Análisis de Paralelización con Memoria Compartida y Memoria Distribuida en Clusters de Nodos con Múltiples Núcleos

Fernando G. Tinetti[1]

III-LIDI
Facultad de Informática, UNLP
50 y 120, 1900, La Plata, Argentina
fernando@info.unlp.edu.ar

Gustavo Wolfmann

Laboratorio de Computación
Fac. Cs. Exactas, Físicas y Naturales
Universidad Nacional de Córdoba
gwolfmann@gmail.com

Resumen

En este trabajo se presentan las alternativas y los resultados de rendimiento obtenidos del análisis de las alternativas de paralelización en clusters de nodos con múltiples núcleos. El objetivo final es mostrar si es necesario tener en cuenta los dos modelos de procesamiento y paralelización (memoria compartida y memoria distribuida) o solamente uno de ellos. La aplicación utilizada es clásica en el contexto de cómputo de alto rendimiento: la multiplicación de matrices. Si bien esta operación es representativa de las aplicaciones de álgebra lineal, se muestran los resultados en términos de las condiciones bajo las cuales se puede optimizar rendimiento y hacia dónde debe estar enfocado el esfuerzo de la paralelización de algoritmos en los clusters de nodos con múltiples núcleos. Estos clusters son considerados como los estándares de bajo costo hoy en día, dado que casi cualquier máquina de escritorio con la que se construyen los clusters está basada en un procesador con más de un núcleo e, inclusive con más de un procesador. En cualquier caso, todas las unidades de procesamiento deberían ser utilizadas al máximo para optimizar el rendimiento obtenido por las aplicaciones paralelas.

Palabras claves: *Paralelización con Memoria Compartida, Paralelización con Memoria Distribuida, Cómputo Paralelo en Clusters, Algebra Lineal en Paralelo.*

Abstract

This article presents the alternatives and performance results obtained after analyzing parallelization alternatives in clusters of nodes with multiple cores. The ultimate goal is to show if both processing and parallelization models (shared memory and distributed memory) need to be taken into account, or if only one of them is enough. The application used is classical in the context of high-performance computing: matrix multiplication. Even though this operation is representative of linear algebra applications, results are shown in terms of the conditions under which performance can be optimized and where algorithm parallelization efforts should be focused on for clusters of nodes with multiple cores. These clusters are nowadays considered as low-cost standards, since almost any desktop computer used to build clusters is based on a multi-core processor, and even on multi-processors. In any case, all processing units should be used to their maximum to optimize the performance of parallel applications.

Keywords: *Shared Memory Parallelization, Distributed Memory Parallelization, Cluster Parallel Computing, Parallel Linear Algebra.*

[1] Investigador Asistente, Comisión de Investigaciones Científicas de la Provincia de Buenos Aires.

1 Introducción

En el contexto de cómputo paralelo, los clusters, o directamente las redes locales de computadoras construidas para cómputo paralelo, siguen teniendo la mejor relación costo/rendimiento para cómputo numérico con grandes requerimientos de cómputo u operaciones numéricas [2]. Sin embargo, desde las primeras propuestas de hardware y software en la segunda mitad de la década de 1990 hasta hoy, han habido cambios cuantitativos y también cualitativos que deberían ser tenidos en cuenta a la hora de paralelizar aplicaciones y de analizar rendimiento. Desde un punto de vista cuantitativo, el incremento de rendimiento de cada procesador de las computadoras de escritorio ha continuado incrementándose, aunque aproximadamente a principios de la década de 2000 ya se comenzaron a visualizar los límites en cuanto al incremento de ciclo de reloj que los procesadores podían utilizar. Más allá de las razones tecnológicas, la evolución fue marcada en un principio hacia arquitecturas superescalares, luego con capacidad de procesar múltiples hilos (o *threads*) y luego directamente con múltiples núcleos independientes al menos hasta el primer nivel de memoria cache [11] [10] [7].

Ya desde la segunda mitad de la década de 2000, los procesadores con más de un núcleo se consideraron estándares y a medida que pasa el tiempo va a ser cada vez menos probable adquirir una computadora de escritorio con un único núcleo de procesamiento. Tal como ha sido la evolución del mercado, las computadoras de escritorio con más de un núcleo de procesamiento pasarán a ser las de menor costo (o las *únicas*, de hecho) disponibles, y el costo variará en función de otras características, como la frecuencia de reloj, el tamaño de memoria principal, los periféricos instalados, etc. Esto necesariamente tiene impacto directo sobre los clusters que se instalan/rán y que se utilizarán para cómputo paralelo: cada nodo o computadora tendrá más de un núcleo (o CPU, como se le ha llamado tradicionalmente). Aunque esta característica puede verse como “natural” en el entorno de cómputo paralelo, dado que siempre se ha conservado la tendencia a utilizar más de un procesador o núcleo, la paralelización de aplicaciones no necesariamente se puede mantener al margen de esta variación en la arquitectura de los nodos de los clusters.

