Programación Paralela en Sistemas Híbridos

Verónica Gil-Costa, Fernando Saez, Cristian Tissera y Marcela Printista
LIDIC, Departamento de Informática
Facultad de Ciencias Físico, Matemáticas y Naturales
Universidad Nacional de San Luis

CONTEXTO

Esta línea de investigación pertenece al Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC) de la UNSL, el cual tiene por objetivo actual el diseño e implementación de nuevas herramientas de computación inteligente para desarrollar Sistemas Inteligentes. También se incluyen entre las áreas de investigación del LIDIC los Sistemas Multiagente, los Sistemas Híbridos y las Técnicas de Distribución y Paralelismo para acelerar los procesos de búsqueda y mejorar la calidad de los resultados.

RESUMEN

Con la aparición de las CPU multi-cores (o Chip-level-Multi-Processor -CMP-), es importante el desarrollo de las técnicas que exploten las ventajas de las CMP para acelerar las aplicaciones paralelas que poseen una gran demanda de cómputo paralelo. En particular, las aplicaciones que requieren de un gran poder computacional de los recursos disponibles, es esencial poder desarrollar estrategias y algoritmos que aprovechen el uso adecuado del hardware. Esto es especialmente crítico cuando se consideran sistemas o aplicaciones en las que los requerimientos ingresan en intervalos variables. En este trabajo se propone el desarrollo de técnicas híbridas basadas en el uso de MPI para la comunicación entre procesadores y OpenMP para la comunicación entre cores de un mismo procesador. OpenMP ha sido desarrollado para tomar ventaja de las facilidades multithreading de los nodos CMP.

Palabras clave: *sistemas multi-core, OpenMP, MPI, simulación paralela, algoritmos paralelos, clusters*

1. INTRODUCCION

Modificar un programa secuencial para distribuirlo o para que se ejecute en paralelo tiene dos atractivas razones: acelerar la velocidad de ejecución y/o mejorar la cantidad de memoria disponible [Hol07]. Actualmente existen dos alternativas para paralelizar aplicaciones secuenciales: (a) utilizar clusters de computadoras, y (b) utilizar sistemas multi-cores. Los sistemas multi-core difieren en varios aspectos respecto de un cluster de computadoras. Un sistema multi-core ofrece a todos los cores un acceso rápido a una única memoria compartida, evitando la transferencia de datos entre las máquinas a través de la red del cluster, pero aún así la cantidad de memoria disponible es limitada. En cambio los clusters de computadoras permiten incrementar el espacio de almacenamiento (principal y secundario) aunque deben pagar el precio de la latencia de la red. Una tercera alternativa (c) consiste en un sistema híbrido que permite combinar las características de ambos sistemas, e incrementar aun más la capacidad y el poder de los sistemas computacionales. Esta última alternativa es el eje central de la línea de investigación propuesta en este trabajo. Sin embargo el diseño de aplicaciones para este tipo de sistema trae aparejado una serie de complejidades que se describen en esta sección.

La tendencia actual de la tecnología destaca la aparición de procesadores CMP [Str69, Hamm97, Hamm00]. En realidad, la mayoría de los sistemas que incorporan estos chips descartan la vieja idea de un sistema multiprocesador como varios nodos mono-procesador. Las tendencias de la tecnología indican que el número de cores (núcleos) en un chip seguirá creciendo a medida que lo indiquen los planes de trabajo de los fabricantes más importantes. Hoy en día, AMD trabaja con chips de cuatro núcleos (Quad tecnología nativa) e Intel ha comenzado a incorporar el procesador Intel Core™ de cuatro y ocho núcleos de procesador en sus sistemas.

