Luis Miguel Benitez Ruiz

00994361

Reporte 3

Design and Evaluation of a Parallel K-Nearest Neighbor Algorithm on CUDA-enabled GPU.

El algoritmo KNN es una técnica de clasificación muy común. Una de sus ventajas es que no requiere entrenamiento y es muy sencillo incluir nuevas observaciones a las ya conocidas. La idea básica es, para un punto cuya clase no se conoce, encontrar sus *k* vecinos más cercanos, y asignarlo a la clase que sea más común entre este conjunto de vecinos. Después de esto, esta nueva observación puede tomarse en cuenta para futuras observaciones no conocidas.

La distancia puede medirse de muchas formas, como euclidiana, manhattan o Mahalanobis, por dar unos ejemplos. La desventaja de este algoritmo es que, para encontrar a los *k* vecinos más cercanos a una observación desconocida *r*, es necesario evaluar la distancia entre *r* y el resto de las observaciones existentes. Puede utilizarse una heurística, pero incluso en este caso la evaluación debe hacerse con cada una de las observaciones conocidas, lo cual puede volverse muy costoso cuando la cantidad de información es muy grande.

Los autores proponen en este artículo un algoritmo paralelo utilizando CUDA. KNN puede beneficiarse mucho de la paralelizacion en GPU, porque la operación para evaluar la distancia entre dos puntos es independiente de un par de puntos a otro, por lo que puede ejecutarse en paralelo sin que haya problemas por la concurrencia. La función de evaluación de distancia también suele ser sencilla y generalmente es completamente algebraica, por lo que no se necesita la alta capacidad para manejar instrucciones condicionales de un CPU.

Para esto se utilizan dos kernel, uno de cálculo de distancia y otro de selección.

El primer kernel es el de cálculo de distancia, que esencialmente se encarga de calcular la distancia entre un punto de referencia *p* y todos los otros puntos conocidos. Los autores consideraron como lo más importante para el rendimiento el maximizar la concurrencia en el cálculo de las distancias, y minimizar el acceso a la memoria global del dispositivo.

El kernel de selección ejecuta el cálculo de las distancias. Cada bloque carga en su memoria compartida un subconjunto único de los objetos conocidos (de referencia= con los que se tiene que comparar *p*. Cada hilo t_i va a calcular la distancia (d_i) entre el objeto de referencia *i* y *p*. Debe de haber un objeto de referencia por cada hilo en el bloque.

Después de esto se ejecuta el kernel de selección. El primer paso es que cada thread t_i , que en el kernel anterior calculo la la distancia d_i, compara d_i contra el resto de las distancias calculadas en ese bloque, y obtiene el rango de d_i. Todos los threads ejecutan esto de forma concurrente, así que todos los rangos se calculan en el mismo tiempo. Los hilos de cada bloque almacenan esta información en colas ordenadas en la memoria compartida.

Después de este paso, un hilo revisa cada una de las colas, elige el elemento más cercano y lo saca de la cola, este es el vecino más cercano. Como las colas están ordenadas, este hilo solo tiene que

hacer *m* comparaciones, donde *m* es el número de bloques. Después de *k* comparaciones, se habrán encontrado los *k* vecinos más cercanos a *p*.

Después de este kernel de selección, la última tarea es asignar la clase de *p* en base a la clase más común entre sus *k* vecinos más cercanos. Debido a que *k* suele ser pequeño, esta tarea se ejecuta en el CPU.

Para sus experimentos, se utiliza una GPU NVIDIA Tesla C1060, que cuenta con 240 cores y 4 GB de memoria RAM en el dispositivo. La comparación se realiza contra un algoritmo secuencial implementado en C.

Para la pruebas iniciales, ejecutan el algoritmo paralelo con 262 144 objetos de referencias, generados algorítmicamente, y calculando 7 vecinos para un nuevo punto. Para la prueba inicial los objetos son unidimensionales, y para pruebas posteriores se duplican sus dimensiones hasta llegar a 64 dimensiones en la penúltima prueba. La última prueba es con objetos de 80 dimensiones. En esta prueba el tiempo de ejecución muestra un crecimiento lineal, y es de solo 8 ms en la prueba de 80 dimensiones, mientras que era de 2 ms en la prueba con objetos unidimensionales.

Más adelante se describe una prueba con datos reales. Se utilizó una base de datos de 32 768 elementos, representando información sobre partículas en experimentos en un colisionador de alta energía. Cada objeto tiene 65 dimensiones. En este caso lo que se varió entre cada prueba fue el numero *k* de vecinos a comparar, comenzando en 16 y duplicándolo (aproximadamente) hasta llegar a 1048 vecinos.

La mejora más alta que se alcanza es de 42.49, donde la ejecución serial tarda 347.15 ms, mientras que la paralela tarda 8.17 ms. De acuerdo a los autores esto indica que este algoritmo es apropiado para resolver problemas de gran escala. Esto se debe en parte a que se utiliza coalescencia de memoria y streams para tratar de aprovechar al máximo el ancho de banda de memoria disponible en la GPU.

Referencias

Lian, S., Liu, Y., Wang, C., & Jian, L. (2010). Design and Evaluation of a Parallel K-Nearest Neighbor Algorithm on CUDA-enabled GPU. 53-60.