GRID COMPUTING- OPENMP

Juan Pablo Luna Felipez, M.Sc
jplunaf@gmail.com.

Docente Ingeniería Informática
Universidad Nacional "Siglo XX"
Llallagua, Bolivia

Resumen - *Grid Computing o computación en Grilla es una tecnología novedosa altamente útil y de altas prestaciones en la era actual donde los requerimientos de procesamiento y almacenamiento de la información crecen de forma vertiginosa.*

Para la implementación de la computación Grid, se hace necesario implementar modelos de programación Grid, constituyéndose la programación paralela en una tecnología indispensable para la gestión de la computación Grid. Es por ello que se emplean de modelos de programación Grid, uno de los cuales es la son los modelos de memoria Compartida.

Una librería altamente empleada dentro de los modelos de Programación de memoria compartida es OpenMP, el cual fue sido desarrollado específicamente para procesamiento de memoria compartida en paralelo y tiene amplio apoyo de los mejores fabricantes de Hardware y Software de computadores.

El presente artículo aborda aspectos relacionados a la programación con OpenMP.

Palabras clave - *Computación en Grilla, Computación paralela, OpenMP, Modelos de Programación paralela. Modelos de programación paralela de memoria compartida.*

Abstract - *The Grid Computing is a highly useful and high performance in today's era where the requirements of processing and storage of information growing at a dizzying new technology.*

For the implementation of the Grid computing, it is necessary to implement models of Grid programming, parallel programming becoming an indispensable technology for managing grid computing. That is why we used Grid programming models, one of which is the models are shared memory.

A library highly used within models of shared memory programming is OpenMP, which was specifically developed for processing in parallel shared memory and has broad support from the best manufacturers of computer hardware and software.

This article discusses issues related to programming with OpenMP.

Keywords - *Grid Computing, Parallel Computing, OpenMP, Parallel Programing Models, Shared Memory Parallel Programing Models*

1. INTRODUCCIÓN

La Grid Computing o computación en Grilla implica el uso de varias computadoras para hacer los cálculos como si fuera un solo hardware. Esto significa que puede ejecutar su aplicación en distintas máquinas al mismo tiempo de forma paralela.

Para realizar la implementación de la Grid computing disponemos de modelos de programación Grid que se basan en modelos de programación paralela, aunque estos quizás no logren satisfacer los retos de la computación Grid.

Entre estos modelos tenemos los modelos de memoria Compartida, y Memoria Distribuida, Modelos Peer To Peer, Modelos Basados en RPC, modelos basados en Objetos, componentes, y Frameworks, modelos orientados a servicios y otros. [8]

Dentro de los modelos de Programación de memoria compartida, se tiene la librería OpenMP.

OpenMP ha sido desarrollado específicamente para procesamiento de memoria compartida en paralelo y tiene amplio apoyo de los mejores fabricantes de Hardware y Software de computadores.

OpenMP surgió como el estándar para computación en paralelo y se usa para explotar el paralelismo de la memoria compartida en máquinas individuales de memoria compartida dentro del grupo [7]

El presente artículo aborda aspectos relacionados a la programación paralela con OpenMP

2. GRID COMPUTING

Según el Grid Computing Information Centre que es una de las asociaciones dedicada exclusivamente al desarrollo de esta tecnología, llama grid a un "tipo de sistema paralelo y distribuido que permite compartir, seleccionar y reunir recursos 'autónomos' geográficamente distribuidos en forma dinámica y en tiempo de ejecución, dependiendo de su disponibilidad, capacidad, desempeño, costo y calidad de servicio requerida por sus usuarios".[11]

La computación grid es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado. En este sentido es una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores, clusters...) y se encuentran conectados mediante redes de área extensa (por ejemplo Internet). Desarrollado en ámbitos científicos a principios de los años 1990[10]

El Grid Computing se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. [9]

Por tanto la computacíon Grid es una tecnología novedosa altamente útil y de altas prestaciones en la era actual donde los requerimientos de procesamiento y almacenamiento de la información crecen de forma vertiginosa.

