APLICACIONES DEL CÓMPUTO DE ALTAS PRESTACIONES

Nilda M. Pérez Otero – Sandra A. Mendez – Cristina V. Ayusa – Marino I. Aucapiña – Victor J. Lopez

Facultad de Ingeniería – Universidad Nacional de Jujuy (UNJu)
{nilperez, smendez, cristina_ayusa, maucapiña, vlopez}@fi.unju.edu.ar

CONTEXTO

El presente proyecto se encuadra en el campo del Procesamiento paralelo y Distribuido particularmente en el Cómputo de Altas Prestaciones y se desarrolla en la Facultad de Ingeniería de la UNJu en el marco del Programa de Investigación Cód. 08/D058.

RESUMEN

Debido al avance de las arquitecturas multiprocesador y la importancia que tiene actualmente el Cómputo de Altas Prestaciones (HPC) se vió la necesidad de proporcionar a los alumnos, docentes investigadores de la provincia de Jujuy y a la región noroeste un lugar de referencia donde se pueda aprovechar los servicios para aplicaciones que requieran una gran capacidad de procesamiento. Es así que se inicia en el año 2008 el proyecto de Investigación "APLICACIONES DEL CÓMPUTO DE ALTAS PRESTACIONES" formado por jóvenes docentes investigadores de la Facultad de Ingeniería de la UNJu con el objetivo de profundizar en el área del cómputo de altas prestaciones, incorporar HPC en la currícula de las carreras informáticas y colaborar con otros grupos de investigación que requieran HPC.

Palabras clave: Cómputo de Altas Prestaciones, HPC, Procesamiento paralelo

1. INTRODUCCION

Los organismos gubernamentales, militares y grandes centros de investigación que cuentan con aplicaciones científicas para la simulación de procesos naturales (previsión del tiempo, análisis de cambios climáticos, entre otros procesos), modelaje molecular, simulaciones físicas (túneles de viento, criptoanálisis), etc. emplean costosas supercomputadoras que permiten realizar los cálculos complejos requeridos por tales aplicaciones. Una supercomputadora o superordenador es una computadora con altísima capacidad de procesamiento y gran capacidad de memoria, diseñada para procesar enormes cantidades de información en poco tiempo y dedicada a una tarea específica en investigaciones científicas y militares. Las 4 tecnologías más importantes en el diseño de supercomputadoras son:

- La tecnología de registros vectoriales que permite la ejecución de innumerables operaciones aritméticas en paralelo.
- El sistema conocido como M.P.P. por las siglas de Massively Parallel Processors o Procesadores Masivamente Paralelos, que consiste en la utilización de cientos y a veces miles de microprocesadores estrechamente coordinados.
- La tecnología de computación distribuida: los clusters de computadoras de uso general y relativo bajo costo, interconectados por redes locales de baja latencia y el gran ancho de banda.
- Cuasi-Supercómputo: con la popularización de la Internet, han surgido proyectos de computación distribuida en los que software especiales aprovechan el tiempo ocioso de miles de ordenadores personales para realizar grandes tareas por un bajo costo. A diferencia de las tres últimas categorías, el software que corre en estas plataformas debe ser capaz de dividir las tareas en bloques de cálculo independientes que no se ensamblarán ni comunicarán por varias horas. En esta categoría destacan BOINC y Folding@home.

El costo de las supercomputadoras, millones de dólares, hace que las universidades e instituciones de investigación cuyos presupuestos no permiten inversiones de tal envergadura opten por sistemas de cómputo de altas prestaciones basadas en clúster de PCs que funcionan como un supercomputador. La implementación de este tipo sistemas es posible debido a las características de las actuales computadoras de escritorio [1]:

- Arquitecturas especializadas VLIW, que permiten ejecutar varias instrucciones simultáneamente, especialmente en el área de procesamiento de gráficos.
- Paralelismo asimétrico a nivel de sistema, en cuanto a que los sistemas suelen tener procesadores de propósito general junto con procesadores especializados (los más potentes son los procesadores gráficos, pero también hay procesadores de física, por ejemplo).
- Arquitecturas multithread de altas prestaciones, que usan sistemas tales como el Hyperthreading para ejecutar varias tareas de forma concurrente.
- Paralelismo simétrico a nivel de sistema, con arquitecturas que incluyen dos (o potencia de dos) núcleos por microprocesador (en la práctica, varios microprocesadores). En el año 2006, por ejemplo, las arquitecturas multi núcleo ya eran habituales tanto en ordenadores de sobremesa como en portátiles. En poco tiempo, este tipo de arquitecturas se extienden a dispositivos pequeños de tipo PDA, e incluso móviles.