Un procesador multi-core combina dos o más núcleos (normalmente una CPU) en un único paquete compuesto de un sólo circuito integrado (CI) denominado *die*. Un procesador dual-core contiene dos núcleos, y un procesador quad-core posee cuatro núcleos. Un microprocesador multi-core implementa multiprocesamiento en un solo paquete físico. Un procesador con todos los núcleos en una sola *die* se denomina procesador monolítico. Los cores pueden compartir una única caché coherente del nivel más alto del dispositivo de memoria caché (por ejemplo, L2 para el Intel Core 2) o puede tener cachés separadas (por ejemplo, los procesadores AMD dual-core). Los procesadores también comparten la misma interconexión con el resto del sistema. Cada core implementa independientemente optimizaciones como ejecución superescalar, pipelining, y multithreading. Un sistema con n cores es efectivo cuando se presenta con n o más hilos simultáneamente. La ganancia obtenida por el uso de sistemas multi-cores, depende de las características del problema que se quiere

resolver y del algoritmo utilizado. La Figura 1 muestra el diseño de un chip dual core.

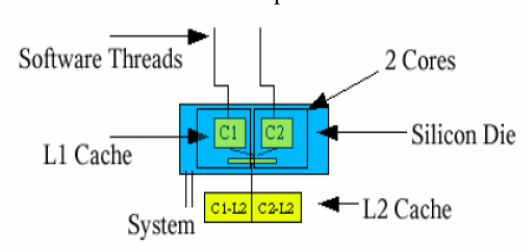

Figura 1: Procesador con CMP.

Las mejoras que se obtienen al utilizar sistemas multi-cores se reflejan a través del paralelismo desarrollado para las aplicaciones. Si se ejecuta un programa secuencial sobre un sistema multi-core no se obtiene ninguna ganancia. Por lo tanto, la ventaja de estos sistemas consiste en dividir eficientemente la ejecución de una aplicación, para que cada parte sea ejecutada por un thread independiente. La cantidad de threads disponibles en estos sistemas también es un tema importante a analizar, debido a que la creación y administración de threads requiere del uso de los recursos tales como la memoria; además los threads deben ser cuidadosamente planificados e incorporados en el stack de ejecución.

Por lo tanto, el uso de chips multi-core en los sistemas de computación de uso común conllevará, sin lugar a dudas, grandes cambios en la forma de desarrollar aplicaciones y software. En esta nueva etapa, será necesario reescribir los programas existentes, paralelizando el código para sacar provecho de los sistemas con múltiples núcleos. Una de las cuestiones básicas cuando se diseñan programas en paralelo es coordinar los accesos a los datos compartidos de manera que los resultados de la ejecución sean correctos. El uso de mecanismos de coordinación entre procesadores basados en el acceso exclusivo a datos (uso de exclusiones mutuas de grano fino) no se percibe como posibles soluciones por la complejidad que supone su uso para la mayoría de los programadores. Las zonas críticas no sólo son complejas de administrar, sino que son posibles fuentes de cuellos de botella cuando varios threads deben acceder a ellas para realizar una operación.
Los trabajos realizados hasta el momento en sistemas multi-core, se focalizan en la eficiente paralelización de un conjunto de instrucciones de un programa secuencial [Fri06, Ben05, Cho07, Goo07], para ellos proponen y evalúan dos métodos principales denominados parallel depth-first (pdf) y work-stealing (ws). También existen algunas variantes que optimizan estas estrategias [Gu]. Sin embargo, estos trabajos no aprovechan las características inherentes en las aplicaciones que se desean paralelizar. Es decir que ninguno de estos trabajos toma ventaja de las características particulares de las aplicaciones que se desean paralelizar. Es importante estudiar la forma en particular que trabajan las aplicaciones, debido que para ciertas operaciones es conveniente utilizar uno u otro tipo de planificación paralela.

Todavía no hay estudios con esta tecnología que evalúen los paradigmas de programación paralela que trabajan en el contexto de las aplicaciones que resuelven requerimientos de los usuarios en un corto plazo y que requieren de un gran poder computacional, y si son las más apropiadas.