Desde la perspectiva de procesamiento, una computadora con múltiples procesadores y/o con múltiples núcleos por procesador puede clasificarse, basándose en [6] como MIMD (*Multiple Instruction stream, Multiple Data stream*) de memoria compartida o multiprocesador. Siguiendo la misma clasificación, un cluster es un MIMD de memoria distribuida o multicomputadora. La diferencia a nivel de procesamiento y relación entre las múltiples unidades de procesamiento o CPUs es evidente: mientras que en los multiprocesadores toda la comunicación y la sincronización se realiza utilizando la memoria compartida, en las multicomputadoras normalmente se utiliza comunicación explícita, vía pasaje de mensajes [9]. En este punto la diferencia en cuanto a la posible paralelización de aplicaciones se hace evidente como mínimo en cuanto a la programación: la implementación de algoritmos en memoria compartida es esencialmente diferente de la implementación en memoria distribuida. Más aún, los algoritmos suelen ser claramente diferentes para la optimización de un tipo de arquitectura en particular [5]. En el caso extremo quizás, existen bibliotecas tan establecidas y aceptadas como ScaLAPACK (Scalable LAPACK, cuya última versión es de Abril de 2007 [22]) que directamente se orientan a un tipo de arquitectura en particular: MIMD de memoria distribuida [3] [22].

La propuesta de este trabajo se orienta a identificar lo más claramente posible en qué medida se puede evitar el hecho de tener dos tipos diferentes de procesamiento paralelo en un cluster cuyos nodos tienen múltiples núcleos. La idea es atacar por separado, siempre que sea posible, la paralelización en un nodo, donde se tiene un MIMD de memoria compartida y la paralelización en

un cluster, donde se tiene un MIMD de memoria distribuida, usualmente programado con pasaje de mensajes. Se utilizará un problema específico ampliamente conocido en el área de procesamiento numérico en paralelo: la multiplicación de matrices. A diferencia de [21], la idea final es analizar la posibilidad de directamente enfocar la paralelización a un entorno de memoria distribuida definiendo los alcances o los límites de este enfoque teniendo en cuenta que en cada nodo se tiene cómputo paralelo con memoria compartida. Claramente, el parámetro de análisis será el rendimiento obtenido por la paralelización.

2 Procesamiento Paralelo: Multiprocesadores y Multicomputadoras

El modelo de procesamiento en un multiprocesador puede considerarse como una extensión relativamente natural a partir del modelo de concurrencia que implementan los sistemas operativos estándares. Coexisten varios procesos en el mismo sistema, que se coordinan y/o compiten para llevar a cabo procesamiento. En el caso de las aplicaciones paralelas el problema es único y suele ser dividido entre los diferentes procesos para que cada uno de ellos resuelva una parte del total. Lo interesante o ventajoso del modelo de memoria compartida es que permite al programador concentrarse casi directamente en la mejor forma de paralelizar, dado que en general el modelo de memoria compartida es conocido en cuanto a los mecanismos de comunicación y sincronización de procesos. En las computadoras de escritorio se incluyeron más recursos de procesamiento, específicamente más de un procesador o CPU, como primera medida orientada a resolver o mejorar el problema de productividad o *throughput*. El resultado fue denominado multiprocesamiento simétrico (o SMP, de *Symmetric MultiProcessing*), dado que todos los procesadores tienen el mismo tipo de acceso o prioridad de acceso a todo el sistema, no solamente a la memoria única y compartida [4] [15]. La evolución tecnológica ha llegado a producir varios núcleos o CPUs completas en un solo dispositivo, donde el principio de cómputo es exactamente el mismo: varios procesadores que comparten memoria.

El modelo de procesamiento en una multicomputadora está esencialmente basado en memoria distribuida, donde los procesos corriendo en procesadores o, más aún, en las computadoras interconectadas deben explícitamente enviar y recibir información por pasaje de mensajes [9] [20]. Tradicionalmente, se ha reconocido que este modelo de hardware es el más escalable, dado que permite llegar actualmente a configuraciones con miles y decenas de miles de procesadores que pueden computar en paralelo [23]. También tradicionalmente se ha reconocido la necesidad de paralelizar o utilizar algoritmos paralelos específicos para este tipo de arquitecturas, siguiendo el modelo de programamción de pasaje de mensajes, hoy implementado mayoritariamente con el estándar MPI (Message Passing Interface) [14].

Desde la perspectiva de cómputo paralelo en clusters, se llega a la situación que se muestra en la Fig. 1, donde se tiene un conjunto de computadoras interconectadas por una red local y en cada computadora se disponen de múltiples núcleos o CPUs. Los algoritmos paralelos en general tienden a adaptarse a este relativamente nuevo entorno de procesamiento paralelo y los algoritmos paralelos de procesamiento numérico en particular no son la excepción. En el área de álgebra lineal en paralelo se han propuesto y analizado varias alternativas, normalmente utilizando operaciones con matrices [21] [13] [16]. En todos estos casos, la idea es tratar de aprovechar el paralelismo dentro de un multiprocesador en términos de resolver subproblemas (en cada una de las computadoras $Comp_0$, ..., $Comp_s$ de la Fig. 1), de forma tal que el cómputo local en cada computadora se optimiza. A la vez, el programa paralelo que trata con todo el problema puede ser considerado básicamente como de pasaje de mensajes en una multicomputadora (el conjunto de computadoras

interconectadas $Comp_0$, ..., $Comp_s$ de la Fig. 1). Este modelo tambien es conocido como *hybrid MPI/OpenMP programming* [24,25].

