Proyecto final Histograma realizado con CUDA

Arquitecturas Avanzadas y de Propósito Específico

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

4 de junio de 2024

Índice general

1. Introducción

Consiste 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 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 cada uno se encargue de ir al elemento que le corresponda V e incremente la caja correcta en el vector histograma H (posiblemente 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 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 que escriben 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 esta suma de la forma más paralela o eficiente, posiblemente utilizando el método de reducción. La implementación se encuentra en el archivo *histogram_2.cu*.

El código es el siguiente:

```
  __global__ void histogram() {
  // Declarar memoria compartida para el histograma local
  extern __shared__ int local_histogram[];

  // Inicializar el histograma local en memoria compartida
```

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

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

  // Realizar la reduccion de los histogramas locales en un unico histograma global
    final
  for (int j = tid; j < M; j += blockDim.x) {
    atomicAdd(&vector_H[j], local_histogram[j]);
  }
}
```

La segunda implementación se divide en dos fases.

3. Pruebas realizadas

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

4. Bibliografía

1. Ng Pante, C. (2001). Titulo. Nombre pagina web. Recuperado de `http://url.com`