Ejecución distribuida de bucles en Grids computacionales

Distributed Execution of Self-Schedulig Loops in Computational Grids

J. Herrera, E. Huedo, R. S. Montero e I. M. Llorente

Resumen

La distribución de bucles es uno de los paradigmas usados más frecuentemente para reducir el tiempo de ejecución de aplicaciones. De hecho, el objetivo del modelo de programación basado en directivas de tipo OpenMp y HPF es especificar una distribución del trabajo realizado por los bucles entre los procesadores disponibles. La llegada de la tecnología Grid aporta nuevas posibilidades de implementación de estos algoritmos. Sin embargo, la incorporación de esta tecnología a estos algoritmos dificulta su desarrollo y aplicación. Nuestra investigación propone la distribución de bucles autoplanificados, tomando las ventajas que ofrece el estándar DRMAA y la herramienta Grid*W*ay. Demostraremos la fiabilidad y eficiencia de nuestra investigación con el cálculo del conjunto de Mandelbrot analizado en una infraestructura Grid de investigación basada en el Globus Toolkit 4.0.

Palabras clave: Computación Grid, bucles, heterogéneos, distribuidos.

La distribución de bucles es uno de los paradigmas usados más frecuentemente para reducir el tiempo de ejecución de aplicaciones

Summary

Loop distribution is one of the most useful techniques to reduce the execution time of parallel applications. Traditionally, loop-scheduling algorithms are implemented based on parallel programming paradigms such as OpenMp and HPF. In this way, Grid computing technology presents new implementation possibilities. However, the use of Grid computing to develop these algorithms increases the efforts of these implementations. We propose a new approach to implement loop distribution schemes in computational Grids. This approach is implemented using the Distributed Resource Management Application API (DRMAA) standard and the GridWay meta-scheduling framework. The efficiency of this approach to solve the Mandelbrot set problem is analyzed in a Globus-based research testbed.

Keywords: Grid computing, self-scheduling, parallel loops, heterogeneous, distributed.

1. Introducción

Se consideran distintos tipos de algoritmos de paralelización de bucles para distribuir las iteraciones sobre recursos de un Grid computacional

Desde la aparición de las arquitecturas paralelas, la distribución de bucles es uno de los métodos más utilizados para aumentar la eficiencia de las aplicaciones paralelas, ya que, dentro de un programa, los bucles constituyen una fuente importante de paralelismo en el caso de que no existan dependencias entre las iteraciones (bucles paralelos). Debido a la extensión y complejidad de las aplicaciones actuales, es necesario incorporar nuevas técnicas que faciliten el uso y la implementación de los bucles paralelos. El objetivo de nuestra investigación consiste en aplicar las técnicas anteriores en un entorno Grid computacional. En particular consideraremos distintos tipos de algoritmos de paralelización de bucles para distribuir las iteraciones sobre recursos de un Grid computacional. Esta implementación se realiza utilizando el estándar DRMAA y la herramienta Grid*W*ay.

DRMAA [7] es una interfaz de programación para diferentes DRMS (*Distributed Resource Management Systems*), en nuestro caso Grid*W*ay [4], que permite ejecutar, controlar y monitorizar los trabajos en el Grid. Por otro lado, la herramienta Grid*W*ay es un meta-planificador que permite la ejecución de trabajos, array de trabajos o trabajos complejos en un Grid dinámico y heterogéneo de forma eficiente y fiable.

La organización del artículo es la siguiente: en la sección 2 se realiza una pequeña introducción a los algoritmos autoplanificados. En la sección 3 se explica el modelo de desarrollo adoptado y se discute algunos aspectos de implementación. La sección 4 detalla los experimentos realizados, así como algunos resultados adicionales. Finalmente, el artículo termina con las conclusiones del trabajo de investigación.

2. Algoritmos autoplanificados simples

Tradicionalmente, los algoritmos de distribución de bucles son uno de los métodos más utilizados a la hora de distribuir fragmentos de trabajos; estos algoritmos se puede clasificar en dos tipos principales [2][5]: *Estáticos* cuando la distribución de fragmentos de trabajos se realiza en tiempo de compilación, y *Dinámicos* cuando la distribución se realiza en tiempo de ejecución. A su vez, los algoritmos de distribución dinámicos se dividen en otros dos subtipos, simples o distribuidos. Los dinámicos distribuidos toman decisiones de distribución en función de determinados factores tales como la velocidad del procesador o la carga computacional de los nodos.

