UNIVERSIDAD NACIONAL DEL SUR

Departamento de Ciencias e Ingeniería de la Computación

Tesis de Doctor en Ciencias de la Computación

PROCESAMIENTO PARALELO DISTRIBUIDO HETEROGÉNEO APLICADO A INGENIERÍA DE PROCESOS.

Gustavo E. Vazquez

Bahía Blanca Argentina

- 2002 -

PREFACIO

Esta tesis es presentada como parte de los requisitos para optar al grado Académico de Doctor en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otras. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Departamento de Ciencias de la Computación de esta Universidad durante el período comprendido entre el 1 de agosto de 1998 y el 1 de agosto de 2002, bajo la dirección de la Dra. Nélida Beatriz Brignole, profesora adjunta del Departamento de Ciencias de la Computación de la Universidad Nacional del Sur e investigadora del CONICET.

Mi más profundo agradecimiento a la Dra. Nélida Beatriz Brignole por su invaluable colaboración y espíritu de trabajo, por la formación brindada y por su estimulo constante en la concreción de las investigaciones, los cuales resultaron fundamentales en el desarrollo y concreción de esta tesis.

Al Departamento de Ciencias de la Computación mi agradecimiento por haberme facilitado los medios para realizar esta tarea. A todo su personal por su permanente cordialidad y predisposición, muy especialmente al Mg. Jorge Ardenghi por su apoyo y colaboración como codirector de beca a lo largo de todos estos años de trabajo.

Al Consejo Nacional de Investigaciones Científicas y Técnicas por la ayuda económica brindada y a la Planta Piloto de Ingeniería Química por haberme facilitado sus instalaciones y recursos y por proporcionar un ambiente de trabajo cordial.

A mi amigo y colega Dr. Ignacio Ponzoni, por amistad y compañerismo a lo largo de estos años de trabajo.

A la Dra. Susana Espinoza y al Ing. Anibal Blanco, quienes con su constante apoyo y camaradería crearon un ámbito de trabajo excepcional.

Y por último quiero expresar mi eterna gratitud a Dios por regalarme la vida, y a mis padres, quienes con su afecto y aliento hicieron posible que concretara esta tesis. A ellos mi agradecimiento por todo el cariño y apoyo brindado.

Gustavo Esteban Vazquez

Bahía Blanca, 15 de Julio de 2002

RESUMEN

El objetivo de esta tesis ha sido diseñar nuevas estrategias de procesamiento paralelo en entornos de cómputo distribuido heterogéneo para facilitar la resolución de problemas tanto estructurales como numéricos del campo de la ingeniería de procesos. Como resultado de estas investigaciones se ha logrado el desarrollo de técnicas robustas y eficientes aplicables a un amplio espectro de problemas de búsquedas en grafos y de optimización con función objetivo y restricciones no lineales.

En términos generales, es posible distinguir dos líneas de investigación para el desarrollo de algoritmos paralelos distribuidos: la paralelización de algoritmos secuenciales existentes y la creación de alternativas intrínsecamente paralelas. En el caso de problemas estructurales, se estudiaron los métodos secuenciales clásicos de búsqueda en grafos y se establecieron las limitaciones para su uso en redes de estaciones de trabajo. Sobre esta base se propuso un nuevo método de distribución semi-dinámica y se la aplicó al algoritmo GS-FLCN para análisis de observabilidad.. Por otra parte, en la línea de los algoritmos intrínsecamente paralelos se desarrolló un nuevo algoritmo de búsqueda totalmente distribuido con el objeto de aumentar la eficiencia de los recorridos para esta aplicación específica. En cuanto a los problemas numéricos, se consideraron estrategias para aplicar el paralelismo a las secciones de cómputo intensivo de algoritmos secuenciales existentes para optimización no lineal con restricciones conocidos como GRG y SQP. Asimismo se desarrolló una nueva técnica de descomposición de dominio con el objeto de ampliar el rango de aplicabilidad de un algoritmo intrínsecamente paralelo concebido originalmente para problemas sin restricciones de modo que se lo pudiera utilizar en forma eficiente para el tratamiento de

los problemas de optimización no lineal con restricciones que surgen en ingeniería de procesos.

En cuanto a las verificaciones de desempeño, se adaptaron las métricas de speed-up con el objeto de tener en cuenta la heterogeneidad de los procesadores y así poder asegurar comparaciones justas. En tal sentido, todos los nuevos algoritmos propuestos lograron un muy buen desempeño en cuanto al tiempo de ejecución en comparación con los algoritmos secuenciales correspondientes. Se analizaron casos de estudio académicos y problemas industriales reales de mediano y gran tamaño pertenecientes al área de ingeniería de procesos.

Por último, cabe destacar que los beneficios derivados de las propuestas descriptas en esta tesis doctoral no se limitan al ámbito de ingeniería de procesos. Tanto las búsquedas en grafos como los problemas de optimización surgen naturalmente en otras ramas de la ingeniería así como también en biología, economía, etc. No solo es factible emplear en otras disciplinas los mismos algoritmos sino también aplicar la filosofía subyacente, tal como el criterio de descomposición de dominio o la distribución semi-dinámica de carga.

TABLA DE CONTENIDOS

LISTA DE PUBLICACIONES

En Revistas Científicas Internacionales con Referato.

Vazquez G.E., Diaz S., Brignole N.B., Bandoni J.A. *"Optimization Of Industrial Problems Using Parallel Processing Under Distributed Environments"* Chemical Engineering Communications, 189, 643-657, 2002. ISSN: 0098-6445. Editorial Taylor&Francis.

Ponzoni I., Vazquez G.E., Sánchez M.C., Brignole N.B. *"Parallel Observability Analysis on Networks of Workstations"*, Comp. & Chem. Eng., 25, 997-1002, 2000. ISSN: 0098-1354. Editorial Elsevier.

Vazquez G. E., Ponzoni I, Sánchez M., Brignole N. "*ModGen: A Model Generator for Instrumentation Analysis.*" Advances in Engineering Software, 32, 1, 2001, 37-48, 2001. ISSN: 0965-9978. Editorial Elsevier.

Vazquez G.E., Rainoldi R., Brignole N.B. *"Non-Linear Constrained GRG Optimization under Heterogeneous Parallel-distributed Computing Environments"*, Computer-Aided Chemical Engineering, 8, 127-132, 2000. ISBN: 0-444-50520-2. Editorial Elsevier.

Vazquez, G. E., Brignole N.B. *"Parallel NLP Strategies using PVM on Heterogeneous Distributed Environments.*" Lecture Notes in Computer Science: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 1697, 533-540, 1999. ISSN: 0302-9743. Editorial Springer.

En Congresos Internacionales con Comité de Revisión.

Vazquez, G.E., Brignole, N.B. "*A Sensitivity-Analysis Approach for Domain Partitioning in Multisplitting Algorithms.*" Mang, H. A., Rammerstorfer, F. G., and Eberhardsteiner, J. Austria. Fifth World Congress on Computational Mechanics. (2002).

Diaz G.A., Caverzán E., Vazquez G.E., De Beistegui R., Ponzoni I., Fillotrani P., Brignole N.B., Brignole E.A. *"An Object-Oriented Software System for Sequential-Modular Process Modelling and Simulation"*. ENPROMER III, 3er Congreso de Ingeniería de Procesos del Mercosur, Santa Fe, Argentina, Sept. 2001.

Caverzán E., Diaz G.A., Ponzoni I., Vazquez G.E., De Beistegui R., Brignole N.B., Fillotrani P., Brignole E.A. *"A System for Process Plant Simulation"*. ICIE 2001, Buenos Aires, Abril de 2001.

