EVALUACIÓN DE UNA RED DE CÁLCULO DISTRIBUIDO USANDO PVM

Andrea M. Yommi y Victorio E. Sonzogni
Centro Internacional de Métodos Computacionales en Ingeniería (CIMEC)
INTEC-(CONICET-UNL)
Güemes 3450, 3000 Santa Fe, Argentina
e-mail: ayommi@intec.unl.edu.ar sonzogni@intec.unl.edu.ar

RESUMEN

Se analizó una red de computadoras para evaluar sus posibilidades de cálculo y de comunicación. Para ello, se realizaron pruebas que permitieron conocer los tiempos de inicio de los mensajes y la velocidad de transmisión de los mismos. También se efectuaron pruebas con programas que realizaban operaciones algebraicas, en particular el producto matricial, tratando de optimizar los tiempos de ejecución tanto en modo secuencial como en paralelo. En ambos casos se midieron los tiempos de ejecución (tiempo transcurrido y tiempo `CPU`) de cálculo y comunicación.

ABSTRACT

A net of computers was analyzed to evaluate its possibilities of calculation and of communication. Some tests allowing to measure the startup time of the messages and their speed of transmission were carried out. Tests were also made with programs that carried out algebraic operations, in particular the matrix product, trying to optimize the times of execution so much in sequential as in parallel ways. In both cases the times of execution (elapsed and `CPU` time) of calculation and communication were measured.

INTRODUCCION

El grado de avance de las arquitecturas de computadoras ha sido muy significativo en las últimas décadas, y el desarrollo tecnológico ha permitido contar con procesadores cada vez más veloces. No obstante, existe un factor que limita el incremento de la velocidad de procesamiento: la velocidad de la luz. Por este motivo, aumentar la potencia de cálculo, ha orientado el desarrollo computacional hacia el paralelismo.
El procesamiento paralelo puede llevarse a cabo mediante la disposición de varios procesadores en paralelo, ya sea en computadoras paralelas como en redes de computadoras. El uso simultáneo de procesadores requiere de un software adecuado el cual dependerá de la arquitectura del hardware y del modelo de programación.
En el `CIMEC, INTEC` se dispone de una pequeña red - no homogénea - que consta de cuatro estaciones de trabajo `ALPHA`, interconectadas con cable Ethernet y sobre la que se utiliza un software específico de comunicación: `PVM` (Parallel Virtural Machine).
Se tiene particular interés en conocer las características de esta red para poder aprovechar las potencialidades de la misma en la resolución de problemas específicos de elementos finitos. Para ello, se han considerado tanto los aspectos de cálculo como de comunicación. Con el propósito de clasificar la red utilizada, se dan a continuación nociones básicas acerca del cálculo paralelo: tipos de computadoras, clasificación, modelos de programación paralela, niveles de paralelismo y lenguajes de programación. Luego se mencionan algunas características del software `PVM`. A continuación se discute un modelo de comunicación entre dos procesadores, se determina el tiempo de inicio (startup) y la velocidad de transmisión entre estos. Luego se mencionan dos

métodos de resolución en paralelo del producto matricial. Se analiza la implementación de los mismos en su versión secuencial y en paralelo. Finalmente, se muestran las medidas de eficiencia y los tiempos obtenidos de cálculo y de comunicación.

CÁLCULO PARALELO

El procesamiento paralelo consiste en la ejecución simultánea de un determinado número de tareas y éste puede llevarse a cabo de diferentes maneras.[1] En el caso de computadoras secuenciales, es decir con un sólo procesador, se pueden efectuar operaciones simultáneas mediante: múltiples unidades funcionales, "pipelining", instrucciones vectoriales y encadenamiento de operaciones, entre otras. Una parte importante del diseño de computadoras está en el flujo de datos desde la memoria hacia las unidades funcionales. Las unidades funcionales se comunican con los registros, los cuales pueden enviar datos a la `CPU` en 1 ciclo reloj. Estos se comunican directamente con la memoria cache, que representa un buffer de pequeña capacidad pero de alta velocidad entre el procesador y la memoria `RAM`. Si estos se utilizan eficientemente, es posible reducir el tiempo de acceso a memoria.

Arquitectura de Computadoras Paralelas

Las computadoras paralelas son aquellas que constan de varias unidades de procesamiento trabajando simultáneamente. Se pueden clasificar según diferentes criterios:[3] *Granularidad del hardware*, *Conexión entre los procesadores*, *Organización de la memoria* y *Control de la ejecución*, entre otros. Se puede pensar que el conjunto formado por las 4 computadoras interconectadas entre sí constituye una arquitectura de computadora de memoria distribuida con pocos procesadores, donde el control de la ejecución el del tipo `MIMD`.

Niveles de paralelismo

La idea de paralelismo puede intentarse de diversas formas. A nivel de instrucción, llevado a cabo por el sistema operativo; a nivel de programa, realizado por el software; a nivel inter-instrucción, realizado por el compilador, y a nivel intra-instrucción, llevado a cabo por el hardware. A su vez, en el *paralelismo a nivel de programa*, si las tareas a paralelizar consisten de ciclos `DO`, se habla de un paralelismo de granularidad fina o microtasking, mientras que si lo que ha de paralelizarse son secciones enteras del programa, se trata de un paralelismo de granularidad gruesa o macrotasking. La relación entre el número de tareas a realizar en cada procesador y la comunicación entre ellos se conoce como "*Granularidad de tareas paralelas*". En computadoras de memoria local es la relación entre la cantidad de operaciones y la cantidad de datos a transmitir en cada tarea.

