Demostración de uso de GRID superscalar

GRID Superscalar's Usage Demo

R. Sirvent, R. M. Badia, P. Bellens et al.

Resumen

En esta presentación realizada durante la Tercera Reunión de la Red Temática para la Coordinación de Actividades Middleware en Grid pudimos ver en uso el entorno de programación GRID superscalar, desarrollado por el grupo de Grid computing del Barcelona Supercomputing Center-Centro Nacional de Supercomputación (BSC-CNS) y la Universidat Politècnica de Catalunya (UPC). Este entorno pretende facilitar la programación de aplicaciones en el Grid, proporcionando un modelo de programación secuencial muy simple. A partir del algoritmo descrito secuencialmente en el código, GRID superscalar genera en tiempo de ejecución una tarea para cada llamada a una función listada en el fichero de interficies IDL. También se analizan las dependencias de datos entre las tareas, creando un grafo de dependencias, y se encarga de mandarlas ejecutar en el Grid, respetando dichas dependencias. El *runtime* implementa diferentes técnicas para incrementar el paralelismo disponible en la aplicación y para minimizar el tiempo de ejecución total. El entorno también proporciona una herramienta llamada Deployment Center; una interficie gráfica que facilita la preparación del entorno de ejecución.

Finalmente, se mostró un prototipo de monitor de ejecución para GRID superscalar que permite al usuario visualizar interactivamente la ejecución de los trabajos.

Palabras clave: Modelos de programación para Grids computacionales, *middleware* de Grid, paralelización automática, *deployment* automático, computación de altas prestaciones.

Summary

During this presentation, which was performed at the third meeting of the Spanish "Red Temática para la Coordinación de Actividades Middleware en Grid", we could see GRID superscalar framework in use, developed by the Grid Computing Group from Barcelona Supercomputing Center–Centro Nacional de Supercomputación (BSC-CNS) and the Universitat Politècnica de Catalunya (UPC). This framework tries to ease the programming of Grid applications by providing a very simple sequential programming model. Starting from the sequential algorithm described in the code, GRID superscalar generates a task for each of the functions listed in the interface IDL file. The data dependences between the tasks are also analyzed and a task dependence graph is built and proceeds to its execution in the Grid, taking into account those dependencies. The runtime implements different techniques in order to increase the application's available parallelism and to minimize the total execution time.

A tool called Deployment Center is also provided: a graphical interface which eases the preparation of the execution environment. Finally, a prototype of an execution monitor for GRID superscalar was presented, which allows the user to visualize the tasks' execution interactively.

Keywords: Programming models for computational grids, grid middleware, automatic parallelization, automatic deployment, high performance computing.

El entorno de programación GRID superscalar pretende facilitar la programación de aplicaciones en el Grid mediante un modelo de programación secuencial muy simple

1. Introducción

Tener que transformar un código programado de manera secuencial en un modelo de programación paralela es complejo: se debe aprender la sintaxis y el modo de utilización de un nuevo modelo de programación y adaptar nuestro algoritmo a él. Incluso en muchas ocasiones el propio usuario debe decidir cómo dividirá y distribuirá el trabajo a realizar en la ejecución paralela. GRID superscalar pretende facilitar el paso de convertir una aplicación secuencial en paralela [1], a la vez que permite su ejecución en el Grid.

Muchas de las ideas que implementa GRID superscalar provienen del mundo de la arquitectura de computadores y el diseño de microprocesadores

Muchas de las ideas que implementa GRID superscalar provienen del mundo de la arquitectura de computadores y el diseño de microprocesadores [2]. Conceptualmente, un computador no es muy diferente de un Grid (dispone de unidades de cálculo, unidades de almacenamiento y de una red que las interconecta). Las diferencias se aprecian en la cantidad de tiempo que hace falta para realizar una operación en ese entorno. En un computador las operaciones se ejecutan en un tiempo del orden

de nanosegundos, cuando en un Grid una operación puede tardar segundos, minutos o incluso horas. Como veremos más adelante, técnicas como la ejecución fuera de orden o el renombramiento de operandos (registros en el caso del computador o ficheros en el caso de GRID superscalar) se aplican en el *runtime* de nuestra herramienta para la ejecución paralela de la aplicación. El resultado para el usuario final es que su aplicación se ha ejecutado en el Grid explotando el paralelismo disponible, cuando para él parece como si se hubiera ejecutado de manera secuencial en su propia máquina.

