UNIVERSIDAD DEL VALLE DE GUATEMALA
CC3069 - Computación Paralela y Distribuida
Sección 21
Ing. Miguel Novella Linares

Proyecto #3

Programación paralela con CUDA

Juan Solorzano, 18151
Mario Perdomo, 18029

GUATEMALA, 13 de abril de 2022

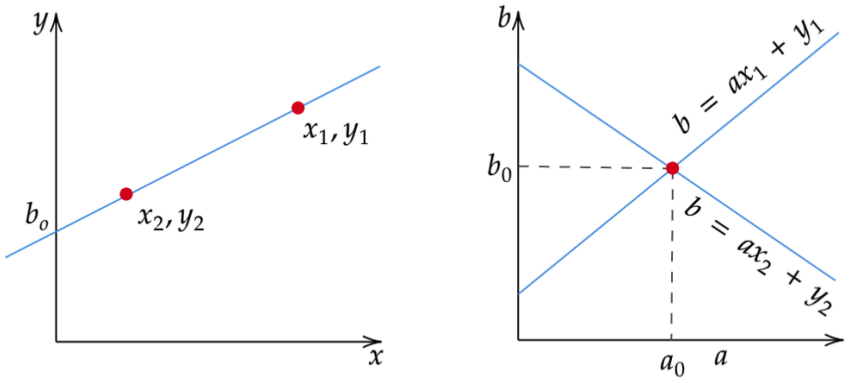
Marco Teórico

El algoritmo de Hough y su transformación es un método de extracción de características utilizado en el análisis de imágenes. La transformada de Hough puede utilizarse para aislar características de cualquier curva regular como líneas, círculos, elipses, etc. La transformada de Hough, en su forma más simple, puede utilizarse para detectar líneas rectas en una imagen.

Una transformada de Hough generalizada puede utilizarse en aplicaciones en las que no es posible una simple descripción analítica de las características. Debido a la complejidad computacional del algoritmo, programadores utilizan funcionalidades paralelas para obtener resultados más eficientes.

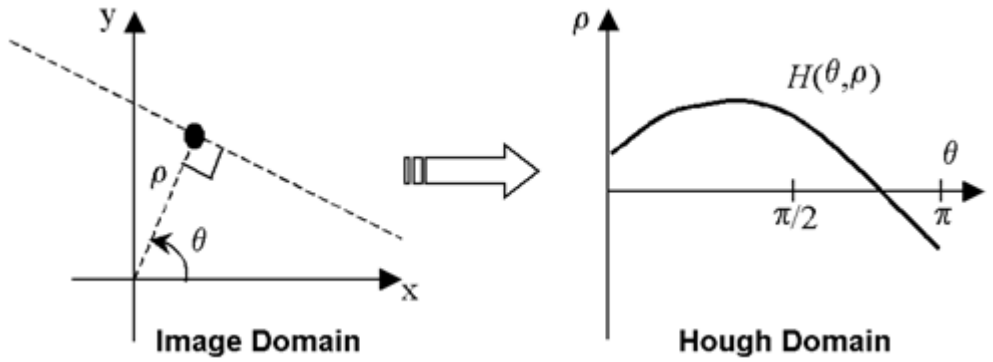
Algoritmo

Una línea recta es el límite más sencillo que podemos reconocer en una imagen. Varias líneas rectas pueden formar un límite mucho más complejo. Transformamos el espacio de la imagen en el espacio de hough. De este modo, convertimos una línea en el espacio de la imagen en un punto en el espacio de hough.



La ecuación de la línea en el espacio de la imagen es de la forma y = mx + c donde m es la pendiente y c es la intersección y de la línea. Esta recta se transformará en un punto de la forma (m, c) en el espacio de hough. Pero en esta representación m va al infinito para las líneas verticales. Por tanto, utilicemos en su lugar las coordenadas polares.

La transformación de Hough construye una matriz de histogramas que representa el espacio de los parámetros (es decir, una matriz M x N, para M valores diferentes del radio ρ y N valores diferentes del ángulo θ). Si tenemos una línea en el espacio de la imagen que se compone de varios segmentos o puntos que están cerca de la misma ecuación de la línea, tendremos muchas líneas de intersección en el espacio de hough.