Modelos de programación paralela

Programar de manera eficiente va a depender del tipo de arquitectura que se tiene. Frente a las arquitecturas ya mencionadas pueden establecerse al menos dos modelos de programación.[3] Por un lado está el *modelo de memoria compartida*, donde el programa es ejecutado por un procesador y cuando una tarea requiere cálculo paralelo, se convoca a los otros procesadores para ejecutarla. Este modelo, además de necesitar una zona de memoria compartida, exige la sincronización entre los procesadores. Los programas pueden realizarse en `Fortran` o `C`, aunque estas máquinas suelen tener extensiones del lenguaje original. Por otro lado está el *modelo de intercambio de mensajes* que es utilizado por computadoras de memoria local. Cada procesador comienza a ejecutar un programa y en determinadas circunstancias se hace necesario el intercambio de información lo cual se lleva a cabo mediante el envío de mensajes de un procesador a otro. Las rutinas básicas de comunicación y sincronización pueden ser propias de cada máquina, pero existe software portable tal como `PVM`,[4] `MPI`, `EXPRESS`, `LINDA`.

Descripción de la red

Cada una de las máquinas que forma la red utilizada consta de un procesador escalar y una memoria local.

La conexión entre los procesadores se realiza mediante un canal de transmisión llamado *Ethernet*, que tiene una velocidad teórica de 10 $Mbit/s$. En esta red, el paralelismo se da a nivel programa donde el modelo utilizado es el de intercambio de mensajes, y el software empleado para la comunicación es `PVM`. La tabla (1) muestra las características de cada computadora de la red.

Tabla 1: Características de la Red

Nombre	Procesador	Mhz	`RAM`
Geminis (G)	Alphastation 255	233	96 Mb
Tauro (T)	Alphastation 255	300	128 Mb
Virgo (V)	Alphastation 500	333	320 Mb
Cancer (C)	Alphastation 500	333	384 Mb

PVM: PARALLEL VIRTUAL MACHINE

`PVM` es un sistema de software que permite que un conjunto heterogéneo de computadoras interconectadas sea visto como una única máquina paralela o también llamada *máquina virtual*. La heterogeneidad de la que se habla puede deberse al tipo de arquitectura de las computadoras, los formatos de datos con que trabajan, o en caso de que todas tengan igual formato de datos, puede haber diferencias en las velocidades de las mismas. Aún cuando la máquina virtual esté formada por computadoras con idénticas características, debe tenerse en cuenta que puede haber otros usuarios haciendo uso de la red, lo cual puede afectar considerablemente el tiempo que toma enviar o recibir un mensaje en la misma, hecho que incide en la pérdida de eficiencia del programa.
La máquina virtual se puede configurar dinámicamente, agregando o eliminando tareas durante la ejecución. La *tarea* es la unidad de paralelismo en `PVM`, y consiste de una secuencia independiente de instrucciones de control que alternan entre cálculos y comunicación. Se pueden asignar múltiples tareas a un procesador, y el modelo de transferencia de mensajes consiste en que las tareas se envían y reciben mensajes entre sí. El tamaño del mensaje está limitado por la cantidad de memoria disponible.
El sistema `PVM` consta de dos partes, un `PVM` *daemon* que reside en todas las computadoras formando la máquina virtual, y una *Biblioteca* `PVM`, que contiene rutinas de interfase que permiten enviar y recibir mensajes, crear procesos, coordinar tareas y modificar esta máquina virtual, desde el programa de aplicación.[4]

Esquemas de Programación y Balance de carga

Un esquema de programación distribuida puede ser *Funcional*, que consiste de varios programas y cada uno ejecuta una función diferente en la aplicación; *Master-Slave,* donde la tarea "master" inicia todas las tareas "slaves" y coordina el trabajo, las entradas y salidas; o `SPMD` *Single-Program - Multiple Data*, donde todas las tareas son las mismas pero cada una sólo conoce y resuelve una parte de los datos. En este último caso, la descomposición de los datos puede hacerse en forma *estática*, dividiendo el problema en tareas y asignándolas a los procesadores una única vez, o *Dinámica*, ya sea mediante una bolsa de trabajo (esquemas Master-Slave) o por coordinación (esquemas `SPMD`).

TRANSMISIÓN DE MENSAJES

Uno de los objetivos perseguidos en este trabajo, fue obtener criterios que permitan en un futuro la programación eficiente de algoritmos paralelos. Esto va a depender del grado de paralelización del problema, de la tecnología del compilador, del software de transferencia de mensajes, de la cantidad de memoria, de la velocidad del procesador y de la red que los conecta. Se han comentado algunos de estos items, aunque resulta de particular interés analizar el modelo de comunicación `PVM` en la red utilizada.
El rendimiento correspondiente a la transferencia de mensajes se mide generalmente en *bandwidth* (Mb/s). Se podría decir que el *bandwidth* representa una medida de la velocidad de transferencia de información, usada