Esta tecnología ofrece grandes ventajas [9]:

- *Descentralización:* pueden agregarse recurso sin importar su localización geográfica.
- *Heterogeneidad:* todo recurso (Hardware y software) puede ser integrado.
- *Escalabilidad:* la infraestructura puede ser aumentada continuamente sin alterar procesos y sin que se resienta su eficiencia.
- *Multipropósito:* la infraestructura puede utilizarse para todo tipo de aplicaciones.

2.1. PROGRAMACIÓN PARALELA

La computación paralela es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo).[12]

La programación paralela se constituye en una tecnología indispensable para la gestión de la computación Grid

2.2. MODELOS DE PROGRAMACIÓN GRID

Los modelos de programación paralela son un conjunto de herramientas software que permiten realizar la ejecución paralela de procesos utilizando bibliotecas invocadas desde los programas tradicionales, extendiendo lenguajes de programación o utilizando modelos de ejecución completamente nuevos.[2]
Siguiendo a Isaza y Duque[8] los modelos de modelos de programación Grid son: modelos de memoria Compartida, Memoria Distribuida, Modelos Peer To Peer, Modelos Basados en RPC, modelos basados en

Objetos, componentes, y Frameworks, modelos orientados a servicios y otros.

2.3. MODELOS DE PROGRAMACIÓN GRID DE MEMORIA COMPARTIDA

Los sistemas de memoria compartida están formados por múltiples procesadores que comparten un mismo espacio de memoria y que se comunican entre ellos Escribiendo y leyendo variables compartidas [2].

Estos sistemas de memoria compartida se clasifican en dos: SMP y NUMA.

Los sistemas SMP son los más simples a la hora de realizar ejecuciones paralelas de programas porque no hay que preocuparse de la localización de datos, en estos sistemas todos los procesadores comparten una conexión común a la memoria y el acceso a cualquier zona de memoria se realiza a la misma velocidad.

Por lo que los sistemas SMP tienen la desventaja de que no escalan bien y están limitados a un pequeño número de procesadores.

En los sistemas NUMA la memoria es compartida pero está distribuida, por lo que la velocidad de acceso a un determinado bloque de memoria por parte de un procesador dependerá de la lejanía o de la cercanía de dicha memoria.

Por tanto los sistemas Numa, disminuyen el cuello de botella generado por el ancho de banda de la memoria y tiene la capacidad de utilizar un mayor número de procesadores que en el caso de SMP.

Dentro de los modelos de memoria compartida, una de las librerías es ampliamente aceptada y utilizada es OpenMP.

3. OPENMP

OpenMP es una API, interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida en múltiples plataformas. [1]

OpenMP es una API que nos permite añadir concurrencia a las aplicaciones mediante paralelismo con memoria compartida. Se basa en la creación de hilos de ejecución paralelos compartiendo las variables del proceso padre que los crea [7]

3.1. ANTECEDENTES

OpenMP surgió como una API creada por varias empresas con la idea de estandarizar y facilitar el desarrollo de programas paralelos escritos en lenguajes estructurados (Fortran en su inicio, C/C++ y Python actualmente). La idea era facilitar la adaptación de los desarrolladores de software tradicional a las nuevas tecnologías emergentes, como los nuevos procesadores de varios núcleos o los ordenadores de memoria compartida distribuida.[2]

La versión 3.0 de OpenMP fue presentada en mayo del año 2008, la ultima versión es la versión 4.0 que fue lanzada en Julio del 2013 [3]

3.2. CARACTERISTICAS DE OPENMP

OpenMP [1] permite añadir concurrencia a los programas escritos en C, C++ y Fortran. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix, Linux y Microsoft Windows.

Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de hardware y de software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras.

El modelo de ejecución de OpenMP está basado en el modelo fork-join, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join).[1]

Es conveniente utilizar OpenMP porque es fácil de implementar, permite la paralelización secuencial, también realiza la conversión paralelo-serie mediante compilador y es portable con poca dependencia de la arquitectura [8]

3.3 VENTAJAS DE OPENMP

Las ventajas de OpenMP respecto a otros tipos de programación paralela son las siguientes [6]:

- Es fácil de usar.
- Está ampliamente adaptado.
- Es escalable.
- Extensa documentación, clara y rica en ejemplos.
- Facilidad para la instalación y completa integración con g++.
- Filosofía sencilla y clara, que ayuda mucho si no se tiene mucha experiencia con hilos