Ponzoni I., Vazquez G.E., Sánchez M.C., Brignole N.B. *"A Computer-Aided DSS for Observability Analysis"*. 4th World Multi-Conference on Circuits, Systems, Communications and Computers CSCC 2000, Atenas, Grecia, 9-16 Julio 2000. Publicado en "Signal Processing, Communications and Computer Science" (ISBN = 960-8052-18-1) Editorial WSES, 2000.

Ponzoni I., Vazquez G. E., Brignole N.B., "*A Parallel Arlgorithm for Observability Analysis on Networks of Workstations*". AIChE 1999 Annual Meeting, Dallas, Estados Unidos. Octubre de 1999. (Publicado en el CD-ROM del Congreso)

Rainoldi R., Vazquez G.E., Brignole N.B., "*Algoritmo GRG para Optimización No Lineal Paralela-Distribuida Heterogénea*", ENPROMER II, Florianápolis, Brasil. 1 al 3 de Septiembre de 1999. Trabajo publicado en CD-ROM del Congreso, 1999.

Vazquez G. E., Ponzoni I, Brignole N.B., "*Parallel Depth-First Search on clusters of Workstations*". SIAM Annual Meeting, Atlanta, Estados Unidos. 12 al 15 de Mayo de 1999. Resumen publicado en los Anales del Congreso, 198, 1999.

Vazquez G. E., Brignole N.B., "*Parallel Distributed Optimization for Chemical Engineering Applications*". SIAM Annual Meeting, Atlanta, Estados Unidos. 12 al 15 de Mayo de 1999. Resumen publicado en los Anales del Congreso, 198, 1999.

Vazquez G. E., Ponzoni I, Sánchez M., Brignole N.B., "*ModGen: A Model Generator for Instrumentation Analysis. Industrial Application using New Observability Technique*". AIChE 1998 Annual Meeting, Miami, Estados Unidos. Noviembre de 1998. (Publicado en el CD-ROM del Congreso)

Vazquez G. E., Díaz S., Brignole N.B., Bandoni A.J., "*Optimization of Industrial Problems Using Parallel Processing Under Distributed Environments*". AIChE 1998 Annual Meeting, Miami, Estados Unidos. Noviembre de 1998. (Publicado en el CD-ROM del Congreso)

Vazquez G. E., Sánchez M., Brignole N. B., "*Implementación de un Generador de Modelos de Plantas Químicas*", ENPROMER'97 *(1st Congress on Process Engineering for the MERCOSUR)*, Bahía Blanca, Argentina. 1 al 4 de Septiembre de 1997. Publicado en los Anales del Congreso, 11-12, 1997.

En Congresos Nacionales con Comité de Revisión.

Maller P.A., Ponzoni I., Vazquez G.E., Gallard R. "*Sistemas de Lindenmayer*", CACIC (*Congreso Argentino de Ciencias de la Computación*), Tandil, Argentina, Octubre de 1999. Trabajo aceptado para su presentación en el Congreso.

Vazquez G. E., Brignole N. B., "*Empleo de modelos reducidos y procesamiento distribuido en optimización dinámica*". Mecánica Computacional, Vol. XVIII, 695-703, 1997.

Vazquez G. E., Ugrin P. E., Brignole N. B., "*Evaluación de Técnicas de Reducción de Ecuaciones Diferenciales Ordinarias*". Mecánica Computacional, Vol. XVI, 361-371, 1996.

Maller P.A., Ponzoni I., Vazquez G.E., "Sistemas de Lindenmayer", *XXIV JAIIO (Jornadas Argentinas de Informática e Investigación Operativa)*, SADIO, Capítulo Estudiantil, Buenos Aires, Argentina. 7 al 9 de agosto de 1995.

CAPÍTULO 1: INTRODUCCIÓN

Los enfoques rigurosos para el tratamiento de problemas complejos en ciencia e ingeniería a menudo conducen a formulaciones cuyo procesamiento eficiente requiere mayor poder de cómputo del que una computadora con un único procesador podría proveer. Una forma de superar esta limitación es mejorar la velocidad de los procesadores y componentes de manera que puedan ofrecer la capacidad de cómputo requerida. Si bien esto es posible, las mejoras alcanzables en el diseño de hardware están restringidas por la velocidad de la luz, las leyes termodinámicas y un gran costo de inversión asociado al diseño y proceso de fabricación de nuevos procesadores.

Una solución alternativa es el uso de múltiples procesadores interconectados que permitan coordinar sus esfuerzos computacionales para resolver un mismo problema. Este concepto constituye la base de los mecanismos del ***procesamiento paralelo***. Naturalmente, para obtener beneficios concretos es necesario asimismo organizar las tareas en forma eficiente, es decir efectuar un diseño de software adecuado.

A partir de la década del 80 surgieron las ***supercomputadoras***, máquinas paralelas dedicadas al cómputo intensivo tales como las famosas Cray, SGI T3E, IBM SP2 y Connection Machine CM-series. Sin embargo, desde sus inicios, el uso efectivo de esta tecnología sólo ha estado disponible para las personas o instituciones que tienen acceso a la misma. Estas arquitecturas son costosas pues su hardware y software normalmente constituyen diseños exclusivos no estándar (proprietary hardware/software). En vista de estas desventajas, desde los comienzos de la década del 90 existe una tendencia creciente

para reemplazarlas por clusters conformados por unidades de cómputo de propósito general. Un ***cluster*** es un conjunto de estaciones de trabajo cualesquiera que se comunican entre sí mediante una red de datos de área local (Local Area Network – LAN).

La configuración distribuida tiene éxito debido a la creciente disponibilidad tanto de estaciones de trabajo como de redes locales de comunicación de datos de alto desempeño. En la práctica los clusters resultan mucho más viables que las supercomputadoras por su bajo costo de inversión inicial y la fácil escalabilidad de los sistemas. Además, otro factor importante que contribuye a facilitar el empleo de este enfoque es la estandarización de muchas de las herramientas usadas por las aplicaciones paralelas, lográndose así un entorno de programación más práctico, independiente de la arquitectura de cada estación de trabajo y/o de la red de comunicación de datos. Ejemplos de estos estándares son las librerías de pasaje de mensajes PVM (Geist y col., 1994), MPI (Pacheco, 1996) y el lenguaje HPF (High Performace Fortran) (Koelbel y Zosel, 1993).

1.1. EVOLUCIÓN DE LA COMPUTACIÓN DISTRIBUIDA

Tradicionalmente, las estaciones de trabajo empleadas en ciencia e industria eran computadoras costosas basadas en procesadores RISC (Reduced Instruction Set Computing) que usaban el sistema operativo UNIX. Por otra parte, la función principal de las PC (siglas con que se conoce a las computadoras que utilizan procesadores Intel) se limitaba a los ámbitos administrativos. Sin embargo, durante los últimos 10 años, la diferencia de rendimiento entre estas plataformas se ha reducido considerablemente. Esto puede ser atribuido a la introducción de la serie de procesadores Pentium de Intel y

de los sistemas operativos Linux y Windows NT en la década de 1990. Naturalmente, esta convergencia de tecnologías ha motivado el interés en utilizar PCs y / o computadoras basadas en procesadores RISC en forma cooperativa como un recurso poderoso y económico para realizar procesamiento paralelo. Se ha comprobado estadísticamente que el poder de cómputo de las estaciones de trabajo se duplica cada 18-24 meses. Se espera que estas mejoras continúen en los años próximos, generando así un mercado masivo de nuevas computadoras de alto desempeño. Esto permite predecir que el procesamiento paralelo distribuido será una alternativa válida para resolver una gama cada vez más amplia de problemas de cómputo intensivo.