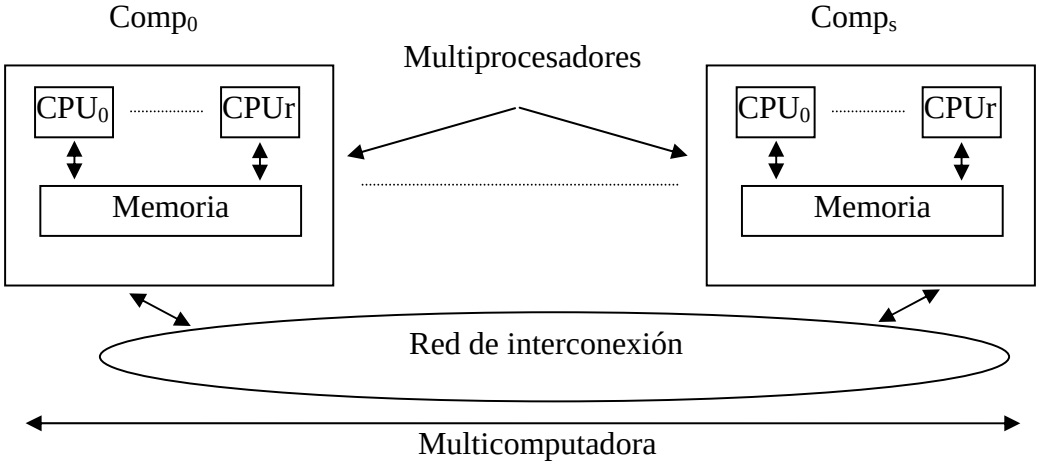


Figura 1: Cluster de Nodos con Múltiples Núcleos.

Dado que el objetivo final del procesamiento numérico en paralelo se enfoca hacia el rendimiento, la idea será optimizar el uso de cada una de las CPU en cada una de las computadoras y, finalmente, utilizar al máximo posible cada computadora del cluster.

3 Algoritmos y Evaluación de Rendimiento

En vez de proponer directamente un algoritmo para optimizar a la vez el rendimiento tanto para la arquitectura distribuida de un cluster como para los multiprocesadores, inicialmente se analizarán las posibilidades de optimización de a un paso por vez. En principio, se analiza en un único núcleo (CPU), luego la optimización en una computadora con multiprocesamiento (con memoria compartida) y finalmente la utilización de un programa paralelo con pasaje de mensajes.

3.1 Optimización en un Núcleo

Un núcleo de un procesador es suficientemente complejo como para justificar en sí mismo el trabajo de optimización, dadas características como la disponibilidad de múltiples unidades escalares (superescalares), las extensiones SIMD (*Single Instruction stream, Multiple Data stream*), y la ejecución especulativa. De hecho, este trabajo ni siquiera debe ser hecho por un programador de aplicaciones, sino que normalmente las empresas que proveen procesadores ponen a disposición bibliotecas optimizadas que ya tienen incorporadas las optimizaciones que corresponden, al menos en cuanto a las operaciones numéricas más importantes o consideradas más utilizadas. Por ejemplo, AMD provee ACML (*AMD Math Core Library*) [1] e Intel pone a disposición la biblioteca MKL (*Intel Math Kernel Library*) [12]. Dentro de las alternativas de código abierto es destacable ATLAS (*Automatically Tuned Linear Algebra Software*) [19]. Por lo tanto, este problema podría considerarse resuelto para el programador de aplicaciones paralelas en un cluster.

3.2 Cómputo Paralelo Optimizado en un SMP o Multiprocesador

Evidentemente aquí el problema es esencialmente de procesamiento paralelo, dado que se trata de

aprovechar al máximo todos los núcleos disponibles en un multiprocesador. Sin embargo, se tienen varios factores que acotan la complejidad de la paralelización en los multiprocesadores actuales. Por un lado, la cantidad de núcleos disponibles en cada computadora no suele exceder los 16-32 núcleos. De hecho, las computadoras de escritorio que tienen la mejor relación costo/rendimiento suele tener entre 2 y 4 núcleos. Esto sin duda es ventajoso desde la perspectiva de análisis de rendimiento, dado que las alternativas de paralelización pueden ser consideradas automáticamente acotadas a relativamente pocas CPUs que comparten memoria. Por otro lado, las mismas empresas u organizaciones que proveen bibliotecas optimizadas para un único núcleo o CPU también proveen alternativas paralelas de sus bibliotecas para sistemas multiprocesadores. De hecho, esto sucede con todas las bibliotecas mencionadas anteriormente: ACML, MKL y ATLAS. Aunque podría considerarse que estas bibliotecas no pueden ser mejoradas, se pueden implementar algoritmos relativamente sencillos y optimizados para multiprocesadores para analizar el rendimiento desde dos puntos de vista, para identificar:

1. Si el rendimiento de las bibliotecas puede ser mejorado con los algoritmos paralelos diseñados específicamente para multiprocesadores.
2. Si los algoritmos son realmente optimizados para multiprocesadores o tienen alguna clase de penalización que *a priori* no se conoce.