2. Uso de GRID superscalar

Para trabajar con GRID superscalar el usuario debe proporcionar al menos tres ficheros. El primero de ellos contiene el programa principal, el algoritmo que quiere resolver. El segundo es el llamado fichero de IDL y define las interficies de las funciones (o tareas) que el usuario quiere ejecutar en el Grid, es decir, el nombre de la función y sus parámetros, añadiendo la dirección del parámetro (si es de entrada, salida o de entrada/salida). Finalmente, el tercero contiene la implementación de las funciones descritas en el fichero IDL. A partir de estos tres ficheros, se realiza el *deployment* de la aplicación (descrito en los siguientes párrafos) y finalmente el usuario puede ejecutar su programa principal. Esto desencadenará que el *runtime* entre en acción, puesto que las funciones del programa principal generarán llamadas al *runtime* en lugar de ejecutar realmente la función. El *runtime* realiza un análisis de dependencias de datos entre las diferentes funciones, basado en los parámetros especificados como ficheros, construyendo así un grafo dirigido acíclico que indicará el orden mínimo que se debe respetar en la ejecución de las funciones. De este grafo, las tareas que tengan sus dependencias resueltas (es decir, que no se deba ejecutar otra tarea antes que ella) serán candidatas a ser ejecutadas en el Grid. GRID superscalar elegirá la más conveniente y también elegirá un recurso en el Grid, intentando minimizar el tiempo de ejecución total y la lanzará para su ejecución remota en el Grid haciendo uso del *middleware* correspondiente.

Los lenguajes de programación que se pueden utilizar con GRID superscalar son C/C++ y Perl

3. Interficie de programación

Entrando más en detalle en la interficie de programación, podemos decir que se estructura en tres partes: el programa principal, el fichero IDL y la implementación de las funciones. Los lenguajes de programación que se pueden utilizar con GRID superscalar son C/C++ y Perl, aunque también disponemos de prototipos usando Java y shell script. Así pues, la idea que GRID superscalar persigue es el ser una especie de lenguaje ensamblador para el Grid, puesto que por encima de él se pueden utilizar diferentes lenguajes de programación.

Para el programa principal se proporciona un conjunto de primitivas muy reducido, puesto que la adaptación del código debe ser muy simple

3.1. Programa principal

Para el programa principal (o máster) se proporciona un conjunto de primitivas muy reducido, puesto que la adaptación del código debe ser muy simple. Las primitivas GS_On y GS_Off se utilizan para inicializar y finalizar el *runtime*. Para el manejo de ficheros en el programa principal se definen las primitivas GS_Open, GS_Close, GS_FOpen y GS_FClose, que con funcionalidad equivalente a las funciones definidas en el lenguaje de programación C para abrir y cerrar ficheros. Su uso es necesario, ya que el *runtime* debe saber cuándo el programa principal hace uso de un fichero, tanto para leerlo como para escribirlo, puesto que los ficheros definen las dependencias de datos entre las tareas. Otras llamadas como GS_Barrier se definen para que el usuario pueda indicar, en un momento dado, que desea esperar a que todas las tareas generadas hasta ese punto acaben su ejecución en el Grid.

También se define la función GS_Speculative_End, para permitir la ejecución especulativa de tareas. El usuario genera tareas en el máster (programa principal) y llama a GS_Speculative_End, que espera a que la ejecución de las tareas generadas hasta ese punto finalice (como un GS_Barrier), pero con la peculiaridad que desde un *worker* se puede llamar a GS_Throw. Esto provocará que las tareas que se debían ejecutar después de la que ha llamado a GS_Throw sean descartadas (si se debían ejecutar no se ejecutarán, si están en ejecución se cancelarán y si se han ejecutado sus resultados son descartados).

3.2. Fichero IDL

En el fichero de IDL se definen las cabeceras de las funciones que el usuario pretende ejecutar en el Grid. El formato en el que se especifica está inspirado en el lenguaje IDL de CORBA [3], sin embargo no tiene ninguna otra relación con CORBA. Se debe especificar el nombre de la función, con todos los parámetros que necesita, con el tipo y la dirección de cada parámetro: si el fichero es de entrada (sólo se lee dentro de la operación), si es de salida (solamente se escribe) o si es de entrada/salida. Las dependencias de datos se tendrán en cuenta en los parámetros definidos como tipo File. Es importante destacar que en el fichero de IDL sólo se deben especificar las funciones que el usuario quiere ejecutar en el Grid. Si el usuario define otras funciones locales, se compilarán y ejecutarán como se hacía anteriormente.

En el fichero de IDL se definen las cabeceras de las funciones que el usuario pretende ejecutar en el Grid

3.3. Implementación de las tareas

Finalmente, la implementación de las funciones (el código de éstas) se debe proporcionar en un fichero separado. En este *worker,* aparte de la función GS_Throw (comentada anteriormente), también se define la función GS_System, que debe utilizarse tal como se utiliza la llamada system de C. Con ella podemos llamar a un ejecutable externo desde el *worker* (típicamente un simulador o algún código heredado). Los cambios que se deben aplicar al programa original son, por tanto, mínimos.