para cuantificar las capacidades de comunicación de sistemas de procesadores. Esta unidad de medición se ha considerado para mensajes de gran tamaño. Para mensajes muy pequeños, se ha tomado al tiempo (μs) como medida de rendimiento. El tiempo de envío de un mensaje muy pequeño o de longitud cero, está acotado por la velocidad de una señal a través de la red (*latencia*). Se puede decir que la *latencia* es el mínimo tiempo necesario para enviar un mensaje, también se lo conoce como tiempo de "*startup*".
El tiempo de transferencia de un mensaje entre dos procesadores se define como una función lineal del tamaño del mensaje:

$$t_n = \alpha + \beta\ n \tag{1}$$

donde α es el tiempo de *startup*, β es el tiempo para transmitir un byte y n es el número de bytes del mensaje. Otra medida que suele ser de interés, es $n_{1/2}$, que se define como aquel valor de n al que le corresponde una velocidad igual a la mitad de la velocidad teórica. Se lo suele definir como $n_{1/2} = \alpha/\beta$.

Método de medición

Para obtener las mediciones que conducen a la obtención de los parámetros de la función lineal dada, se utilizó un programa entre dos nodos, del tipo Master-Slave. En este programa, el proceso master se enrola en `PVM` e inicia el otro proceso. Realiza una serie de tests antes de tomar tiempos. Define un mensaje y con éste realiza 20 iteraciones. En cada una, inicializa el tiempo, inicializa el "send buffer", empaqueta el mensaje y lo envía al proceso slave. Este último, recibe el mensaje, libera el buffer y le envía al master un mensaje vacío. Luego espera una señal que indique que el mensaje ha llegado y toma nuevamente el tiempo. Finalmente se toman los tiempos promedios. El modelo de comunicación utilizado es *send* - *recv* bloqueante. Los mensajes enviados son vectores de enteros con tamaños que varían entre 0 y 1 Mb. La rutina usada para medir el tiempo real de envío de un mensaje es una función de la biblioteca `C`, llamada *gettimeofday*. Para estimar la latencia se utilizaron mensajes cuyos tamaños varían desde 0 a 1100 bytes. La medición de estos tiempos puede verse afectada debido al redondeo realizado por las rutinas de tiempo.

Latencia y Bandwidth

El programa fue implementado 5 veces para las 6 combinaciones posibles de las 4 máquinas que constituyen la red. Para cada par de computadoras se estima un valor de α (latencia) y de la velocidad $\theta = 1/\beta$. El valor de α se toma como el tiempo promedio de 10 corridas que tarda en enviarse un mensaje de longitud 0. Debido a que la velocidad teórica de una red Ethernet es $v = 10\ Mbit/s$, se mide θ en las mismas unidades. La tabla (2) muestra las mediciones de la latencia, el bandwidth, la velocidad y $n_{1/2}$ para cada par de computadoras. Tomando los promedios de los valores de α y β obtenidos para cada par de computadoras,

Tabla 2: Propiedades de la Red

Computadoras	α (n=0)	$\theta = 1/\beta$ (n=10^6)	β $(s/Mbit)$	β $(\mu s/bytes)$	$n_{1/2}$	θ_{max}	n_{max}
G - V	801	6.99	0.1430	1.19	1110	8.38	9040
V - T	1059	6.56	0.1523	1.27	3040	6.95	11040
G - T	1156	6.57	0.1522	1.27	3816	6.66	11040
C - T	1035	6.83	0.1463	1.22	2609	7.34	9040
G - C	780	7.07	0.1414	1.18	1040	8.17	9040
C - V	625	7.24	0.1381	1.15	700	8.70	9040

se puede estimar que el tiempo de envío de un mensaje viene dado por

$$t_n = 0.000909 + 0.146\ n \tag{2}$$

donde t_n está medido en segundos y n en $Mbit$.
El bandwidth promedio resulta ser $\theta = 6.88\ Mbit/s$ para un mensaje de 1 Mb. Este valor, está por debajo del valor teórico que puede alcanzarse con una red Ethernet, circunstancia que también se ha reportado en la literatura.[5]

Se observa que la máxima velocidad es alcanzada con la transferencia de un mensaje de aproximadamente 10 Kb, alcanzando un promedio de 7.7 $Mbit/s$. Se puede observar también que para las combinaciones que incluyen a la computadora Tauro, el valor de α es superior que para las combinaciones que no la incluyen. El promedio de las que incluyen a Tauro, da para α el valor 1083 μs, mientras que las demás combinaciones registran en promedio, un valor de α igual a 735.3 μs, digamos un 30 % inferior. Esto en parte podría deberse a la distancia que debe recorrer el mensaje por la red para llegar desde Cancer, Virgo o Geminis a Tauro, dado que esta última se encuentra físicamente bastante más alejada del resto. Esta diferencia no es significativa para tamaños grandes. En la figura (1) se muestran las curvas correspondientes al bandwidth, medido en $Mbit/s$ y en la figura (2) se muestran los gráficos correspondientes al tiempo (en microsegundos) necesario para enviar mensajes de pequeña longitud.