En esta investigación nos hemos centrado en los algoritmos de distribución dinámicos simples, también llamados algoritmos autoplanificados [1]. Concretamente estudiaremos estos algoritmos dentro de un entorno computacional Grid utilizando la interfaz DRMAA y la herramienta Grid*W*ay. Los algoritmos autoplanificados se basan en el modelo maestro-esclavo. En él el nodo (maestro) asigna de forma dinámica tareas, en nuestro caso fragmentos de iteraciones, al resto de los nodos esclavos. Cuando un nodo esclavo termina su ejecución envía los resultados al nodo maestro y solicita nuevas tareas. A continuación describimos los algoritmos autoplanificados más utilizados [1].

- ***Chunk Self-Scheduling* (CSS)**. La longitud del fragmento que se distribuye, es siempre fija y la define el usuario. Este algoritmo se denomina autoplanificador puro.
- ***Guided Self-Scheduling* (GSS).** El tamaño del fragmento va disminuyendo. El usuario puede elegir el tamaño mínimo de fragmento que se asociará a cada procesador.
- ***Trapezoid Self-Scheduling* (TSS).** La longitud del fragmento decrece linealmente, siguiendo un determinado parámetro de decrecimiento. Dicho parámetro se calcula a partir de otros dos definidos por el usuario.
- ***Factoring Self-Scheduling* (FSS).** La longitud del fragmento se mantiene en cada fase de procesamiento. Después de cada fase, el tamaño del fragmento va disminuyendo.
- ***Fixed Increase Self-Scheduling* (FISS).** En cada fase de procesamiento se mantiene la longitud del fragmento. Después de cada fase el tamaño crece linealmente.

Los algoritmos autoplanificados se basan en el modelo maestro-esclavo, donde el nodo (maestro) asigna de forma dinámica tareas al resto de nodos esclavos

3. Algoritmos autoplanificados en Grid

El modelo de desarrollo usado en este trabajo se muestra en la figura 1, partimos de un problema computacional, en nuestro caso es la distribución de iteraciones sobre recursos Grid. Para desarrollarlo, escribiremos el programa en lenguaje C con sentencias DRMAA, enlazaremos el ejecutable con la implementación DRMAA sobre Grid*W*ay y lo ejecutaremos como si de un programa secuencial se tratase. Dicho programa, de forma totalmente transparente al usuario, distribuye iteraciones sobre recursos del Grid, utilizando la herramienta GridWay, que a su vez utilizará Globus ToolKit 4.0 [3] para ejecutar el programa en cada nodo. Finalmente el usuario recibirá en su máquina los resultados obtenidos en cada nodo.

FIGURA 1: MODELO DE DESARROLLO

Para ejemplificar este modelo de desarrollo consideraremos la distribución de iteraciones de una suma de matrices, tal y como muestra la figura 2. El *diagrama 1* de la figura 2

Cuando un nodo esclavo termina su ejecución envía los resultados al nodo maestro y solicita nuevas tareas

muestra la suma de matrices implementada en código C puro, que consiste en este caso en dos bucles anidados que son independientes en cada iteración. Distribuiremos en el Grid la ejecución del bucle interior (indice j). Siguiendo nuestro esquema de implementación, dicha aplicación se divide en dos programas. El *diagrama 2* representa la implementación del primer programa (bucle esclavo), es decir el bucle interno del *diagrama 1*, utilizando código C puro. Por otro lado, el código que implementa el bucle maestro es un programa en C con sentencias DRMAA, en la figura destacadas en negrita. Dicho programa ejecuta el bucle externo de la iteración y lanza a través de sentencias DRMAA el ejecutable cuyo código se muestra en la figura 2.

FIGURA 2: IMPLEMENTACIÓN DE SUMA DE MATRICES

El código que implementa el bucle maestro es un programa en C con sentencias DRMAA

En nuestra investigación hemos realizado los mismos pasos para implementar un conjunto de algoritmos autoplanificados en un entorno Grid. Concretamente, hemos utilizado dichos algoritmos para resolver el conjunto no homogéneo de Mandelbrot [6] y generar un fractal.

