Mecánica Computacional Vol 20. pp. 468 — 473
Compilado por Fernando G. Flores
Córdoba, 30 de octubre - 2 de noviembre de 2001

ALGORITMOS PARALELO/DISTRIBUIDOS PARA BÚSQUEDAS EN PROFUNDIDAD SOBRE GRAFOS

Federico Fapitalle[1], Gustavo E. Vázquez[1], Ignacio Ponzoni[1] y Nélida B. Brignole[1,2]

[1]Departamento de Ciencias de la Computación
Universidad Nacional del Sur, Av. Alem 1253, 8000 Bahía Blanca, Argentina

[2]Planta Piloto de Ingeniería Química
Complejo CRIBABB, Camino La Carrindanga, km. 7, CC 717
CONICET-UNS, 8000 Bahía Blanca, Argentina

RESUMEN

En este artículo se presenta un algoritmo paralelo distribuido descentralizado para realizar búsquedas en profundidad de caminos sobre grafos. El método se basa en un nueva arquitectura paralelo distribuida, propuesta en este trabajo, en la cual se distribuyen las tareas de cómputo sobre tres tipos de nodos de procesamiento: el Master, los Supervisors y los Workers. Básicamente, el Master organiza la distribución de los distintos subespacios de búsqueda entre los Supervisores. Cada Supervisor encomienda la exploración de los subcaminos correspondientes a su subespacio a diferentes Workers que están a su cargo. Cada Worker efectúa la exploración de una parte del espacio de búsqueda y le envía a su Supervisor cada uno de los subcaminos hallados. Por último, el Supervisor se encarga de recombinar sus subcaminos con los subcaminos almacenados por otros Supervisores. El nuevo algoritmo fue implementado en lenguaje C utilizando la librería de pasaje de mensajes PVM y su desempeño fue evaluado en términos de eficiencia y speed-up.

ABSTRACT

A descentralized parallel-distributed algorithm to carry out depth-first searches along graphs is presented. The method is based on a new parallel-distributed architecture that is proposed in this article. In this formulation the computing tasks are distributed among three kinds of nodes: the Master, the Supervisors and the Workers. The Master organizes the distribution of the various search subspaces among the Supervisors. In turn, each Supervisor delegates the exploration of the subpaths inside the assigned subspace to the Workers under its control. So, each Worker explores a given part of the search space, sending its Supervisor information about the subpaths it could find. Finally, the Supervisor has to recombine its own subpaths with those stored by the other Supervisors. The new algorithm was implemented in C using the PVM message-passage library and its performance was evaluated in terms of speed-up and efficiency.

INTRODUCCIÓN

Los grafos son estructuras matemáticas esenciales para resolver una amplia variedad de problemas que surgen en distintas disciplinas. Los modelos generados mediante este enfoque se usan, por citar

algunos ejemplos, en: análisis sintáctico (parsing) de lenguajes de programación en ciencias de la computación, predicción de isómeros estructurales en química, diseño de instrumentación de plantas en ingeniería de procesos, planeamiento y distribución en investigación operativa, análisis de cadenas de Markov en teoría de probabilidad y diseño de chips VLSI (Very Large Scale Integration) en ingeniería eléctrica [1].

Para algunos de estos problemas modelados con grafos, la mejor solución se obtiene mediante la exploración del espacio de búsqueda generado a partir de recorrer los distintos caminos contenidos en el grafo [2]. Unas de las técnicas más frecuentemente empleadas para realizar esta tarea son las búsquedas en profundidad (*depth-first search*). Estos procedimientos de naturaleza combinatorial exploran un espacio de búsqueda arboreo que crece exponencialmente con la cantidad de nodos y aristas del grafo. Debido a esto, la aplicación de estos métodos sobre problemas grandes pueden requerir una gran capacidad de cómputo. En estos casos, una alternativa interesante para reducir los tiempos de ejecución consiste en descomponer el problema a fin de resolverlo utilizando procesamiento paralelo.

Existen varios algoritmos paralelos para búsquedas en profundidad propuestos para arquitecturas multiprocesador con memoria compartida [1],[3]. Estas técnicas, basadas en la generación de forestaciones, sólo pueden ser implementadas sobre computadoras multiprocesador muy específicas. Dado los costos, características de escalabilidad y falta de estándares de las arquitecturas MIMD (*Multiple Instruction Multiple Data*), este tipo de técnicas se ven acotadas en su rango de aplicabilidad y portabilidad. En contraste, un algoritmo diseñado para esquemas de procesamiento paralelo distribuido resultaría más beneficioso debido a la gran disponibilidad y bajo costo de infraestructura que tienen actualmente las redes de estaciones de trabajo.

