Mecánica Computacional Vol. XXIV
A. Larreteguy (Editor)
Buenos Aires, Argentina, Noviembre 2005

RESOLUCIÓN PARALELO DISTRIBUIDA DE SISTEMAS DE ECUACIONES BASADA EN TÉCNICAS DE DESCOMPOSICIÓN DE GRAFOS

Axel J. Soto[1,2], Ignacio Ponzoni[1, 2], Gustavo E. Vazquez[1]

[1] Laboratorio de Investigación y Desarrollo en Computación Científica (LIDeCC)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Av. Alem 1253 – 8000 - Bahía Blanca - ARGENTINA

[2] Planta Piloto de Ingeniería Química (PLAPIQUI)
Universidad Nacional del Sur - CONICET
Complejo CRIBABB – Camino La Carrindanga km. 7 – CC 717 - Bahía Blanca - ARGENTINA
asoto@plapiqui.edu.ar; ip@cs.uns.edu.ar; gev@cs.uns.edu.ar;

Key Words: procesamiento paralelo, resolución de sistemas de ecuaciones lineales, matrices ralas, teoría de grafos.

Abstract. *En este trabajo se presenta la implementación paralelo distribuida de un nuevo método para la resolución de sistemas de ecuaciones algebraicos lineales ralos, el cual está basado en técnicas de descomposición de grafos. Nuestra hipótesis de partida fue que la técnica de descomposición de dominios, combinada con el procesamiento paralelo distribuido, permitiría la reducción de los tiempos de cómputo requeridos para resolver sistemas de ecuaciones de gran dimensión. Así surgió la propuesta de particionar el sistema original en un conjunto de subsistemas de menor dimensión que puedan ser resueltos en forma simultánea. Se decidió efectuar dicho reordenamiento empleando la técnica de descomposición de grafos denominada Método Directo. Este método reestructura la matriz original a un forma triangular inferior en bloques, donde los bloques sobre la diagonal corresponden a subsistemas de ecuaciones no singulares. Dado que cada uno de los subsistemas puede depender o no de los anteriores, nuestro algoritmo calcula las relaciones de dependencia mediante la generación de un grafo dirigido acíclico (DAG). El trabajo experimental se realizó sobre un cluster de monoprocesadores conectados a través de una red Ethernet. En la implementación del algoritmo se usó la librería de pasaje de mensajes PVM, mientras que la estrategia de paralelización siguió un esquema master-worker. Los resultados obtenidos fueron altamente satisfactorios en términos de speed-up, eficiencia y estabilidad. A su vez, los mismos permiten observar que el método propuesto posee un mejor desempeño cuanto más grandes y ralas sean las matrices asociadas a los sistemas.*

1 INTRODUCCIÓN

La complejidad de los modelos computacionales en una amplia variedad de problemas reales ha ido creciendo sustancialmente, y en la actualidad resulta de vital importancia contar con herramientas adecuadas para abarcar las necesidades de cómputo de dichos modelos. En particular, la resolución de sistemas de ecuaciones de gran dimensión resulta de destacado interés, ya que un sinnúmero de modelos reales requieren de la resolución de sistemas algebraicos en alguna de sus instancias de cálculo[1].

Un factor clave en el desarrollo actual de sistemas de software a gran escala, consiste en el uso de procesamiento paralelo como alternativa viable de ejecución de aplicaciones de alta intensidad computacional[2]. El procesamiento paralelo se puede definir, en esencia, como la conjunción de dos o más procesadores que computan, en forma solapada en el tiempo, subtareas correspondientes a un problema más grande. Dentro del procesamiento paralelo se pueden identificar dos ramas o modelos principales, donde la diferencia entre uno y otro reside en la forma en la que cada procesador accede a su memoria[3].

En el primero, llamado modelo de memoria compartida, o conocido también como sistemas multiprocesador, los procesadores intervinientes en el sistema acceden a una única memoria común a todos. En contraste, en el modelo de memoria distribuida, también denominado como sistemas multicomputador o cluster de computadoras, cada procesador posee su propia memoria local. Por tal motivo, si un procesador desea acceder a la memoria local de otro, se requiere de un intercambio de mensajes entre ambos a través de una red que los comunica. En la Figura 1 se esquematiza la diferencia entre ambos modelos.

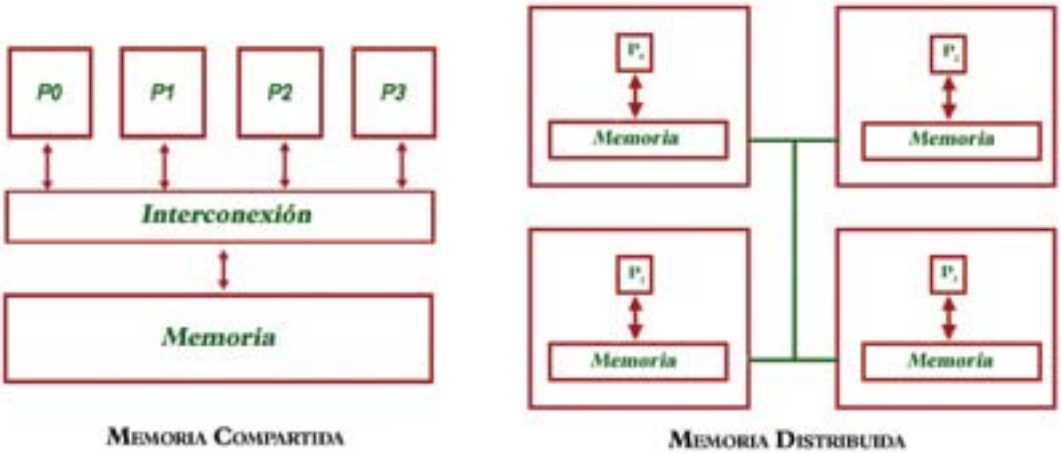

Figura 1: Modelos de procesamiento paralelo.

En el modelo de memoria compartida los procesadores se comunican con la memoria a través de un bus o sistema de *switches* (llaves) de alta velocidad. Esto les permite lograr un mejor desempeño en comparación con los sistemas de memoria distribuida. Otra ventaja que presenta este modelo es su uso más eficiente de la memoria, dado que no hay necesidad de replicación de datos. No obstante, este tipo de arquitecturas presentan dos importantes dificultades: el alto costo actual de este tipo de hardware y la escasa

portabilidad para migrar un programa codificado en un sistema multicomputador hacia otra plataforma de memoria compartida.

En contrapartida, los sistemas distribuidos poseen varias ventajas. En primer lugar, este modelo permite desarrollar software más portable debido a los estándares existentes en los protocolos de comunicación. Por otra parte, los clusters poseen bajo costo y buena escalabilidad dado que usualmente están integrados por computadoras personales interconectadas por una red Ethernet. Por estas razones, la tendencia actual en procesamiento paralelo apunta hacia el empleo de este tipo de sistemas multicomputador[4,5].