4. Experimentos y resultados

Para estudiar los efectos de los algoritmos autoplanificados en un entorno computacional Grid, hemos implementado la resolución del conjunto de Mandelbrot [6]. Para ello hemos tomado como dominio de aplicación una ventana de [-1.7,0.8] x [-1.0,1.0] para un tamaño de la imagen de 60000 x 50000 píxeles con 6 bits por píxel, es decir, tendremos una imagen de 2.25 Gigabytes.

La ejecución del programa consiste en tres fases: ejecución, recopilación y conversión

Para distribuir el fractal dividiremos la imagen en bandas horizontales. La longitud de cada banda está definida por el tipo de algoritmo autoplanificado, es decir $longitud_i = 60000 \times long_{fragmento}$. La tabla 1 muestra un ejemplo del cálculo de la longitud del fragmento suponiendo que se utilizan 5 nodos.

Tabla 1

ALGORITMO	TAMAÑO DEL FRAGMENTO						
CSS(2.000)	2.000	2.000	2.000	2.000	2.000	2.000	2.000 ...
GSS(1.000)	10.000	8.000	6.400	5.120	4.096	3.277	2.622 ...
TSS (5.000, 1.000)	5.000	4.750	4.500	4.250	4.000	3.750	3.500
FSS	5.000	5.000	5.000	5.000	5.000	2.500	2.500 ...
FISS	2.000	2.000	2.000	2.000	2.000	3.334	3.334

Tanto los experimentos como la demostración realizada se han ejecutado en una infraestructura Grid de investigación basada en el Globus Toolkit 4.0 que se describe en la tabla 2 a continuación.

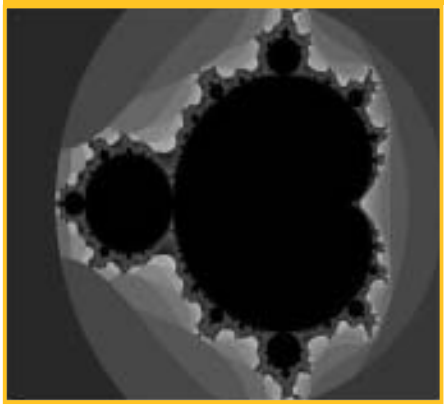
Tabla 2

MÁQUINA	MODELO	HZ	SO	MEMORIA	NODOS	GRAM
hydrus	Intel Pentium 4	3.2 Ghz	Linux 2.6	470MB	4	PBS
ursa	Intel Pentium 4	3.2 Ghz	Linux 2.6	470MB	1	fork
draco	Intel Pentium 4	3.2 Ghz	Linux 2.6	470MB	1	fork
cygnus	Intel Pentium 4	2.5 Ghz	Linux 2.6	512MB	1	fork

La ejecución del programa consiste en tres fases:

(i) Ejecución, cálculo del conjunto de mandelbrot utilizando el algoritmo GSS con un tamaño de fragmento mínimo de 200 en 7 nodos.
(ii) Recopilación, agruparemos todas las bandas de la imagen en un único fichero con formato ppm.
(iii) Conversión, generaremos a partir del fichero anterior, un fichero en formato jpeg. De esta forma, la imagen generada, ver figura 3, muestra un fractal que representa los puntos que entran dentro del conjunto de mandelbrot.

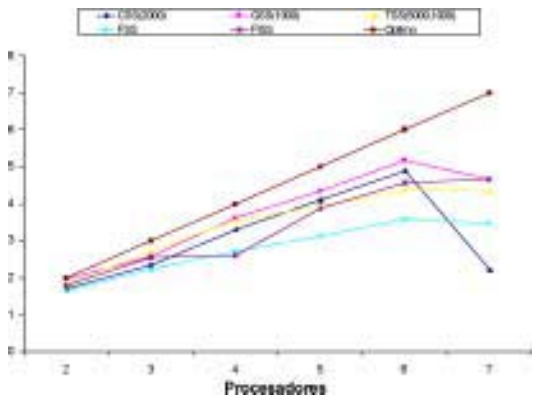
FIGURA 3: FRACTAL DE MANDELBROT GENERADO