3.4. ETAPAS PARA LA APLICACIÓN DE OPENMP

Siguiendo a Boll para aplicar OpenMp debe seguir 3 etapas [8]:

- Detectar las regiones del código que mayor tiempo consumen
- Estudiar cuales de ellas pueden paralelizarse
- Paralelizar
- En todos los pasos anteriores tener en cuenta la Ley de Amdahl's y tratar de estimar el costo/beneficio

3.5 COMPONENTES DE OPENMP

OpenMP comprende de tres componentes complementarios:

- Un conjunto de **directivas de compilador** usado por el programador para comunicarse con el compilador en paralelismo.

- Una **librería de funciones en tiempo de ejecución** que habilita la colocación e interroga sobre los parámetros paralelos que se van a usar, tal como número de los hilos que van a participar y el número de cada hilo.

- Un número limitado de las **variables de entorno** que pueden ser usadas para definir en tiempo de ejecución parámetros del sistema en paralelo tales como el número de hilos.

3.6 FUNCIONAMIENTO DE OPENMP

Siguiendo al Omedo[2], el funcionamiento de OpenMP es el siguiente:

OpenMP se basa en el uso de directivas para marcar las zonas que se quieran realizar de forma paralela.

El usuario debe elegir que variables quiere que sean compartidas entre todos los hilos que ejecutarán una zona paralela y que variables quiere que sean privadas.

Cuando una variable se marca como privada, cada proceso ligero realizará una copia local de la misma y solo tendrá valor dentro del hilo, por lo que el acceso a las mismas será óptimo.

Cuando una variable se marca como compartida, cada hilo accede a la posición de memoria donde se encuentra dicha variable de forma ordenada, de manera que se garantiza que no se sobrescriban los valores. Este proceso será lento, por lo que es recomendable marcar todas las variables posibles como privadas para que el acceso a las mismas sea óptimo.

OpenMP dispone de una cláusula de reducción que es muy útil cuando dentro del bucle se llevan a cabo operaciones de acumulación en una variable, con la cláusula de reducción se consigue que cada hilo cree una copia privada de la variable de reducción, lleve a cabo las operaciones necesarias y una vez terminada la zona paralela se sumarán todos los resultados parciales.

3.7. SINTAXIS DE OPENMP

Para implementar la programación con OpenMP se incluye directivas de OpenMP, con ello el código incluido entre las directivas se marcara como paralelo.

OpenMP tiene la siguiente sintaxis básica para una directiva:

pragma omp <directiva> {cláusula [, ...] ...}

Es decir un bloque de código para volverlo paralelo inicia con la directiva #pragma omp <directiva>, y a continuación se crean los hilos para la programación paralela, al final se destruyen todos los hilos creados prosiguiendo la ejecución secuencial

3.8. DIRECTIVAS DE OPENMP

Al incluir una directiva OpenMP se obliga a que exista una sincronización en todo el bloque de código que se encuentra en la directiva y este se define como paralelo y se crearan hilos de acuerdo a las características que se coloque en la directiva, al final del bloque se realizara una sincronización de todos los hilos, este tipo de ejecución se denomina fork-join.

De acuerdo con Rodríguez[7], las directivas que se pueden emplear en OpenMP son varias, entre las que podemos mencionar:

• Parallel: Indica que la parte de código es paralela y puede ser ejecutada por varios hilos.
• for: Es parecido a parallel, pero esta optimizado para los bucles for. Su formato es:
• section y sections: El código que comprende puede ejecutarse en paralelo pero por solo hilo
• single: La parte de código que define esta directiva, sólo se puede ejecutar un solo hilo de todos los lanzados, y no tiene que ser obligatoriamente el hilo padre.
• master: La parte de código definida, sólo se puede se ejecutada por el hilo padre.
• Critical: Sólo un hilo puede estar en esta sección.
• Atomic: Sirve para indicar que la operación se refiere a sólo una posición de memoria, y tiene que ser actualizada sólo por un hilo simultáneamente. Operaciones tipo x++ o --x son las que usan esta cláusula.
• Flush: Esta directiva resuelve la consistencia, al exportar a todos los hilos un valor modificado de una variable que ha realizado otro hilo en el procesamiento paralelo.
shared: Los datos de la región paralela son compartidos y por tanto son visibles y accesibles por todos los hilos.