Dentro del ámbito del procesamiento paralelo distribuido son escasos los trabajos reportados referidos a búsquedas en profundidad. En particular, Vázquez *et al.* [4] presentan una técnica paralelo distribuida centralizada la cual efectúa una descomposición del espacio de búsqueda en subárboles de exploración, la cual fue posteriormente aplicada en problemas de instrumentación de plantas industriales [5]. Si bien este método logró una importante reducción en los tiempos de cómputo, la política de descomposición de dominios utilizada no explota todo el potencial de paralelización que poseen las búsquedas en profundidad. En particular, no se factoriza la exploración de subcaminos repetidos.

En este artículo se propone un nuevo algoritmo paralelo distribuido para búsquedas en profundidad que sigue una política de trabajo descentralizada. El principal objetivo de esta propuesta es evitar la exploración repetida de subcaminos a fin de optimizar el recorrido del grafo.

BÚSQUEDAS EN PROFUNDIDAD PARALELO/DISTRIBUIDAS DESCENTRALIZADAS

El método se basa en un nuevo paradigma de paralelización propuesto en este trabajo, el cual distribuye las tareas de cómputo sobre tres tipos de nodos de procesamiento: el Master, los Supervisors y los Workers. Básicamente, el Master organiza la distribución de los distintos subespacios de búsqueda entre los Supervisors. Cada Supervisor encomienda la exploración de los subcaminos correspondientes a su subespacio a un Worker que tiene a su cargo. El Worker efectúa el recorrido del subespacio que le fue asignado, y envía a su Supervisor cada uno de los subcaminos hallados. Por último, el Supervisor se encarga de recombinar los subcaminos obtenido de su Worker con los subcaminos almacenados por otros Supervisors. Finalmente, cada camino que contenga una solución es enviado al Master. Desde un punto de vista general, nuestra propuesta apunta a una descentralización en la "construcción" de los caminos, incluyendo actividades de recombinación de subcaminos. La verificación de si un camino es, o no, solución del problema también está descentralizado dado que es llevado adelantes por los Supervisors.

Descripción del método

La arquitectura del método propuesto introduce un nivel intermedio a la jerarquía Master-Worker, llamado el nivel Supervisor. Cada tarea Worker del sistema está subordinada a una tarea Supervisor.

La comunicación de la tarea Worker es bidireccional, y se limita a la tarea Supervisor asociada a ella. La tarea Master tiene comunicación sólo con las tareas Supervisor. Por último, cada tarea Supervisor tiene un canal de comunicación con:

- la tarea Master,
- una tarea Worker subordinada a ella,
- cualquier otra tarea en el mismo nivel jerárquico, i.e., cualquier tarea Supervisor.

Esta arquitectura se muestra en la fig. 1.

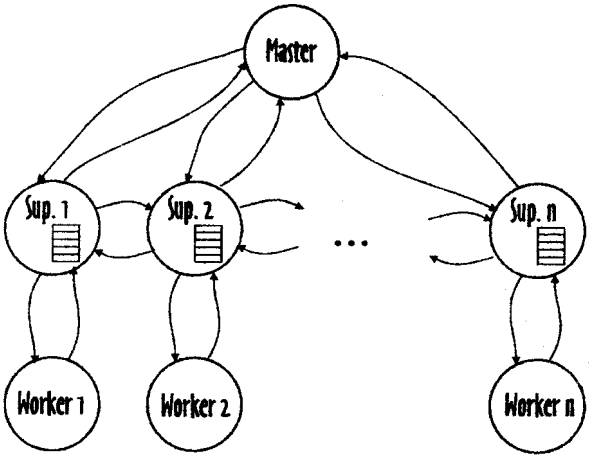

Figura 1. Arquitectura del sistema

Cada subcamino encontrado por la tarea Worker es enviado a la tarea Supervisor asociada a ella, donde se almacena en un registro interno. Este subcamino será la primer mitad de un camino completo, que se formará cuando el Supervisor que lo almacena lo envíe a la tarea Supervisor que almacena la segunda mitad del camino, y éste concatene ambas partes, formando un camino de la longitud deseada. Esto es, cada subcamino de hoja x del registro es enviado a la tarea que supervisó la exploración del subárbol con raíz x, dado que dicha tarea tiene almacenado todo subcamino que comienza con el nodo x. Una vez que se ha obtenido un camino completo, se procede a verificar su viabilidad como solución.