En la figura 4 se muestra el *speedup* obtenido al ejecutar un conjunto de seis experimentos. En cada experimento se ha utilizado un número distinto de nodos y de algoritmos autoplanificados, de forma que se muestra la adaptabilidad de estos algoritmos a un Grid computacional. Desde este punto de vista, el algoritmo que mejor se adapta a nuestro entorno Grid es el GSS, ya que su *speedup* es muy semejante, en la mayoría de los casos, al *speedup* óptimo. Por otro lado, la figura también demuestra que en la mayoría de los experimentos, el incremento en el número de nodos es directamente proporcional al incremento del *speedup,* con la excepción de los experimentos que utilizan 7 procesadores. En este caso, la diferencia se debe a que se incorpora el nodo con capacidad de procesamiento inferior, es decir, aquel que tiene el procesador más lento, concretamente nos referimos a la máquina *cygnus*. Esta incorporación repercute sobre el tiempo total de ejecución, es especial en aquellos algoritmos donde le tamaño del fragmento es grande, como en el caso del CSS.

En cada experimento se ha utilizado un número distinto de nodos y de algoritmos autoplanificados

FIGURA 4: SPEEDUP FRENTE AL NÚMERO DE PROCESADORES



5. Conclusiones

Hemos presentado la implementación de un conjunto de algoritmos autoplanificados para la distribución de fragmentos de iteraciones utilizando tecnología Grid, en particular la interfaz DRMAA

Nuestra implementación aporta nuevas características a este tipo de algoritmos: fiabilidad, dinamismo y transparencia

y la herramienta Grid*W*ay. De esta manera, nuestra implementación aporta nuevas características a este tipo de algoritmos, a saber: *(i) fiabilidad*, si la ejecución falla, el proceso migra a otro nodo que esté disponible, por lo tanto no se reinicia la ejecución de la aplicación; *(ii) dinamismo*, las tareas esclavas pueden migrar a recursos más apropiados durante la ejecución, además, si durante la ejecución se incorpora un nuevo recurso, éste se puede utilizar durante el resto de la ejecución; *(iii) transparencia*, la tolerancia a fallos, la migración y ejecución de los trabajos es transparente al usuario; *(iv) despliegue*, que permite explotar los recursos de GT4.0 pre-WebServices, GT4.0 WebServices y EGEE.

J. Herrera, R. S. Montero
(jherrera@fdi.ucm.es), (rubensm@dacya.ucm.es)
E. Huedo, I. M. Llorente
(huedoce@dacya.ucm.es),(llorente@dacya.ucm.es)
Dpto. de Arquitectura de Computadores y Automática, UCM
Lab. de Comp. Avanzada, Simulación y Aplicaciones Telemáticas
Centro de Astrobiología (CSIC-INTA)

Referencias

[1] Chronopoulos, A.T., Andonie, R.: *A Class of Loop Self-Scheduling for Heterogeneus Clusters*. In: Proceedings of the 2001 IEE International Conference on Cluster Computing. Volume II - Software. (2002).
[2] Cierniak, M., Li, W., Zaki, M.J.: *LooP Scheduling for Heterogeneity*. Technical Report TR540 (1994)
[3] Foster, I., Kesselman, C.: Globus: *A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications 11 (1997) 115-128
[4] Herrera, J., Huedo, E., Montero, R.S., Llorente, I. M.: *Developing Grid-Aware Applications with DRMAA on Globus-based Grids*. Lecture Notes in Coputer Science, Europar04 3149 (2004) 429-435
[5] Li, H., Tandri, S., Stumm, M., Sevcik, K.C.: *Locality and Loop Scheduling on NUMA Multiprocessors*. In: Proceedings of the 1993 International Conference on Parallel Processing. Volume II - Software., Boca Raton, FL, CRC Press (1993) II-140-II-147
[6] Mandelbrot, B.B.: *Fractak Geometry of Nature*. W.H. Freeman & Co. (1988)
[7] Rajic, H., et al.: Distributed Resource Management Application API Specication 1.0. Technical report, DRMAA Working Group The Global Grid Forum (2004)