La implementación de las funciones (el código de éstas) se debe proporcionar en un fichero separado

4. Deployment Center

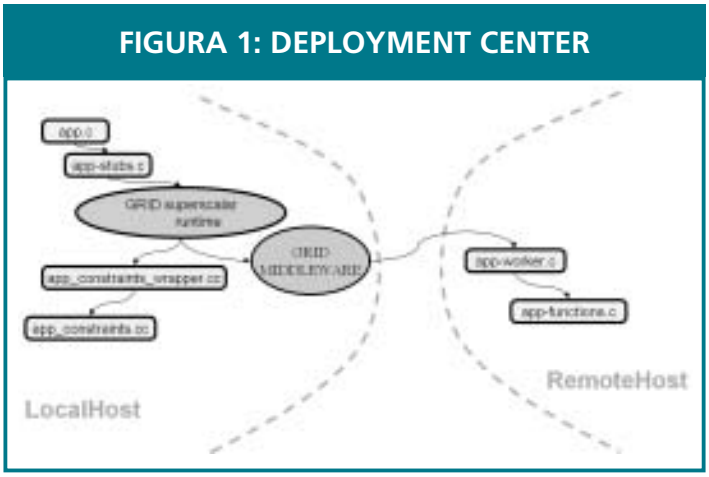
A partir de estos tres ficheros (el programa principal, el fichero IDL y el fichero con las funciones) se procederá a realizar el *deployment* de la aplicación (copia y compilación remota de los ficheros fuente). Éste es el principal objetivo de la herramienta llamada Deployment Center.

Esta herramienta es una interficie gráfica desarrollada en Java que implementa diferentes funcionalidades. Permite especificar los detalles de las máquinas que formaran el Grid (arquitectura, sistema operativo, ubicación de las librerías,...) y seleccionarlas para su uso en la computación. Esto generará de manera sencilla el fichero de configuración que GRID superscalar necesita para la ejecución. También incorpora un proceso de comprobación del estado de las máquinas definidas que consiste en ejecutar un *ping* hacia la máquina, se comprueba que el servicio de Globus [4] funciona correctamente y se intenta realizar el *deployment* de un pequeño ejemplo (se manda el código fuente y se compila en la máquina). Después del proceso de comprobación, si se selecciona la máquina para ser utilizada en la computación, el *deployment* de la aplicación comenzará de manera automática.

El *deployment* incluye otro paso que es la generación automática de código. A partir del fichero de IDL, y con la herramienta llamada *gsstubgen,* se generarán *stubs* (parte máster) y *skeletons* (parte *worker*) que harán de intermediarios al llamar a las funciones. En la parte máster, en lugar de

ejecutar la función original, se llamará al *stub,* que a su vez llamará al *runtime* de GRID superscalar para que gestione la función que se está generando. En el *worker* se ejecutará el *skeleton* que acabará invocando la implementación de la función proporcionada por el usuario (ver figura 1).

FIGURA 1: DEPLOYMENT CENTER



La herramienta *gsstubgen* también genera otros ficheros que permiten definir restricciones que se deben cumplir al ejecutar las funciones definidas

La herramienta *gsstubgen* también genera otros ficheros para usuarios avanzados que permiten definir restricciones que se deben cumplir al ejecutar las funciones definidas (de esta manera se pueden filtrar los recursos de los que disponemos. Por ejemplo: sólo ejecutar en máquinas con arquitectura *powerpc,* etc.) y también se puede añadir una estimación del tiempo de ejecución de la función para conseguir una planificación más precisa.

5. Runtime

El *runtime* de GRID superscalar posee diferentes características. Realiza un análisis de las dependencias de datos entre las operaciones que se deben ejecutar en el Grid (basado en ficheros) y a partir de él explota el posible paralelismo de la aplicación automáticamente de manera transparente al usuario, permitiendo la ejecución fuera de orden de las operaciones generadas (siempre respetando las dependencias de datos). Algunas dependencias pueden ser eliminadas con técnicas de renombramiento de ficheros.

Siempre que haya recursos disponibles, el *runtime* lanzará a ejecutarse en el Grid toda tarea que no tenga dependencias predecesoras

Siempre que haya recursos disponibles, el *runtime* lanzará a ejecutarse en el Grid toda tarea que no tenga dependencias predecesoras (es decir, que no tenga que esperar a que otras tareas finalicen para poder ser ejecutada). De las posibles tareas candidatas a ser ejecutadas, se evaluará el coste de transferir los ficheros que la tarea necesita a la máquina remota donde debe ejecutarse y el coste de ejecución en la máquina correspondiente. Si los ficheros ya están disponibles en una máquina (porque otra ejecución los ha transferido o generado allí), se elimina el coste de transferencia, aprovechando así la localidad de los ficheros. Se ejecutará la que dé una estimación menor.