Dentro de la computación paralelo-distribuida existen varias técnicas para resolver diferentes familias de problemas, y cada una de estas técnicas puede resultar más (o menos) provechosa en función de las características del problema a resolver[6]. En general, varios de los paradigmas de paralelización surgen de dos grandes enfoques: descomposición de dominios y descomposición funcional. El primero, se centra en la división de los datos a utilizarse, los cuales son luego procesados simultáneamente, mientras que el segundo enfoque descompone el programa en distintas subtareas que se resuelven posteriormente en paralelo. Es importante destacar que la mayoría de los problemas a resolver requieren una combinación de ambos enfoques para alcanzar un máximo grado de paralelismo.

Por otra parte, recientes progresos en el campo de la teoría de grafos[7] condujeron a la hipótesis de utilizar técnicas de descomposición de grafos asociadas al reordenamiento en bloques de matrices, de manera de poder particionar sistemas de ecuaciones algebraicas (SEAs) de gran dimensión. En sí, estás técnicas permiten que cualquier SEA pueda ser reordenado en forma estructural, independientemente de su grado de no linealidad y de la cantidad de soluciones que posea, es decir, puede ser reordenado aun en presencia de singularidades numéricas.

Así surgió nuestra hipótesis de utilizar estas técnicas de descomposición de grafos para obtener subsistemas del SEA original, donde, según sea la dependencia entre dichos subsistemas, y aprovechando el cómputo paralelo distribuido, éstos puedan ser resueltos en forma independiente unos de otros. En otras palabras, nuestra idea es emplear el enfoque de descomposición de dominios sobre la base de los métodos de particionamiento de grafos presentados en Ponzoni (2001)[7] para la resolución de SEAs ralos y de gran dimensión.

En este trabajo se presenta un nuevo método numérico que permite la resolución de SEAs lineales en forma paralela. En esta primera versión, se restringió el método para ser utilizado en la resolución de sistemas de ecuaciones lineales con solución única; no obstante, y como se explicará más adelante, la potencialidad del método permitiría trabajar también con sistemas no lineales y con singularidades.

Este artículo está organizado como sigue: en la sección 2, se resume la técnica de descomposición de matrices utilizadas en el nuevo método; en la sección 3 se explican los algoritmos paralelos diseñados; en la sección 4 se analizan los resultados obtenidos; y para finalizar, en la sección 5 se presentan las conclusiones extraídas del trabajo, con sus mejoras y variantes para considerar a futuro.

2 DESCOMPOSICIÓN DE MATRICES EMPLEANDO TEORÍA DE GRAFOS.

El enfoque aquí propuesto está inspirado en una técnica diseñada para análisis de observabilidad en Ingeniería de Procesos, conocida como Método Directo[8]. Básicamente, esta técnica efectúa reordenamientos estructurales en la matriz de ocurrencia correspondiente al SEA que modela el funcionamiento de un proceso industrial. El núcleo central de este procedimiento emplea algoritmos de descomposición de grafos que permutan la matriz de ocurrencia a una forma triangular inferior en bloques, donde los bloques sobre la diagonal corresponden a subsistemas del SEA original.

Por esta razón, se decidió emplear estos algoritmos de particionamiento de grafos como estrategia de descomposición de dominios para un método paralelo de resolución de SEAs. El objetivo de esta sección es explicar detalladamente cómo se realiza el reordenamiento estructural de la matriz, y su consecuente descomposición en bloques.

2.1 Estructura general del método

El análisis de observabilidad dentro de un proceso industrial se realiza a partir del estudio del modelo matemático que representa el funcionamiento de la planta en estado estacionario. En términos generales, dicho modelo consiste en un SEA (lineal o no-lineal) el cual representa balances, relaciones termodinámicas y correlaciones experimentales.

El método directo es un algoritmo que permite la clasificación de las variables involucradas en el SEA asociado a la instrumentación de una planta, de manera de poder identificar si los sensores (variables medidas) ubicados dentro de un proceso industrial son suficientes para conocer todas las variables de interés, o para identificar mediciones que sean redundantes.

Básicamente, el método distingue cuatro categorías de variables según su factibilidad de cálculo, siguiendo la clasificación presentada en Romagnoli y Sánchez[9]:

1. Variables Redundantes: variables medidas que también pueden ser calculadas a partir de las ecuaciones del modelo matemático y las restantes mediciones.

2. Variables No Redundantes: variables medidas que no pueden ser calculadas a partir de las ecuaciones del modelo matemático y las restantes mediciones.

3. Variables Observables: variables no medidas que pueden ser calculadas a partir de las ecuaciones del modelo matemático y las mediciones.

4. Variables No Observables: variables no medidas que no pueden ser calculadas a partir de las ecuaciones del modelo matemático y las mediciones.

Del mismo modo, las ecuaciones del SEA pueden clasificarse en tres grupos:

***1. Ecuaciones Asignadas*:** aquellas que se emplean para despejar variables observables.

2. Ecuaciones Redundantes: son aquellas cuyas variables son todas observables o medidas, y que no se emplean para despejar variables observables.

3. Ecuaciones No Asignadas: son aquellas que contienen al menos una variable no observable y por ende no pueden emplearse para despejar variables observables.

El método directo permite descomponer SEAs lineales o no lineales, con o sin singularidades numéricas. Esta característica representa una ventaja, ya que el chequeo de situaciones de singularidad numérica se determina directamente sobre los subsistemas que corresponden a las ecuaciones y variables de interés. Además, como en general los subsistemas originados por el método son de tamaño pequeño, el costo computacional de detectar singularidades numéricas resulta mucho más bajo que si se chequeara sobre el SEA completo.

A partir del esquema de descomposición del método directo, surgió nuestra idea de utilizarlo para resolver SEAs en paralelo. El método propuesto en este trabajo es una primera versión limitada a la resolución de sistemas lineales con solución única, aunque dadas las potencialidades del método de descomposición, es factible su extensión a sistemas no lineales y con singularidades numéricas.

En esencia, la estructura del método directo consta de siete etapas: *Inicialización, Descomposición Gruesa, Descomposición Fina, Test de Admisión, Reasignación, Reducción y Reordenamiento*. Los últimos cuatro pasos están específicamente vinculados al tratamiento de singularidades numéricas. Por esto, y dado que en este trabajo los sistemas utilizados en la resolución poseen solución única, en esta primera versión sólo se utilizaron las tres primeras etapas del método de reordenamiento.

