Cálculo de las rutas más corta entre los nodos de un grafo usando el algoritmo A diamante.

Calculation of the shortest routes between the nodes of a graph using the algorithm A diamond.

Autor 1: Luis David Giraldo Grajales Autor 2: Juan Sebastian Herrera Giraldo

*Ingeniería de sistemas y computación, Universidad Tecnológica de pereira, Pereira, Colombia*

Correo-e: [juseherrera@utp.edu.co](mailto:juseherrera@utp.edu.co), [davidgigra@utp.edu.co](mailto:davidgigra@utp.edu.co)

***Resumen*— A partir de una serie de archivos cuyo contenido son los pesos y costos que se tienen de un grafo. Se hará un análisis y comparación entre diferentes resultados obtenidos de la ejecución del algoritmo A diamante, de manera secuencial y paralela.**

**Estos resultados serán comparados con los resultados obtenidos de otras personas que han ejecutado dicho algoritmo en otras unidades de cómputo.**

***Palabras Clave* — Computación de alto desempeño, grafo, A diamante, nodos, aristas, HPC, hilos, paralelo, secuencial.**

***Abstract*— From a series of files whose content is the weights and costs of a graph. An analysis and comparison will be made between different results obtained from the execution of the A diamond algorithm, in a sequential and parallel manner.**

**These results will be compared with the results obtained from other people who have executed said algorithm in other computing units.**

***Key Word* —High performance computing, graph, A diamond, nodes, edges, HPC, threads, parallel, sequential.**

1. INTRODUCCIÓN

Computacionalmente un grafo se puede representar como una matriz de adyacencias, donde se guardan los pesos o costos que se tiene de ir de un nodo a otro. El cálculo de la ruta más corta puede ser encontrado con algoritmos como Dijkstra[1], Bellman-Ford[2], A diamante, entre otros más.

A pesar de que estos algoritmos permiten calcular el menor costo, no son cien por ciento eficientes, debido a que en algunas implementaciones los grafos que se usan para modelar los nodos y aristas son muy grandes y requieren de muchos recursos computacionales principalmente tiempo y memoria.

Por esta razón se desea utilizar dichos algoritmos mencionados anteriormente y la computación de alto desempeño, para aprovechar al máximo el beneficio que nos brindan las GPUs de procesar mucha información de manera paralela, permitiendo mayor eficiencia en los tiempos y costos computacionales.

Este artículo se enfocará en el algoritmo A diamante, cuya representación inicial es una matriz dispersa (Sparse Matrix)[3][4], y a medida que se va operando sobre ella, poco a poco deja de ser dispersa y pasa a ser una matriz densa.

Como resultado final, esta matriz densa contiene todos los costos que se tienen al ir desde un Nodo A hasta un Nodo B.

La matriz densa es un claro resultado equivalente al entregado por el algoritmo de Dijkstra.

1. CONTENIDO
2. Recolección de datos.

Los datos utilizados en este análisis fueron archivos obtenidos de DIMACS (the Center for Discrete Mathematics and Theoretical Computer Science), cuyo contenido es: Cantidad de nodos, adyacencias que hay entre un nodo a otro, y sus respectivos costos.

1. Aplicación del algoritmo sobre los datos.

Una vez se cargan los datos en memoria, se procede a encontrar el mínimo que hay entre la suma de los elementos de la fila n,m y la columna m,n. como se muestran en la figura 1.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | c |  | a | b | c |
| d | e | f | \* | d | e | f |
| g | h | i |  | g | h | i |

figura 1.1 Matrices NxN.

Teniendo como resultado otra matriz, con los mínimos calculados.

=

|  |  |  |
| --- | --- | --- |
| min (a+a, b+d, c+g) | min(a+b, b+c, c+h) | min(a+c, b+f, c+i) |
| min(d+a, e+d, f+g) | min(.., .., ..) | min(.., .., ..) |
| min(.., .., ..) | min(.., .., ..) | min(g+c, h+f, i+i) |

figura 1.2 Calcular el primer del algoritmo.