La conectividad es un hecho en todo tipo de ordenadores: desde la conectividad inalámbrica (WiFi, habitualmente) o por red, hasta Bluetooth en los móviles, o la conectividad intrínseca a los mismos.

Como ya se indicó, la computación de altas prestaciones hace uso de computadores paralelos o sistemas distribuidos. Éstos disponen de múltiples procesadores que son capaces de trabajar conjuntamente en una o varias tareas a la vez para resolver un problema computacional. Las arquitecturas que dan soporte a estos sistemas se pueden clasificar de acuerdo a [2] [3] [4] [5] [6] en:

- SIMD (Simple Instruction Multiple Data): un solo flujo de instrucciones se aplica concurrentemente a múltiples conjuntos de datos. En una arquitectura SIMD procesos homogéneos (con el mismo código) ejecutan sincrónicamente la misma instrucción sobre sus datos, o bien la misma operación se aplica sobre unos vectores de tamaño fijo o variable.
- MISD (Multiple Instruction Simple Data): los procesadores tratan de forma diferente el mismo conjunto de datos. Son útiles en casos donde se deban realizar muchas operaciones diferentes sobre el mismo conjunto de datos.
- MIMD (Multiple Instruction Multiple Data): paralelismo funcional y/o de datos. No sólo se distribuyen los datos entre los diferentes procesadores, sino también las tareas a realizar. Varios flujos (posiblemente diferentes) de ejecución se aplican a diferentes conjuntos de datos.
- SPMD (Simple Program Multiple Data): en paralelismo de datos [7], se utiliza el mismo código con distribución de datos. Se crean varias instancias de las mismas tareas, cada una ejecutando el código de forma independiente.

Otra forma de clasificar las arquitecturas es en base a los mecanismos de control y la organización del espacio de direcciones [8] [9] [10], por ejemplo, para las arquitecturas MIMD:

- Sistemas MIMD con memoria compartida (Shared Memory – MIMD), o también denominados Multiprocesadores. En este modelo, los procesadores comparten el acceso a una memoria común por lo que existe un único espacio de direcciones de memoria para todo el sistema, accesible para todos los procesadores. Éstos se comunican a través de la memoria compartida. Los desarrolladores no tienen que preocuparse sobre la posición actual de almacenamiento de los datos.
- Sistemas MIMD con memoria distribuida (Distributed Memory – MIMD), o también llamados Multicomputadores. En este modelo, cada procesador ejecuta un conjunto separado de instrucciones sobre sus propios datos locales. La memoria, no centralizada, está distribuida entre los procesadores del sistema, cada uno con su propia memoria local donde posee su propio programa y datos asociados. El espacio de direcciones de memoria no está compartida entre los procesadores. Una red de interconexión conecta los procesadores (y sus memorias locales), mediante enlaces (links) de comunicación, usados para el intercambio de mensajes entre los procesadores. Los procesadores intercambian datos entre sus memorias cuando se pide el valor de variables remotas.

Debido a la fuerte dependencia de las arquitecturas de las máquinas que se usen y de los paradigmas de programación usados en su implementación, aún no existe una metodología claramente definida para la creación de aplicaciones paralelas [11]. Una metodología simple de creación de aplicaciones paralelas [9], es la que estructura el proceso de diseño en cuatro etapas: Partición, Comunicación, Aglomeración y Mapping (PCAM). Las dos primeras se enfocan en la concurrencia y la escalabilidad, y su objetivo es desarrollar algoritmos que prioricen estas características. En las dos últimas etapas, la atención se desplaza hacia la localidad y las prestaciones ofrecidas. Como resultado de esta metodología y dependiendo de los paradigmas y arquitecturas físicas, se puede obtener:

- una aplicación paralela, con un mapping estático entre tareas y procesadores a la hora de iniciar una aplicación o
- una aplicación que crea tareas (y las destruye) en forma dinámica, por medio de técnicas de balanceo de carga.

Por esto, una de las decisiones importantes en las aplicaciones paralelas es, precisamente, la elección del paradigma de programación.

Los paradigmas de programación paralelos son una clase de algoritmos que solucionan diferentes problemas bajo las mismas estructuras de control [12]. Bajo la idea de paradigma se encapsula una determinada forma de realizar el control y las comunicaciones de la aplicación. Hay diferentes clasificaciones de paradigmas de programación, pero un subconjunto habitual [13] en todas ellas es:

- Master-Worker: también conocido como Task-farming, consiste en dos tipos de entidades, un master y múltiples esclavos (workers). El master es responsable de descomponer el problema a computar en diferentes trabajos más simples (o en subconjuntos de datos), los cuales distribuye sobre los workers y finalmente recolecta los resultados parciales para componer el resultado final de la computación. Los workers ejecutan un ciclo muy simple: reciben un mensaje con el trabajo, lo procesan y envían el resultado parcial de vuelta al master. Usualmente sólo hay comunicación entre el master y los workers. Uno de los parámetros importantes de este modelo es el número de workers utilizados.