2.1.1 Inicialización y Descomposición Gruesa

En la inicialización se construye el grafo bipartito $G = (R, C, A)$ asociado a la matriz de ocurrencia N del SEA a resolver, donde los nodos en R y C están vinculados a las filas (ecuaciones) y columnas (variables) de N respectivamente, y el conjunto de aristas A representa los elementos no nulos de la matriz N. Una vez construido G, se efectúa un primer particionamiento, al cual se lo denomina descomposición gruesa. En esta etapa se busca un pareamiento maximal del grafo bipartito G (*maximum bipartite matching*)[10]. En la Figura 2 se puede apreciar la relación entre la matriz de ocurrencia y un SEA, mientras que en la Figura 3 se observa su grafo bipartito asociado.

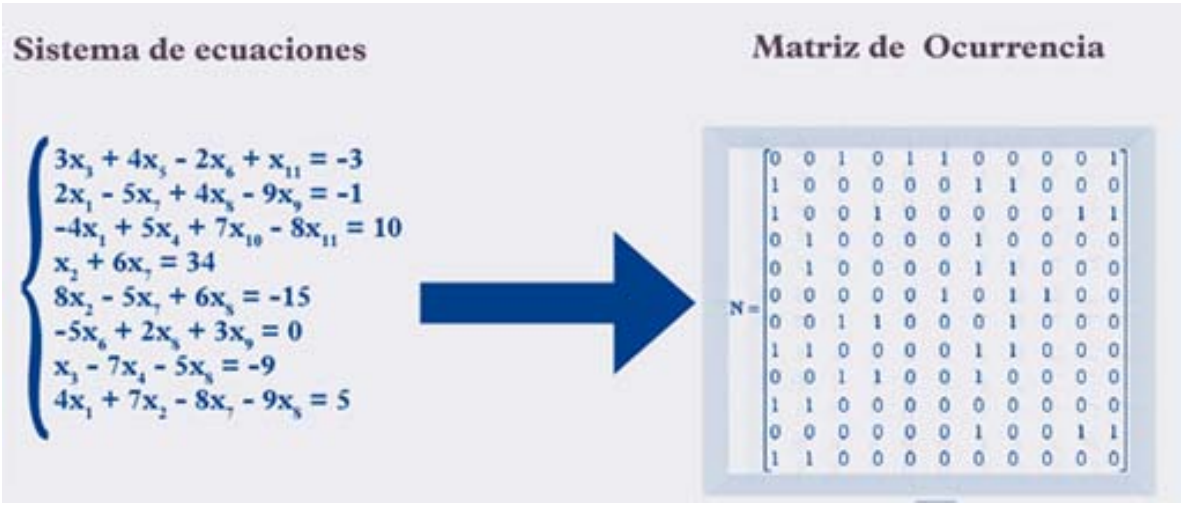

Figura 2: Matriz de ocurrencia asociada a un SEA.

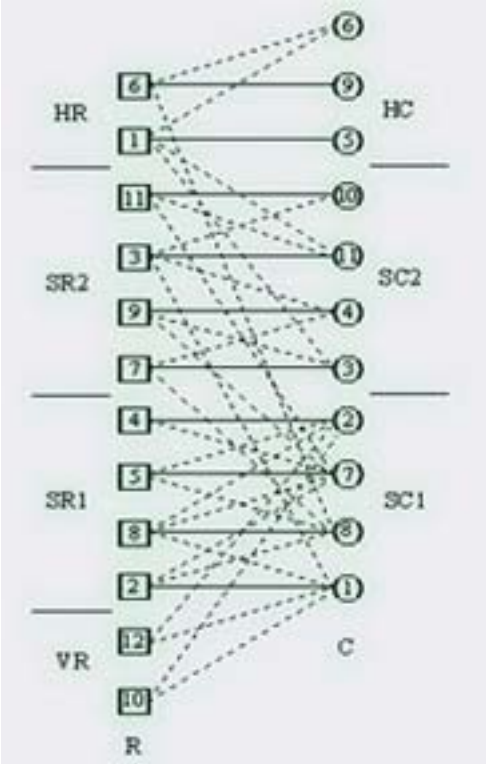

Figura 3: Grafo Bipartito. Las aristas de trazo lleno indican aquellas que pertenecen al pareamiento maximal.

Un pareamiento de un bigrafo ***G*** es un subconjunto de sus aristas tal que ninguna de ellas poseen nodos en común. Se dice que un nodo está apareado si existe una arista en el pareamiento que sea incidente a él. Un camino alternante sobre el bigrafo ***G*** relativo a un pareamiento ***P*** es un camino (secuencia finita de aristas conectadas) cuyas aristas alternan en el pareamiento ***P***. Por ejemplo, si la primera arista del camino alternante está en ***P***, luego las aristas de las posiciones impares del camino pertenecen a ***P*** y las pares no. Un pareamiento maximal, $\boldsymbol{P_m}$, constituye un pareamiento tal que la cardinalidad de su conjunto de aristas es máxima entre todos los pareamientos posibles. Luego de obtenido el $\boldsymbol{P_m}$, se procede a clasificar los nodos de la siguiente manera:

VR = {nodos no apareados de ***R***}

SR1 = {nodos apareados de ***R*** alcanzables desde algún nodo no apareado de ***R*** mediante un camino alternante}

HR = {nodos apareados de ***R*** alcanzables desde algún nodo no apareado de ***C*** mediante un camino alternante}

SR2 = ***R*** \ (VR ∪ SR1 ∪ HR)

SC1 = {nodos apareados de ***C*** alcanzables desde algún nodo no apareado de ***R*** mediante un camino alternante}

HC = {nodos de ***C*** tanto apareados como no apareados alcanzables desde algún nodo no apareado de ***C*** mediante un camino alternante}

SR2 = ***C*** \ (SC1 ∪ HC)

De este modo, las variables indeterminables quedan asociadas a la clase HC, mientras que las variables observables corresponden a los grupos SC1 y SC2. Asimismo, las ecuaciones asignadas quedan asociadas a las clases SR1 y SR2, mientras que las redundantes y no asignadas se corresponden con los grupos VR y HR respectivamente. En el bigrafo de la Figura 3 se detalla la clasificación realizada sobre el mismo, y en la Figura 4 se muestra como queda finalmente particionada la matriz de ocurrencia como producto de la descomposición gruesa. En síntesis, la descomposición gruesa permite obtener dos bloques cuadrados estructuralmente no singulares determinados por los conjuntos (SR1, SC1) y (SR2, SC2).