Otro aspecto que no ha sido estudiado hasta el momento en sistemas multi-core, es el efecto que tienen los cambios de tráfico de los requerimientos ingresados al sistema. Si el procesador posee varios cores que pueden trabajar independientemente pero compartiendo los datos de memoria, cuando los requerimientos de los usuarios ingresan en intervalos cortos o con una gran afluencia en el sistema, es conveniente utilizar threads que trabajen en forma independiente sobre cada requerimiento. Sin embargo, cuando los requerimientos ingresan entre intervalos de tiempo lo suficientemente grandes, es conveniente que los requerimientos se dividan en tareas atómicas y que los threads que se encuentran ociosos “roben” tareas de los threads que tienen una carga de trabajo elevada. De esta manera es posible optimizar el uso de los recursos disponibles en el sistema.

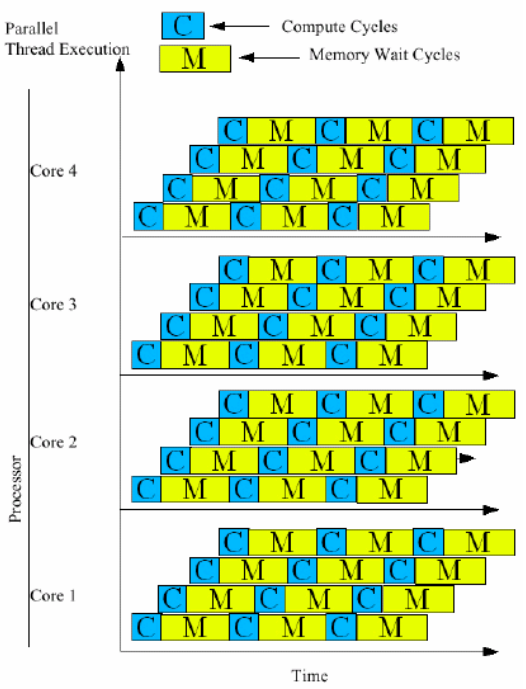

Figura 2: Con CMP se incrementa la velocidad de procesamiento sin modificar la velocidad de acceso a la memoria.

Este mismo sistema híbrido de comunicación puede ser utilizado a nivel de procesadores del cluster de la siguiente manera. Cuando el tráfico de los requerimientos es alto, conviene administrar y procesar estos requerimientos en masa (*bulk*). De esta manera se logra reducir los costos (latencia) de enviar mensajes por la red de intercomunicación y por lo tanto mejorar los tiempos de respuestas. Por otro lado, cuando los tiempos entre arribos de los requerimientos son los suficientemente grande (es decir que llegan pocos requerimientos al sistema), es conveniente utilizar un modo de comunicación completamente asíncrono, donde cada procesador resuelve sus tareas independientemente [Mar08a, Mar08b].

Las técnicas de planificación tanto a nivel micro (dentro de cada procesador) y macro (en un cluster de computadoras) deben ser capaces de obtener un buen balance de carga. Si los todos los cores/procesadores logran realizar aproximadamente la misma cantidad de operaciones, esto permite mejorar el tiempo de respuesta a los requerimientos, así como incrementar el thoughput. Pero para ello es esencial un uso equitativo de los recursos por parte de todos los requerimientos. Para lograr una planificación adecuada de las operaciones a realizar, es posible utilizar un *scheduler* que sea capaz de predecir el comportamiento de los cores/recursos dependiendo del tipo de requerimiento que se esté procesando. Si el sistema opera en modo síncrono la tarea del scheduler es sencilla y puede predecir la cantidad de tareas que cada core/procesador debe realizar en cada etapa realizando mediciones y operando cada Nq consultas terminadas. Esto se puede llevar a cabo utilizando un modelo de colas G/G/∞, donde el cociente de la tasa de llegada de eventos y la tasa de servicios (λ/μ) determina el número de servidores activos. La tasa de llegada se define como $\lambda=\Delta n$, durante un período de tiempo Δ en el que se procesaron n requerimientos. La tasa de servicio se define como $\mu=n/S$, donde S es la suma de las diferencias de los tiempos de salida menos los tiempos de arribo δq[Tpo_Salida – Tpo_arribo].