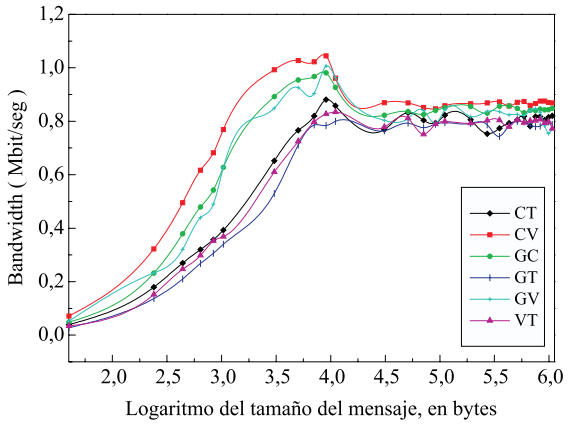

Figura 1. Bandwidth

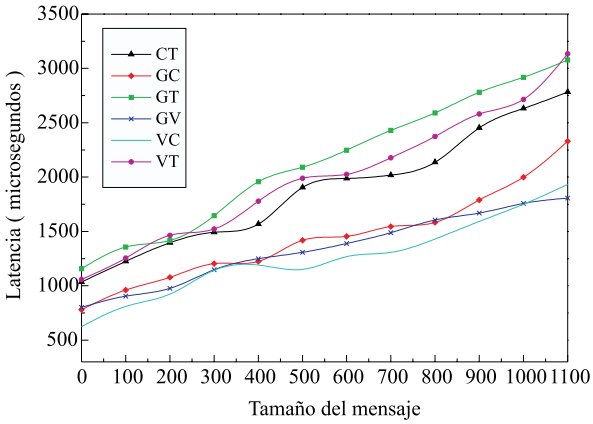

Figura 2. Latencia

PRODUCTO MATRICIAL

Los cálculos matriciales se construyen bajo determinados niveles de operaciones algebraicas lineales. El producto escalar involucra multiplicar y sumar escalares. El producto matriz-vector se construye usando productos internos y el producto matriz-matriz es una colección de productos matriz-vector. En esta sección se considera el problema de multiplicar dos matrices cuadradas de orden $n \times n$. Básicamente, si $\mathbf{A} = (a_{i,j})$ y $\mathbf{B} = (b_{i,j})$, sea $\mathbf{C} = \mathbf{AB}$, $\mathbf{C} = (c_{i,j})$, donde $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$ para $i, j = 1, \ldots, n$. La programación se realizó en lenguaje `C++`, mediante alocación dinámica de las matrices. En `C` y `C++`, las matrices se almacenan en memoria por filas; por lo tanto, se puede pensar en $\mathbf{A}$ y $\mathbf{B}$ como vectores: $\mathbf{A} = (a_i)$, $\mathbf{B} = (b_i)$, $i = 0, \ldots, n^2 - 1$ formados por n filas una al lado de la otra. Bajo esta sintaxis, el producto $\mathbf{AB}$ viene dado por el Algoritmo [1].

```
For (i = 0; i < n; i++)
{
   p = i * n;
  For (j = 0; j < n; j++)
    For (k = 0; k < n; k++)
      c[p + j] += a[p + k] * b[k * n + j];
}
```

Algoritmo [1]. Producto usual

```
s = 10;      nr = n % s;      ns = n - nr;
For (i = 0; i < n; i++)
  For (j = 0; j < n; j++)
    For (k = 0; k <= ns - s; k+= s)
      c[i * n + j] += a[i * n + k] * b[k * n + j] + ··· +
                    + a[i * n + k + 9] * b[(k + 9) * n + j];
    For (k = ns; k < n; k++)
      c[i * n + j] += a[i * n + k] * b[k * n + j];
```

Algoritmo [2]. Producto usando desenrollado

Se observa que $\mathbf{c}[p + j]$ se actualiza para cada valor de k. Una manera de reutilizar estos valores y reducir el tráfico de datos a la memoria, incrementando la razón del número de operaciones en relación al número de

movimientos es utilizar la técnica del "*desenrollado(s)*", donde s indica el número de contribuciones en cada paso para la matriz $\mathbf{C}$.[1] En el caso del producto matricial, una serie de pruebas condujo a tomar $s = 10$. El código dado anteriormente viene dado por el Algoritmo [2].

RESULTADOS DE LA IMPLEMENTACIÓN SECUENCIAL

Ambos esquemas se corrieron secuencialmente en las 4 máquinas para matrices de orden $n \times n$, con n variando desde 25 hasta 1000. En todos los casos se midieron tanto el tiempo real transcurrido como el tiempo `CPU`. Este último se obtiene usando la función *clock()* del sistema operativo `UNIX`, la cual devuelve el tiempo (en microsegundos) empleado por el procesador.
Dado n, cada valor de tiempo obtenido (real y `CPU`) es el resultado de tomar el promedio de tiempos resultantes de multiplicar 20 veces las matrices $\mathbf{A}$ y $\mathbf{B}$ las cuales fueron generadas en forma aleatoria. Las figuras (3) y (4) muestran los tiempos transcurridos (en segundos) registrados con cada computadora en función del tamaño de la matriz para ambos algoritmos. El tiempo transcurrido coincide con el tiempo `CPU` en Cancer, Virgo y Tauro, indicando una utilización eficiente del procesador y la memoria cache. No ocurre lo mismo en Geminis. El tiempo real para los 2 algoritmos se ve incrementado en un 122 % respecto al tiempo `CPU`, lo que muestra un defecto en la reutilización de datos en memoria. Efectivamente, Geminis posee recursos más limitados que las otras computadoras.