• private: Los datos de la región paralela son privados para cada hilo, lo que significa que cada hilo tendrá su copia local que la usará como variable temporal.
• Default: Permite al programador que todas las variables de la región paralela sean compartidas o no para C/C++
• Firstprivate: Como private pero se inicializa con el valor original.

• reduction: Hace que una operación sea privada en cada iteración y al final suma cada resultado

3.9. PRINCIPALES FUNCIONES DE OPENMP

De acuerdo con Rodriguez[7] las principales funciones de OpenMP son las siguientes:

omp_set_num_threads: Esta función determina por defecto el número de hilos para ser usado en la subsiguiente región en paralela que no tiene la cláusula num_threads especificada.

omp_get_num_threads: Esta función retorna el número hilos actuales en el grupo que ejecuta una región en paralelo la cual es llamada.

omp_get_max_threads: devuelve el máximo valor que puede ser retornado por el llamado a la función omp_get_num_threads

omp_get_threads_num: retorna el número del hilo con el cual este grupo de hilos se está ejecutando la función. El número de hilo está entre cero (0) y omp_set_num_threads() - 1 (incluido). El número del hilo maestro es siempre el cero (0).

omp_get_num_procs: devuelve el máximo número de procesadores que puede asignado para el programa.

omp_in_parallel: devuelve un valor diferente de cero si ésta es llamada desde una extensión dinámica de una región paralela es ejecutada en paralelo, en otro caso devuelve un valor de cero.

omp_get_dynamic: Esta función devuelve un valor diferente de cero si el ajuste de hilos dinámico es activo, en caso contrario devuelve cero.

omp_set_dynamic: activa o desactiva el ajuste dinámico del número de hilos disponible para la ejecución de una región en paralelo.

omp_set_nested: activa o desactiva paralelismo anidado.

omp_get_nested: devuelve un valor diferente de cero si el paralelismo anidado está activo y devuelve cero si está inactivo.[7]

3.10. VARIABLES DE ENTORNO

Siguiendo a Rodriguez [7] a continuación se presentan las variables de entorno relacionadas en OpenMP las

cuales controlan la ejecución del código en paralelo:

OMP_SCHEDULE: Específica cómo se va a distribuir el trabajo en cada uno de los hilos, esta función solo se utiliza para los constructores “for” o “parallel for” y este puede ser estático o dinámico.

OMP_NUM_THREADS: Asigna la cantidad de hilos que se van a utilizar durante la ejecución. Si se hace un llamado a función: omp_set_num_thread el número de hilos va a cambiar.

OMP_DYNAMIC: La variable de entorno OMP_DYNAMIC activa o desactiva un ajuste dinámico del número de hilos disponibles para una ejecución .for. o de una región paralela.

OMP_NESTED: Esta función activa o desactiva el paralelismo anidado excepto que el paralelismo anidado esté activo o inactivo llamado por la función omp_set_nested.[7].

3.11. EJEMPLO DE PROGRAMAS CON OPENMP

A Continuación se presenta un par de ejemplos de programacíon con OpenMP

Ejemplo de programa hola mundo

```
#include <omp.h>
#include <iostream>
using namespace std;
int main (){

int nthreads;
int thread;

#pragma omp parallel private(nthreads, thread)
  {
            thread = omp_get_thread_num();
            nthreads = omp_get_num_threads();
            cout<<"Hola Mundo soy la hebra ="<<
            thread <<" de "<<nthreads<<" que
            somos"<<endl;
  }
}
```

Por defecto OpenMP lanzara tantos hilos como núcleos de procesamiento tenga la maquina donde se ejecuta, o como maquinas se disponga.