- SPMD (Single Program Multiple Data): se crea un número de tareas fijas idénticas al inicio de la aplicación, y no se permite la creación o destrucción de tareas durante la ejecución. Cada tarea ejecuta el mismo código, pero con diferentes datos. Normalmente esto significa partir los datos de la aplicación entre los nodos (o máquinas) disponibles en el sistema que participan en la computación. Este tipo de paralelismo también se conoce como Paralelismo Geométrico, Descomposición por Dominios o Paralelismo de Datos.
- Pipelining: este paradigma está basado en un aproximación por descomposición funcional, donde cada tarea se tiene que completar antes de comenzar la siguiente, en sucesión. Cada proceso corresponde a una etapa del pipeline, y es responsable de una tarea particular. Todos los procesos son ejecutados concurrentemente y el flujo de datos se realiza a través de las diferentes fases del pipeline, de manera que la salida de una fase es la entrada de la siguiente. Uno de los parámetros importantes aquí es el número de fases.
- Divide and Conquer: el problema a computar se divide en una serie de subproblemas. Cada uno de estos subproblemas se soluciona independientemente y sus resultados se combinan para producir el resultado final. Si el subproblema consiste en instancias más pequeñas del problema original, se puede entonces, realizar una descomposición recursiva hasta que no puedan subdividirse más. Normalmente en este paradigma son necesarias operaciones de partición (split o fork), computación, y unión (join). Normalmente se trabaja con dos parámetros, la anchura del árbol, el grado de cada nodo, y la altura del árbol, el grado de recursividad.
- Paralelismo especulativo: se utiliza como alternativa si no puede obtenerse paralelismo con alguno de los paradigmas anteriores. Consiste en realizar algunos intentos de paralelización de cómputo, en base a elegir diferentes algoritmos para resolver el mismo problema. El primero en obtener la solución final es el que se escoge entre las diferentes opciones probadas. Otra posibilidad es, si el problema tiene dependencias de datos complejas, puede intentarse ejecutar de forma optimista, con una ejecución especulativa de la aplicación, con posibles vueltas atrás, si aparecen problemas de incoherencias o dependencias no tenidas en cuenta o previstas, que obliguen a deshacer cómputo realizado previamente si algo falla. En caso de acertar, o si no aparecen problemas, se puede incrementar el rendimiento de forma significativa.

Cada paradigma de programación se refiere a una clase de algoritmos que tienen la misma estructura de control [12] [14] y que se pueden implementar usando un modelo genérico de programación paralela. Los modelos de programación solucionan la distribución de código y la interconexión entre las unidades de ejecución y las tareas. Cualquiera de estos modelos puede implementar los diferentes paradigmas de programación paralela. Sin embargo, las prestaciones de cada combinación resultante (paradigma en un determinado modelo) dependerán del modelo de ejecución subyacente (la combinación de hardware paralelo, red de interconexión y software de sistema disponibles).

Se puede clasificar los modelos de programación, según [8] [9], en:

- Memoria compartida: en este modelo [15] [16] los programadores ven sus programas como una colección de procesos que acceden a variables locales y un conjunto de variables compartidas. Cada proceso accede a los datos compartidos mediante una lectura o escritura asíncrona. Por tanto, como más de un proceso puede realizar las operaciones de acceso a los mismos datos compartidos en el mismo tiempo, es necesario implementar mecanismos (semáforos o bloqueos) para resolver los problemas de exclusiones mutuas que se puedan plantear. En este modelo el programador ve la aplicación como una colección de tareas, que normalmente se asignan a threads de ejecución en forma asíncrona. OpenMP [17] es una de las implementaciones más utilizadas para programar bajo este modelo.
- Paralelismo de Datos: el paralelismo de datos, es un paradigma en el que [3] se realizan operaciones semejantes (o iguales) sobre varios elementos de datos simultáneamente, por medio de la ejecución simultanea en múltiples procesadores. Este modelo [18] es aconsejable para las aplicaciones que realizan la misma operación sobre diferentes elementos de datos. La idea principal es explotar la concurrencia que deriva de realizar las mismas operaciones sobre múltiples elementos de las estructuras de datos. Un programa paralelo de datos consiste en una lista de las mismas operaciones a realizar en una estructura de datos. Entonces, cada operación en cada elemento de datos puede realizarse en forma de una tarea independiente. En cuanto a los lenguajes que soportan este modelo, se puede citar a Fortran 90, y su extensión High Performance Fortran [19] [9].
- Paso de Mensajes: este es el modelo más ampliamente usado [8] [20]. En él los programadores organizan sus programas como una colección de tareas con variables locales privadas y la habilidad de enviar, y recibir datos entre tareas por medio del intercambio de mensajes. Definiéndose así por sus dos atributos