La tarea Master administra una cola de prioridad para los nodos que aún no han sido explorados. Inicialmente, en ella se encuentran todos los nodos del grafo, y a todos se les asigna la misma prioridad. A medida que el cómputo avanza, la tarea Master recibirá mesajes, de una tarea Supervisor, consultando el identificador de tarea (*task id*) del Supervisor que tiene a cargo la exploración de determinado nodo. Si este nodo no ha sido enviado para exploración aún, se le asigna una prioridad mayor en la cola. De esta forma, cuando un nodo es muy solicitado, tendrá más chances de ser el próximo nodo que se explore.

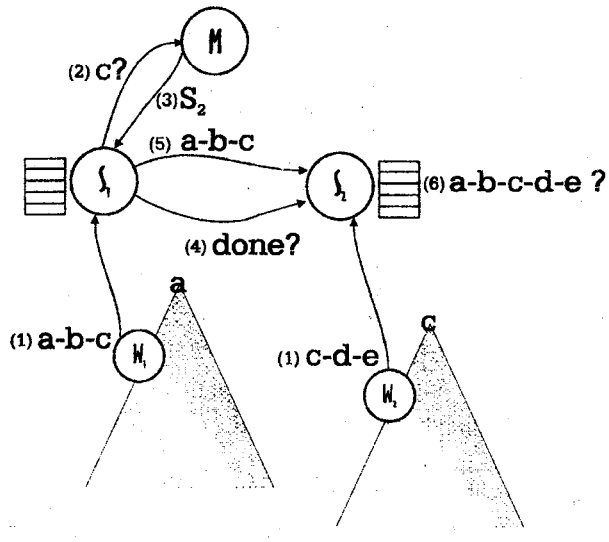

Figura 2. Construcción de caminos

Protocolo de finalización

La terminación de la ejecución de este método ocurre en dos situaciones: se finaliza con éxito si se encuentra un camino que corresponde a una solución del problema, o se finaliza con fracaso en caso contrario.

Cuando se finaliza con éxito, la tarea Master envía un mensaje de finalización de ejecución. Cada tarea Supervisor recibe el mensaje de la tarea Master, y lo propaga hacia la tarea Worker subordinada, y finaliza su propia ejecución. La tarea Worker, a su vez, cuando recibe este mensaje, termina su ejecución.

Cuando la finalización es con fracaso, la terminación se lleva a cabo con un protocolo de finalización de dos etapas. En la primera, cuando no hay más nodos a procesar, cada pedido de nuevo nodo de la tarea Worker obtiene como respuesta un mensaje de terminación de ejecución. La primera fase entonces finaliza cuando todas las tareas Worker han terminado. En la segunda etapa, en el momento que una tarea Supervisor no tiene más caminos en su registro, envía una intención de finalización al Master. Cuando el Master obtiene la intención de finalización de todas las tareas Supervisor, envía a cada una de ellas un permiso de terminación y finaliza su propia ejecución.

Implementación experimental del nuevo algoritmo

Se desarrolló la implementación de una versión prototipo del algoritmo propuesto, con el propósito de realizar pruebas experimentales. El marco de trabajo del desarrollo fué una red local Ethernet de estaciones de trabajo (PCs), junto con el sistema PVM (*Parallel Virtual Machine*) [6].

El sistema PVM provee una completa interface de programación de aplicaciones (API), bajo el paradigma de pasaje de mensajes. Además, PVM permite desarrollar aplicaciones en un conjunto de estaciones de trabajo, potencialmente heterogéneas, interconectadas en una red, que luce lógicamente al usuario como una única computadora paralela. De los lenguajes de programación que soporta actualmente PVM, C, C++ y FORTRAN, el lenguaje C fue el seleccionado para implementar en forma completa este prototipo.

ANÁLISIS DE DESEMPEÑO

Se realizaron pruebas empíricas de desempeño del algoritmo, para las cuales se modificó el mismo con el fin de que ningún camino completo fuera considerado solución. Así se fuerza a que las ejecuciones finalicen utilizando el protocolo de terminación de dos fases. De este modo, toda ejecución del algoritmo examina en forma exhaustiva todos los posibles caminos del grafo. Estas modificaciones fueron realizadas con el propósito de comparar en la forma más justa posible, dos ejecuciones sobre el mismo grafo.