Específicamente en este trabajo se implementó un algoritmo de multiplicación de matrices similar al detallado en [18] que se muestra de manera esquemática en la Fig. 2, para computadoras paralelas de memoria compartida con cuatro núcleos. A cada uno de los núcleos se le asigna la respectiva porción de A y la totalidad de B, de forma tal que cada procesador computa un bloque filas de C (en la multiplicación C = A x B), accediendo al bloque correspondiente de A y a toda la matriz B.

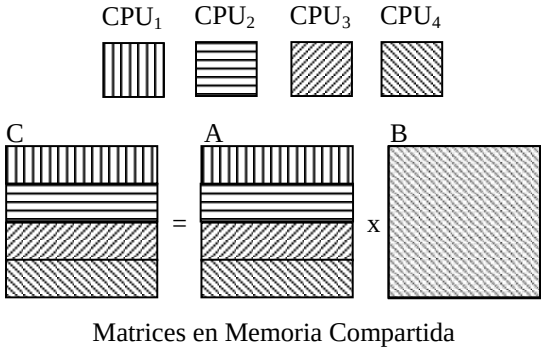


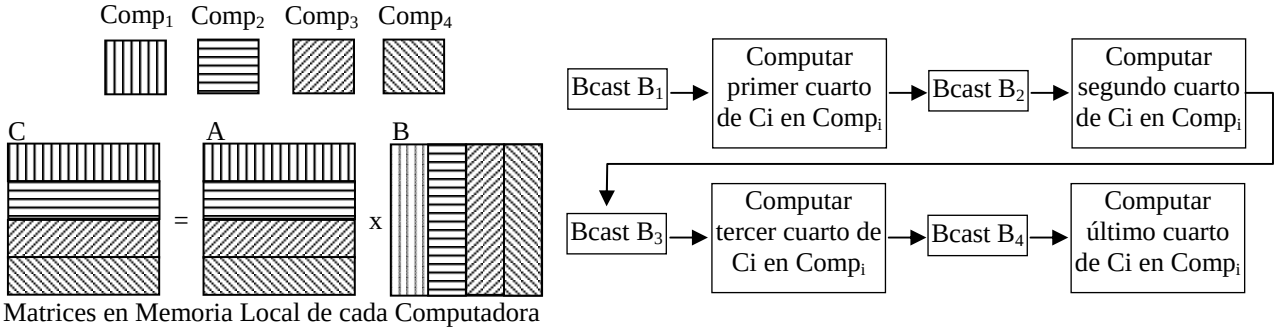
Figura 2: Multiplicación de Matrices en Multiprocesadores.

Si bien la forma de cómputo paralelo dada en la Fig. 2 no se estima a priori como la óptima, sólo podrá corroborarse por medio de la experimentación y analizar objetivamente con los resultados reales sobre al menos un nodo con múltiples núcleos real.

3.3 Cómputo Paralelo Optimizado en un Cluster

Dado que la idea es considerar por separado el cluster de la característica de multiprocesamiento de cada uno de sus nodos, se puede utilizar, al menos en principio, un algoritmo específicamente diseñado para clusters. En este punto se podría tomar cualquiera de los existentes, pero se elige el basado en mensajes broadcasts presentado en [18] por su sencillez. En este punto de deja abierta la posibilidad de implementar otro algoritmo si no se tienen buenos resultados de rendimiento en el momento de la experimentación sobre un cluster real. La Fig. 3 muestra de manera esquemática el

algoritmo de multiplicación de matrices para cuatro computadoras, cada una con su propia memoria local. Más específicamente, la Fig. 3.a) muestra la distribución de las matrices y la Fig. 3.b) muestra la secuencia de pasos a realizar para completar el cómputo. El primer bloque de columnas de la matriz B, que está asignado a $Comp_1$, se denomina B_1, el segundo bloque, que está asignado a $Comp_2$, se denomina B_2 y así siguiendo hasta el bloque de columnas B_4 asignado a $Comp_4$. De manera similar se denotan los bloques de filas de A y C, como A_1, …, A_4, y C_1, …, C_4 respectivamente. Las comunicaciones son necesarias específicamente por los datos de la matriz B, que se necesitan en todas las computadoras y es por eso que se elige la comunicación broadcast. Cada una de las operaciones A_i x B_j da por resultado una porción de C_i, específicamente una cuarta parte en la Fig. 3 de la porción de C_i que debe calcular la $Comp_i$.



a) Distribución de Datos en cada $Comp_i$ b) Comunicaciones y Cómputo en cada $Comp_i$

Figura 3: Multiplicación de Matrices en Clusters.