Para cada combinación de parámetros, ρ y θ, encontramos entonces el número de píxeles distintos de cero en la imagen de entrada que caerían cerca de la línea correspondiente, e incrementamos la matriz en la posición (ρ,θ) adecuadamente.

En este proyecto, se estara implementando el programa CUDA houghBase(), dondellama al kernel con el número de bloque y el recuento de hilos adecuados en cada bloque con sus respectivos píxeles de una imágen.

El programa principal también inicializa la asignación de memoria tanto en el dispositivo como en el host, y el conjunto de datos se transmite del host al dispositivo utilizando la función cudaMemCpy().

1. El índice variable va de 0 al número máximo de hilos N.
2. Los puntos de datos se denotan con los índices r[índice], xCoord e yCoord.
3. El acumulador se construyó utilizando el factor de escala y el binning adecuados.
4. Para evitar los racing conditions en la versión de shared memory, el acumulador se desarrolla con la función CUDA AtomicAdd().

Preguntas

- Respecto a las variables xCoord y yCoord, explicar que se está realizando en esas operaciones y porque se calcula de tal forma.

  xCoord e yCoodr son las coordenadas correspondientes al plano. Al considerar el plano xy, denominado espacio de parámetros, se obtiene una única recta para cada punto; por ejemplo, si consideramos dos puntos (xi,yi) y (xj,yj), cada punto tendrá asociada una única recta, que al intersectarse darán los parámetros de x e y de la recta que contiene los puntos colineales de (xi,yi) y (xj,yj).

- En un párrafo describa cómo se aplicó la memoria Constante a la versión CUDA de la Transformada. Incluya sus comentarios sobre el efecto en el tiempo de ejecución.

  Se agregó memoria constante al utilizar cudaMemcpyToSymbol y declarar las variables __constant__ float d_Cos[degreeBins] y __constant__ float

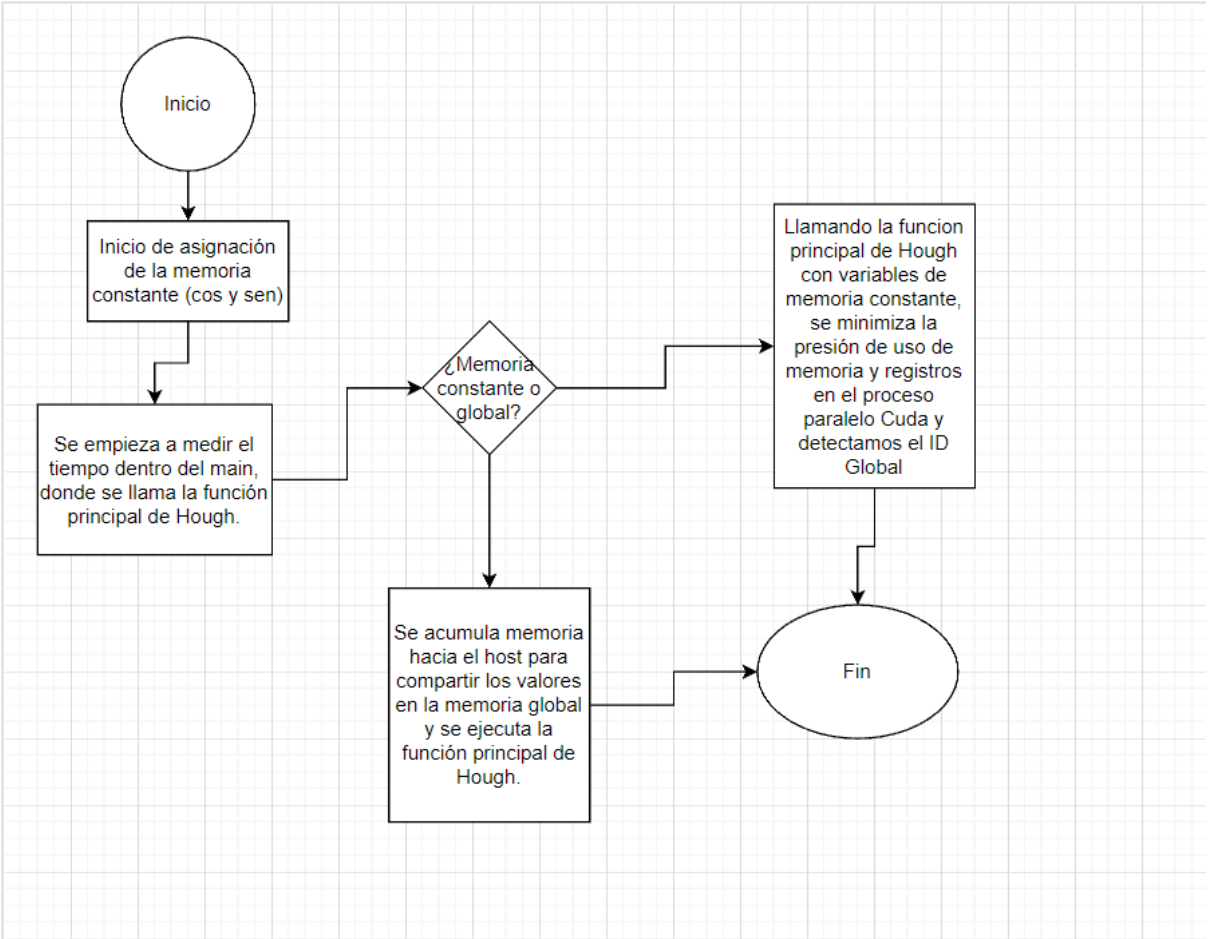
d_Sin[degreeBins]. Al utilizar __constant__ se está diciendo que las variables estarán en memoria constante.
Como se ve en la bitácora de tiempos, los métodos tienen un efecto muy similar. Al calcular los tiempos promedios para memoria constante y global se encontró que el uso de memoria global es más rápido por una diferencia de solo 0.004.