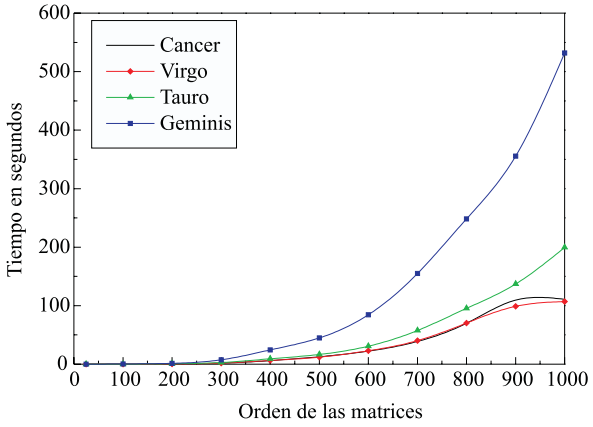

Figura 3. Algoritmo [1]

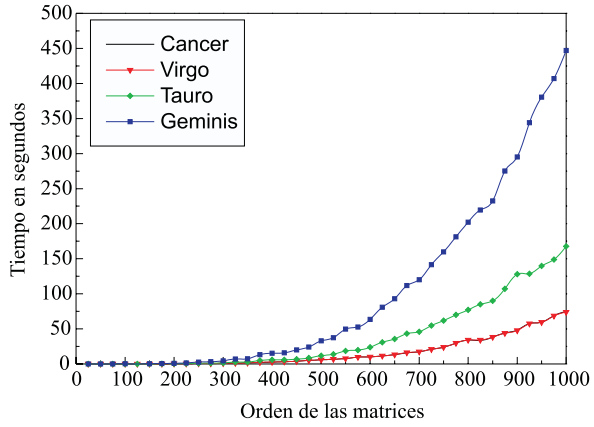

Figura 4. Algoritmo [2]

Los algoritmos utilizados necesitan almacenar tres matrices e inicializar dos de ellas. Esto representa una reserva de $3n^2$ datos. Por otro lado, la cantidad de operaciones que implica multiplicar $\mathbf{A}$ y $\mathbf{B}$ es $2n^3$. Intuitivamente, se podría pensar que si cada procesador demora t_i segundos en realizar el producto y $t_s = max\{t_i,\ i = 1, \ldots, p\}$, siendo p el número de procesadores disponibles en la red, la ejecución paralela de tal producto debería quedar reducida en la cantidad p, es decir $t_p = t_s/p$ segundos. En una implementación paralela eficiente, esto reduciría considerablemente el tiempo de ejecución.

ALGORITMOS PARALELOS

Veremos dos tipos de descomposición de las matrices. La primera consiste en particionar $\mathbf{A}$ y $\mathbf{B}$ en bloques y la segunda particiona la matriz $\mathbf{A}$ en bloques filas.

Descomposición por Bloques

Sean $\mathbf{A}$, $\mathbf{B}$ y $\mathbf{C}$ matrices de orden $n \times n$, las cuales pueden ser particionadas en m^2 bloques de tamaño $r \times r$. Cada tarea genera un bloque de $\mathbf{A}$ y de $\mathbf{B}$ y los distribuye entre las otras tareas, teniendo que multiplicar m bloques. Debido a que la red cuenta con 4 computadoras, las matrices fueron particionadas en $m^2 = 4$ bloques de tamaño $n/2 \times n/2$. Es decir

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix} \quad ; \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix} \quad \text{y} \quad \mathbf{C} = \mathbf{AB}$$

donde $\mathbf{C}_{i,j} = \sum_{k=1}^{m} \mathbf{A}_{i,k}\mathbf{B}_{k,j}$, siendo (i,j) el bloque de $\mathbf{C}$ asignado a la misma.[2]
Cada producto $\mathbf{A}_{i,k}\mathbf{B}_{k,j}$ requiere $n^3/4$ operaciones (una operación es una suma o un producto), de modo que cada tarea realiza $n^3/2$ operaciones. Además recibe dos bloques y envía otros dos a las tareas vecinas. Hemos visto que el tiempo en segundos para enviar k bytes es $t_n = \alpha + \beta k$. Se trabajó con valores tipo "float", que ocupan 4 bytes. Cada bloque tiene $n^2/4$ datos, es decir, un total de n^2 bytes. Por lo tanto, cada tarea envía un total de $2n^2$ bytes. Suponiendo que el receptor no queda ocioso esperando el mensaje, se puede estimar el tiempo total de comunicación (en microsegundos),(t_{com}) que emplea cada tarea, y el tiempo de cálculo para realizar R operaciones (t_{cal}) por

$$t_{com} = 2(\alpha + \beta n^2) = 2(909 + 1.21n^2) \quad ; \quad t_{cal} = \frac{n^3}{2R}$$

Luego, la razón entre estos tiempos cuantifica la proporción de comunicación relativa respecto al volumen de cálculo; y se observa que, si n aumenta, r también.

$$r = \frac{t_{cal}}{t_{com}} = \frac{n^3}{4R(\alpha + \beta n^2)} = O(n) \tag{3}$$

Observación: El valor de R, o de *mflops* de cada procesador, está relacionado con el tiempo de ciclo de reloj de cada máquina. En efecto,

$$R = \frac{N°operaciones}{t} = \left[\frac{N°operaciones}{1\ ciclo\ reloj}\right]\left[\frac{1\ ciclo\ reloj}{t}\right] = \gamma Mhz$$

Dada la homogeneidad de los procesadores de las computadoras de la red, se puede pensar que todas realizan igual número de operaciones en 1 ciclo de reloj. Por lo tanto, la relación de *mflops* entre dos cualesquiera de ellas, será la relación entre sus *Mhz*, y esto da una medida de las velocidades de cada una. Se pudo observar que Cancer y Virgo tienen aproximadamente el mismo valor de r, mientras que los de Tauro y Geminis son un 90 % y 70 %, respectivamente, del r de Cancer.

