Merge sort secuencial Versus paralelo

Autores
Angelica Aguirre Castro
Estudiante de Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira

Kenny Bedoya Cifuentes
Estudiante de Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira

Introducción

En este informe se presenta el resultado de análisis de dos algoritmos de ordenamiento que usan el modelo de Merge sort sobre un arreglo de tamaño dinámico, el cual es esencial para la comparación de tiempos de ejecución al variar la cantidad de elementos que contiene el mismo, obteniendo así un análisis de desempeño el cual se expone a continuación. Uno de los algoritmos fue implementado para ser ejecutado de manera secuencial, mientras que el otro fue implementado para una ejecución en paralelo, el cual fue ejecutado sobre una GPU Nvidia Gforce GTx 980 (tarjeta puesta a disposición para el curso de High Performance Computing, HPC, por parte del semillero de Investigación Sirius de la Facultad de Ingenierías de la Universidad Tecnológica de Pereira).

Abstract

In this report the results of analysis of two sorting algorithms that use the model merge sort on an array of dynamic size, which is essential for comparing execution times by varying the amount of items containing the same show, thus obtaining a performance analysis which follows. One of the algorithms was implemented to be executed sequentially, while the other was implemented for parallel execution, which was executed on a GPU Nvidia Gforce GTx 980 (card made available for the course of High Performance Computing, HPC , by the hotbed of Sirius Research of the Faculty of Engineering of the Technological University of Pereira).

Palabras clave: algoritmo de ordenamiento, CPU, GPU, secuencial, paralelo, arreglo.

Keywords: sorting algorithm, CPU, GPU, sequential, parallel, array.

Desarrollo investigativo

En las ciencias de la computación los algoritmos de ordenamiento son aquellos los cuales tienen como propósito poner elementos de un arreglo en una secuencia establecida, es decir, dicho algoritmo recibe como parámetro de entrada un arreglo, el cual es devuelto reordenado según la secuencia establecida (ej: números enteros en manera ascendente).

Debido a que el algoritmo de ordenamiento *Merge sort* es clasificado como externo y estable, se expondrá lo que ello significa. Que sea un algoritmo externo se refiere a que puede manejar grandes cantidades de información, por lo que es un ordenamiento que se requiere cuando la información de entrada no cabe en la memoria (generalmente la RAM), por lo que es requerido usar una memoria externa de acceso más lento como lo son los discos duros. Un ordenamiento estable mantiene el orden relativo que tenían originalmente los elementos con claves iguales. Por ejemplo, si una lista ordenada por fecha se reordena en orden alfabético con un algoritmo estable, todos los elementos cuya clave alfabética sea la misma quedarán en orden de fecha.

Debido a que el objetivo es realizar un comparativo entre dos algoritmos que realizan un ordenamiento por mezcla (Merge sort), uno secuencial y el otro paralelo, se debe saber el funcionamiento y manera de ejecución de los tipos de algoritmos, ya que el secuencial usa la CPU (Central Processing Unit) para su ejecución, mientras que el paralelo usa la GPU (Graphics Processing Unit). Por tanto es necesario indicar que las tareas atendidas por una CPU se hacen de manera secuencial, es decir, se atiende una tarea a la vez, luego de terminar con una de ellas atiende a la siguiente hasta terminar de atender todas las tareas. Mientras que la GPU puede ejecutar varias tareas al tiempo según la cantidad de hilos que el algoritmo use y/o capacidad de la GPU.

Implementación

El ordenamiento por mezcla (Merge sort) es un algoritmo de ordenamiento eficiente y de propósito general que produce un ordenamiento estable, el cual fue creado por John Von Neumann en 1945. A continuación se describe el funcionamiento conceptual del algoritmo:

1. Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:
2. Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.
3. Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
4. Mezclar las dos sublistas en una sola lista ordenada.

Además dicho algoritmo incorpora dos ideas esenciales para su funcionamiento:

1. Una lista pequeña necesitará menos pasos para ordenarse que una lista grande.
2. Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas, que a partir de dos listas desordenadas. Por ejemplo, sólo será necesario entrelazar cada lista una vez que están ordenadas.