En este punto se tienen los lineamientos generales del cómputo paralelo tanto en cada nodo con múltiples núcleos como en el cluster en general. Debe notarse que los algoritmos se definieron por separado justamente porque el objetivo se orienta a independizar cada optimización del resto, reduciendo la complejidad total de la paralelización.

4 Experimentación y Análisis de Resultados

En la experimentación, se utilizó un cluster de nodos con múltiples núcleos que puede ser considerado como estándar: cada computadora tiene dos procesadores y cada procesador tiene dos núcleos (*dual core processors*). Más específicamente, se tienen cuatro computadoras ~~de escritorio~~, cada una de ellas con dos procesadores AMD Opteron 2200, y cada uno de los procesadores contiene dos núcleos integrados. La Tabla 1 resume las características de cada una de las computadoras del cluster utilizado en la experimentación.

Procesadores	Proc. x comp.	Núcleos por proc.	RAM	Frec. reloj	Sist. Operativo
AMD Opteron 2200	2	2	4 GB	2.2 GHz	Sun Solaris 10

Tabla 1: Características de los Nodos del Cluster.

La red de interconexión es de relativamente bajo costo: Ethernet de 1 Gb/s con un único *switch* que interconecta las cuatro computadoras. El tamaño de las matrices a multiplicar se eligió de forma tal que requiere un tiempo significativo de cómputo y a la vez no genera inconvenientes de falta de memoria como para utilizar el espacio de memoria *swap* del sistema operativo. Específicamente, los experimentos se llevaron a cabo con matrices cuadradas de 13000 x 13000 elementos (punto

flotante en precisión simple). El entorno de desarrollo es relativamente estándar: una de las implementaciones de MPI, en este caso la provista por Sun, HPC Cluster tools [17], que está basada casi directamente en la implementación de código abierto Open MPI [8]. El compilador de Fortran (lenguaje estándar en procesamiento numérico) también es provisto por Sun, en este caso específico Sun Studio 12: Fortran 95 Compiler.

4.1 Código Optimizado en un Núcleo: Procesamiento *Secuencial*

Como se aclara antes, se aprovechará la optimización de procesamiento secuencial provista por bibliotecas como ACML, MKL y ATLAS. En el caso específico de las computadoras utilizadas para la experimentación, se utilizaron las alternativas:

- ACML, dado que los procesadores son AMD.
- ATLAS, por ser una biblioteca de código abierto, independiente del procesador utilizado.
- La propia función intrínseca provista por Fortran 95: `matmul()`.

La Tabla 2 muestra los resultados de rendimiento obtenidos para la multiplicación secuencial (en un único núcleo de matrices). Claramente, la biblioteca ACML provee los mejores resultados de rendimiento, seguida por la función intrínseca implementada por el compilador de Fortran de Sun. Quizás sea demasiado bajo el rendimiento obtenido por la biblioteca ATLAS, pero de todas maneras se la utilizará como referencia de una biblioteca de biblioteca numérica de código abierto.

Mult. de Matrices	ACML (Mflop/s)	ATLAS (Mflop/s)	Intrínseca (Mflop/s)
13000 x 1300	8139	2298	7586

Tabla 2: Multiplicación de Matrices: Rendimiento de Procesamiento Secuencial.

Más allá de las posibles diferencias de rendimiento entre las bibliotecas, está claro que aún utilizando ATLAS, el uso de estas bibliotecas optimiza múltiples factores implementados en los procesadores (cómputo superescalar, jerarquía de memoria con varios niveles de memoria cache, etc.). Si se implementa la multiplicación de matrices con las tres iteraciones anidadas *clásicas*, el rendimiento obtenido es al menos un orden de magnitud peor, con lo cual las tres bibliotecas proveen una mejora más que aceptable.

4.2 Código Paralelo en un Multiprocesador: Procesamiento *Paralelo en Memoria Compartida*

Como se aclara antes, se realizaron experimentos con las versiones paralelas de memoria compartida de las bibliotecas ACML y ATLAS, y también se implementó un algoritmo paralelo de multiplicación de matrices para multiprocesadores. Es interesante notar las características en cuanto a complejidad de cada implementación:

- ACML: solamente se cambia la forma en que se lleva a cabo la etapa de link, dado que se usa la versión de la biblioteca para multiprocesadores. En cada ejecución, se indica por un parámetro la cantidad de núcleos que se utilizarán de manera efectiva para computar.
- ATLAS: solamente se cambia la forma en que se lleva a cabo la etapa de link, dado que se usa la versión de la biblioteca para multiprocesadores. Inicialmente, la biblioteca utiliza todos los núcleos disponibles, no hay forma de indicar que utilice solamente algunos de ellos a menos que se reinstale la biblioteca. En este caso en particular, se utilizarán los cuatro núcleos disponibles.
- Algoritmo paralelo: en este caso, la división del procesamiento se realiza con el programa detallado antes. El costo de esta implementación puede considerarse muy bajo, dado que, de hecho, no llevó más de 50 líneas de código Fortran 90/95 haciendo uso de las directivas OpenMP, que justamente simplifican la tarea de paralelización en un multiprocesador.