Detalles del algoritmo

Para identificar las tareas, éstas se unen a un grupo. La primera que se une es la que inicia las otras tareas, a las que les envía el tamaño del bloque y el número total de bloques. Todas llaman a la rutina *pvm_barrier* y se bloquean hasta que todas la han llamado. Cada tarea determina cuál bloque de $\mathbf{C}$ ha de calcular, inicializa los bloques de $\mathbf{A}$ y $\mathbf{B}$ usando valores aleatorios y realiza los cálculos. Las tareas correspondientes a los bloques de la diagonal envían su bloque de $\mathbf{A}$ a las tareas de su fila. Todas las tareas realizan el cálculo $\mathbf{A}_{i,i}\mathbf{B}_{i,j}$ y lo agregan al bloque $\mathbf{C}_{i,j}$. Después se rotan las columnas de $\mathbf{B}$ y cada tarea envía $\mathbf{B}_{i,j}$ a la tarea que está encima y recibe $\mathbf{B}_{i+1,j}$ de la que está debajo. Se realiza nuevamente el producto. Luego, el bloque $\mathbf{A}_{i,i+1}$ se envía a todas las tareas de la fila i y se continúa así durante m pasos.

Implementación

Como en el caso secuencial, $\mathbf{A}$ y $\mathbf{B}$ son matrices de orden $n \times n$ con n variando entre 25 y 1000. Se consideró la ejecución en paralelo usando los algoritmos [1] y [2], en 2 y 4 procesadores. En todos los casos se midieron los tiempos de ejecución total (Real y `CPU`) registrados en cada procesador, discriminando los tiempos de cálculo, comunicación e inicialización. Cada valor es el resultado de multiplicar $\mathbf{A}$ y $\mathbf{B}$ 15 veces.
Se observó que con ambos algoritmos, el tiempo total transcurrido está influenciado por la potencia de Geminis, que destina al cálculo más tiempo que la suma de los tiempos de las otras 3 computadoras. Esto ocasiona que estas últimas queden ociosas, viendose ésto reflejado en los tiempos de comunicación. La

utilización del desenrollado implica una reducción porcentual del tiempo total transcurrido, en promedio, del 35%. Los tiempos de cálculo de la `CPU` para cada máquina son los mismos que el tiempo total transcurrido, a excepción de Geminis en la cual la `CPU` trabaja menos de la mitad del tiempo transcurrido, hecho que también sucede en la versión secuencial.
Debido al comportamiento dispar de las computadoras de la red, se implementó el programa sólo en Virgo y Cancer, asignándoles 2 bloques a cada una. En la tabla (3) se muestran los tiempos de cálculo y comunicación para $n = 1000$, los cuales son prácticamente similares en ambas computadoras. Como antes, la implementación del desenrollado implica una reducción del tiempo total del 33.3 % para $400 \leq n \leq 1000$.
Al utilizar 2 procesadores en vez de 4, el tiempo total transcurrido se ve reducido en promedio un 41.3 % ($500 \leq n \leq 1000$) utilizando el algoritmo [1] y un 37.4 % usando el algoritmo [2], debido en parte a la disminución de los tiempos de comunicación entre Virgo y Cancer. En las figuras (5),(6) y (7) se observan los tiempos totales para las corridas con 4 y 2 procesadores.