En la figura 1.2 se observa cómo se calcula el primer resultado de la primera iteración, ahora imaginemos una matriz suficientemente grande como para realizar sobre ella n iteraciones, para calcular su , esto implica mucha complejidad y consumo de recursos computacionales.

¿Qué solución matemática permite reducir la complejidad de este algoritmo?.

Si recordamos propiedades de las matrices, podemos decir:

sea A una matriz.

es equivalente a tener , y del mismo modo es equivalente a tener .

podemos afirmar que es igual a

teniendo esto claro, podemos aplicarlo al algoritmo y reducir su complejidad de cómputo, a una complejidad

Considerando esta complejidad, se puede paralelizar este algoritmo y mostrar su gran rendimiento tanto de memoria como de tiempos.

1. Resultados Obtenidos.

**Procesador i5-4570 CPU @ 3.20 Ghz 4 cores**

**Algoritmo secuencial:** 51.484833 minutos.

**Algoritmo paralelo:**  16,157616667 minutos.

**Procesador intel i3 @ 2.4 Ghz 2 cores**

**algoritmo secuencial:** 180 minutos.

**algoritmo paralelo:** indefinido.

**Cluster: CIDT**

1. Análisis de los datos.

Anteriormente se muestran diferentes resultados, probados en diferentes unidades de cómputo y cuyas matrices no poseen las mismas dimensiones.

El primer resultado obtenido, fue con una matriz que simula la ciudad de nueva york y contiene 264346 nodos y 733846 arcos. Podemos observar que es una matriz medianamente grande y cuya implementación del algoritmo en paralelo fue muy buena, obteniendo un **Speedup** de 3.1864X.

El segundo resultado obtenido, fue con una matriz de 250 nodos y 31336 arcos, estos no representan ninguna ciudad, pero son datos válidos para evaluar el algoritmo.

Como se puede observar el valor de la **Speedup** no se puede calcular debido a que la implementación tiene mucha responsabilidad en la paralelización de este algoritmo.

El tercer resultado obtenido se realizó sobre la CPU del cluster, de manera secuencial, con un grafo de 1000 nodos y 4650 arcos, el cual no representa ninguna ciudad, pero representa un grafo pequeño. El cluster cuenta con muy buen equipo de cómputo, y permitió realizar el algoritmo en un tiempo secuencial de 3, 280442 minutos.

¿Que indica este valor de **Speedup**?

Este valor indica qué tan rápido se realiza la aplicación del algoritmo respecto a la versión secuencial. Un ejemplo claro de SpeedUp es cuando se trabaja con entrenamiento de redes neuronales, y dichos entrenamientos se demoran días e incluso semanas. En pocas palabras este valor indica que número de veces es más rápido se hace una operación con respecto a la otra, y se representa #X.

1. Comparación con otros artículos.

Apesar de haber muy pocos artículos relacionados con el algoritmo A diamante, una articulo como lo es Efficient Sparse Matrix-Vector Multiplication on cuda[5], mencionan como paralelizar una matriz dispersa en cuda. Allí mencionan diferentes maneras de como optimizar dichas matrices, mientras que en este artículo

1. CONCLUSIONES

- Muchos algoritmos a pesar de ser paralelizables no obtienen los resultados esperados; como prueba de ello en el análisis de los 250 nodos, el resultado fue bastante malo, debido a la forma de cómo se implementa el algoritmo.

- Todo computador tiene recursos limitados, y por ello una buena implementación de algoritmos y código permiten sacar al máximo el rendimiento de las máquinas.

REFERENCIAS

1. Mehlhorn, Kurt; Sanders, Peter (2008). "Chapter 10. Shortest Paths" (PDF). Algorithms and Data Structures: The Basic Toolbox. Springer. doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.1: The Bellman-Ford algorithm, pp.588–592.
3. Bank, Randolph E.; Douglas, Craig C. "Sparse Matrix Multiplication Package".
4. NVIDIA Corporation. NVIDIA CUDA Programming Guide, June 2008. Versión 2.0.
5. Bell Nathan, Garland Michael, “Efficient Sparse Matrix-Vector Multiplication on CUDA, December 2008.