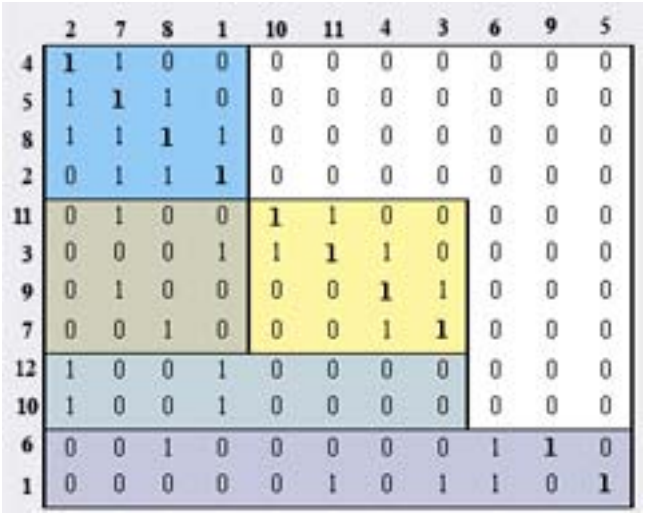

Figura 4: Descomposición gruesa: Azul (SR1, SC1), Amarillo (SR2, SC2), Gris Claro (VR) y Gris Oscuro (HR).

2.1.2 Descomposición Fina

La siguiente etapa de este proceso, llamada descomposición fina, particiona los bloques (SR1, SC1) y (SR2, SC2) en subsistemas cuadrados no reducibles mediante un algoritmo de detección de componentes fuertes de un digrafo[11]. Para esto, primero se debe asociar un digrafo a cada uno de los bloques de la matriz de la siguiente forma: por cada elemento no nulo A_{uv} de la fila ***u*** columna ***v*** del bloque ***A***, se genera una arista ***e = (u,v)*** en el dígrafo que va del nodo ***u*** al ***v***. Luego, se aplica el algoritmo de detección de componentes fuertes a cada uno de los digrafos generados, y cada uno de los componentes fuertes detectados corresponde a un subconjunto de asignación (o subsistema) de la descomposición final de la matriz. En la Figura 5 se muestra un pequeño ejemplo de este último proceso. De este modo, entre las fases de descomposición fina y gruesa, se logra la descomposición de un SEA en subsistemas irreducibles y estructuralmente no singulares.

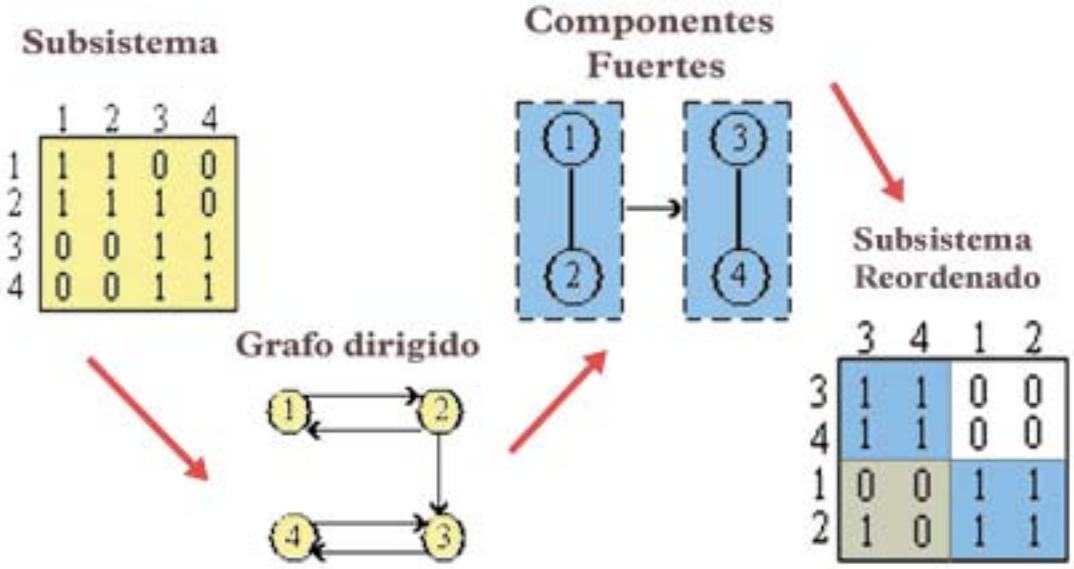

Figura 5: Proceso de descomposición fina. Aquí el subsistema {1, 2, 3, 4} (amarillo) se descompuso en {3, 4} y {1, 2} (azul)

3 MÉTODO NUMÉRICO PARALELO DISTRIBUIDO PARA LA RESOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES.

En la sección anterior se examinó un método de descomposición en bloques de una matriz bivaluada. En nuestro método numérico, esta matriz se corresponde con la matriz de ocurrencia de un SEA lineal. Por ende, los bloques generados representan la división del SEA en una serie de subsistemas lineales más chicos, no singulares y con solución única.

Las ventajas de transformar un SEA de gran dimensión en una secuencia de subsistemas de menor orden son esencialmente dos. La primera es la posibilidad de paralelización. Cada uno de estos subsistemas delimitan la granularidad de paralelismo del método, ya que, cada uno corresponde a unidades de resolución independientes. Lo ideal sería que todos los subsistemas pudieran ser resueltos en forma simultánea, pero como se verá más adelante, existen dependencias entre los subsistemas que obligan al secuenciamiento en la resolución de algunos de ellos.

La segunda ventaja de la descomposición reside en la reducción de: tiempos de cómputo, propagación de errores y condicionamiento numérico, puesto que se trabaja con subsistemas de menor orden. Más aún, los métodos directos (tales como Gauss Jordan, QR y otros) se vuelven imprácticos a medida que las dimensiones del sistema crecen debido al alto esfuerzo computacional que requieren. En contrapartida, los métodos iterativos (como SOR y Jacobi) si bien son aplicables a matrices ralas, en sistemas demasiado grandes no es fácil hallar un vector de inicialización que logre la convergencia hacia una solución.

El contenido de esta sección está organizado del siguiente modo: en la primera parte se explica la problemática asociada a las dependencias entre los subsistemas, luego se introduce el esquema de paralelización elegido y finalmente se detallan los funcionamientos de las dos componentes básicas de la aplicación (el *master* y el *worker*) y cómo interactúan entre sí.

3.1 Digrafo de dependencias - DAG (Directed Acyclic Graph)

Antes de seguir adelante, repasemos nuevamente los pasos de la descomposición fina. A partir de los dos grandes bloques cuadrados generados por la descomposición gruesa, se construye un digrafo asociado a cada uno de los bloques (no confundir este digrafo con el que ahora explicitaremos). Al digrafo se le aplica un algoritmo de detección de componentes fuertes, para particionar al mismo en un conjunto de componentes conexas. Cada una de estas componentes conexas se corresponden con los subsistemas de la descomposición final de la matriz.

De cada uno de estos subsistemas generados, no todos pueden ser resueltos en forma independiente. Puede que uno o más subsistemas requieran de la resolución de otro, para comenzar su proceso de cálculo. En este caso diremos que existe una dependencia, ya que se exige la finalización de un subsistema para el arranque de otros. Asimismo, esta dependencia puede estar supeditada a la finalización de más de un subsistema. En términos formales:

Definición 3.1: Un subsistema S_i es dependiente de un subsistema S_j, si al menos una de las ecuaciones de S_i contiene una incógnita con coeficiente no nulo, que también es una incógnita con coeficiente no nulo en S_j. Para el mismo caso, diremos que S_j es dominante de S_i.