- En un párrafo describa cómo se aplicó la memoria compartida a la versión CUDA de la transformada. Incluya sus comentarios sobre el efecto en el tiempo de ejecución.

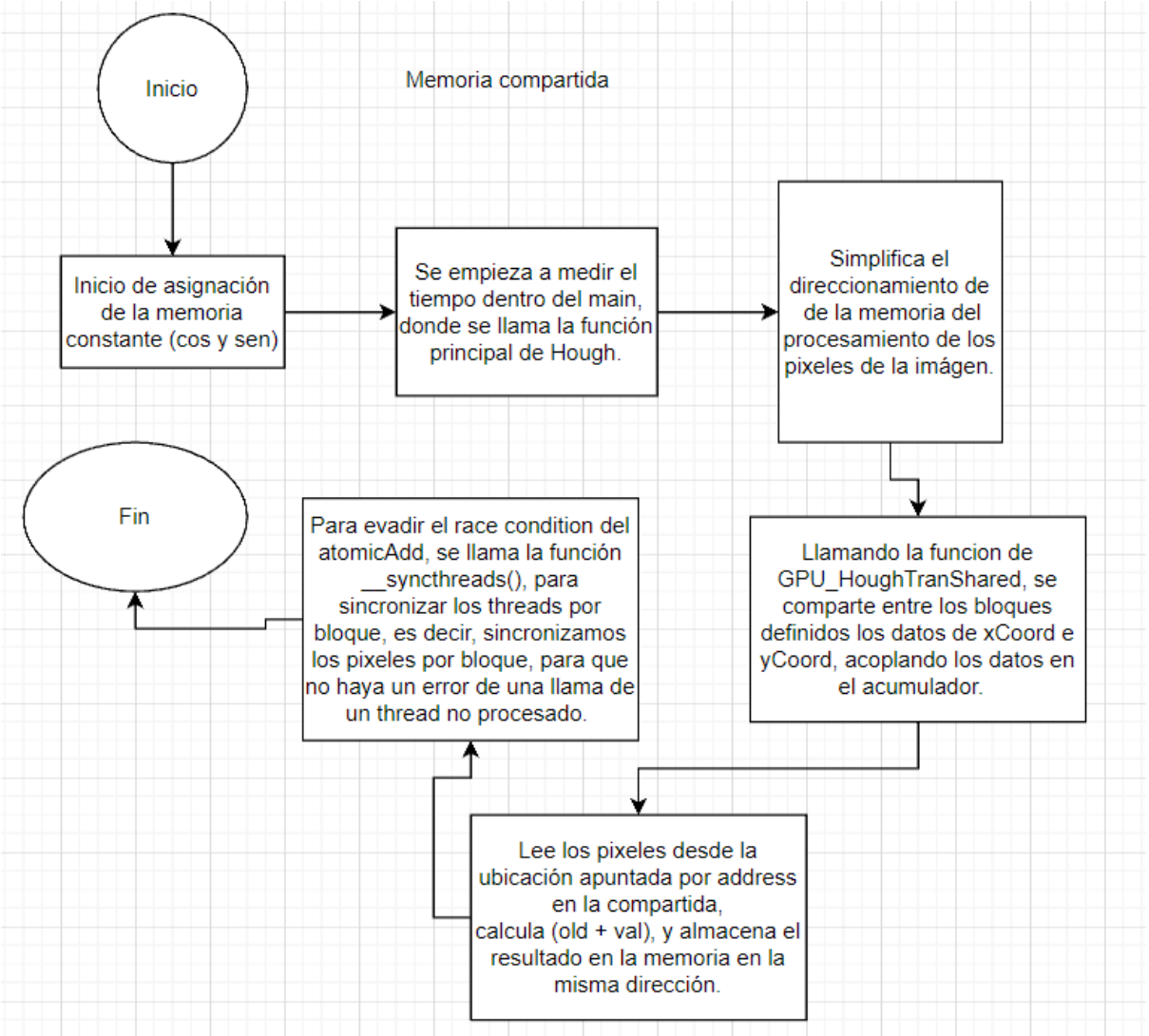
  Para la memoria compartida se creó la función GPU_HoughTranShared. Dentro de esta función se asignó la memoria compartida estática en __shared__ int localH. Luego se inicializa la memoria compartida en el kernel con el for loop donde se asigna el valor 0 al arreglo localH. Luego se realiza el mismo proceso que se realizó anteriormente asegurándonos de hacer syncthreads antes y después de la lógica. Finalmente se realiza la operación atómica para actualizar el valor local al global. Esta vez sí se ve una diferencia en los tiempos de ejecución siendo este alrededor de 1.4 veces más tardado con un tiempo promedio de 2.170067.

