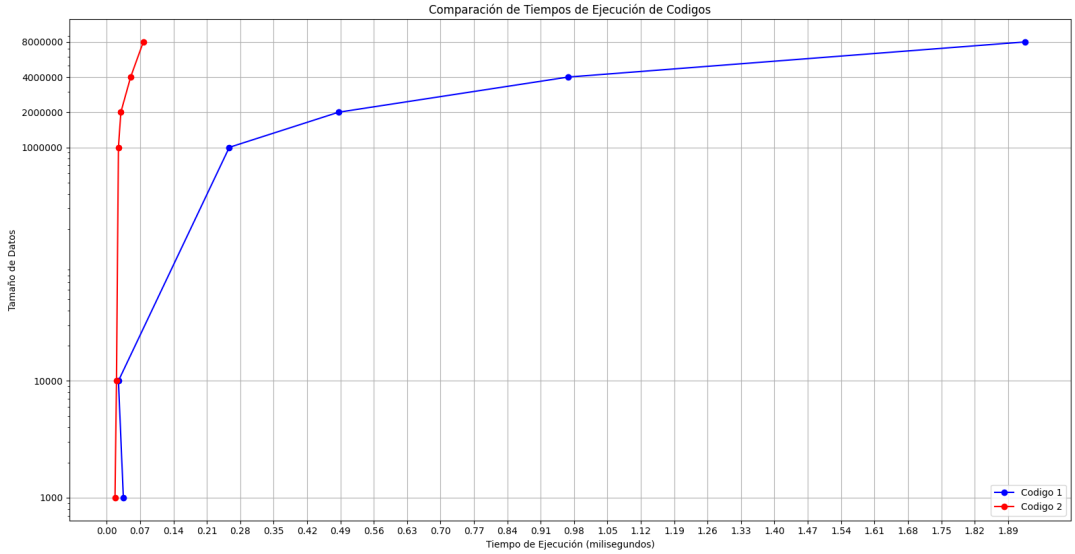

Figura 3.2: Tiempo máximo de ejecución

Tamaño de datos	Tiempo Código 1	Tiempo Código 2
1000	0.034912	0.017408
10000	0.024576	0.02048
1000000	0.257024	0.024576
2000000	0.4864	0.029664
4000000	0.96768	0.050176
8000000	1.92614	0.076736

Cuadro 3.2: Tiempo máximo de ejecución

3.3. Tiempos mínimo de ejecución

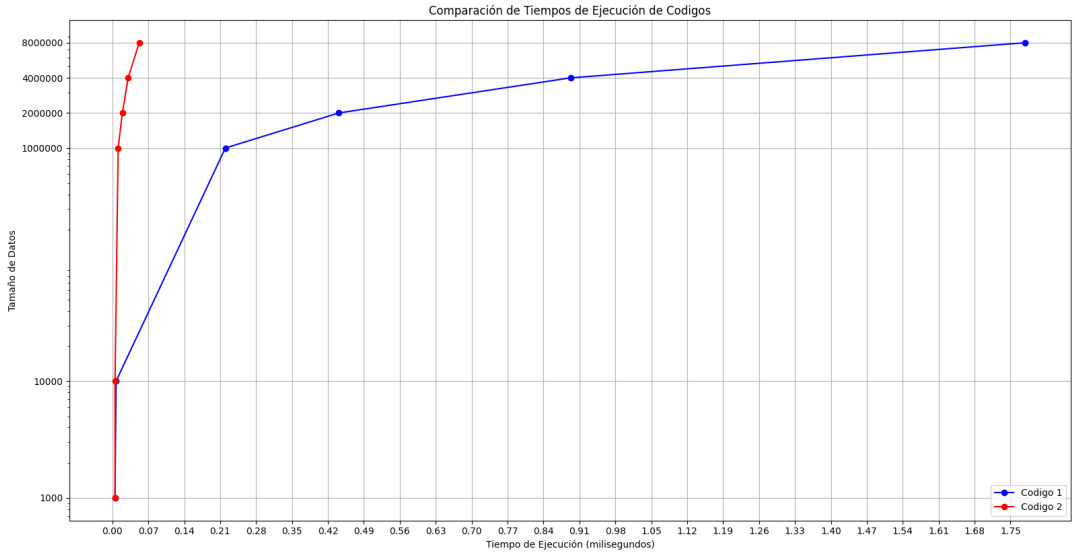

Figura 3.3: Tiempo mínimo de ejecución

Tamaño de datos	Tiempo Código 1	Tiempo Código 2
1000	0.00512	0.00528
10000	0.007168	0.005152
1000000	0.22016	0.011264
2000000	0.441344	0.019424
4000000	0.892928	0.03072
8000000	1.77869	0.052224

Cuadro 3.3: Tiempo mínimo de ejecución

4. Conclusiones

En conclusión, la segunda implementación es más eficiente, en comparación con la primera implementación, debido a que se divide en dos fases. En la primera de ellas, se calcula un histograma local para cada hilo. Mientras que en la segunda, se realiza la suma de los histogramas locales en un único histograma global final. Por consecuencia, la primera implementación es menos eficiente, debido a que se utiliza un único histograma compartido por todos los hilos.

Además, se ha podido observar que el tiempo de ejecución de la primera implementación es mayor que el tiempo de ejecución de la segunda. Donde en la primera, el tiempo de ejecución aumenta de forma exponencial a medida que el tamaño de datos aumenta. En comparación con la segunda implementación, donde el tiempo de ejecución aumenta de forma lineal a medida que el tamaño de datos aumenta.