Informe Algoritmo de Dijkstra con CUDA

Comparación entre modelos de GPU y secuencial para aplicar Algoritmo de Dijkstra

Rafael Pinzón Rivera
Christian Patiño Grisales

Universidad Tecnológica de Pereira
Pereira
Octubre 2015

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El algoritmo de dijkstra determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos. Algunas consideraciones:

1. Si los pesos de mis aristas son de valor 1, entonces bastará con usar el algoritmo de BFS.
2. Si los pesos de mis aristas son negativos no puedo usar el algoritmo de dijsktra, para pesos negativos tenemos otro algoritmo llamado Algoritmo de Bellmand-Ford.

1.1. Desarrollo del algoritmo

Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértices evaluaremos sus adyacentes, como dijkstra usa una técnica greedy – La técnica greedy utiliza el principio de que para que un camino sea óptimo, todos los caminos que contiene también deben ser óptimos- entre todos los vértices adyacentes, buscamos el que esté más cerca de nuestro punto origen, lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás. Después escogemos al siguiente más cercano (con las distancias ya actualizadas) y repetimos el proceso. Esto lo hacemos hasta que el vértice no utilizado más cercano sea nuestro destino. Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vértice se le conoce como relajación (relaxation).

Dijkstra es muy similar a BFS, si recordamos BFS usaba una Cola para el recorrido para el caso de Dijkstra usaremos una Cola de Prioridad o Heap, este Heap debe tener la propiedad de Min-Heap es decir cada vez que extraiga un elemento del Heap me debe devolver el de menor valor, en nuestro caso dicho valor será el peso acumulado en los nodos.

1.2. Complejidad

Orden de complejidad del algoritmo:

$$O(|V|2 + |A|) = O(|V|2) \tag{1.1}$$

sin utilizar cola de prioridad,

$$O((|A| + |V|)log|V|) = O(|A|log|V|) \tag{1.2}$$

utilizando cola de prioridad (por ejemplo un montículo). Por otro lado, si se utiliza un Montículo de Fibonacci, sería

$$O(|V|log|V| + |A|) \tag{1.3}$$

.

Podemos estimar la complejidad computacional del algoritmo de Dijkstra (en términos de sumas y comparaciones). El algoritmo realiza a lo más n-1 iteraciones, ya que en cada iteración se añade un vértice al conjunto distinguido. Para estimar el número total de operaciones basta estimar el número de operaciones que se llevan a cabo en cada iteración. Podemos identificar el vértice con la menor etiqueta entre los que no están en Sk realizando n-1 comparaciones o menos. Después hacemos una suma y una comparación para actualizar la etiqueta de cada uno de los vértices que no están en Sk. Por tanto, en cada iteración se realizan a lo sumo 2(n-1) operaciones, ya que no puede haber más de n-1 etiquetas por actualizar en cada iteración. Como no se realizan más de n-1 iteraciones, cada una de las cuales supone a lo más 2(n-1) operaciones, llegamos al siguiente teorema.

TEOREMA: El Algoritmo de Dijkstra realiza O(n2) operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértices.

Capítulo 2

Pruebas y Gŕaficas

2.1. Image 1

Tamaño			**GPU**			**Promedio**
8	0,00035	0,00035	0,00035	0,00036	0,00035	0,00035
12	0,00105	0,00105	0,00105	0,00105	0,00105	0,00105
16	0,00279	0,00279	0,00279	0,00279	0,00279	0,00279
20	0,00629	0,00630	0,00629	0,00629	0,00629	0,00629
24	0,00985	0,00986	0,00986	0,00985	0,00985	0,00985
100	0,58689	0,58690	0,58688	0,58689	0,58688	0,58689

Cuadro 2.1: Dijkstra GPU. tiempo: segundos.

Tamaño			**Secuencial**			**Promedio**
8	0,00003	0,00003	0,00004	0,00003	0,00004	0,00004
12	0,00004	0,00004	0,00004	0,00004	0,00004	0,00004
16	0,00006	0,00006	0,00006	0,00005	0,00005	0,00005
20	0,00007	0,00007	0,00007	0,00007	0,00008	0,00007
24	0,00010	0,00010	0,00010	0,00010	0,00010	0,00010
100	0,00222	0,00295	0,00296	0,00294	0,00156	0,00252

Cuadro 2.2: Dijkstra Secuencial. tiempo: segundos.