Diagramas

Memoria Constante y Global



Memoria Compartida



Resultados

Ejecución del programa

Memoria global

```
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
 Time elapsed during the Hough Formula: 1.566976

student20-24@ip-172-31-8-99:~$
```

Memoria constante

```
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
 Time elapsed during the Hough Formula: 1.577984

student20-24@ip-172-31-8-99:~$
```

Memoria compartida

```
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
 Time elapsed during the Hough Formula: 2.166816

student20-24@ip-172-31-8-99:~$
```

Iteración	Global (tiempo en s)	Constante (tiempo en s)	Compartida (tiempo en s)
1	1.566976	1.577984	2.166816
2	1.582208	1.566720	2.169856
3	1.587232	1.576192	2.166208
4	1.568768	1.588448	2.166496
5	1.564704	1.584864	2.174976
6	1.570464	1.570304	2.165632
7	1.566656	1.572864	2.170368
8	1.578976	1.583232	2.164640
9	1.572704	1.579872	2.168736
10	1.576736	1.575648	2.186944
Promedio	1.573542	1.577613	2.170067

Conclusión

En este proyecto se encontraron retos que se relacionaban más al descuido por parte del grupo. Se siguió la guía proveída pero en la segunda entrega no colocamos la llamada a cudaMemcpyToSymbol lo cual causó resultados erróneos en los tiempos de ejecución para memoria constante. Esto se soluciono al agregar la llamada adecuadamente y cambiar la bitácora de resultados a los valores nuevos.

Luego de utilizar los tres tipos de memoria en CUDA, se encontró que la memoria global obtiene los menores tiempos de ejecución a comparación de la memoria constante y compartida.

Referencias Bibliográficas

- Harris, M. (2012). How to Implement Performance Metrics in CUDA C/C++. Extraido de https://developer.nvidia.com/blog/how-implement-performance-metrics-cuda-cc/
- Lee, S. (2020). Lines Detection with Hough Transform. Extraído de: https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549
- Harris, M. (2013). Using Shared Memory in CUDA C/C++ (Extraído de: https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/