Por ejemplo, en la Figura 6 C_3 es dependiente de C_1 y C_2, dado que para computar C_3 es necesario conocer las variables 4 y 9, cuyos valores son conocidos recién luego de resolver los subsistemas dominantes de C_3.

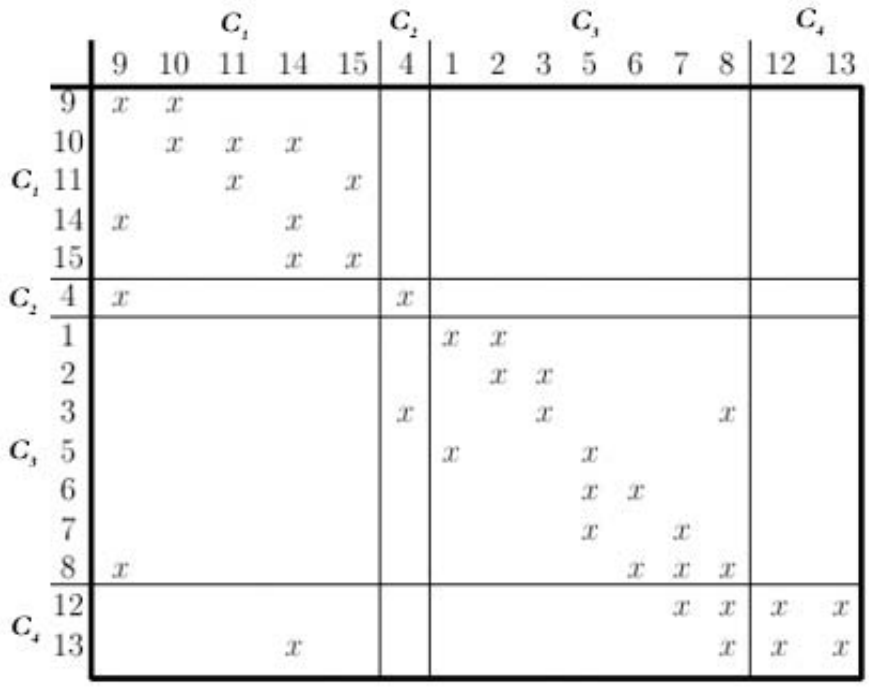

Figura 6: Identificación de dependencias. Los subsistemas C_1 y C_2 son dominantes de C_3 por causa de las variables 4 y 9

Para representar todas las dependencias de los subsistemas de la matriz, utilizamos un digrafo acíclico de dependencias (DAG)[12], en el que cada vértice se corresponde con un subsistema y cada arista con una dependencia. La arista ***(i, j)***, modela que el subsistema ***j*** es dependiente del subsistema ***i***. En particular, el DAG empleado en nuestro trabajo no es modelado como un DAG clásico, sino que se lo modela diferenciado por niveles según se hace en Pissanetzky[13]. Para entender esta clasificación por niveles, veamos primero la siguiente definición:

Definición 3.2: Decimos que una arista ***(p, q)*** es una salida de la componente fuerte $C = (V_c, E_c)$ o, que deja C, si $p \in V_c$ y $q \notin V_c$. Una arista ***(p, q)*** es una entrada de C o, entra a la componente fuerte C, si $p \notin V_c$ y $q \in V_c$.

Una propiedad importante de los DAGs es que, cuando un digrafo es particionable en más de una componente fuerte se puede apreciar que existe al menos una de ellas que no tiene salidas. Puesto que, si todas las componentes tuvieran una salida, seríamos capaces de encontrar un camino que vaya de una componente a otra, luego a otra y así sucesivamente hasta volver a la primera, lo cual contradice la definición de componente fuerte. En general, puede haber más de una componente que no tenga salidas.

Decimos entonces, que todas las componentes fuertes que no tengan salidas pertenecen al nivel 1. Todas las componentes fuertes donde todas sus salidas entren a alguna componente de nivel 1 pertenecen al nivel 2. Nuevamente, todas las componentes fuertes donde todas sus salidas entren a componentes de niveles 1 ó 2 pertenecen al nivel 3. Nótese que una componente de nivel 3 puede no tener salidas a componentes de nivel 1, pero siempre debe tener a nivel 2, pues de sólo tener salidas al 1, pertenecería al nivel 2. Resulta claro observar que, una salida de un subsistema ***i*** a uno ***j***, marca, también, que el subsistema ***i*** es dependiente del ***j***. Esta clasificación de las componentes por niveles es única. Formalmente definimos:

Definición 3.3: Una componente C_i pertenece al:

- Nivel 1 ó L_1: Si C_i no tiene salidas.
- Nivel $(k + 1)$ ó L_{k+1} : Si $\exists$ (u, v) salida desde C_i y $v \in C_j$, donde $C_j \in L_{k'}$ y C_j es la componente dominante de C_i que pertenece al nivel más alto.

En la Figura 7 se muestra el DAG estructurado por niveles asociado a la matriz de la Figura 6.

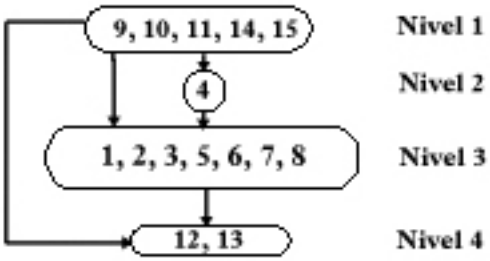

Figura 7: DAG diferenciado por niveles asociado a la matriz de la Figura 6.

A partir de esta nueva clasificación de las componentes fuertes (o subsistemas) por niveles, podemos representar al DAG de una manera que resulta más representativa para llevar un orden en el cómputo de los subsistemas. Nótese que los subsistemas de nivel 1 pueden ser resueltos en primera instancia y en forma simultánea. A medida que cada subsistema es calculado, nuevos subsistemas pueden empezar a resolverse, siempre y cuando el cómputo de todos sus dominantes haya terminado.

3.2 Esquema de paralelización

El paradigma *Master-Worker*[14] fue elegido para el diseño del algoritmo paralelo. La tarea del *master* consiste en realizar la descomposición estructural de la matriz, crear los *workers* y coordinar el secuenciamiento de las tareas entre ellos. Cada *worker* al iniciar, carga los datos numéricos de la matriz y, mediante comunicaciones con el *master*, se encarga del proceso de resolución de los subsistemas de ecuaciones. Existe un *worker* por cada unidad de procesamiento disponible, de manera que un *worker*, a lo largo de su ejecución puede llegar a resolver más de un subsistema.