básicos: un espacio de direcciones distribuido y soporte únicamente al paralelismo explícito. Los mensajes pueden ser enviados vía red o usando memoria compartida si está disponible, dependiendo del modelo arquitectural del sistema (o de la virtualización software empleada). Las comunicaciones entre dos tareas ocurren a dos bandas, donde los dos participantes tienen que invocar una operación. Se puede denominar a estas comunicaciones como operaciones cooperativas ya que deben ser realizadas por cada proceso, el que tiene los datos y el que quiere acceder a ellos. También, en algunas implementaciones [21] [22], pueden existir comunicaciones de tipo (one-sided), si es sólo un proceso el que invoca la operación, colocando todos los parámetros necesarios, y la sincronización se hace de forma implícita. Como ventajas, el paso de mensajes permite un enlace con el hardware existente, ya que se corresponde bien con arquitecturas que tengan una serie de procesadores conectados por una red de comunicaciones (ya sea interconexión interna, o red cableada). En cuanto a la funcionalidad, incluye una mayor expresión disponible para los algoritmos paralelos, proporcionando control no habitual en el paralelismo de datos, o en modelos basados en paralelismo implícito por compilador paralelizante. PVM [23] y MPI [24] [25] constituyen ejemplos de librerías que soportan el modelo de paso de mensajes.

2. LINEAS DE INVESTIGACION y DESARROLLO

Aunque a nivel mundial el tema de altas prestaciones está en auge, a nivel regional, particularmente en el NOA, los profesionales de informática no han incursionado en el área del Cómputo de Altas Prestaciones y los investigadores de las diferentes disciplinas científicas que requieren utilizar aplicaciones que requieren gran capacidad de cómputo no tienen acceso a estos sistemas.

A continuación se enumeran los ejes del tema de investigación:

- Evaluación de las aplicaciones existentes para la instalación y configuración de un cluster de computadoras.
- Instalación y Configuración de un cluster de PCs con dos sistemas operativos.
- Evaluación de librerías MPI para el desarrollo de programas paralelos.
- Evaluación de Gestores de Recursos para Cluster de PC (PBS, SGE).
- Selección y Prueba de aplicaciones clásicas que requieren procesamiento paralelo (TSP, 9 Reinas).
- Análisis del rendimiento de la aplicaciones de prueba en el cluster.

3. RESULTADOS OBTENIDOS/ESPERADOS

Dada la reciente formación del grupo de investigación aun se trabaja en la evaluación del clúster y el análisis del rendimiento de las aplicaciones que se ejecutan en este.

Los resultados obtenidos hasta ahora son:

- Instalación y configuración de un clúster formado por 12 PCs.
- Instalación y Configuración de la librería de paso de mensajes LAM-MPI.
- Estudio y selección de los gestores de recursos TORQUE, SGE y LFS. De este estudio se optó por elegir PBS (versión gratuita de TORQUE). Aunque no se lo utiliza en la actualidad por un problema logístico.

Actualmente se trabaja en:

- La paralelización de aplicaciones científicas y su evaluación en el cluster.
- La implementación del gestor de recursos.

Además, se espera seguir avanzando en la optimización del cluster para poder proporcionar su servicio a quienes lo requieran.

4. FORMACION DE RECURSOS HUMANOS

La mayoría de los miembros del grupo de investigación realizan estudios de posgrado en HPC en la Universidad Nacional de La Plata y se cuenta con la asesoría constante de los Dres. Emilio Luque y Dolores Rexachs, profesores de la Universidad Autónoma de Barcelona y de la UNLP y del personal de la UAB que trabaja en el área del cómputo de Altas Prestaciones. Además, algunos integrantes de este proyecto integran un grupo de investigación (formado en el 2009) dedicado a la Tolerancia a Fallos en Cómputo de Altas Prestaciones, donde tienen como tarea, entre otras, la formación en cuestiones de HPC de alumnos avanzados de la carrera Ingeniería Informática.