Sin embargo, disponer de nodos de cómputo veloces en un cluster no asegura que la ejecución de un algoritmo paralelo sea eficiente. Resolver un problema de manera cooperativa implica que esos nodos deben comunicarse, tanto para transmitir resultados intermedios como para realizar tareas de coordinación. Por lo tanto, la red de transmisión de datos también influye en la eficiencia de un cluster. En este sentido existen dos factores fundamentales que establecen su calidad: el ancho de banda en las comunicaciones y la latencia en la transmisión de la información. El ***ancho de banda*** se refiere a la cantidad de información que puede transmitirse por unidad de tiempo, mientras que la ***latencia*** mide la cantidad de tiempo que necesita un dato para ser transmitido desde un nodo del cluster hacia otro. De la misma manera que las computadoras, las redes de comunicación de datos han evolucionado sostenidamente en los últimos 15 años. Originalmente, su función era proveer a los usuarios servicios de red básicos (compartir impresoras, archivos de datos, etc.). Sin embargo, la demanda de las nuevas aplicaciones hizo que las redes de transmisión mejoraran sus prestaciones. Por ejemplo, la transmisión de información multimedia (audio, video) requiere un gran ancho de banda, mientras que las aplicaciones de videoconferencia, telefonía o acceso a

computadoras remotas necesitan latencia mínima para que su uso en tiempo real sea fluido.

Todos estos avances desarrollados en las redes de comunicación no sólo permiten un uso efectivo de los cluster para resolver problemas de cómputo intensivo, sino también pueden extender su alcance a nuevas aplicaciones. Por ejemplo, utilizar la memoria RAM de las estaciones de trabajo como un cache suplementario, permitiendo mejorar notablemente el desempeño de la memoria virtual y del sistema de archivos. Otra aplicación es la implementación de un sistema RAID (Redundant Array of Inexpensive Disks) por software, utilizando los discos de las estaciones de trabajo para montar un sistema de archivos RAID económico y escalable usando la red local como soporte.

1.2. VENTAJAS DEL PROCESAMIENTO PARALELO DISTRIBUIDO

Los clusters de estaciones de trabajo ofrecen las siguientes ventajas:

- Los clusters de estaciones de trabajo son más fáciles de integrar en redes locales existentes que las computadoras paralelas.
- Pueden usarse nodos de cómputo no dedicados.
- Las redes de estaciones de trabajo son más económicas y altamente disponibles comparadas con las plataformas de supercomputadoras.
- Los clusters pueden crecer fácilmente; las capacidades de un nodo pueden ser incrementadas agregando nuevos periféricos o reemplazando su procesador.

1.3. EL PROCESAMIENTO PARALELO EN INGENIERÍA DE PROCESOS

En el área de ***ingeniería de procesos*** muy frecuentemente se requiere resolver con precisión problemas industriales cuyos modelos constituyen casos de estudio complejos y/o de gran tamaño. En este marco, el procesamiento paralelo constituye una alternativa efectiva que recientemente ha comenzado a explotarse con vistas a lograr mayor eficiencia. Este potencial se aprecia significativamente en aplicaciones que requieren muchas horas de cómputo o que exceden los límites de tiempo prácticos cuando son resueltos con una computadora de un solo procesador. En esta rama de la ingeniería - que abarca problemas de simulación estacionaria y dinámica, síntesis, diseño, optimización, control e instrumentación de procesos - puede observarse una amplia variedad de formas de emplear el paralelismo. Una gran cantidad de aplicaciones pueden catalogarse como problemas de cálculo numérico, mientras que otras involucran la paralelización de información estructural. Asimismo, el desarrollo de los algoritmos paralelos puede efectuarse o bien en forma directa (modo implícito), generando automáticamente código paralelo a partir de los algoritmos secuenciales mediante un compilador adecuado, o bien con un enfoque metodológico (modo explícito), concibiendo algoritmos paralelos especializados.

En algunos casos, es posible idear una estrategia de paralelización a partir de la observación criteriosa del problema físico a resolver, es decir explotando la naturaleza "separable" del problema original. Por ejemplo, Chimowitz y Bielinis (1987) emplearon una estrategia modular simultánea para simulación de plantas de procesos dividiendo el flowsheet en zonas que luego se resolvieron en paralelo como bloques independientes entre sí. Para lograr compatibilidad entre las variables de entrada/salida de cada bloque

resultó necesario iterar hasta convergencia. Este enfoque presenta desventajas prácticas pues es altamente dependiente de la instancia del problema, lo cual torna difícil garantizar una distribución uniforme del esfuerzo computacional.

La estrategia más ampliamente usada consiste en representar matemáticamente el fenómeno físico de interés, para luego abordar la paralelización de la técnica elegida para resolver el modelo. Según la naturaleza del problema, la resolución puede involucrar módulos de cálculo numérico y/o manejo de información estructural.

En el ámbito de los ***problemas de cálculo***, la mayoría de las veces se trabaja sobre los métodos numéricos generales que constituyen el corazón del problema, resultando la paralelización independiente de la instancia a resolver. La opción de granularidad más fina consiste en implementar los métodos numéricos utilizando librerías, tales como Scalapack (Blackford y col., 1997), que proveen versiones paralelas de las operaciones básicas del álgebra matricial. Por ejemplo, existen implementaciones de métodos de cálculo de autovalores para matrices simétricas y hermíticas que emplean Scalapack (Scalapack, 2002). La principal limitación de este enfoque es que las librerías de este tipo están diseñadas especialmente para computadoras paralelas, siendo conveniente su uso en clusters sólo cuando los procesadores son similares en arquitectura y desempeño y la red de comunicación de datos es suficientemente veloz.

Otros autores aprovechan el paralelismo a un nivel más alto. En el campo de la resolución de sistemas de ecuaciones algebraicas lineales, Larsen y Madsen (1999) presentaron un esquema que paraleliza los métodos iterativos clásicos de Jacobi y Gauss-Seidel, mientras que, en el caso de los métodos directos, Duff (1997), Ingle y Mountziaris (1995) y Scott (2001) desarrollaron distintas versiones paralelas de algoritmos multifrontales. Otra alternativa muy común es considerar sistemas lineales

especiales, desarrollando algoritmos específicos aplicables exclusivamente a determinados patrones de ralidad (Duff y col., 1997). Con respecto al campo de los problemas de optimización, existen dos grandes líneas de trabajo: los métodos de búsqueda directa y los de tipo gradiente. Dennis y Torczon (1991) diseñaron un método paralelo para búsqueda multidireccional, que luego evolucionó hacia técnicas de búsquedas con patrones (Hough y col., 2001). Estos trabajos han sido desarrollados como herramientas de propósito general, no habiéndose reportado experiencias sobre aplicaciones específicas en ingeniería. En cuanto a los métodos tipo gradiente, High y LaRoche (1995) introdujeron el paralelismo en un método de optimización SQP (Successive Quadratic Programming) mediante la evaluación simultánea de la función objetivo y las restricciones. Nuevamente, todas las implementaciones citadas en este párrafo fueron diseñadas especialmente para correr sobre computadoras paralelas, existiendo muy pocos antecedentes de algoritmos paralelos concebidos para su ejecución sobre clusters. En el ámbito de los métodos de optimización tipo gradiente, podemos mencionar a Mittelman (1996), quien desarrolló un algoritmo para problemas no lineales sin restricciones que descompone el dominio de búsqueda y resuelve en cada nodo del cluster uno de los bloques asociados.