Los casos de estudio fueron generados aleatoriamente, clasificando los grafos por su cantidad de nodos y densidad. Dado que la meta es trabajar con grafos con matrices de incidencia rala, se construyeron ejemplos con densidad muy baja. Asimismo, para cada ejemplo se efectuaron corridas sobre 1, 2, 4 y 6 nodos de procesamiento. Los resultados obtenidos para cada cantidad de procesadores se muestran en las tablas I a IV.

Tabla I. Tiempos de Ejecución para 1 procesador

	Cantidad de Nodos				
Densidad	30	40	50	60	70
5%	0:00.62	0:01.03	0:02.99	0:03.18	0:08.21
10%	0:01.08	0:02.71	0:10.80	0:26.78	0:48.05

Tabla II. Tiempos de Ejecución para 2 procesadores

	Cantidad de Nodos				
Densidad	30	40	50	60	70
5%	0:01.26	0:01.91	0:04.98	0:17.19	0:38.08
10%	0:05.68	0:02.77	0:12.34	0:21.51	0:25.90

Tabla III. Tiempos de Ejecución para 4 procesadores

	Cantidad de Nodos				
Densidad	30	40	50	60	70
5%	0:03.61	0:04.75	0:07.94	0:16.19	0:30.14
10%	0:01.88	0:10.38	0:16.76	0:37.13	0:41.83

Tabla IV. Tiempos de Ejecución para 6 procesadores

	Cantidad de Nodos				
Densidad	30	40	50	60	70
5%	0:02.95	0:07.22	0:13.71	0:26.55	0:50.92
10%	0:04.89	0:11.50	0:22.70	0:54.82	01:09.8

En total se reportan corridas para grafos de cinco tamaños distintos, con cantidad de nodos variando entre 30 y 70, y de densidades diferentes, 5 y 10% . Los tiempos mostrados en las tablas se presentan en minutos (minutos:segundos.centésimas).

Del análisis de los resultados se observa que los tiempos de cómputo crecen en forma geométrica con la cantidad de nodos y densidad del grafo. Esta característica resulta destacable dado que en los recorridos en profundidad los espacios de búsqueda crecen de manera exponencial. La justificación de este menor incremento en los tiempos es que nuestro algoritmo evita el revisitado de subcaminos durante la exploración. Además, dado que la búsqueda de caminos de longitud n se reduce a hallar

subcaminos de longitud $n/2+1$, los espacios de exploración se reducen en forma logarítmica.

No obstante estos buenos resultados, también se aprecia que esta primera implementación experimental sufre de una significativo sobrecarga en comunicación. Esto se puede mejorar mediante un refinamiento de los protocolos establecidos para el envío y recepción de mensajes entre los distintos niveles de la arquitectura distribuida.

CONCLUSIONES

Se presentó un nuevo algoritmo paralelo distribuido para realizar búsquedas en profundidad sobre grafos. El énfasis estuvo puesto en la descentralización de los recorridos, recombinación de caminos, y evitar revisitar subcaminos. Los resultados obtenidos revelaron que se logró el objetivo de disminuir el grado de crecimiento del tiempo de la búsqueda conforme se incrementa el tamaño del grafo. Se obtuvo un crecimiento de tiempos geométrico en lugar del incremento exponencial que sufre la gran mayoría de estas búsquedas.

Si bien estos primeros resultados son satisfactorios, es necesario continuar trabajando en el refinamiento de la implementación del algoritmo. Por un lado es necesario optimizar los protocolos de comunicación, para reducir la sobrecarga de mensajes. Por otra parte, la optimización de las estructuras de datos utilizadas por los Supervisores y el estudio de políticas de balance de carga entre los nodos de procesamiento pueden incrementar notoriamente el desempeño de nuestra propuesta.

REFERENCIAS

[1] **Chaudhuri, P.**, *Parallel Algorithms: Design and Analysis*, Prentice Hall, 1999.

[2] **Ponzoni, I., Sánchez, M.C., y Brignole, N.B.**, *A New Structural Algorithm for Observability Classification*, Industrial & Engineering Chemistry Research, Vol. 18, 1999, págs. 3027-3035.

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

[4] **Vázquez, G.E., Ponzoni, I., y Brignole, N.B.**, *Parallel Depth-First Search on Clusters of Workstations*, SIAM Annual Meeting, Atlanta (U.S.A.), 10 al 12 de mayo,1999, pág. 198.

[5] **Ponzoni I., Vazquez G.E., Sánchez M.C., y Brignole N.B.**, *Parallel Observability Analysis on Networks of Workstations*, Computers & Chemical Engineering. Vol 25, No.7-8, 2001, págs. 997-1002.

[6] **Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, R, Sunderam, V.**, *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.