Para la ejecución de una tarea en concreto, primero se deben transferir a la máquina seleccionada los ficheros que ésta necesite y que no tenga disponibles y se debe crear un directorio temporal de trabajo para evitar problemas con otras tareas que puedan estar ejecutándose en ese mismo momento. Después de la ejecución, los resultados se almacenan en la misma máquina que los ha generado, dejándolos así disponibles para las siguientes tareas que deban ejecutarse. Cuando ha

finalizado, el runtime actualiza su grafo de dependencias entre tareas eliminando las dependencias que tenía la tarea que ha finalizado, hecho que puede dejar nuevas tareas disponibles para ser ejecutadas en el Grid.

GRID superscalar también es capaz de trabajar con discos compartidos entre diferentes máquinas (por ejemplo, con NFS [5]), y con réplicas de datos, minimizando así las transferencias necesarias para las ejecuciones. Se incluyen también otras técnicas como el *checkpointing* a nivel de tarea, que permite reiniciar la ejecución desde un punto intermedio de la computación si por cualquier motivo se ha tenido que detener, y la gestión de excepciones con las primitivas GS_Speculative_End y GS_Throw, como ya se ha descrito en párrafos anteriores. El runtime de GRID superscalar funciona sobre diferentes *middlewares* de Grid (actualmente disponemos de versiones sobre Globus 2.4, Globus 4.0, Ninf-G2 [6] y ssh/scp).

6. Demostración realizada

La demo de GRID superscalar utilizó como ejemplo de partida un algoritmo de multiplicación de matrices. La peculiaridad de este ejemplo es que las matrices están divididas en bloques más pequeños (lo que se conoce por hipermatrices) y cada bloque se almacena en disco en un fichero. Esto es muy útil cuando las matrices son tan grandes que no caben en la memoria principal: se divide la matriz en bloques más pequeños que sí caben en memoria, y se opera con bloques en lugar de con la matriz entera. Esta implementación se conoce con el nombre de *out-of-core*.

Se mostró a los asistentes el uso del Deployment Center para definir las máquinas del Grid y realizar la compilación remota de manera automática de los ficheros fuente necesarios para la ejecución. Tras haber utilizado el Deployment Center también se mostró la ejecución de la aplicación en las máquinas del BSC-CNS y se pudo ver un nuevo prototipo de monitor para GRID superscalar, donde se visualizan las tareas generadas, las dependencias entre ellas, y el estado en el que está cada tarea (si está disponible, en ejecución, en qué máquina, si ha finalizado, etc.).

GRID superscalar también es capaz de trabajar con discos compartidos entre diferentes máquinas

La demo de GRID superscalar utilizó como ejemplo de partida un algoritmo de multiplicación de matrices

Raúl Sirvent Pardell
(Raul.Sirvent@bsc.es)
Rosa María Badia Sala
(Rosa.M.Badia@bsc.es)
Pieter Bellens
(pbellens@ac.upc.edu)
Vasilis Dialinos
(dialinos@ac.upc.edu)
Jesús Labarta Mancho
(jesus@ac.upc.edu)
Josep M. Pérez Cáncer
(Josep.M.Perez@bsc.es)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación
Dept. d'Arquitectura de Computadors
UPC

Referencias

[1] Badia, Rosa M.; Labarta, Jesús; Sirvent, Raúl; Pérez, Josep M; Cela, José M.; Grima, Rogeli. "Programming Grid Applications with GRID superscalar". Journal of Grid Computing, 1 (2). 2003. 151-170.

[2] Hennessy, J. L.; Patterson, D. A.; Goldbert, D. "Computer Architecture: A Quantitative Approach". Morgan Kaufmann. 2002.

[3] Object Management Group. "IDL Syntax and Semantics". Consultado en: *www.omg.org/cgi-bin/doc?formal/02-06-39 15-5-2003.*

[4] Foster, I.; Kesselman, C. "Globus: A Metacomputing Infrastructure Toolkit". Int. Journal of Supercomputer Applications, 11(2). 1997. 115-128.

[5] Shepler, S.; Callaghan, B.; Robinson, D.; Thurlow, R.; Beame, C.; Eisler, M.; Noveck, D. "Network File System (NFS) version 4 Protocol". Request for Comments 3530. April 2003.

[6] Tanaka, Y.; Nakada, H.; Sekiguchi, S.; Suzumura, T.; Matsuoka, S. "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing". Journal of Grid computing, 1 (1). 2003. 41-51.