Finalmente, en el ámbito de los ***problemas estructurales***, la mayoría de los trabajos reportados corresponden a operaciones sobre grafos. Gau y Stadtherr {Gau, 2002 17 /id /d}, por ejemplo, desarrollaron un algoritmo de particionamiento de grafos que permite reordenar matrices ralas en paralelo, para llevarlas a una forma cuasi-tridiagonal que facilita la posterior resolución del sistema de ecuaciones lineales asociado. Otro ejemplo interesante corresponde a la optimización de tipo “branch and bound” . En este caso, el árbol de búsqueda abarca todo el espacio de soluciones del problema y el algoritmo intenta ubicar la solución óptima mediante una selección

“inteligente” de las ramas a explorar. Gau y Stadtherr (2002), por ejemplo, utilizaron un método branch and bound paralelo sobre un cluster de estaciones de trabajo para análisis de intervalos de Newton aplicado a la estimación de parámetros de un modelo de equilibrio líquido-vapor.

De la revisión bibliográfica se desprende que la mayoría de los trabajos existentes han sido desarrollados para su ejecución en computadoras paralelas, aprovechando las capacidades especiales de esa arquitectura. Esto constituye una severa limitación en el rango de aplicabilidad de las metodologías en vista de los altos costos asociados a este hardware. Más aún, en la práctica no es conveniente usar las implementaciones así concebidas sobre clusters. Por un lado, no es posible emular mediante clusters una arquitectura paralela con memoria global en forma eficiente. En el caso de computadoras paralelas con memoria distribuida, el desempeño eficiente de los esquemas de comunicación entre procesadores hace que la granularidad de algoritmos desarrollados sea muy fina. Al portar estos algoritmos a un cluster, el desempeño resulta pobre pues los costos de comunicación aumentan significativamente. En vista de este panorama y considerando asimismo las ventajas comparativas de los clusters en cuanto a economía y flexibilidad, surge la necesidad de profundizar conocimiento en el desarrollo de algoritmos paralelo distribuidos especialmente diseñados para su uso sobre clusters para aplicaciones complejas en el campo de la ingeniería de procesos.

1.4. OBJETIVOS

El objetivo global de este trabajo es desarrollar nuevos algoritmos paralelos que permitan resolver eficientemente problemas de cómputo intensivo, tanto estructurales como numéricos, que surgen frecuentemente en el área de ingeniería de procesos. Las

investigaciones se concentraron en el diseño e implementación de algoritmos paralelos sobre redes de estaciones de trabajo en vista de las ventajas asociadas al uso de clusters. Para lograr el aprovechamiento óptimo de los recursos computacionales existentes, se incluyeron consideraciones asociadas al empleo de procesadores heterogéneos.

Desde el punto de vista de la eficiencia, el objetivo básico es reducir el tiempo de cómputo requerido para resolver los problemas. Esto implica encontrar distribuciones óptimas de carga, repartiendo adecuadamente las tareas para minimizar el tiempo ocioso de los procesadores.

Más específicamente, se pueden mencionar objetivos asociados al tipo de problema a resolver. Por ejemplo, en el caso de aplicaciones en línea resulta útil ampliar el espectro de problemas que pueden ser resueltos con una herramienta. En tal sentido, las ganancias en tiempo derivadas de una paralelización criteriosa permiten lograr resultados dentro de los límites prácticos de tiempo disponibles para un mayor número de instancias.

Los objetivos específicos son profundizar en dos clases de problemas, uno numérico y otro estructural, que son de fundamental importancia en la ingeniería de procesos. El primero corresponde a optimización no lineal con restricciones no lineales; este método es ampliamente utilizado en diseño, síntesis, control, etc. El segundo se refiere al reordenamiento estructural de matrices que es aplicable a instrumentación de plantas de procesos. Cabe destacar que, en ambos casos, su aplicabilidad de las investigaciones realizadas no está limitada únicamente a los problemas de ingeniería de procesos mencionados. Los métodos de optimización no lineal con restricciones no lineales son ampliamente utilizados en todas las disciplinas de la ingeniería.

Las metodologías más apropiadas para el desarrollo de algoritmos paralelos en clusters con nodos heterogéneos son la distribución dinámica de tareas independientes y el particionamiento de dominios de un problema. En el primer caso el paralelismo es introducido evaluando tareas en forma paralela y, dependiendo de cada aplicación particular, actualizando los resultados intermedios. En el segundo caso, el problema original se transforma en otro cuyo modelo es separable; luego, el paralelismo se aplica en cada uno de esos bloques independientes.

Como resultado se han desarrollo técnicas robustas y eficientes aplicables a búsquedas en grafos y algoritmos paralelos en problemas de optimización con función objetivo y restricciones no lineales.

En el caso de problemas estructurales se estudiaron los métodos de búsquedas en grafos clásicos y se establecieron sus limitaciones de aplicación usando redes de estaciones de trabajo. En este sentido se propuso un método de distribución de tareas basado fundamentalmente en el uso de una técnica semi-dinámica y además se desarrollaron nuevos algoritmos distribuidos. En cuanto a los problemas numéricos, se discutieron y desarrollaron estrategias para la paralelización de algoritmos de optimización existentes y se desarrolló una nueva técnica de descomposición de dominio para un algoritmo intrínsicamente paralelo.

Todas las técnicas desarrolladas lograron un muy buen desempeño, respecto de las metodologías existentes. Se emplearon en problemas industriales reales de ingeniería de procesos y se verificaron sus desempeños.

1.5. CONTENIDOS Y ESTRUCTURA DE LA TESIS

La tesis está organizada en siete capítulos. En el primero se introdujo brevemente la problemática y se delinearon los principales objetivos de estas investigaciones. El Capítulo 2 muestra conceptos generales sobre temas asociados al procesamiento paralelo, tales como una clasificación para las distintas arquitecturas de cómputo, consideraciones de diseño de algoritmos paralelos, métricas para evaluación de desempeño y herramientas de propósito general para implementación de desarrollos.

En el Capítulo 3 se estudia la paralelización de problemas estructurales que se resuelven mediante búsquedas en grafos y se aborda en detalle una aplicación compleja del área de diseño de instrumentación que requiere efectuar reordenamientos estructurales de matrices ralas mediante búsquedas en profundidad (depth-first search). Sobre la base de estas investigaciones se propone un nuevo algoritmo de búsqueda en grafos totalmente distribuido, el cual se describe en el Capítulo 4.

A continuación se consideran estrategias para la paralelización de problemas de tipo numérico a través del análisis de problemas de optimización, con especial énfasis en el tratamiento de problemas no lineales con restricciones. El Capítulo 5 se refiere la creación de algoritmos inherentemente paralelos mientras que el Capítulo 6 se concentra en la paralelización de algoritmos secuenciales clásicos.

Finalmente, en el Capítulo 7, se exponen las principales conclusiones de este trabajo y se enuncian lineamientos y sugerencias para futuras investigaciones.

CAPÍTULO 2: CONCEPTOS GENERALES DE PROCESAMIENTO PARALELO EN ENTORNOS DE COMPUTO DISTRIBUIDOS

Desde el nacimiento de las primeras computadoras paralelas hasta el día de hoy, la arquitectura de las mismas se ha diversificado considerablemente. Por lo tanto, es necesario definir una terminología simple que permita categorizar las diferentes arquitecturas de cómputo.