Con la directiva #pragma omp for podemos indicar que las iteraciones de un bucle for se va a dividir entre los distintos hilos existentes.

```
#include <omp.h>
#include <iostream>

#define N 10
#define nthreads 4
 using namespace std;
int main ()
{
  int thread;
  omp_set_num_threads(nthreads);
  #pragma omp parallel private(thread)
  {
     thread = omp_get_thread_num();
     #pragma omp for
     for (int i = 0 ; i < N; i++){
         cout<<"Soy el proceso "<<thread<<" ejecuto la
iteracion "<<i<<std::endl;
     }
  }
}
```

3.12. OPENMP Y VISUAL STUDIO

De acuerdo con la compañía Microsoft[4] OpenMP se puede emplear desde el entorno de Visual Studio, ya que el net.Framework soporta la programacíon paralela com OpenMP.

La interfaz de programación de aplicaciones de OpenMP c y C++ permite escribir aplicaciones que utilizan eficazmente varios procesadores.

Visual C++ admite el estándar de OpenMP 2,0.

3.13. PROBLEMAS CON OPENMP

Algunos problemas[6] que se encontraron con OpenMP, fue que en una estructura de programa con bucle infinito, se produce error en el ensamblador, ya sea el ciclo “while(true)” o “for(;;)” en todas las secciones omp section. La solución es poco limpia, pero bien sencilla, simplemente agregar una sección más.

4. CONCLUSIONES

El presente artículo permite acercarse a OpenMP, una

librería que permite añadir concurrencia a programas mediante paralelismo con memoria compartida.

Se presenta las ventajas, características, componentes, sintaxis, directivas, funciones, variables de entorno y algunos problemas de OpenMP.

Esta tecnología se emplea para Grid Computing como un modelo de programación de memoria compartida, que es ampliamente aceptado y utilizado.

También enfatizar que la paralelización con OpenMP en máquinas con memoria compartida no es compleja y no requiere grandes cambios en el programa, siendo esta librería una buena opción para la Grid Computing.

5. BIBLIOGRAFIA

[1] Wikipedia, "OpenMp", disponible en https://es.wikipedia.org/wiki/OpenMP, OnLine, consultado el 16-10-2015

[2] Omedo Camacho Miguel, "Diseño y evaluación de un complemento para refactorización paralela de código C, usando OpenMP", Universidad Carlos III de Madrid, 2012

[3] OpenMP, "OpenMP specifications", disponible en "http://openmp.org/wp/openmp-specifications/, OnLine, visitado el 20/10/2015

[4] Microsoft, ",OpenMp en VisualC++", disponible en https://msdn.microsoft.com/es-es/library/tt15eb9t.aspx, OnLine, consultado el 18 – 10 -2015.

[5] Universidad de Granada, "Programación distribuida y paralela", disponible en: http://lsi.ugr.es/jmantas/pdp/tutoriales/tutorial_omp.php?tuto=01_omp_holamundo, OnLine, consultado el 15-10-2105

[6] OpenFantasyWorld,"OpenMP", disponible en: https://openfw.wordpress.com/2013/01/25/openmp-en-openfantasyworld/, OnLine, visitado el 21-10-2105

[7] Rodriguez Gabriel, "Introducción al OpenMP", disponible en: lisidi.cs.uns.edu.ar/chaco/Trabajos/**OPENMP**.pdf, On Line, Consultado el 21-10-2015

[8]Isaza Gustavo A, Duque Mendez Nestor Dario "Arquitecturas y Modelos de Programación Grid", Universidad Tecnologica de Pereira, 2007, ISSN 0122-1701, Off Line, Revisado el 12-10-2015

[8] Diego Boll, "Paralelización mediante OpenMP", disponible en: uba.ar/materias/scmp/2014/cuat2/**OpenMP,** On Line, consultado el 22-10-2105

[9] Cemtic,"Grid Computing", texto guía del programa de doctorado en Ciencias de la Computación, 2015.

[10]Wikipedia, "Computación Grid", disponible en https://es.wikipedia.org/wiki/Computaci %C3%B3n_grid, On Line, consultado el 18-10-2015

[11] Santiago Banchero Mariano f., Jorge González Pablo j. Lavallén, "¿Qué es la computación Grid?" Universidad Nacional de Luján, disponible en: www.tyr.unlu.edu.ar/tyr/TYR.../**computacion_grid**-banchero-otros.pdf, On Line, revisado el 20-10-2015

[12] Wikipedia, "Computación paralela", disponible en: https://es.wikipedia.org/wiki/Computaci %C3%B3n_paralela, On Line, revisado el 21-10-2015.