Este año se dictarán cursos para los jóvenes de la facultad en temas de HPC, dándoles la oportunidad de realizar las pruebas en el cluster real de reciente creación.

También se tiene previsto ofrecer opciones de trabajos finales de carrera en HPC a los alumnos de Ingeniería en Informática.

5. BIBLIOGRAFIA

[1] J. Merelo, J. L. Jiménez Laredo. “Introducción a las Arquitecturas de Altas Prestaciones”. www.geneura.ugr.es/~jmerelo/asignaturas/AAP/AAP-Tema-1.mhtml.

[2] S.G. Akl. The Design and Analysis of Parallel Algorithms. Prentice Hall, Englewood Clifs, NJ, 1989.

[3] George S. Almasi and Alan Gottlieb. Highly Parallel Computing, 2nd ed. Benjamin/Cummings division of Addison Wesley Inc., Redwood City, CA, 1st edition, 1994.

[4] Oliver A. McBryan. An overview of message passing environments. Parallel Computing, 1994.

[5] Aad J.vander Steen and Jack J. Dongarra. Overview of recent supercomputers. Technical Report UT-CS-96-325, Department of Computer Science, University of Tennesee, April 1996.

[6] Aad van der Steen and Jack Dongarra. Overview of high performance computers. In Handbook of Massive Data Sets. Kluwer Academic Publishers Group, Norwel, MA, USA, and Dordrecht, The Netherlands, 2002.

[7] W. Daniel Hilis and Guy L. Steele. Data paralel algorithms. Communications oftheACM, 29(12):1170–1183, December 1986.

[8] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings, Redwood City, CA, 1994.

[9] I Foster. Designing and Building Parallel Programs. Addison Wesley, 1995.

[10] David E. Culler, Jaswinder Paul Singh and Anoop Gupta. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kauffman Publishers, inc., San Francisco, CA, 1999.

[11] Jorba Esteve, Josep. "Análisis automática de prestaciones de aplicaciones basadas en paso de mensajes". Tesis doctoral – Universidad Autónoma de Barcelona. Barcelona. España. 2006. www.tesisenxarxa.net/TESIS_UAB/AVAILABLE/TDX-1013106-132034//jje1de1.pdf

[12] Per Brinch Hansen. Model programs for computational science: A programming methodology for multicomputers. Concurrency: Practice & Experience,5(5):407–423, August1993.

[13] R. Buyya, editor. High Performance Cluster Computing: Architectures and Systems. Prentice Hal, June 1999.

[14] D.I. Moldovan. Parallel Processing -From Applications to Systems. Morgan Kaufmann, San Mateo, 1993.

[15] Moti Ben-Ari. Principles of Concurrent Programming. Prentice-Hal International, Inc., Englewood Clifs,1990.

[16] Satish Chandra, James R. Larus, and Anne Rogers. Where is time spent in message-passing and shared-memory programs? Technical Report TR-463-94, Princeton University, Computer Science Department, July1994.

[17] F. Wolf and B. Mohr. Automatic performance analysis of hybrid mpi/openmp applications. In Parallel, Distributed and Network Based Processing, 2003. Proceedings. Eleventh Euromicro Conference on, pages 13–22, 2003.

[18] Philip J. Hatcher and Michael J. Quinn. Data–Parallel Programming on MIMD Computers. The MIT Pres, Cambridge, MA, 1991.

[19] B. Chapman, P. Mehrotra, and H. Zima. Extending HPF for advanced data-parallel applications. IEEE Parallel and Distributed Technology, 2(3):15–27, 1994.

[20] Bary Wilkinson and Michael Allen. Parallel Programming. Prentice Hall, 1999.

[21] Message Passing Interface Forum. Mpi-2: Extensions to the message-passing interface. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html, 1997.

[22] Weihang Jiang, Jiuxing Liu, Hyun-Wook Jin, D.K. Panda, W.Gropp, and R. Thakur. High performance mpi-2 one-sided communication over infiniband. In Cluster Computing and the Grid, 2004. CC Grid 2004. IEEE International Symposium on, pages 531–538, 2004.

[23] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S. Sunderam. PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Network Parallel Computing. Scientific and Engineering Computation Series. MIT Pres, Cambridge, MA, 1994.

[24] Wiliam Gropp, Ewing Lusk Anthony Skjelum, and Nathan Doss. A high-performance, portable implementation of the MPI message passing interface standard, December 26 1996.

[25] J. Dongarra, S. Otto, M. Snir, and D. Walker. A message passing standard for mpp and workstations. Communications of the ACM, 39(7):84–92, 1996.