2.1. TAXONOMÍA DE ARQUITECTURAS DE CÓMPUTO

Michael Flynn (Dowd y Severance, 1998) desarrolló la taxonomía de arquitecturas de cómputo que se muestra en la Figura 2.1. Esta categorización se basa en la cantidad de secuencias simultáneas de instrucciones y datos que pueden presentarse en un sistema de procesamiento. Con respecto a la posibilidad de ejecución simultánea de secuencias de instrucciones, cada arquitectura en particular corresponde a una de las siguientes configuraciones: una única cadena (SI, single instruction) ó múltiples cadenas (MI, multiple instruction). Análogamente, con relación al acceso a los datos, algunas arquitecturas manejan una única secuencia (SD, single data) mientras que otras permiten acceder simultáneamente a múltiples secuencias de datos (MD, multiple data).

SISD Estaciones de trabajo uniprocesador	**MISD** Categoría sin sentido práctico
SIMD Computadoras con procesadores vectoriales	**MISD** Computadoras Paralelas y clusters

Figura 2.1: Taxonomía de arquitecturas de cómputo

Las estaciones de trabajo uniprocesador clásicas se clasifican como sistemas SISD, pues en cada instante de tiempo existe una única cadena de instrucciones que actúa sobre una única secuencia de datos. Por su parte, un procesador de tipo vectorial consta de una matriz de procesadores, donde todos ejecutan la misma instrucción sobre sus propios datos locales, razón por la cual le corresponde la categoría SIMD. Por último, los clusters de estaciones de trabajo y las computadoras paralelas son sistemas MIMD porque en un mismo instante de tiempo existen múltiples cadenas de instrucciones que están siendo ejecutadas sobre diferentes secuencias de datos. Cabe destacar que ninguna arquitectura existente pertenece a la categoría MISD porque el esquema carecería de sentido práctico.

2.2. CLASIFICACIÓN DE ARQUITECTURAS PARALELAS

En el ámbito del procesamiento paralelo, pueden identificarse fácilmente las siguientes cuatro categorías según el tipo de hardware empleado:

- MIMD con acceso uniforme a la memoria (uniform memory access, UMA)

- MIMD con acceso no uniforme a la memoria (non-uniform memory access, NUMA)

- MIMD con memoria distribuida

- SIMD con memoria distribuida

Un sistema MIMD con acceso uniforme a la memoria (UMA) posee múltiples procesadores que comparten una memoria global y acceden a la misma con la misma velocidad. Si bien la visualización global de la memoria facilita la tarea de programación de las aplicaciones, para lograr un funcionamiento correcto debe garantizarse de manera explícita el acceso sincronizado a los datos. Ejemplos de esta arquitectura son la SGI Power Challenge, la HP/Convex C-Series, la DEC AlphaServer y la Sun Enterprise.

En el caso de un sistema MIMD con acceso no uniforme a la memoria (NUMA) , los procesadores siguen teniendo una visión global para su acceso. Sin embargo, y tal como su nombre lo indica, los tiempos de acceso a memoria no son uniformes. Esto se debe a que la memoria se encuentra físicamente distribuida. Si bien, en principio, todos los procesadores pueden acceder a cualquier lugar de memoria, cada uno de ellos tiene un área de almacenamiento local de más rápido acceso. Por lo tanto, para alcanzar un desempeño máximo los programas deben diseñarse de modo que empleen datos almacenados en una memoria local siempre que sea posible. En el modelo NUMA pueden distinguirse dos formas de implementar el acceso a la memoria caché de cada nodo: de modo consistente o inconsistente. En el primer caso, el caché está siempre actualizado pues el hardware asegura que los caché locales siempre contengan los valores actualizados de las locaciones remotas. Esto conlleva un gasto en tiempo asociado al mantenimiento de la información. Algunos ejemplos de modelos NUMA con

caché consistente son las computadoras SGI Origin-2000, HP Exemplar X-Class y Sequent NUMAQ-2000. En el modo inconsistente, la arquitectura es mucho más simple y el acceso a los datos de la memoria es más veloz. Sin embargo la programación en estos modelos es más difícil pues debe evitarse el acceso simultáneo de más de un procesador a una misma dirección de memoria para no producir resultados inconsistentes. Un sistema actual que soporta este esquema es la Cray T3E.

En un sistema MIMD con memoria distribuida, los múltiples procesadores tienen acceso sólo a su memoria local y existe un hardware de transmisión de datos que permite el intercambio de información entre los procesos residentes en los distintos nodos del sistema. A esta clasificación pertenecen los clusters de estaciones de trabajo y las computadoras paralelas con memoria distribuida. La gran diferencia entre estos sistemas reside en el hardware de interconexión usado. En un cluster la comunicación se realiza a través de una red, preferentemente de alta velocidad, mientras que en una computadora paralela existe un hardware dedicado. Algunos ejemplos de clusters son los proyectos The Computational Plant (1890 nodos) (Sandia National Laboratories, 2002) y Netfinity Supercluster (1024 nodos) (NCSA, 2002). En cuanto a las computadoras paralelas con memoria distribuida, las más conocidas son la IBM SP, la Connection Machine CM-5 y la serie de computadoras desarrolladas a medida por Intel según su modelo ASCI (Advanced Scientific Computing Initiative).

Finalmente, una arquitectura SIMD consiste en un conjunto de procesadores, usualmente del orden de miles de nodos, que ejecutan un misma instrucción pero sobre diferentes datos. Cada procesador tiene una pequeña memoria local y un hardware que permite la comunicación entre ellos. La principal desventaja de esta arquitectura es que no permite a los procesadores ejecutar distintas instrucciones en un mismo ciclo de reloj.

Así, cuando la instrucción no involucra múltiples datos, el sistema desperdicia poder de cómputo pues quedan procesadores ociosos. Por esta razón, el uso de estos sistemas ha disminuido en los últimos tiempos. Las computadoras Connection Machine CM-2, MasPar MP-1 y MP-2 son ejemplos de esta arquitectura.

2.3. PROCESAMIENTO PARALELO

Consideremos un algoritmo que puede ser implementado para correr en forma secuencial ó paralela. Si se asume que el mismo tipo de procesador fue usado para la implementación del algoritmo secuencial y paralelo, intuitivamente debería esperarse que el tiempo requerido para resolver el problema decrezca a medida que se agregan más procesadores. Así, el tiempo de cómputo de un algoritmo que se ejecuta en una computadora con un único procesador puede reducirse hasta P veces (en el caso ideal) cuando este esfuerzo es distribuido en un ambiente de P procesadores. Si P es suficientemente grande, la implementación paralela de un algoritmo podría resolver problemas en mínimas fracciones de tiempo comparado con su versión secuencial. La Figura 2.2 muestra este tiempo óptimo paralelo, el cual puede denominarse tiempo ideal, en comparación con el tiempo real de cómputo asociado a un problema típico.

