Dayana Rodriguez

Taller Grid Computing

Introducción

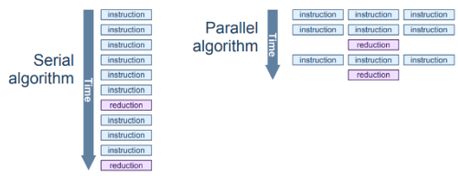
Grid computing se enmarca dentro de la tecnología de computación distribuida englobando conceptos como sistemas operativos distribuidos, programación multiprocesador, redes de ordenadores, etc. El término Grid se refiere a una infraestructura que posibilita el empleo de forma integrada y cooperativa de ordenadores, redes, bases de datos e instrumentos científicos que pertenecen y son administrados por múltiples organizaciones; de manera que el sistema permita a los usuarios un acceso uniforme a recursos distribuidos en un entorno heterogéneo. De esta forma se hace posible manejar problemas que presentan un alto costo computacional o de memoria - como la modelación de la estructura espacial de proteínas o el procesamiento de datos recolectados por radio telescopios – empleando un grupo de ordenadores conectados en red de forma que se integren en una arquitectura virtual capaz de distribuir tareas a través de una plataforma paralela. Los sistemas de Computación Grid permiten realizar tareas de cálculo que involucren grandes cantidades de datos fragmentado estos en conjuntos más pequeños; además brindan la capacidad para ejecutar muchas más instrucciones por unidad de tiempo que los ordenadores corrientes – incluso que muchos supercomputadores – empleando un modelo de procesos que pueden ser llevados en paralelo.

Investigación

Paralelizmo con OpenMP

OpenMP es una biblioteca que admite multiprocesamiento de memoria compartida. El modelo de programación OpenMP es SMP (multiprocesadores simétricos o procesadores de memoria compartida): eso significa que al programar con OpenMP todos los hilos comparten memoria y datos. Código paralelo con marcas OpenMP, a través de una directiva especial, secciones que se ejecutarán en paralelo. La parte del código que está marcado para ejecutarse en paralelo hará que se formen hilos. La banda de rodadura principal es el hilo maestro. Todos los hilos esclavos se ejecutan en paralelo y ejecutan el mismo código. Cada hilo ejecuta la sección paralelizada del código de forma independiente. Cuando termina un hilo, se une al maestro. Cuando todos los hilos terminaron, el maestro continúa con el código que sigue a la sección paralela. Cada subproceso tiene un ID adjunto que se puede obtener utilizando una función de biblioteca de tiempo de ejecución **(llamada omp_get_thread_num ()).** El ID del hilo maestro es 0.

OpenMP es compatible con C, C ++ y Fortran.Las funciones de OpenMP se incluyen en un archivo de encabezado llamadoLas partes de OpenMP en el código se especifican usando #pragmasOpenMP tiene directivas que permiten al programador:especificar la región paralela (crear hilos) especificar cómo paralelizar bucles especificar el alcance de las variables en la sección paralela (privada y compartida) especificar si los hilos se sincronizarán especificar cómo se divide el trabajo entre hilos (programación) OpenMP oculta los detalles de bajo nivel y permite al programador describir el código paralelo con construcciones de alto nivel, que es lo más simple posible.



Cuello de Botella: Se utilizó para averiguar el rendimiento en cuanto tiempo se demoró en ejecutar el programa o los hilos en paralelo.

```
unsigned t0, t1;

t0=clock()
// Code to execute
t1 = clock();

double time = (double(t1-
t0)/CLOCKS_PER_SEC);
cout << "Execution Time: " << time <<
endl;
```

Variables privadas: los datos dentro de una región paralela son privados para cada hilo, lo que significa que cada hilo tendrá una copia local y la usará como una variable temporal. Una variable privada no se inicializa y el valor no se mantiene para su uso fuera de la región paralela. Por defecto, los contadores de iteración de bucle en las construcciones de bucle OpenMP son privados

-privadas a los threads, no se inicializan antes de entrar y no se guarda su valor al salir.

#pragma omp parallel for private(….)

Variables compartidas: los datos dentro de una región paralela son compartidos, lo que significa que todos los subprocesos son visibles y accesibles simultáneamente. Por defecto, todas las variables en la región de

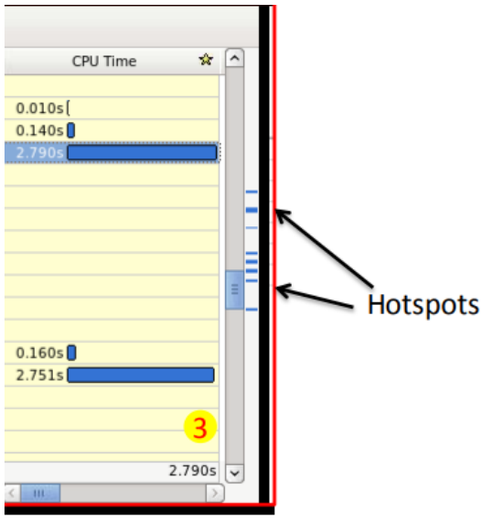
trabajo compartido se comparten, excepto el contador de iteración de bucle

-Los threads se comunican utilizando variables compartidas. (misma dirección en el contexto de cada hebra)

Bloques no paralelizables:

Hotspots: se utiliza porque se puede indicar por dónde comenzar con la optimización en serie o paralelización de memoriacompartida.

-Agregar operación de transmisión



Tareas:

• Evita la sobrecarga de asignar/liberar tareas
• Evita copiar datos y volver a ejecutar constructores y destructores.

Continuación pasando

• Adelantar reduce el espacio del stack y permite pasar por alto el planificador Pasando por alto el planificador

• La tarea puede regresar un apuntador a la próxima tarea a ejecutar – Por ejemplo, el padre regresa un apuntador a uno de sus hijos

• Guarda push/pop en deque (y lo bloquea/desbloquea).

Codigo

Librerías

```
//para los encabezados
precompilados
#include <pch.h>
#include <omp.h>
#include <iostream>
#include <ctime>
```

para calcular el tiempo en ejecución

```
unsigned t0, t1;
t0 = clock();
```

Funciones relacionadas con cantidad de hilos:

int omp_get_thread_num(); // nos devuelve el ID del thread

int omp_set_num_thread(); // establecemos cantidad de threads

int omp_get_num_thread(); // nos devuelve cantidad de threads

Resultados : View