Reporte proyecto 3

Hernández Jiménez Juan Carlos

Calzada Martínez Jonathan Omar

García Lazcano Carlos David

29/05/19

Objetivos

General

Comprender el funcionamiento de la programación paralela, las cuales son a nivel tarea, datos y e instrucción, e implementarla a dos métodos de ordenamiento: BubbleSort y MergeSort.

Especifico

Que el alumno pueda discernir cuando el paralelismo es eficiente, a diferentes niveles de este, que su contrparte serial.

Introducción

La programación paralela surge a través de la necesidad de reducir el tiempo para realizar trabajos a nivel de cómputo, esto haciendo uso de la programación concurrente en la cual participan varios procesadores con diferentes tareas que trabajan cooperativamente para resolver un problema, como un ejemplo clásico se dice que "un hombre hace una casa en 10 horas" pero no se puede afirmar que "9 mujeres pueden hacer un bebé en un mes", partiendo de esto hay situaciones en las que el paralelismo no es más eficiente que un algoritmo serial, por lo que se dió el grupo a la tarea de detrminar esto con diferente cantidad de datos y sí en algún momento puede dar peores rendiemientos.

Paralelismo

El Paralelismo es una función que realiza el procesador para ejecutar varias tareas al mismo tiempo, puede realizar varios cálculos simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo. Existen 4 niveles de paralelismo, pero solo se veran y explicar 3, los cuales son: a nivel de datos, tarea e instrucción. El empleo de la computación paralela se convierte cada día en más grande y rápida, muchos problemas considerados anteriormente muy largos y costosos se han podido solucionar. El paralelismo se ha utilizado para muchas temáticas diferentes, desde bioinformática para hacer plegamiento de proteínas, hasta economía para hacer simulaciones en matemática financiera.

Nivel de Datos

Paralelismo de datos es un paradigma de la programación concurrente que consiste en subdividir el conjunto de datos de entrada a un programa, de manera que a cada procesador le corresponda un subconjunto de esos datos. Cada procesador efectuará la misma secuencia de operaciones que los otros procesadores sobre su subconjunto de datos asignado. En resumen: se distribuyen los datos y se replican las tareas.Idealmente, esta ejecución simultánea de operaciones, resulta en una aceleración neta global del cómputo. El paralelismo de datos es un paradigma suficientemente adecuado para operaciones sobre vectores y matrices, dado que muchas de ellas consisten en aplicar la misma operación sobre cada uno de sus elementos.

Nivel de Tarea

Consiste en asignar distintas tareas a cada uno de los procesadores de un sistema de cómputo. En consecuencia, cada procesador efectuará su propia secuencia de operaciones. el paralelismo de tareas se representa mediante un grafo de tareas, el cual es subdividido en subgrafos que son luego asignados a diferentes procesadores. De la forma como se corte el grafo, depende la eficiencia de paralelismo resultante.

La partición y asignación óptima de un grafo de tareas para ejecución concurrente es un problema NP-completo, por lo cual en la práctica se dispone de métodos heurísticos aproximados para lograr una asignación cercana a la óptima.

Nivel de Instrucción

El paralelismo a nivel de instrucción consiste en un técnica que busca que la combinación de instrucciones de bajo nivel que ejecuta un procesador puedan ser ordenadas de forma tal que al ser procesadas en simultáneo no afecten el resultado final del programa, y más bien incrementen la velocidad y aprovechen al máximo las capacidades del hardware. Un pipeline (canalizador) de instrucciones es el que permite que por cada ciclo de reloj del procesador múltiples instrucciones se encuentren en distintas fases de ejecución.

MergeSort

Algoritmo basado en la técnica de diseño de algoritmos Divide y Vencerás. Consiste en dividir el problema a resolver en subproblemas del mismo tipo que a su vez se dividirán, mientras no sean suficientemente pequeños o triviales. Ordenar una secuencia S de elementos:

- Si S tiene uno o ningún elemento, está ordenada
- Si S tiene al menos dos elementos se divide en dos secuencias S1 y S2, S1 conteniendo los primeros n/2, y S2 los restantes.
- Ordenar S1 y S2, aplicando recursivamente este procedimiento.
- Mezclar S1 y S2 ordenadamente en S
- Mezclar dos secuencias ordenadas S1 y S2 en S:
- Se tienen referencias al principio de cada una de las secuencias a mezclar (S1 y S2).
- Mientras en alguna secuencia queden elementos, se inserta en la secuencia resultante (S) el menor de los elementos referenciados y se avanza esa referencia una posición.