La curva correspondiente al tiempo real de procesamiento muestra un descenso hasta llegar a un mínimo que corresponde a una determinada cantidad de procesadores. A partir de esa configuración, la incorporación de más procesadores degenera el desempeño del sistema. Esto se debe a dos aspectos principales:

- limitaciones propias del algoritmo
- limitaciones en la implementación

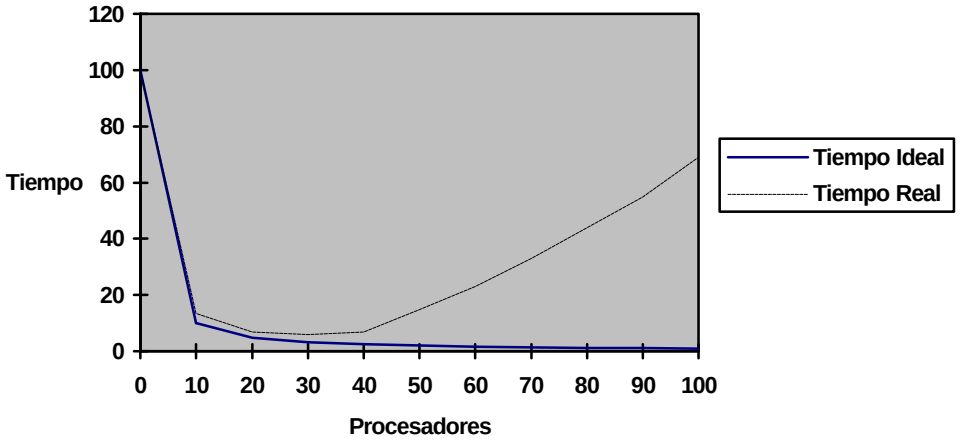

Figura 2.2: Tiempo ideal y tiempo real de una implementación paralela

Las limitaciones del algoritmo están asociadas a su grado de paralelizabilidad. No todos los algoritmos permiten ser paralelizados pues el paralelismo implica la ejecución simultánea de tareas, cuya factibilidad depende de las depedencias existentes entre la información y los resultados intermedios. Así, un algoritmo inherentemente secuencial impone una restricción insalvable para su realización paralela. Por otra parte, cada algoritmo tiene sus propias características que hacen que su paralelización sea más o menos beneficiosa.

Existen limitaciones de implementación cuando un procesador no puede dedicar tiempo a realizar cómputo provechoso. Estas penalidades surgen cuando el procesador cuando el procesador ejecuta una operación de comunicación o cuando queda ocioso.

- Necesidad de comunicación: esta es una penalidad propia al modelo paralelo si se considera que los procesadores tienen necesidad de comunicarse para resolver un problema cooperativamente.

- Tiempo ocioso: ocurre cuando un procesador no puede realizar cómputo productivo, ya sea porque no tiene más tareas para ejecutar, porque su tarea

actual se encuentra suspendida esperando una sincronización, o porque los datos que necesita para continuar con su operación no han arribado. La estrategia de utilización plena de todos los procesadores se denomina balanceo de carga.

Si bien estas limitaciones no pueden ser eliminadas en su totalidad, diversas estrategias tales como el correcto balanceo de carga de los procesadores o el uso de protocolos no bloqueantes para las comunicaciones, permiten mejorarlas para ofrecer un mayor desempeño del sistema paralelo.

2.4. GRANULARIDAD DE LOS PROCESOS

Para lograr una mejora en la velocidad de procesamiento mediante el paralelismo, es necesario dividir el cómputo en subtareas que puedan ser ejecutadas en forma simultánea, no importa qué clase de sistema de cómputo paralelo se disponga. El tamaño de estas tareas puede ser descripto mediante su granularidad. Con granularidad gruesa, cada tarea contiene gran cantidad de código que se ejecuta secuencialmente y consume un tiempo de cómputo sustancial. De la misma manera, granularidad fina implica que el código consiste de pocas instrucciones o, eventualmente, una sola. En general se busca incrementar la granularidad de las tareas en la implementación paralela para compensar el costo de creación de los procesos asociados y el de comunicación de los datos.

En el caso particular donde los sistemas paralelos se basan en clusters es imprescindible reducir el costo de comunicación. Esto se debe fundamentalmente a que la latencia en los mensajes es una penalidad de magnitud significativa. A medida que dividimos el problema en partes llegamos a un punto donde el tiempo de comunicación domina al tiempo total de ejecución. La siguiente fórmula:

$$\frac{\text{Tiempo total de cómputo}}{\text{Tiempo total de comunicación}} = \frac{t_{comp}}{t_{comun}} \qquad (2.1)$$

puede ser usada como una métrica para la granularidad. Se observa claramente que es importante maximizar la relación cómputo/comunicación mientras se mantiene el paralelismo.

Además, la granularidad en algunas ocasiones está relacionada con la cantidad de procesadores disponibles en el sistema. Por ejemplo, cuando la metodología de resolución de un problema implica una descomposición de su dominio, a mayor cantidad de nodos de cómputo menor sería el tamaño de las tareas y mayor la cantidad de comunicaciones (debido a la necesidad de sincronización y/o reportes de resultados parciales). Así, el valor de la relación (2.1) baja al aumentar el número de procesadores. Esto muestra que en los sistemas paralelos existe un punto óptimo de operación, y cuando se lo alcanza, la adición de nuevos nodos de cómputo aumenta el tiempo de ejecución, tal como se observa en la Figura 2.2.

2.5. MÉTRICAS

2.5.1. FACTOR DE SPEED-UP

Consideremos un sistema de cómputo paralelo con *n* procesadores homogéneos. Una medida relativa de desempeño que compara un sistema de cómputo paralelo con respecto a otro uniprocesador es el factor de ***speed-up***, *S*(*n*), definido como:

$$S(n) = \frac{\text{Tiempo de ejecución usando un procesador}}{\text{Tiempo de ejecución usando un sistema paralelo de n procesadores}} = \frac{t_s}{t_p}$$

siendo t_s el tiempo del mejor algoritmo secuencial ejecutado en un procesador con un desempeño comparable al de los nodos de procesamiento del sistema paralelo. Asociada a la métrica de speed-up se define la eficiencia *E* para un sistema con *n* procesadores como:

$$E = \frac{S(n)}{n}$$

donde *E* expresa que porcentaje del poder de cómputo total del sistema paralelo está siendo aprovechado para el cómputo efectivo en una implementación e instancia de problema particular.

Cabe destacar que estas definiciones están desarrolladas considerando el caso homogéneo, es decir un conjunto de procesadores idénticos. En condiciones normales el speed-up máximo que puede lograrse con *n* procesadores es *n* y se denomina ***speed-up*** lineal. Si la implementación paralela alcanza un valor superior a *n* podemos estar en un caso donde no se ha testeado el algoritmo paralelo frente al mejor secuencial o bien existe alguna diferencia en el desempeño de los procesadores. Sin embargo, en configuraciones especiales de una arquitectura paralela puede suceder que aún haciendo una medición correcta se supere el valor de *n* y en este caso hablamos de ***speed-up superlineal***. La situación más común en que puede darse esta situación es cuando se aprovecha la mayor memoria disponible del sistema multiprocesador. Supongamos que la cantidad de memoria que posee una computadora de un único procesador es igual a la memoria que tiene cada nodo de un sistema multiprocesador; dado que el total de memoria disponible en la máquina paralela es mayor que en la máquina uniprocesador, la primera puede tener almacenados en RAM más datos del problema en un mismo instante, provocando un menor acceso a la unidad de almacenamiento secundario pues

utilizará menos el sistema de memoria virtual. La Figura 2.3 muestra los comportamientos del speed-up real, lineal y superlineal.