Sin embargo, cuando el sistema opera en modo asíncrono, la panificación de los nuevos requerimientos tiene una complejidad adicional, debido a que en un período de tiempo determinado los cores/procesadores pueden tener un número diferente de requerimientos en espera o en servicio. Además, no existe ningún tipo de control que permite visualizar el estado del sistema en un instante dado. En este caso es posible utilizar un sistema de planificación síncrono ficticio que permita predecir el comportamiento de del sistema real.

Teniendo en cuenta esta tendencia, el objetivo principal de este trabajo es estudiar, cuál es la forma más eficiente para explotar esta nueva tecnología, aplicándola sobre un cluster de computadoras. De esta forma se logra obtener un sistema hibrido que combina las propiedades de "divide y vencerás" en un nivel macro (a nivel de cluster con memoria distribuida), y en un nivel micro (dentro de cada procesador con memoria compartida).

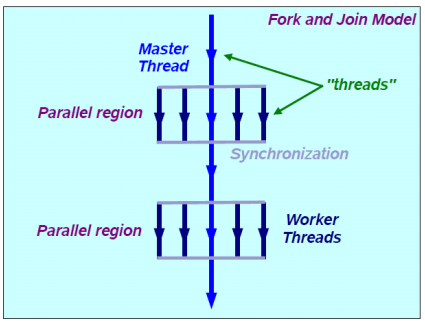

Figura 3: Programación con OpenMP.

2. LINEAS DE INVESTIGACION y DESARROLLO

La línea de investigación involucra una serie de desarrollos individuales que en su conjunto logran obtener el objetivo planteado. Para ello es necesario estudiar formas eficientes de programación paralela utilizando el lenguaje OpenMP sobre arquitecturas multi-cores. Como se mencionó anteriormente, es de suma importancia reducir las regiones críticas o zonas de memoria compartidas para evitar los cuellos de botellas. También es necesario estudiar las ventajas y desventajas provistas por los sistemas de clusters de computadora que se comunican mediante librerías paralelas como MPI (asíncrona) y/o BSP (síncrona).

Para optimizar la eficiencia de los algoritmos paralelos, es necesario determinar las características particulares de las aplicaciones. Además, es necesario utilizar estructuras de datos que sean capaces de manejar eficientemente grandes cantidades de datos (miles de Terabytes).

2.1. Línea 1: Algoritmos de Búsquedas Avanzados

Una aplicación que requiere de un gran poder computacional y que puede beneficiarse de las características provistas por los sistemas multicores son las aplicaciones que soportan búsquedas de objetos multimediales. Estas deben ser capaces de resolver requerimientos en un tiempo razonable y al mismo tiempo dependen de operaciones computacionales complejas generalmente de alta dimensionalidad. Un ejemplo de estos sistemas es el sitio Flickr (http://www.flickr.com/) que utiliza estructuras de datos poco convencionales basadas en

árboles y mapas. Este tipo de estructuras es poco conveniente al momento de utilizar un sistema multi-core, debido a los accesos simultáneos a celdas de memoria. Por lo tanto, es necesario diseñar cuidadosamente las estructuras de almacenamiento y de indexación [GilC08a, GilC08b].

2.2. Línea 2: Programación Paralela Estructurada
Esta línea trata con nuevas formas de programación paralela que tienen por objetivo reducir el tiempo de diseño, testing y codificación de aplicaciones paralelas. En esta dirección estamos desarrollando un sistema de programación esqueletal que simplifique la tarea de desarrollar software paralelo. Con el advenimiento de las arquitecturas actuales (cluster con nodos multicores) es necesario la adaptación de estos sistemas para obtener las ventajas del modelo híbrido (Pasaje Mensajes y de Memoria compartida). Además, esto implica abordar nuevos problemas que surgen en este entorno, como la sincronización, planificación, mapeo de procesos sobre procesadores, balance de carga, etc [Sae08a, Sae08b].