La Fig. 4 muestra los resultados de rendimiento obtenidos en términos del factor de Speed Up *Relativo* respecto de cada alternativa secuencial con todas las alternativas paralelas para un multiprocesador. Es decir que cada alternativa paralela se compara con su propia versión secuencial, que no es necesariamente la de mejor rendimiento secuencial. La idea de esta definición es la de analizar si la paralelización escala correctamente en función de la cantidad de procesadores. Las dos primeras alternativas son denominadas "implícitas" en el gráfico, dado que no hay que programarlas para que sean paralelas, , tan solo cambia el proceso de enlazado(link). Tanto ACML como ATLAS tienen la posibilidad de utilizar en paralelo los núcleos disponibles en el caso de ACML se puede indicar cuántos y es por esto que hay resultados para 1, 2, 3 y cuatro núcleos. En el caso de ATLAS se tienen los resultados para 1 y 4 núcleos. Las versiones denominadas "explícitas" corresponden al algoritmo paralelo para multiprocesadores detallado antes, que utilizan la multiplicación de matrices secuencial provista por ACML, ATLAS y el compilador (función intrínseca) respectivamente.

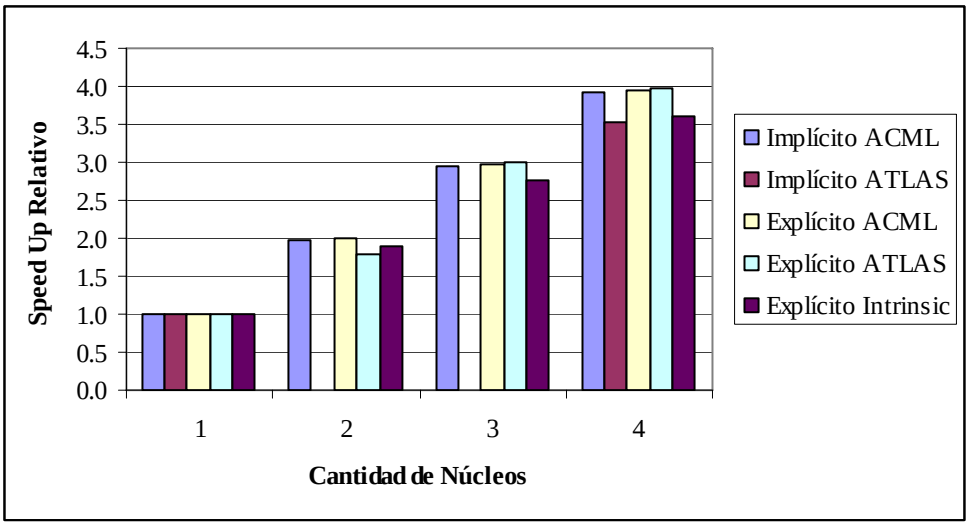


Figura 4: Rendimiento de la Multiplicación de Matrices en un Multiprocesador.

Es interesante comparar, por ejemplo, los resultados de ACML "implícita" con la versión programada de manera explícita para cómputo paralelo en multiprocesador, es decir "Implícito ACML" con "Explícito ACML". Los resultados son casi idénticos, con una leve ventaja de la alternativa programada explícitamente paralela para multiprocesador. Esta comparación provee dos conclusiones destacables:

- Es muy posible que la propia biblioteca ACML implemente un algoritmo paralelo como el que se implementó en este trabajo o, como mínimo, el algoritmo que implementa no es mejor que el que se utiliza en este trabajo.
- La paralelización explícita para multiprocesador provee excelentes resultados, aún levemente mejores que los provistos por la biblioteca creada por la misma empresa que fabrica los procesadores con múltiples núcleos.

En cualquier caso, los resultados obtenidos con el algoritmo paralelo para multiprocesadores son altamente satisfactorios, por cuanto se utilizan al máximo (en más de un 90%) todos los núcleos disponibles.

4.3 Código Paralelo en el Cluster: Procesamiento *Paralelo en Memoria Distribuida*

En este caso, se utiliza el algoritmo paralelo que se describe antes y no hay alternativa implícita o

provista por ninguna de las bibliotecas mencionadas, dado que son básicamente para procesamiento local en cada nodo, sea estrictamente secuencial cuando existe un procesador con un único núcleo o un multiprocesador con varios núcleos cada uno, como en el caso del cluster utilizado. La idea en este caso, es utilizar en cada nodo las versiones disponibles de cómputo paralelo para multiprocesador y la distribución de datos y cómputo en el cluster de acuerdo al algoritmo que se detalló antes para clusters, basado en mensajes broadcast. Más específicamente, en el caso del cómputo local en paralelo (en los múltiples nodos) se utilizaron dos alternativas: la proporcionada por la biblioteca ACML en su versión para multiprocesadores (que se podría llamar *implícitamente paralela*) y la obtenida con el algoritmo paralelo para multiprocesadores, utilizando en cada núcleo la función de multiplicación de matrices secuencial de la misma biblioteca ACML. La Fig. 5 muestra los resultados de rendimiento obtenidos en el cluster. Nótese que las dos series de resultados ("Implícito ACML" y "Explícito ACML" en la Fig. 5) son opciones diferentes de cómputo en cada nodo, el algoritmo paralelo en el cluster es único.