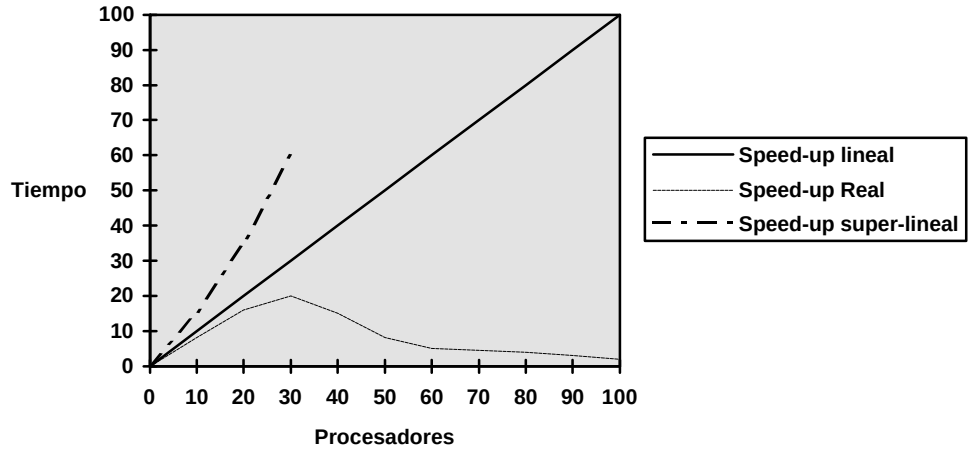

Figura 2.3: Representación del speed-up real, lineal y superlineal

2.5.2. SPEED-UP MÁXIMO – LEY DE AMDAHL

Típicamente, todos los algoritmos secuenciales que desean ser paralelizados contienen una fracción *f* de cómputo que es inherentemente serial, como se esquematiza en la Figura 2.4 (a). Por lo tanto, la situación ideal para la versión paralela es que el resto de las tareas puedan ser computadas en paralelo por todos los procesadores. Asumiendo que este particionamiento de tareas es óptimo y no existe costo por la distribución de estas tareas, el tiempo de ejecución de la implementación paralela es:

$$t_p = ft_s + (1 - f)t_s / n$$

y se esquematiza en la Figura 2.4 (b)

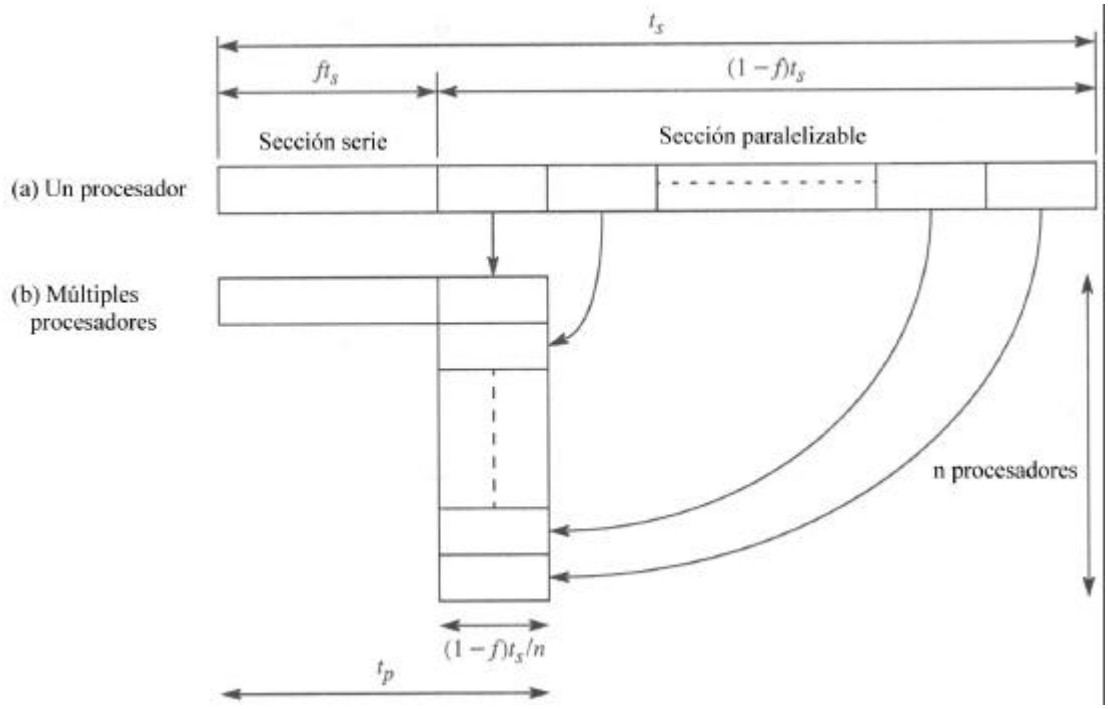

Figura 2.4: Máximo nivel de paralelización idealmente alcanzable

Por lo tanto, el speed-up máximo alcanzable queda determinado por la siguiente ecuación, conocida como la ***Ley de Amdahl*** (Amdahl, 1967) :

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s / n} = \frac{n}{1+(n-1)f}$$

La Figura 2.5 muestra el speed-up $S(n)$ graficado en función del número de procesadores y de la fracción de tiempo de cómputo secuencial f. Como es lógico, se observa una mejora del speed-up a medida que se aumenta el número de procesadores. Sin embargo:

$$\lim_{n \to \infty} S(n) = \frac{1}{f}$$

lo cual significa que aún disponiendo de un número infinito de procesadores, el speed-up máximo está limitado por la cantidad de tiempo de cómputo no paralelizable. Por

ejemplo, si sólo un 5% del tiempo total de cómputo corresponde al código serial, el speed-up teórico máximo que podrá lograrse es 20, sin importar la cantidad de procesadores que se usen.

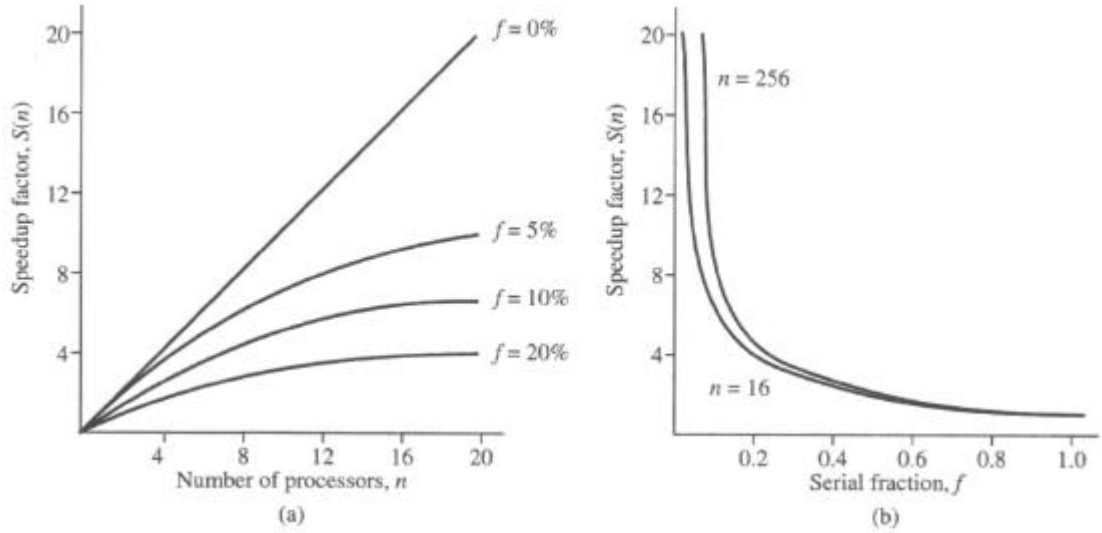

Figura 2.5: Efecto del total de cómputo serial sobre el speed-up

2.5.3. EFICIENCIA

Una métrica comúnmente usada para determinar el porcentaje de tiempo de cómputo útil en un sistema, es la eficiencia relativa al desempeño logrado con un único procesador, que se mide como:

$$E(n) = \frac{S(n) * 100}{n}$$

La Figura 2.6 muestra los tiempos de cómputo ideal y real para el ejemplo de la Figura 2.2, expresados en términos de eficiencia según la cantidad de procesadores empleados. Se observa que en el caso ideal cada procesador logra un 100% de eficiencia, es decir, dedica 100% de su tiempo a realizar cómputo útil. Los valores de eficiencia

para un caso real son siempre menores debido a limitaciones del algoritmo y la penalidades de implementación (ver Sección 2.1).