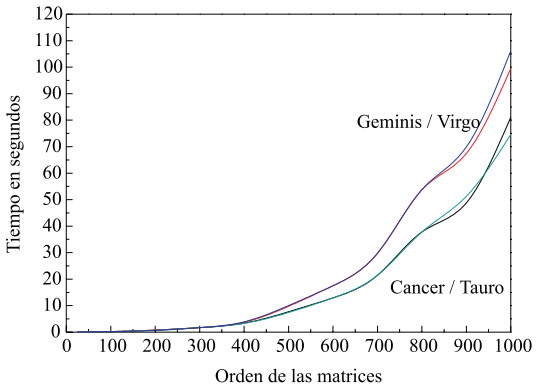

Figura 5. Algoritmo [1] en 4 Procesadores

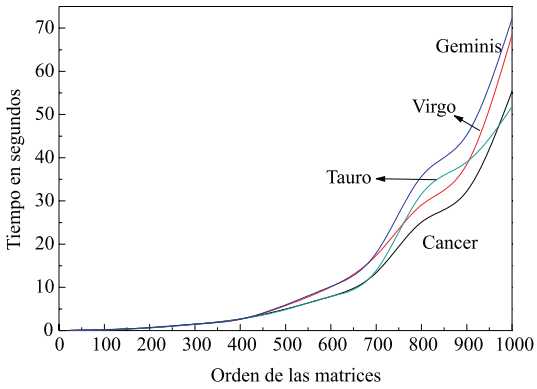

Figura 6. Algoritmo [2] en 4 Procesadores

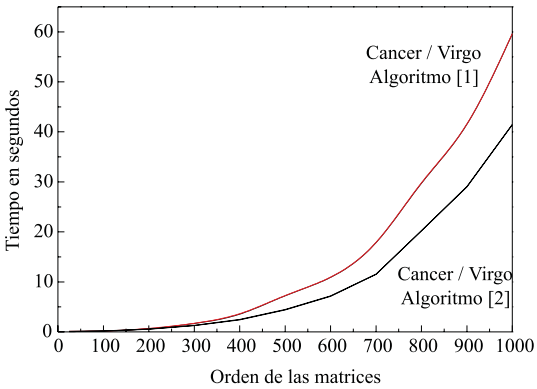

Figura 7. Algoritmos [1] y [2] en 2 Procesadores

Tabla 3: Algoritmo por bloques ($n = 1000$)

Alg. [1]	Cálculo	Comunicación	Total
Virgo	47.8	11.2	59.6
Cancer	49.4	9.6	59.6
Alg. [2]	Cálculo	Comunicación	Total
Virgo	30.6	10.3	41.5
Cancer	30.6	10.2	41.4

Medidas de eficiencias paralelas

En las implementaciones paralelas, lo que resulta de interés es el tiempo total transcurrido. Con el propósito de medir la performance de los algoritmos paralelos, se consideran dos medidas.
Se define la *Aceleración* o *Speedup* de un programa paralelo, al cociente $S_p = \frac{t_s}{t_p}$, donde t_s es el tiempo de ejecución secuencial de un programa y t_p el tiempo de ejecución en paralelo con p procesadores. La *Eficiencia* de un algoritmo paralelo usando p procesadores está dada por $E_p = \frac{S_p}{p} = \frac{t_s}{pt_p}$. En condiciones ideales, $S_p \to p$ y $E_p \to 1$. En la práctica $S_p \leq p$ y $E_p \leq 1$.
Para calcular el Speedup, se tomó como tiempo secuencial al obtenido con Cancer y como tiempo paralelo el mayor de los tiempos registrados en las computadoras. Las figuras (8) y (9) muestran la evolución del Speedup y la Eficiencia en función de n usando 2 y 4 procesadores.

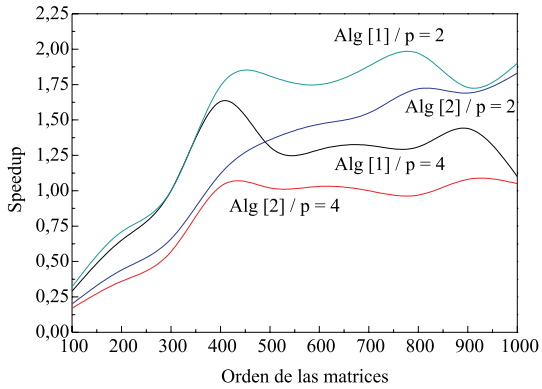

Figura 8. Speedup

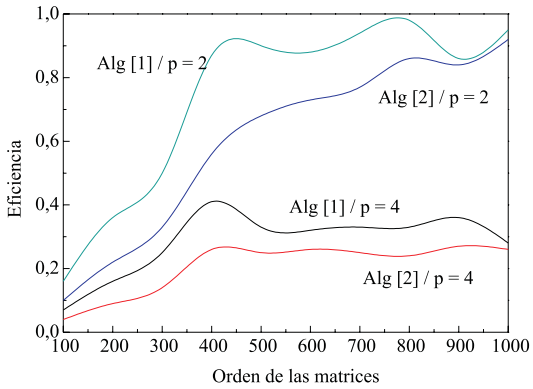

Figura 9. Eficiencia

Descomposición por filas

Se propone realizar el producto matricial mediante la descomposición de $\mathbf{A}$ y $\mathbf{C}$ en m bloques fila. La diferencia con la descomposición por bloques es que ahora los tamaños de los bloques no necesariamente son iguales y entonces se puede asignar a cada procesador una cierta cantidad de filas según sus potencias de cálculo, permitiendo un mejor balanceo del problema.

$$\begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_m \end{bmatrix} \begin{bmatrix} \\ \mathbf{B} \\ \\ \end{bmatrix} = \begin{bmatrix} \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_m \end{bmatrix}$$

Cada tarea debe resolver un bloque fila de la matriz $\mathbf{C}$: $\mathbf{C}_k = \mathbf{A}_k\mathbf{B}$, $k = 1, \ldots, m$ donde $\mathbf{A}_k$ y $\mathbf{C}_k$ son de orden $p_k \times n$ y $\sum_{k=1}^{m} p_k = n$. El producto anterior requiere un total de $2p_k n^2$ operaciones.
El programa se realizó siguiendo un esquema del tipo Master-Slave donde el master crea las matrices $\mathbf{A}$ y $\mathbf{B}$ en forma aleatoria, se queda con un bloque de $\mathbf{A}$ y asigna los restantes a las otras tareas. La matriz $\mathbf{B}$ se envía a todas las tareas "slaves". Una vez que cada tarea tiene los datos, realiza el producto mediante el algoritmo [2] con desenrollado igual a 10 y lo envía al master. Este esquema no requiere sincronización de las tareas.