BubbleSort

Es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas burbujas. También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

Desarrollo

Como primer paso nos dimos la tarea de encontrar un algoritmo previamente realizado y analizar si este puede paraleliszarse para este ejercicio. Ya que el profesor nos brindó el código del algoritmo de Bubble-Sort investigamos cómo poder paralelizarlo de una manera adecuada; posteriormente se modificaron los elemntos a ordenar, cabe recalcar que la generación de datos se dio de manera aleatoria. Una vez hecho con diferente cantidad de elementos se tomó en cuenta la diferencia de tiempos haciendo una gráfica con la cual se podrían tomar decisiones sobre si debe paralelizar este algoritmo, a escalas mucho más grandes. De la misma manera se aplico el concepto para el algoritmo MergeSort. Aunque para el MergeSort no logrmos visualizar del todo la forma de paralelizar, procedimos a buscar uno ya paralelo en internet, encontrando uno que trabajaba unicamente con dos hilos, aunque tratamos de modificar esta parte para poder hacer el proceso con n hilos, nos resultó más complejo, por lo cual se decidio dejarlo con dos hilos y hacer el resto del análisis con ello. Mientras que, con BubbleSort fue más sencillo visualizar como podiamos hacer paralelo el algoritmo, llevando a cabo las comparaciones necesarioas se pudo notar que si hay una mejoria muy significativa en éste ordenamiento, pero unicamente cuando se trata de grandes cantidades de datos ya que al inicio con pocos datos el algoritmo serial resultaba ligeramente más rápido que el paralelo.

Conclusiones

Hernández Jiménez Juan Carlos:

Al momento de paralelizar los algoritmos de ordenamiento, notamos que vale más la pena hacerlo para Bubble sort, ya que en este el tiempo entre el serial y el paralelo ya es bastante cuando más datos son, como ejemplo, cuando hicimos el ordenamiento con 2,000,000 datos, el tiempo del serial fue de 116 segundos mientras que el paralelo con 8 procesadores lógicos tardó solamente 30 segundos. Asímismo se observó que al paralelizar el de Merge sort, no es tan conveniente, debido a que los tiempos no cambian demasiado, siendo el tiempo del ordenamiento en paralelo es poco más de la mitad del tiempo que se tarda en ejecutar la version serial, aunque puede ser a que solamente se utlilizaron dos hilos y no se implementó para tener más de dos. Cabe destacar que, aunque la parelelizacion es compleja y a veces no se ve a simple vista, en algunos caso es conveniente hacerla para procesar millones de datos como se vio en el caso de Bubble aunque se hallan más algoritmos muhisimo más eficientes a que.

Calzada Martínez Jonathan Omar:

En general creo que se cumpplieros los objetivos del proyecto ya que se logro entender el funcionamiento del problema. Ya que logramos hacer el analisis de los codigos y poder comprenderlos. Al hacer este proyecto aprendimos que el algoritmo de bubblesort paralelizado es muy efecto solo si son muchos datos los que se van ordenar sino el ordenamiento no es tan efectivo y se complica todo y es preferible hacerlo de manera seríal o no paralelizable La paralelización de los algoritmos es buena pero es necesario checar si es conveniente ya que hay ocasiones que no conviene por el nivel de complejidad que tienen. Por ejemplo en el caso de merge casi no hay diferencia en el en la eficiencia y en la diferencia de datos por lo que el nivel de complejidad no es conveniente programar y es mejor hacerlo de manera usual.

García Lazcano Carlos David:

Al principio dudamos sobre el algoritmo que ibamos a paralelizar, puesto que debido a que se necesitan comunicar podría tardar demasaido, porque al implementar el paralelismo con pocos datos tardaba menos el serial, pero conforme ibamos aumentando el número de datos a considerables cantidades, el paralelo empezó a funcionar mejor, se debio de analizar con sumo detalle para poder paralelizar de alguna manera, tambien en el aspecto de Merge solo se pudo paralelizar con dos hilos, puesto que hacerlo con más se nos complico demasiado. Sin duda alguna el paralelismo se debe de analizar con mucho ciudado, puesto que hacerlo de una mala manera puede volver el código un tormento. El objetivo se cumplió actualmene he interesado más por ver como este tipo de algoritmos pueden manejarse, debido a que muchas aplicaciones manejan tantos datos que es necesario manejar esto..

Bibliografia

- Paralelismo
- https://www.ecured.cu/Paralelismo
- Paralelismo de Datos
- https://esacademic.com/dic.nsf/eswiki/900010
- Paralelismo de tareas
- https://esacademic.com/dic.nsf/eswiki/900011
- Paralelismo a Nivel de Instrucción
- http://www.di-mare.com/adolfo/cursos/2008-1/ppInstParalel.pdf
- MergeSort https://www.ecured.cu MergeSort
- BubbleSort
- https://www.ecured.cu/Ordenamiento de burbuja