El pseudocódigo del *master* se bosqueja en el siguiente algoritmo:

```
Inicio
1   Crear tantos workers como procesadores haya;
2   Aplicar método directo de descomposición en bloques de la matriz;
3   Repetir mientras haya subsistemas sin resolver;
        Si existen workers libres
3a       Asignar subsistema a worker libre;
3b       Mandar datos de dicho subsistema al worker;
        Sino {No hay procesadores libres}
            Si hay terminación de subsistema
3c              Recibir resultados;
    Fin repetir;
4   Mandar señal de terminación para todos los workers;
Fin
```

Al comenzar se crea un número de *workers* igual a la cantidad de procesadores disponibles. En el paso 2 se aplica el método directo de descomposición explicado en la sección anterior. El ciclo de la sentencia 3, corresponde al intercambio de mensajes con los *workers*. Posterior a la asignación de un subsistema a un procesador se deben transmitir a los *workers* los datos del mismo (pasos 3a – 3b), es decir, se debe indicar qué filas y qué columnas de la matriz componen el subsistema, y en caso de tener subsistemas dominantes, las soluciones de los mismos. Por otro lado, cuando un *worker* termina, el *master* recibe los resultados del subsistema computado por éste (paso 3c).

La asignación de los subsistemas a los *workers*, se hace según una política de prioridades, de manera que los subsistemas *"cuello de botella"* sean resueltos lo antes posible. Por último, es necesario mandar una señal de finalización (paso 4), indicando a los *workers* cuando no hay más subsistemas por resolver.

El algoritmo del *worker* se esquematiza de la siguiente manera:

```
Inicio
1   Recuperar coeficientes y términos independientes de toda la matriz;
2   Repetir mientras no se reciba señal de finalización
2a      Recibir datos del subsistema a resolver;
2b      Armar subsistema;
2c      Resolver subsistema;
2d      Mandar resultados al master;
    Fin repetir;
Fin
```

En primer lugar, el *worker* traspasa la matriz de coeficientes completa, la cual debe estar en la memoria secundaria de todos los nodos, a una estructura de datos acorde para almacenar una matriz rala. Es importante mencionar, que si bien se requiere de un tiempo importante para realizar la entrada de los datos, este tiempo no influye demasiado en el tiempo de cómputo global, dado que se hace en paralelo con el proceso de descomposición de la matriz de ocurrencia en el *master*.

Luego, el *worker* entra en un ciclo, en el que, mientras no se reciba un mensaje de finalización, resuelve los subsistemas que le envía el *master*. A partir de estos datos enviados, el *worker* arma la matriz numérica correspondiente al subsistema a computar y lo resuelve utilizando el método de Gauss-Jordan. En el caso de que el subsistema sea de nivel mayor que 1, se reciben también los resultados de variables correspondientes a sus subsistemas dominantes.

Finalmente, nótese que una característica importante del esquema de paralelización adoptado, es que el *master* y el *worker* no sólo realizan actividades diferentes (paralelismo funcional), sino que también trabajan con conjuntos de datos distintos (paralelismo de datos). El *master* sólo posee información de la matriz de ocurrencia, mientras que los *workers* trabajan con los datos numéricos de la matriz.

4 TRABAJO EXPERIMENTAL

4.1 Diseño de casos de estudio

El primer paso para la evaluación del desempeño del algoritmo paralelo consistió en decidir la batería de casos de estudio a emplear durante los experimentos. En tal sentido, fue necesario establecer qué características debía reunir la matriz de ocurrencia de un SEA para que su resolución en paralelo sea provechosa. En primer lugar, se puede apreciar que el nivel de ganancia obtenido al paralelizar la resolución de un SEA utilizando nuestra técnica, es dependiente del grado de ralidad y número de subsistemas arrojados por la descomposición.

La ralidad del SEA (porcentaje de elementos no nulos) está relacionada con la cantidad de subsistemas obtenidos por la descomposición. Si bien no es una relación directa, los SEAs ralos son más propensos a obtener un mayor número de subsistemas y por ende una descomposición de granularidad más fina.

Asimismo, la cantidad de subsistemas generados y el tamaño de los mismos constituyen medidas importantes para el buen desempeño del algoritmo. En general, para un tamaño de

matriz fijo, cuanta más cantidad haya y más chicos sean los subsistemas, la posibilidad de paralelismo aumenta y el tiempo de cómputo requerido para resolver cada subsistema disminuye considerablemente. Sin embargo, una cantidad excesiva de subsistemas de muy bajo orden puede ir en detrimento del desempeño. En tal sentido, cabe destacar que existe un punto de equilibrio en lo referido a la granularidad ideal de descomposición para un SEA, la cual depende en gran medida de las características técnicas de la red de comunicación y la capacidad de cómputo de las estaciones de trabajo.

Existen además otros factores que afectan los tiempos de resolución del SEA, que son independientes de la cantidad de subsistemas, y que están determinados por las características del DAG. Entre ellos, la *profundidad* del DAG (i.e. cantidad de niveles) nos marca de forma muy clara un tope en la cantidad de paralelismo alcanzable por el algoritmo, ya que, independientemente del número de procesadores que se tengan, la profundidad del DAG indica la cantidad mínima de subsistemas que deben resolverse en forma secuencial.

Otro rasgo distintivo de las matrices, determinado por las características del DAG, es el *promedio de las dependencias directas*, calculado como el promedio de la cantidad de dominantes que posee cada subsistema. Este número nos da una medida de la independencia de cada subsistema y, en definitiva, los subsistemas podrán empezar a resolverse antes cuanto más independientes sean. Por último, el *promedio de las dependencias indirectas* se calcula promediando, para cada uno de los subsistemas, la suma de la cantidad total de dominantes y la cantidad de subsistemas que dominan a sus dominantes y así sucesivamente, hasta llegar a los dominantes del primer nivel. Es importante destacar que las dependencias redundantes y/o transitivas son eliminadas en este análisis ya que su presencia o ausencia no incide en el comportamiento del algoritmo.

Si bien este promedio está relacionado con la media de las dependencias directas, con la ayuda de las indirectas se puede deducir, para una misma profundidad, si los distintos niveles están bien balanceados en cuanto a la cantidad de subsistemas que éstos comprenden. Esto se debe a que, cuando se calcula las dependencias indirectas, las componentes de niveles más altos tienen mayor ponderación en el promedio, ya que es más larga la cadena de dominantes que necesitan hasta llegar al nivel 1. Lo ideal, para un número grande de procesadores y para una determinada profundidad, es que la cantidad de subsistemas por niveles sea lo más pareja posible. De esta manera, se procura que todos los procesadores tengan poco tiempo de espera ociosa y un balance de carga equilibrado.

