Práctica CUDA

Daniel González Alonso Santos Ángel Pardo Ramos

27 de mayo de 2016

1. Introducción

A continuación, presentaré las mejoras que hemos realizado a la versión secuencial proporcionada para poder ser ejecutado de forma paralela bajo Cuda.

Partiendo de que para las entregas anteriores de OpenMP y de MPI ya habíamos realizado mejoras en la versión secuencial, hemos decidido aplicar esa serie de mejoras en esta entrega, aunque no han podido ser todas las deseadas debido a la escasez de tiempo para la realización de esta práctica. Recordaremos brevemente cuales fueron esas mejoras:

- Diseñamos una nueva función para actualizar el mapa para la primera antena recibida por los argumentos, debido a que no es igual que en el resto.
- Modificamos la forma en que se actualiza el mapa. Vimos que los valores del mapa siguen un patrón con forma de rombo, por lo que se puede reducir el número de veces que comparamos valores y calculamos la distancia de manhattan, pudiendo así optimizar notablemente está función.

2. Cambios para la Práctica de Cuda

Lo primero que hicimos, fue reservar memoria para el puntero del mapa, el mapa solo es accesible desde la GPU, es decir, en el Host no disponemos de un mapa.

Acto seguido, comenzamos a mejorar la función actualizar, la cual, como hemos descrito anteriormente, aprovechaba el patrón en forma de rombo para hacer menos iteraciones. En nuestra implementación, empleamos únicamente un hilo por cada lateral del rombo, de forma que en cada iteración solo tiene que calcular la distancia de manhattan una vez. Cabe destacar que a la hora de iterar por el lateral del rombo, cada hilo comienza y acaba en las intersecciones del rombo con el mapa para evitar iteraciones vacías. Al tener solo un hilo por cada lateral, tenemos un grid de 1x1 bloques y un solo bloque de 4x4 hilos.

Respecto a la función para calcular la distancia de manhattan, ésta solo es accesible desde el device.

Para la función que se encarga de calcular el máximo, se eliminó por completo todo el código de las versiones anteriores, y se imitó el funcionamiento de un ejercicio propuesto en el laboratorio. La función se basa en hacer una reducción de forma paralela sobre un vector compartido. Para usar esta función, tenemos un grid de cuatro bloques, cada uno con 512 hilos. Estos hilos ejecutan la función `reduce_kernel` en la que copian una posición del mapa a memoria shared en función de su identificador y de su número de bloque. Posteriormente, únicamente los hilos pares comparan el elemento correspondiente a su identificador con el elemento siguiente. En las siguientes iteraciones, actúan la mitad de hilos que en la iteración anterior, sucesivamente, hasta que al final la última reducción la lleve a cabo solo el hilo 0. Este valor, posteriormente, es copiado a una variable compartida.

La elección del número de hilos y de bloques ha sido siempre un número múltiplo de dos, ya que siempre funcionan mejor con estas cantidades. Tampoco hemos alcanzado un grado de desarrollo suficientemente avanzado como para discutir cual era el mejor número de hilos y de bloques que se debían utilizar, a lo cual nos basamos en los datos que venían en ejercicios del laboratorio, siendo el número de hilos 512 y de bloques 4 en la función de calcular el máximo, números que se podían emplear en la arquitectura Pre-Fermi de los ordenadores de la escuela.

El código no ha podido ser subido a LeaderBoards debido a que presenta ciertos errores que no han podido ser corregidos para la entrega a tiempo.