2.3 Línea 3: Simulación en Ciencia Computacional
En esta línea se aborda la simulación de alta performance. El propósito es realizar una aplicación paralela utilizando un sistema híbrido basado en OpenMP y MPI, para desarrollar un modelo de simulación basado en autómatas celulares (AC) y agentes inteligentes, destinado al estudio de las dinámicas pedestres en situaciones de emergencia, especialmente aquellas que involucran la evacuación forzada debido a la amenaza del fuego dentro de un área definida cerrada con un número específico de salidas.

3. RESULTADOS OBTENIDOS/ESPERADOS

Los resultados obtenidos hasta el momento son:

- Diseño e implementación de algoritmos eficientes que permiten resolver consultas ingresadas por los usuarios en un sistema de búsqueda. El sistema ha sido implementado para soportar consultas de texto y multimediales sobre un cluster de computadoras[GilC08c].
- Los sistemas de búsquedas desarrollados operan en forma síncrona y asíncrona [Mar08a].
- Se han implementado estrategias que permiten resolver eficientemente los requerimientos de los usuario en sistemas multi-core y que realizan una planificación de las tareas para optimizar el uso de los recursos.
- El desarrollo de esqueletos paralelos que resuelven algoritmos del tipo "Divide y Vencerás" y "Autómata Celular" para el modelo de pasaje de mensajes[Sae07].
- Un modelo teórico para predecir el tiempo de ejecución de una aplicación que se instancia en el esqueleto "Divide y Vencerás" [Sae08b].
- Un modelo secuencial de la simulación basada en autómatas celulares para el estudio de procesos de evacuación [Tis06a, Tis06b].

Los resultados esperados son:

- Combinar diferentes modos de comunicación (síncrono/asíncrono) sobre el sistema que opera sobre un cluster de computadoras.
- Obtener un sistema híbrido que permite utilizar los beneficios provistos por los sistemas multi-core y los clusters de computadoras. Para ello es posible utilizar programas que utilicen tanto la librería OpenMp como MPI/BSP.
- Aplicar algoritmos de scheduling inteligentes que se adapten a los cambios del ambiente (tráfico de requerimientos variable) para balancear la carga de trabajo a nivel macro y micro.
- Aprovechar las características de las estructuras de datos utilizadas como índices, para explotar la localidad de los datos y reducir los costos de comunicación (latencia), y los fallos a memoria caché.
- Adaptar otros esqueletos algorítmicos a los sistemas de alta performance actuales (híbridos).

4. FORMACION DE RECURSOS HUMANOS

Los investigadores de esta línea de trabajo se desarrollan en el Laboratorio de Investigación y desarrollo en Inteligencia Computacional (LIDIC). Actualmente, el mismo cuenta con un subsidio de Ciencia y Técnica de la UNSL y un Proyecto PICT del FONCyT.

En la temática de la línea se están desarrollando tres doctorados en la UNSL:

1- El doctorando es becario de CONICET y el trabajo se ha realizado en cooperación con investigadores del Centro de Investigación de la Web-Chile (Próximo a terminar).
2- El doctorando es becario de CONICET. (En desarrollo, plan de tesis aprobado).
3- El doctorando es docente exclusivo de la UNSL y el trabajo se desarrolla en cooperación con la Universidad Autónoma de Barcelona. (En desarrollo, se esta confeccionando su plan de tesis).

Subsidios externos:
Durante el año 2007 y 2008 estos investigadores han sido parcialmente subsidiados por el proyecto CYTED GRID (finalizado) para realizar viajes de

estudios tendientes a completar su formación de posgrado.
Durante el 2008, el Centro Yahoo! Research, becó en 6 oportunidades a una investigadora para realizar investigaciones conjuntas.
Durante el 2008, una investigadora fue becada por el centro de investigación High Performance Computing Center Stuttgart (HLRS) de la Universidad de Stuttgart, Alemania. Financiado por el programa HPC-Europa Transnational Access para realizar una estadía de perfeccionamiento de 2 meses.