En la figura 8 y Tabla 1 se muestra un ejemplo de un DAG al que se le calculan las métricas antes mencionadas. En este caso el DAG es de *profundidad* 4 ya que ese es su número de niveles. El nivel 1 queda integrado por los nodos S_1, S_2 y S_3, el nivel 2 por S_4, S_5 y S_6, el nivel 3 por S_7 y S_8, y finalmente el nivel 4 por S_9 y S_{10}. Notar que para el cálculo del promedio de las dependencias se excluye la arista que va de S_3 a S_{10} por considerarse una arista redundante. Luego, la cantidad de dependencias que restan son 9. Este valor dividido por la cantidad de subsistemas nos da el *promedio de las dependencias directas*.

Para el caso del *promedio de las dependencias indirectas*, por cada subsistema se calcula la cantidad de dependencias que necesitan ser resueltas desde el principio del algoritmo hasta el comienzo de su cómputo. Por ejemplo, el subsistema S_9 requiere que 4 dependencias sean resueltas para comenzar su ejecución. Aplicando la suma 1 + 1 + 2 + 4 + 3 + 4 + 4,

correspondiente a la cantidad de dependencias indirectas de cada uno y dividiendo nuevamente por los 10 subsistemas, se obtiene este último promedio.

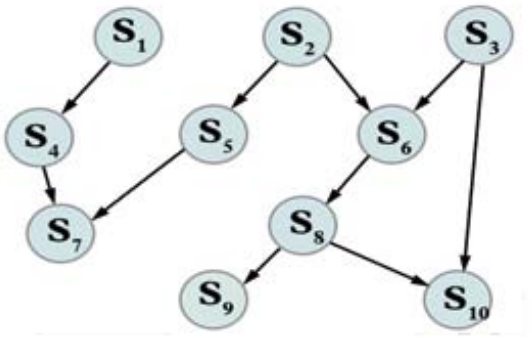

Figura 8: Análisis de un DAG.

Profundidad	4
Prom. dep. directas	0,9
Prom. dep. indirectas	1,9

Tabla 1: Características del DAG de la Figura 8.

Teniendo en cuenta los aspectos anteriores, se implementó un generador de matrices ralas que permite crear casos de estudios con distintas características a fin de efectuar una evaluación criteriosa del algoritmo paralelo. Para esto, el generador recibe como datos de entrada una secuencia de parámetros que define el tipo de matriz con el que se desea trabajar. El algoritmo chequea que los subsistemas originados por la descomposición posean solución única.

Entre los datos que el generador recibe, destacamos como los más relevantes aquellos que delimitan las dimensiones y la ralidad. Estas dos características se definen tanto para la matriz de ocurrencia original como para los subsistemas generados por la descomposición. El porcentaje de elementos no nulos en la matriz de ocurrencia define la cantidad de subsistemas finales, mientras que este porcentaje en los subsistemas marcará el número de dependencias que el DAG poseerá. El resto de los parámetros están relacionados con la matriz de coeficientes y con el vector de soluciones del SEA. Vale aclarar que en todas las matrices generadas de esta forma, existe una cuota de aleatoriedad que permite obtener diversidad para matrices de iguales características.

4.2 Análisis de los resultados

Empleando el generador descripto anteriormente, se decidió trabajar con dos tamaños de matrices, 5000×5000 y 7500×7500, clasificando los casos de estudios en dos grupos, *débilmente acoplados* y *muy débilmente acoplados*, según se sintetiza en la Tabla 2. Dadas sus características, ambos grupos representan SEAs cuya paralelización puede resultar potencialmente beneficiosa. En resumen, se generaron matrices representativas de cuatro

categorías de SEAs distintos. En la Tabla 3 puede apreciarse cuáles son los valores, en promedio, que caracterizan a estos cuatro tipos.

Tipo de Matriz	Débilmente acopladas (DA)	Muy débilmente acopladas (MDA)
Profundidad	Min 10 Max 13	Min 3 Max 5
Prom. dep. directas	Min 1,0 Max 2,0	Min 0,25 Max 0,6
Prom. dep. indirectas	Min 10, 0 Max 35,0	Min 0,25 Max 0,8

Tabla 2: Clasificación utilizada en los casos de estudio.

	D.A.	D.A.	M.D.A.	M.D.A.
Orden	5000	7500	5000	7500
Ralidad (%)	0,20	0,12	0,19	0,13
Ralidad subsistemas(%)	7,08	6,25	6,75	6,83
Cantidad de subsistemas	36,33	52,67	36,33	53,33
Tamaño promedio subsistemas	140,00	143,33	139,17	141.67
Promedio depcias. directas	1,41	1,64	0,40	0,39
Promedio depcias. indirectas	17,49	27,13	0,52	0,50
Profundidad	11,00	12,00	3,67	3,67

Tabla 3: Resumen de las características de los SEAs utilizados.

En la Tabla 4 se detallan los resultados de nuestros ensayos. Los tiempos, medidos en milisegundos, para cada grupo son los promedios obtenidos para cada clase de matrices. El trabajo experimental se realizó sobre un cluster de monoprocesadores conectados a través de una red Ethernet. Para la implementación del algoritmo se utilizó la librería de pasaje de mensajes PVM[15,16].

	D.A.	D.A.	M.D.A.	M.D.A.
Tiempo 1 worker	556,67	863,00	551,67	860,00
Tiempo 2 workers	338,67	500	318,67	499,67
Tiempo 4 workers	241,00	315,67	190,67	279,67
Tiempo 6 workers	222,67	289,00	150,33	201,33
Tiempo 8 workers	226,00	275,33	135,67	181,00
Speed-up 2 workers	1,64	1,73	1,73	1,72
Speed-up 4 workers	2,33	2,73	2,9	3,09
Speed-up 6 workers	2,54	2,99	3,74	4,27
Speed-up 8 workers	2,49	3,14	4,11	4,75

Tabla 4: Tiempos (en milisegundos) y speedups obtenidos de la resolución en paralelo.

Analizando la tabla, podemos ver que los tiempos y *speed-ups*[17] escalan mejor cuanto más grandes son las dimensiones de la matriz y que, obviamente, las matrices *muy débilmente acopladas* tienen un comportamiento superior a las *débilmente acopladas*. Estos resultados

eran de esperar, dado que cuando mayor dimensión y nivel de desacople poseen las matrices, mayor es el grado de concurrencia logrado en la etapa de resolución del sistema y esto se refleja en mejores índices de *speed-up*.

Asimismo, la Figura 9 permite una observación más detallada de los *speed-ups* obtenidos para cada cantidad de procesadores. En la misma se aprecia que el número de *workers* que optimiza la relación de *speed-up* es dependiente de las características estructurales del DAG asociado al sistema a resolver.