Para nuestro análisis fueron usados dos algoritmos que implementan el ordenamiento por mezcla (mergesort), el primero un algoritmo secuencial implementado en el lenguaje de programación C, en donde se evidencia el uso de ciclos que realizan los respectivos recorridos sobre los arreglos. A continuación se muestra el código de la función de ordenamiento implementada y usada.

```
void MergeSort(int *A,int n) {
        int mid,i, *L, *R;
        if(n < 2) return; // condición inicial, el arreglo debe tener al menos dos elementos.

        mid = n/2;  // mitad del arreglo

        // se crean los dos sub-arreglos (izquierdo y derecho)
        L = (int*)malloc(mid*sizeof(int));
        R = (int*)malloc((n- mid)*sizeof(int));

        for(i = 0;i<mid;i++) L[i] = A[i]; // creando el arreglo izquierdo
        for(i = mid;i<n;i++) R[i-mid] = A[i]; // creando el arreglo derecho

        MergeSort(L,mid);  // se ordena el arreglo izquierdo
        MergeSort(R,n-mid);  // se ordena el arreglo derecho
        Merge(A,L,mid,R,n-mid);  // se mezclan los dos sub-arreglos
      free(L);
      free(R);
}
```

El segundo es un algoritmo paralelo implementado en el lenguaje de programación Cuda C, en donde se evidencia que la cantidad de hilos y bloques (necesarios para que la ejecución sea en paralelo) se calcula según la cantidad de elementos que contenga el arreglo a ordenar. A continuación se muestra el código de la función de ordenamiento implementada y usada.

```
__global__ void gpu_mergesort(long* source, long* dest, long size, long width, long
slices, dim3* threads, dim3* blocks) {
   unsigned int idx = getIdx(threads, blocks);
   long start = width*idx*slices,
        middle,
        end;

   for (long slice = 0; slice < slices; slice++) {
      if (start >= size)
         break;

      middle = min(start + (width >> 1), size);
      end = min(start + width, size);
      gpu_bottomUpMerge(source, dest, start, middle, end);
      start += width;
   }
}
```

Resultados

La GPU utilizada, características

Luego de la implementación de los dos algoritmos, fueron ejecutados 10 veces cada uno para cada tamaño del arreglo para luego promediar dichos datos y obtener los condensados de tiempos de ejecución que se muestran a continuación.

Tamaño	Ts	Tp	X = Ts / Tp
10	0.000004	0.00003	0.1333333333
100	0.0000256	0.0000378	0.6772486772
1000	0.0002184	0.000046	4.747826087
10000	0.0025498	0.0000568	44.89084507
100000	0.0205928	0.0000645	44.89084507
500000	0.090101	0.0000713	1263.68864
1000000	0.1790582	0.0000764	2343.693717

Convenciones

Ts: tiempo de ejecución algoritmo secuencial

Tp: tiempo de ejecución algoritmo paralelo

X: aceleración

De la anterior tabla de tiempos, podemos observar como la implementación paralela logra una aceleración cada vez mayor al aumentar la cantidad de elementos del arreglo.

Conclusiones

Se evidencia que el debido uso de una GPU para la ejecución de algoritmos que trabajan con mucha información, en nuestro caso un algoritmo de ordenamiento, aumenta la velocidad de ejecución con respecto a la ejecución del mismo en una CPU.

Cabe mencionar que es necesario conocer las especificaciones de la GPU a utilizar debido a que no se deben sobrepasar los límites de memoria e hilos de la misma, para así identificar los límites tanto de información a cargar como de las técnicas de las que se deben hacer uso en caso tal de necesitar una mayor capacidad en memoria

Trabajos futuros

Debido al crecimiento de la información que almacenan las diversas bases de datos con las que los usuarios interactúan en su diario vivir, vemos que el uso de algoritmos de ordenamiento y búsqueda implementados para una ejecución en paralelo, pueden ser usados para el análisis de dicha información, con la cual se puede obtener una nueva información para uso estratégico (Big data).