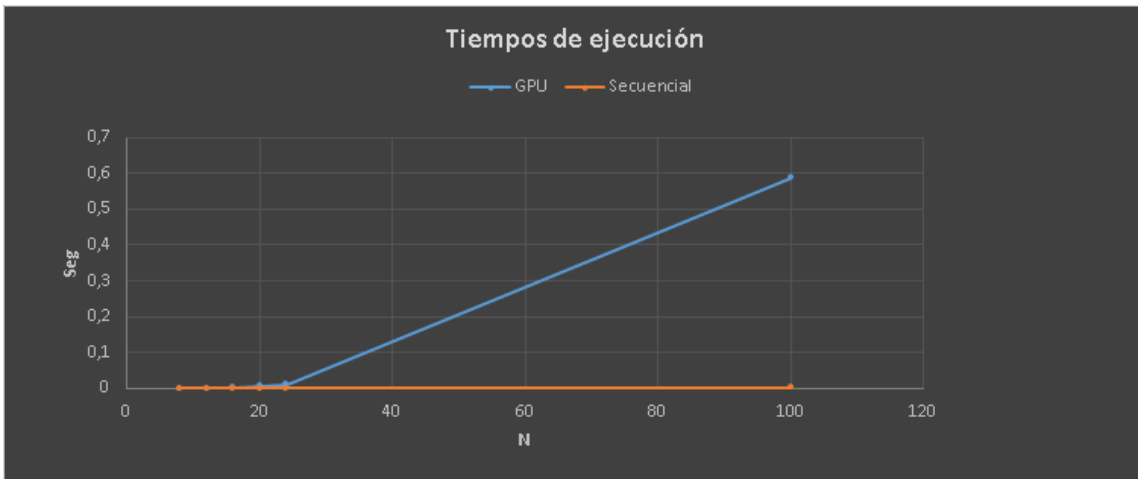

Figura 2.1: Gráfica de Tiempos

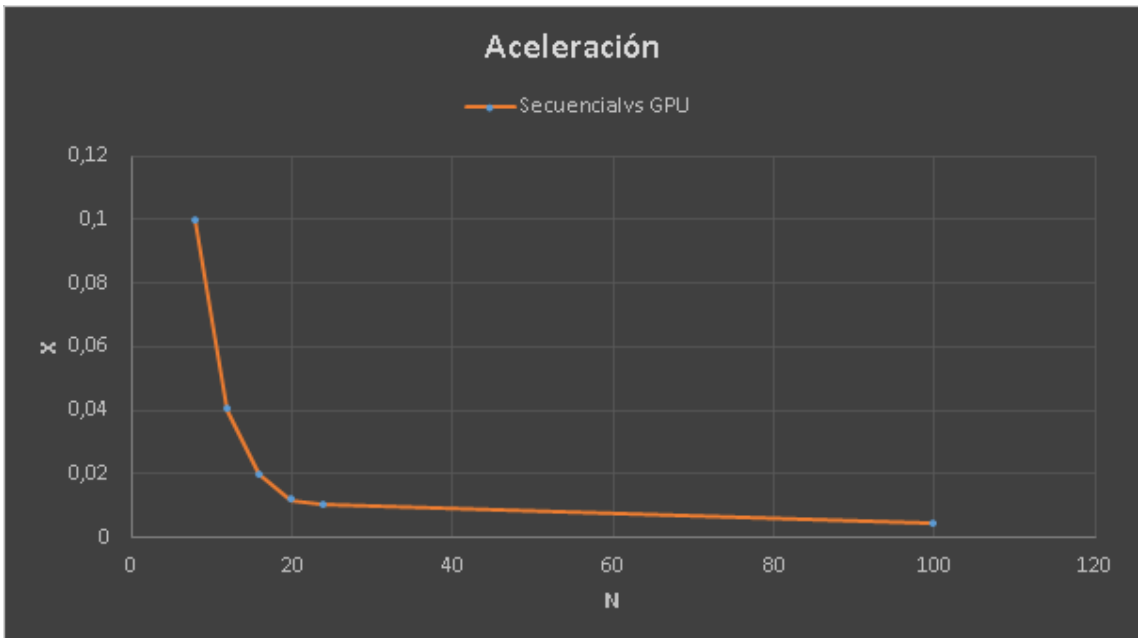

Figura 2.2: Gráfica de Aceleración: Secuencial VS GPU

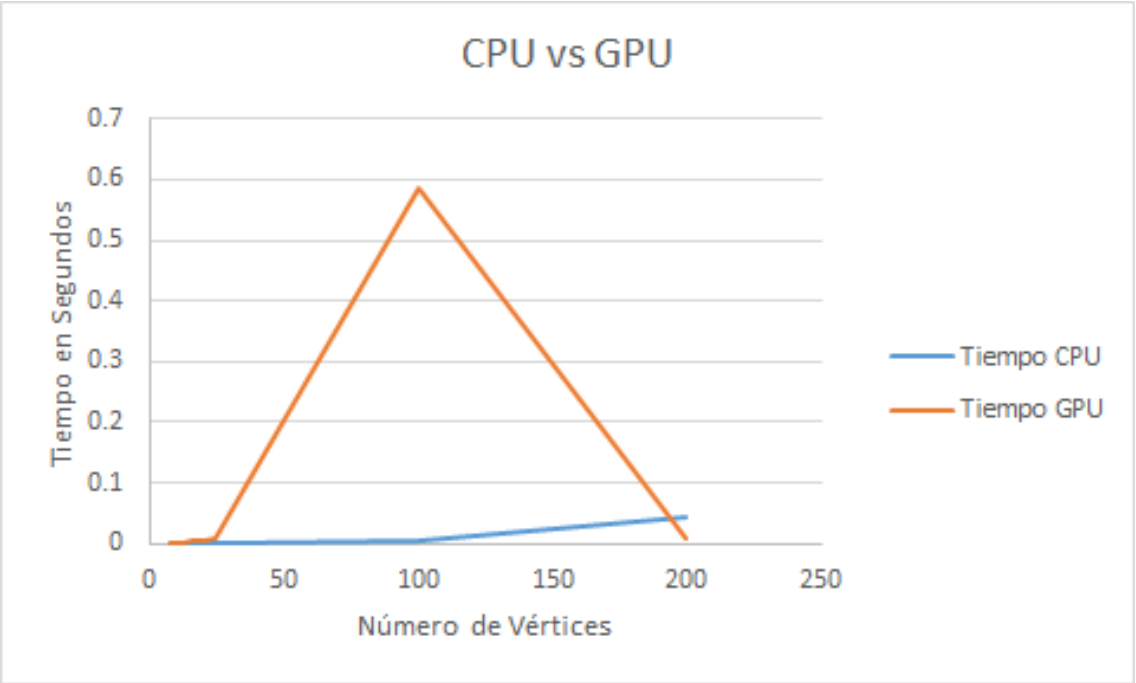

Figura 2.3: Gráfica de Tiempos con GPU Constante: Secuencial VS GPU

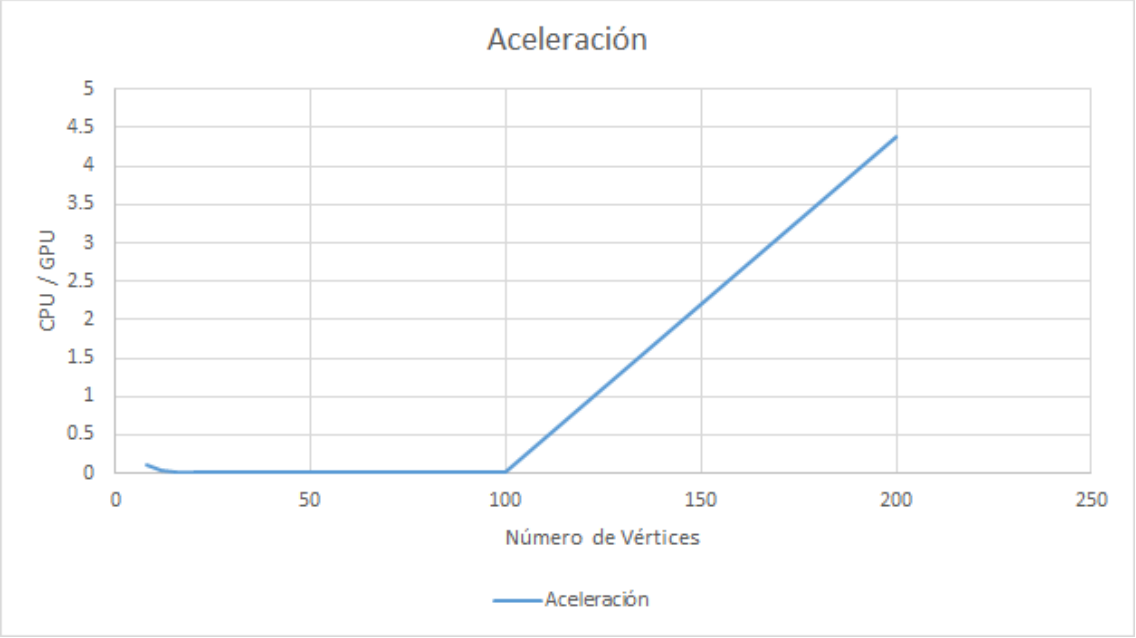

Figura 2.4: Gráfica de Aceleración con GPU Constante: Secuencial VS GPU

Capítulo 3

Conclusiones

1. Es necesario manejar una cantidad mayor de nodos para que se pueda apreciar la mejora en los tiempos de ejecución del algoritmo paralelo.

2. Para poder paralelizar alguna parte del algoritmo es necesario saber que los datos no generan problemas de concurrencia, es decir que estos no sean dependientes del procesamiento de los anteriores datos.

3. Al momento de usar memoria constante para mejorar el algoritmo paralelo hay que tener en cuenta que solo se pueden almacenar datos que no vayan a cambiar en ningun momento de la ejecuación.