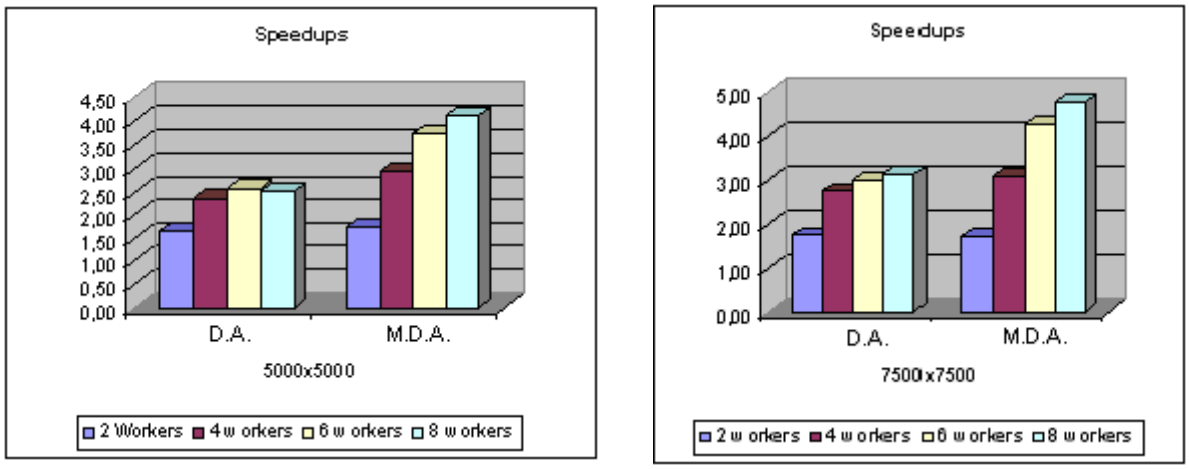

Figura 9: Escalabilidad de los resultados.

5 CONCLUSIONES

El objetivo de este trabajo ha sido el desarrollo de un nuevo método numérico que permitiera la resolución de sistemas de ecuaciones algebraicos lineales con solución única. El método numérico está orientado a sistemas de gran tamaño y con altos porcentajes de ralidad, situaciones comunes en innumerables modelos industriales y científicos. Utilizando el método de clasificación de variables y descomposición en bloques propuesto en Ponzoni *et al.*[8], se desarrolló una implementación paralela que es aplicada sobre los bloques obtenidos en la descomposición, a fin de resolver el sistema de ecuaciones.

Los resultados obtenidos fueron altamente satisfactorios en términos de *speed-up*, eficiencia y estabilidad para matrices que presentan buenas características de descomposición. La resolución de dichos sistemas, de no utilizarse algún criterio de descomposición de dominios, se volvería inviable.

Como trabajo a futuro, existen potenciales mejoras a realizar. En primer lugar, ubicamos nuestra intención de extender el método numérico para permitir la resolución de sistemas de ecuaciones no lineales. El método de clasificación de variables utilizado nos provee el marco necesario para la aplicación de nuestra metodología al cómputo de cualquier tipo de sistema de ecuaciones algebraico.

Por otra parte, resta también realizar un estudio más detallado de la estabilidad numérica de los resultados. Los cómputos en cada uno de los subsistemas están sujetos a los cálculos previos

hechos en sus subsistemas dominantes. Si los subsistemas están mal condicionados, pequeñas variaciones en la entrada nos significan discrepancias importantes en los resultados. La precisión en los resultados es un tema de suma importancia en este tipo de cómputos a gran escala.

Por último, resultaría valioso avanzar en la clasificación de los casos de estudio, a fin de identificar que otros parámetros de tipificación de las matrices, además de los ya considerados, poseen incidencia en el grado de concurrencia que se puede obtener con nuestra metodología. En tal sentido, la definición de métricas más precisas basadas en estos parámetros permitiría evaluar analíticamente la potencial ganancia de emplear nuestro metodología, antes de su efectiva aplicación sobre un nuevo caso de estudio. De esto modo, se contaría con una herramienta efectiva para establecer los alcances de nuestra propuesta.

6 AGRADECIMIENTOS

Los autores desean expresar su agradecimiento a la Agencia Nacional de Promoción Científica y Tecnológica de la Argentina por la subvención otorgada en el marco del Programa de Modernización Tecnológica, Contrato de Préstamo BID 1201/OC-AR, al Proyecto de Investigación Científica y Tecnológica Nº11-12778, denominado "Procesamiento paralelo distribuido aplicado a ingeniería de procesos", aprobado por Resolución ANPCYT Nº117/2003.

También queremos agradecer a la Secretaría de Ciencia y Tecnología de la Universidad Nacional del Sur por la subvención otorgada al Proyecto de Grupos de Investigación (PGI 24/N012): "Computación Científica Aplicada al Diseño de Instrumentación".

7 REFERENCIAS

[1] D.P. Bertsekas y J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific, (1997).

[2] P. Brinch Hansen, *Studies in Computational Science: Parallel Programming Paradigms*, Prentice Hall, (1995).

[3] P. Chaudhuri, *Parallel Algorithms: Design and Analysis*, Prentice Hall, (1992).

[4] R. Buyya, *High Performance Cluster Computing: Architectures and Systems,* Vol. I, Prentice Hall, (1999).

[5] R. Buyya, *High Performance Cluster Computing: Programming and Applications* Vol. II, Prentice Hall, (1999).

[6] A. Chalmer y J. Tidmus, *Practical Parallel Processing*, Thomson Computer Press, (1996).

[7] I. Ponzoni, "Aplicación de Teoría de Grafos al Desarrollo de Algoritmos para Clasificación de Variables", *Tesis deDoctor en Ciencias de la Computación*, UNS, B. Blanca., Argentina, (2001).

[8] I. Ponzoni, M.C. Sánchez, N.B. Brignole. "A Direct Method for Structural Observability Analysis", *Ind. Eng. Chem. Res.*, **43**, 577-588, (2004).

[9] J.A. Romagnoli y M.C. Sánchez, *Data Processing and Reconciliation for Chemical Process Operations*, Academic Press Inc, (1999).

[10] M. Karpinski y W. Rytter, *Fast Parallel Algorithms for Graph Matching Problems,* Oxford University Press, (1998).

[11] R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms", *SIAM J. Comput*., **1** 146-160, (1972).
[12] F. Harary, *Graph Theory*. Addison-Wesley, p. 200, (1994).
[13] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press Inc, (1984).
[14] B. Wilkinson y M. Allen, *Parallel Programming*, Prentice Hall, (1999).
[15] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek y V. Sunderam., *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, (1994).
[16] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing", *Concurrency-Pract. Ex.*, **2**, 315-339, (1990).
[17] L. Crowl, "How to Measure, Present and Compare parallel Performance", *Parallel Distrib. Tech.*, **2**, 9-25, (1994).