5. BIBLIOGRAFIA

[Ben05] Concurrent cache-oblivious B-trees. M. A. Bender, J. T. Fineman, S. Gilbert, and B. C. Kuszmaul. In ACM SPAA, 2005.

[Cho07] The cache-oblivious gaussian elimination paradigm: Theoretical framework, parallelization and experimental evalua-tion. R. Chowdhury and V. Ramachandran. In ACM SPAA, 2007.

[Fri06] The cache complexity of multithreaded cache oblivious algorithms. M. Frigo and V. Strumpen. In ACM SPAA, 2006.

[GilC08a] Parallel query processing on distributed clustering indexes. V. Gil-Costa, M. Marin and N. Reyes Journal of Discrete Algorithms (7) 03-17, 2009 (Elsevier).

[GilC08b] An Empirical Evaluation of a Distributed Clustering-Based Index for Metric Space Databases, G.V. Costa, M. Marin and N. Reyes. In International Workshop on Similarity Search and Applications (SISAP 2008), Cancun, Mexico, April 11-12, pp. 386-393, IEEE-CS Press, 2008

[GilC08c] Distributed Sparse Spatial Selection Indexes, G.V. Costa and M. Marin. In 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing(PDP 2008), Toulouse, France, Feb. 13-15, pp. 440-444, IEEE-CS Press, 2008

[Goo07] Sorting in parallel external-memory multicores. M. T. Goodrich, M. Nelson, and N. Sitchinava Technical report, U.C. Irvine, 2007.

[Gu] Provably Good Multicore Cache Performance for Divide-and-Conquer Algorithms. Guy E., Blelloch, Rezaul A. Chowdhury, Phillip B. Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch.

[Ham00] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra CMP. IEEE Micro, 20(2), 2000.

[Hamm97] L. Hammond, B. Nayfeh, and K. Olukotun. A single-chip multiprocessor. IEEE Computer, 30(9), 1997.

[Hol07] The Design of a Multi-Core Extension of the SPIN Model Checker. Gerard J. Holzmann and Dragan Bošnački. PDMC 2007, J. Electronic Notes in Theoretical Computer Science, pp. 33-46, 2007.

[Mar08a] M. Marin, G.V. Costa, (Sync|Async)+ MPI Search Engines, In Euro PVM/MPI: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Paris, France, Sep. 30-Oct. 3, Lecture Notes in Computer Science 4757, pp. 117-124, Springer, 2007

[Mar08b] High-Performance Distributed Inverted Files, M. Marin, G.V. Costa. In ACM 16th Conference on Information and Knowledge Management (CIKM 2007), Lisbon, Portugal, Nov 6-9, pp. 935-938, ACM, 2007.

[Sae07] Conceptos Fundamentales de Diseño en Sistemas de programación Esqueletal. XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007). Universidad Nacional del Nordeste, Corrientes y Resistencia, Argentina, Octubre 2007.

[Sae08a] Predicting the Performance of Hypercube Divide and Conquer Skeleton. F. Saez, M.Printista. Jornadas Chilenas de computación. Punta Arenas, Chile. Noviembre de 2008.ISBN 978-956-319-507-1

[Sae08b] A Performance Model for Divide and Conquer Skeletons on Homogeneous Cluster". F. Saez, M. Printista. 7th International Information and Telecommunication Technologies Symposium (I2TS´08). Brasil. Págs. 139-147.

[Str69] V. Strassen. Gaussian elimination is not optimal. Numerische Mathematik, 13(4), 1969. Springer.

[Tis06a] Tesis de Licenciatura en Cs. de la Computación: "Simulación basada en autómatas celulares para el estudio de procesos de evacuación". Director: Marcela Printista, Co-director: Dr. Marcelo Errecalde. Resolución 490/06.

[Tis06b] Tissera C., Errecalde M. Printista A.M (2006). Simulación De Evacuaciones basada en Autómatas Celulares. (XII CACIC 2006). Págs. 1837 a 1848. ISBN 950-609-050-5. Argentina.