Implementación

Se ejecutó el programa en 2 y 4 procesadores, para matrices de orden $n \times n$, con n variando desde 25 a 1000. Los resultados usando 2 procesadores se obtuvieron en Cancer y Virgo. A cada uno le fue asignado en promedio, el 50 % de las filas de la matriz $\mathbf{A}$, debido a las características computacionales de las mismas. No sucede lo mismo con 4 procesadores, donde la heterogeneidad de las computadoras no ha permitido una división equitativa de $\mathbf{A}$. En promedio, los porcentajes asignados fueron: 50 % a Cancer, 36 % para Virgo, 10 % para Tauro y sólo 4 % a Geminis. La tabla (4) muestra los tiempos registrados con 2 y 4 procesadores para $n = 1000$.
Al igual que con el algoritmo por bloques, es notoria la influencia de Geminis y de Tauro (en menor medida), especialmente para $n \leq 700$, donde la reducción promedio de tiempo de ejecución al usar 2 procesadores es del 37 %. Sólo para $n = 1000$, el tiempo con 2 procesadores supera en un 8 % al tiempo con 4 procesadores. Lo interesante es que las tareas se sincronizan de modo tal que los tiempos totales son iguales en ambos procesos. Utilizando 4 procesadores, se logran equiparar los tiempos totales asignando a Geminis y Tauro no más del 20 % del cálculo computacional total.
Respecto al algoritmo por bloques, se logra una mejoría en los tiempos totales con 4 procesadores de aproximadamente un 22 % para $n \geq 800$, y respecto a la versión secuencial obtenida en Cancer, conviene la utilización del algoritmo por filas para matrices de orden 400 en adelante. Usando 2 procesadores, los tiempos totales se ven incrementados en promedio un 26.3 % ($400 \leq n \leq 1000$) respecto al algoritmo por

bloques, y es conveniente aplicarlo si $n \geq 600$, de otro modo se prefiere la versión secuencial. Las curvas dadas en la figura (10) muestran los tiempos totales transcurridos en la ejecución del algoritmo por filas tanto en 2 como en 4 procesadores.

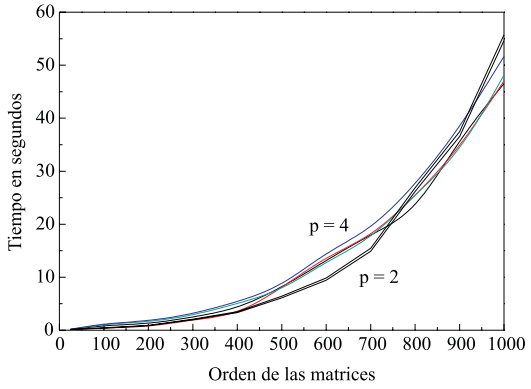

Figura 10. Algoritmo [2]

Tabla 4: Algoritmo por filas ($n = 1000$)

Alg. [2]	Cálculo	Comunic.	Total
Virgo	44.4	10.2	54.6
Cancer	37.9	17.8	55.7
Alg. [2]	Cálculo	Comunic.	Total
Virgo	36.5	10	46.5
Cancer	35.9	11.2	47.1
Tauro	39.9	8.2	48.1
Geminis	16.3	35.3	51.6

Para calcular el Speedup se tomó el mejor tiempo secuencial - obtenido en Cancer - dividido el tiempo paralelo. Comparando los valores obtenidos con el algoritmo por bloques, el algoritmo por filas tiene menor Speedup y es menos eficiente cuando se utilizan 2 procesadores, aunque esta situación se revierte al utilizar 4 procesadores.

CONCLUSIONES

La implementación en paralelo de un algoritmo eficiente demanda un estudio exhaustivo de la red de computadoras que se tiene. Se tiende a pensar que contar con una mayor cantidad de procesadores, hará disminuir el tiempo total de ejecución, lo cual no necesariamente es cierto. En los casos particulares testeados, se ha observado que los mejores resultados se logran trabajando con 2 procesadores y no con 4. Además, con ambos algoritmos se logran buenos resultados empleando la técnica del desenrollado. Si se comparan los resultados con la versión secuencial, resulta que conviene paralelizar a partir de $n = 400$. Desde el punto de vista del Speedup y la Eficiencia, se observa que el Algoritmo [1] usando 2 procesadores es el que alcanza los resultados teóricos esperados.

AGRADECIMIENTOS

Este trabajo se ha realizado con el apoyo de la Universidad Nacional del Litoral, del CONICET, y con subsidios de los proyectos: *PICT 51 FONCyT* y *PIP 198/98 y 266/98 CONICET*.

Referencias

[1] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van der Vorst, "Solving Linear Systems on Vector and Shared Memory Computers", SIAM, Philadelphia, 1991.

[2] G. H. Golub y C. Van Loan, "Matrix Computation", 2nd. edition, The John Hopkins University Press, Baltimore, MD, 1993.

[3] Victorio E. Sonzogni, "Cálculo Paralelo en Análisis Estructural", Mecánica Computacional, Vol. 14, pag 50-59, Santa Fe, 1994.

[4] A. Geist et al., "PVM3 User's Guide and Reference Manual", Rep. ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee USA, 1993.

[5] J. J. Dongarra, Tom Dunigam, "Message - Passing Performance of Various Computers", Rep. University of Tennessee and Oak Ridge National Laboratory, 1997 - "www.netlib.org/utk/papers/commperf.ps".