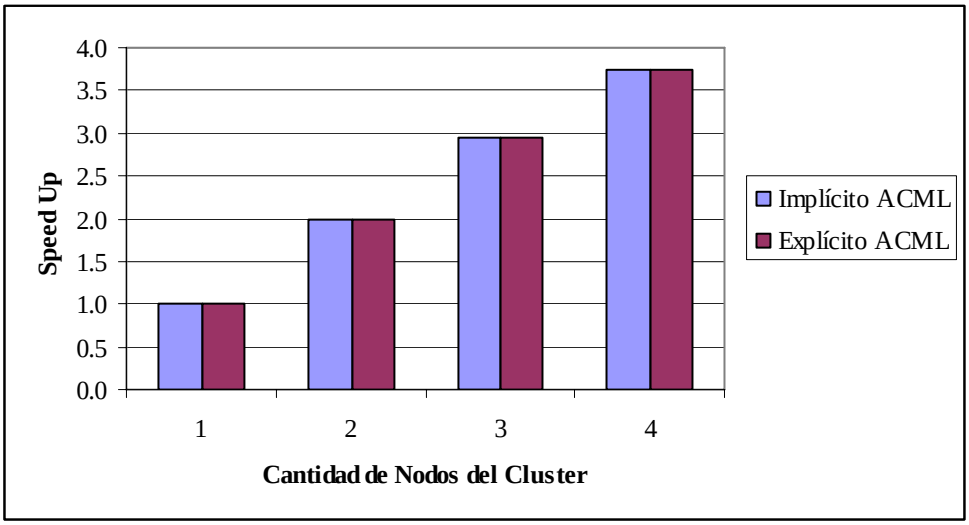


Figura 5: Rendimiento de la Multiplicación de Matrices en el Cluster.

A partir de los resultados que se muestran en la Fig. 5 queda claro que el rendimiento de la multiplicación de matrices es el mismo, sea que en cada nodo se utilice una biblioteca optimizada para múltiples nodos (en este caso, ACML) o se implemente un algoritmo sencillo pero muy eficiente para multiprocesadores (al menos para la cantidad de núcleos que hay disponibles en multiprocesador que se utilizó en los experimentos). A partir de los datos que se muestran en la Fig. 4, es claro que si hay un problema de rendimiento en el cluster es por el algoritmo paralelo de memoria distribuida implementado en el cluster, no en cada nodo con múltiples núcleos.

5 Conclusiones y Trabajos Futuros

En este artículo se presenta la idea de separar o atacar por separado el problema de optimización de rendimiento paralelo en un cluster de nodos con múltiples núcleos. Dado que el rendimiento ha sido tradicionalmente aceptado como la métrica en función de la cual analizar las propuestas de algoritmos paralelos, se utiliza un problema estándar y conocido con el cual evaluar el rendimiento obtenido por experimentación: la multiplicación de matrices. Se ha mostrado para este caso, que en términos de optimización secuencial (en un único núcleo), el problema está resuelto por diferentes bibliotecas como ACML, MKL y ATLAS. Es de destacar que estas bibliotecas y otras similares no

solamente resuelven satisfactoriamente la optimización para la multiplicación de matrices sino también al menos para todas las operaciones involucradas con las aplicaciones de álgebra lineal.

El rendimiento paralelo en multiprocesadores (es decir en computadoras que tienen múltiples núcleos y/o múltiples CPUs que comparten memoria) es optimizado tanto en el caso de las bibliotecas como ACML, que directamente proveen una versión de las funciones de cómputo para multiprocesadores como en el caso de la paralelización explícita. Aunque, por supuesto, la paralelización explícita para multiprocesadores es más costosa que directamente utilizar una función de biblioteca, se demuestra que algoritmos paralelos sencillos e intuitivos proveen al menos tan buen resultado como el provisto por bibliotecas optimizadas como ACML. El costo de la paralelización es mínimo para la multiplicación de matrices (aproximadamente 50 líneas de código Fortran 90/95 con directivas OpenMP). Si bien no se puede asegurar esto en general, es decir para cualquier operación matricial, la sencillez con la que se logra en el caso de la multiplicación de matrices puede considerarse como una indicación a favor de la posibilidad de paralelizar de manera muy sencilla en el entorno de un multiprocesador.

