Comparativa del Algoritmo Quicksort

Arquitecturas Avanzadas de Computo

Bruno Baruffaldi

Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

1. Objetivo:

El siguiente informe es un trabajo final correspondiente al curso Arquitecturas avanzadas de cómputo dictado durante la escuela de ciencias informáticas en la uba en el 2018, cuyo objetivo es realizar una comparación entre los tiempos de ejecución de un algoritmo de ordenamiento implementado sobre distintas arquitecturas y tratando de aprovechar al máximo la paralelización de tareas.

2. Introducción:

Existen un conjunto de distintas arquitecturas que fueron creadas para resolver tareas específicas, pero actualmente pueden llegar a ser útiles para la resolución de problemas de cualquier índole. Dichos problemas, dependiendo de la cantidad de operaciones que realizan, pueden llegar a tardar un tiempo prolongado para su resolución. Sin embargo, en algunos casos estos tiempos pueden ser ampliamente mejorados con la programación en paralelo. Existen diferentes formas de ejecutar programas en paralelo y la mayoría depende del hardware que se utiliza.
En un comienzo la programación en paralelo era simulada debido al alto costo del hardware, pero aun así se obtenían mejores resultados. Hoy en dia, en la mayoría de los dispositivos informáticos que utilizamos cotidianamente es habitual que posean un procesador multinúcleo, que combina dos o más microprocesadores en un solo circuito integrado, lo que permite que el programador cuente con una forma de paralelismo real a nivel de threads. Para facilitar al programador hacer uso de esta ventaja se desarrollaron distintas interfaces de programación como OpenMP que permite aadir concurrencia a programas desarrollados en C,C++ y Fortran.
Otra arquitectura interesante son las unidades de procesamiento gráfico o GPU, que son dispositivos desarrollados para el procesamiento de gráficos u operaciones de punto flotante que cuentan con una gran cantidad de núcleos. Con el tiempo, estos dispositivos empezaron a ser utilizados para el cálculo científico de propósito general (en inglés GPGPU - General-Purpose Computing on Graphics Processing Units) dado que de esta forma muchas aplicaciones mejoraron enormemente sus tiempos de ejecución. Por estos motivos, empresas como NVIDIA introdujeron en sus tarjetas gráficas la arquitectura CUDA para el cálculo paralelo de propósito general.

3. Algoritmo:

En este informe se decidió utilizar el algoritmo de ordenamiento Quicksort y comparar sus tiempos de ejecución de distintas implementaciones que buscan explotar las ventajas de diferentes arquitecturas.
El algoritmo quicksort es uno de los algoritmos de ordenamiento más eficientes y funciona de la siguiente forma:

1. Elige un valor del arreglo al cual llamaremos pivote.
2. Resitua a los demás elementos de la lista a cada lado del pivote, insertando a todos los elementos menores que el pivote del lado izquierdo y a los mayores del lado derecho.
3. De esta forma, las lista inicial queda separada en dos sublistas, una con los elementos menores que el pivote y otra con los elementos mayores por lo cual pueden ser ordenadas de forma independientes. Notar que el pivote se encuentra en su posición correspondiente respecto del arreglo ordenado.
4. El algoritmo repite el proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado el proceso el arreglo está ordenado.

Este algoritmo fue creado por el científico británico Tony Hoare, quien sugirió que cuando la lista de números a ordenar no es demasiado grande es mejor utilizar otro algoritmo.
La programación en paralelo consiste en dividir un problema en subproblemas(lo más independientes posibles), con el fin de optimizar el tiempo de ejecución del programa. Esto es exactamente lo que hace el algoritmo quicksort y es lo que se trató de explotar en las distintas implementaciones para reducir el tiempo de ejecución.

4. Experimentos:

Los programas utilizados están disponibles en un repositorio de Github y fueron ejecutados en una máquina con las siguientes características:
CPU
GPU
Para obtener estos resultados se fueron modificando las macros de los programas para establecer el número de elementos a ordenar, la cantidad de hilos y otros valores para tratar de sacar el mayor provecho al algoritmo.

La siguiente tabla muestra una comparativa entre los tiempos de las distintas implementaciones en milisegundos respecto a la cantidad de elementos a ordenar.

Cantidad	Secuencial	OpenMp	Cuda
1,000	0.196	0.210	3.555
5,000	1.202	1.335	6.226
10,000	2.391	2.524	9.962
50,000	12.236	13.202	12.248
100,000	28.779	27.818	19.947
500,000	143.603	93.471	37.349
1,000,000	304.991	203.211	46.617
5,000,000	1,698.34	1,122.590	88.937
10,000,000	3,683.713	2,473.136	158.088
50,000,000	20,287.507	13,316.552	401.351
100,000,000	41,183.724	32,087.728	878.485
500,000,000	224,981.828	148,800.042	3,803.709
1,000,000,000	467,772.651	291,855.056	7,313.001

Tener en cuenta que los arreglos fueron creados con números pseudo-aleatorios y no sobre alguna distribución en particular.

5. Conclusión:

Como se observa en los experimentos cada implementación hace mejor o peor tiempo dependiendo del número de elementos que posea el arreglo.
Si la cantidad de elementos es pequea (menos de 100000), sería conveniente utilizar la versión secuencial del algoritmo sin ninguna optimización en cuanto a la paralelización pues el algoritmo corre lo suficientemente rápido y no pierde tiempo en la creación de nuevos hilos.
Si la cantidad de elementos pasa a ser un poco más grande (entre 500000 y 5000000) es útil utilizar varios núcleos del procesador para ejecutar varios hilos y paralelizar la operaciones.
Sin embargo, a si la cantidad de números es muy grande la mejor opción es la implementación GPU del algoritmo. Esta última versión es varias veces mas rapida que la anterior, si bien en un principio se puede pensar que esto es debido a la superioridad del dispositivo GPU con respecto a la CPU sobre la cual se llevaron a cabo los experimentos. No obstante, si bien esto puede llegar a influir en los experimentos no se puede negar la amplia mejora en los tiempo de ejecución del algoritmo.