El rendimiento paralelo en un cluster de nodos con múltiples núcleos puede ser optimizado sin la necesitad de recurrir explícitamente a algoritmos paralelos especialmente diseñados para tener en cuenta los dos *niveles* o las dos clases de procesamiento paralelo: en memoria compartida dentro de cada nodo y en memoria distribuida en el cluster. De hecho, el algoritmo paralelo utilizado en los experimentos es directamente un algoritmo paralelo propuesto para clusters y no se le hizo ninguna modificación específica o *ad hoc* para el cluster de nodos con múltiples núcleos. Este argumento es válido siempre que se disponga de bibliotecas optimizadas para el cómputo en memoria compartida.

Como suele suceder en estos entornos, siempre es útil y en muchos casos también es necesaria la experimentación con mayor cantidad de nodos. Aunque no parece probable a priori en base a los resultados obtenidos, siempre es mejor analizar el rendimiento con mayor cantidad de nodos del cluster como para poder enfocar factores como la escalabilidad del algoritmo. Aunque queda pendiente para el caso en que se disponga de clusters con mayor cantidad de nodos, es posible que los problemas de escalabilidad se den por el algoritmo paralelo para memoria distribuida y no por la combinación de paralelismo en memoria compartida y memoria distribuida.

También queda pendiente (y quizás sea lo primero a analizar en el futuro) el análisis de resultados para otras operaciones y/o métodos de álgebra lineal diferentes de la multiplicación de matrices. Como se aclara antes, la facilidad de la paralelización de la multiplicación de matrices puede ser tomada como un buen indicio, pero no hay demostración posible al menos hasta ahora que esta facilidad se tendrá para las demás operaciones y métodos de álgebra lineal. Aún se sigue utilizando el método de “caso por caso” y debería ser utilizado también en este caso particular.

REFERENCIAS

[1] AMD Corp., ACML - AMD Core Math Library User Guide, http://developer.amd.com/cpu/Libraries/acml/downloads/Pages/default.aspx#docs

[2] M. Baker, R. Buyya, “Cluster Computing at a Glance”, in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 3-47, 1999.

[3] L. Blackford, J. Choi, A. Cleary, E. D' Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.

[4] D. P. Bovet, M. Cesati, Understanding the Linux Kernel, O'Reilly Media, Inc., 3rd Ed., 2005, ISBN 0-596-00565-2.

[5] D. Culler, J. P. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann, 1998, ISBN 1558603433.

[6] M. Flynn, “Some Computer Organizations and Their Affectiveness”, IEEE Trans. on Computers, 21 (9), 1972.

[7] D. Geer, “Chip Makers Turn to Multicore Processors”, Computer, May 2005, IEEE Computer Society, pp. 11-13.

[8] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, A. Lumsdaine, “Open MPI: A High-Performance, Heterogeneous MPI”, Proc. Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, Barcelona, Spain, 2006.

[9] A. Grama, G. Karypis, V. Kumar, A. Gupta, Introduction to Parallel Computing, 2nd. Ed. Addison Wesley, 2003, ISBN 0201648652.

[10] P. Gepner, M. F. Kowalik, “Multi-Core Processors: New Way to Achieve High System Performance”, Proc. International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, 13–17 September 2006, ISBN 0-7695-2554-7.

[11] J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, 4th Edition, Morgan Kaufmann, 2006, ISBN 0123704901.

[12] Intel Corp., MKL Math Kernel Library, http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm

[13] M. Krishnan, J. Nieplocha, “SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable Shared Memory Systems”, ipdps, p. 70b, 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Papers, IEEE Computer Siciety, 2004.

[14] MPI Forum, “MPI: a message-passing interface standard”, International Journal of Supercomputer Applications, 8 (3/4), pp. 165-416, 1994.

[15] C. Schimmel, UNIX(R) Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers, Addison-Wesley Professional, 1994, ISBN 0201633388.

[16] M. Schmollinger, M. Kaufmann, “Algorithms for SMP-Clusters Dense Matrix-Vector Multiplication”, Proc. of the International Parallel and Distributed Processing Symposium (IPDPS’02), IEEE Computer Society.

[17] Sun Microsystems, Inc., Sun HPC ClusterTools 7.1 - Overview, http://www.sun.com/software/products/clustertools/

[18] F. G. Tinetti, Cómputo Paralelo en Redes Locales de Computadoras, Tesis Doctoral (Doctorado en Informática), Universidad Autónoma de Barcelona, Marzo de 2004. Disponible en http://ftinetti.googlepages.com/tesisdoctoral

[19] R. C. Whaley, A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS", Software: Practice and Experience, Vol. 35, No. 2, pp. 101-121, Feb. 2005. http://math-atlas.sourceforge.net/

[20] B. Wilkinson, M. Allen, Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 2nd Ed., Prentice Hall, 2004, ISBN 0131405632.

[21] M.-S. Wu, S. Aluru, R. A. Kendall, "Mixed Mode Matrix Multiplication", Proc. of the IEEE International Conference on Cluster Computing (CLUSTER'02), IEEE Computer Society's Task Force on Cluster Computing (TFCC), Chicago, USA.

[22] ScaLAPACK Home Page, http://www.netlib.org/scalapack/scalapack_home.html

[23] TOP500 supercomputing Sites, http://www.top500.org/