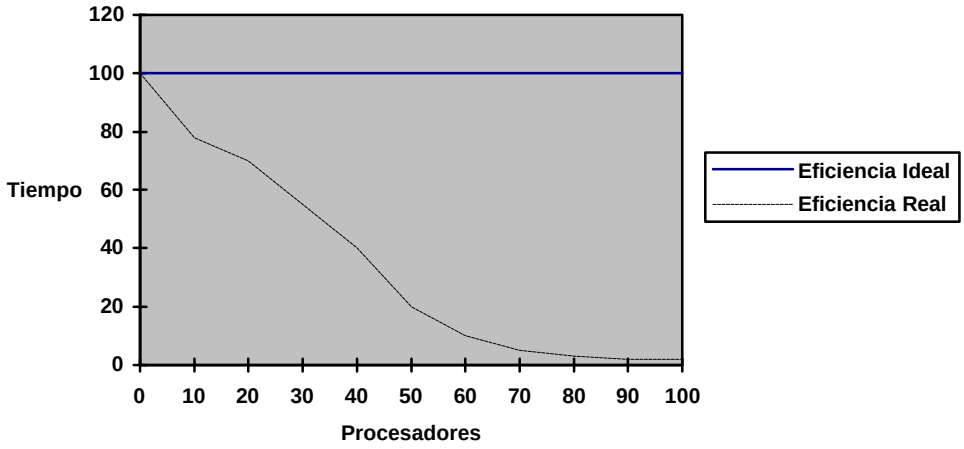

Figura 2.6: Eficiencia ideal y real

2.6. SOPORTE PARA LA IMPLEMENTACIÓN DE PROCESAMIENTO PARALELO DISTRIBUIDO HETEROGÉNEO

En un ambiente de procesamiento distribuido es necesario disponer de un mecanismo que permita la comunicación entre procesadores. Este mecanismo se denomina protocolo de transmisión de datos. Existen numerosos protocolos que permiten estas operaciones; sin embargo, el sistema operativo UNIX popularizó el estándar mundial de mayor aceptación: TCP/IP . Para la comunicación entre tareas, TCP/IP provee servicios UDP (User Datagram Protocol) y TCP (Transmission Control Protocol). El primero está diseñado para el envío de mensajes no seguros y sin conexión, mientras que el segundo supera esas desventajas sacrificando eficiencia.

Si bien es posible el desarrollo de aplicaciones paralelas en ambientes distribuidos usando los servicios de transporte de los protocolos de transmisión de datos, la

complejidad de los mismos hacen que esta forma de implementación sea no viable. Por esta razón, es recomendable el uso de abstracciones que permitan liberar al desarrollador de aplicaciones paralelas de la complejidad subyacente en los mecanismos de comunicación. En este sentido, se considera el uso de uno de los estándares para los servicios de comunicación y coordinación de tareas en ambientes distribuidos: el sistema Parallel Virtual Machine (PVM).

PVM usa el modelo de pasaje de mensajes para permitir el desarrollo de software en entornos de procesamiento de memoria distribuida sobre una gran variedad de computadoras con arquitecturas paralelas. Este sistema está basado en el protocolo TCP/IP. La idea fundamental en PVM es permitir que un conjunto de computadoras simule una máquina virtual paralela. PVM provee un ambiente unificado en el cual el software paralelo puede ser desarrollado de una manera eficiente usando el hardware existente. PVM maneja en forma transparente el ruteo de mensajes, conversión de datos y distribución de tareas sobre un conjunto de computadoras, heterogéneas ó no, interconectadas a través de diferentes redes de comunicación de datos.

El modelo de computación PVM es sumamente atractivo pues es simple y a la vez general. El usuario escribe su aplicación como una colección de tareas cooperativas. Las tareas acceden a los recursos PVM a través de una librería de rutinas, las cuales proveen mecanismos de inicialización y terminación de tareas sobre la red, así como también la comunicación y sincronización entre las tareas. Las primitivas de pasaje de mensajes de PVM están orientadas para permitir una operación heterogénea, involucrando constructores fuertemente tipados para buffering y transmisión. Los constructores de comunicación incluyen operaciones para envío y recepción de estructuras, además de primitivas de alto nivel, tales como broadcast y barrier syncronization.

Las estructuras de control de las tareas PVM son arbitrarias. En cualquier punto de ejecución de una aplicación, una tarea puede comenzar/detener otras tareas, ó agregar/eliminar computadoras de la máquina virtual. Cualquier proceso puede comunicarse y/o sincronizarse con otro. Cualquier estructura de control y dependencia puede ser implementada en el sistema PVM mediante el uso apropiado de los constructores provistos y por las sentencias de control de flujo del lenguaje utilizado.

CAPÍTULO 3: PARALELIZACIÓN DE ALGORITMOS DE BUSQUEDA EN GRAFOS Y SU APLICACIÓN AL REORDENAMIENTO ESTRUCTURAL DE MATRICES

Los grafos son estructuras matemáticas esenciales para resolver una amplia variedad de problemas que surgen en distintas disciplinas. Algunas aplicaciones son los modelos gráficos para redes eléctricas y arquitecturas de computadoras, modelos para optimización de redes, asignación de tareas, etc. (Yellen y Gross, 1998).

Estas aplicaciones involucran exploraciones del espacio de búsqueda mediante el recorrido de los distintos caminos contenidos en el grafo que modela al problema. Estas búsquedas son de naturaleza combinatorial pues se debe recorrer un espacio arbóreo que crece exponencialmente con la cantidad de nodos y aristas del grafo. Por esta razón la paralelización de estos algoritmos puede conducir a mejoras sustanciales en los tiempos de ejecución.

Dentro de la Ingeniería de Procesos, los grafos y búsquedas se utilizan para estudios de instrumentación y monitoreo de plantas. Esta disciplina que se ocupa de la obtención, registro y análisis de datos de planta, tiene un gran incentivo económico ya que un buen monitoreo conduce a una operación eficiente y segura. El diseño de instrumentación consiste en definir la cantidad, tipo y ubicación de los sensores requeridos para lograr suficiente conocimiento del estado real de la planta en cualquier

instante. En particular, el enfoque estructural moderno para efectuar esta tarea se basa en el reordenamiento de la matriz de ocurrencia del problema hacia un patrón específico que revela a simple vista la clasificación de todas las variables del proceso. En este capítulo se presenta el desarrollo e implementación de un nuevo algoritmo paralelo distribuido para efectuar estas permutaciones en forma eficiente y robusta.

3.1. DESCRIPCIÓN DEL PROBLEMA DE INSTRUMENTACIÓN

Dentro del área de monitoreo de plantas, algunos objetivos prácticos de sumo interés son diseñar la configuración de instrumentos de medición para una nueva planta o equipo individual, o bien modificar la configuración existente (*revamp*) con el objeto de mejorar el nivel de conocimiento del estado de un proceso. Un análisis de instrumentación adecuado y riguroso genera importantes mejoras en la práctica, ya sea desde el punto de vista económico a través del ahorro en recursos, como así también desde el punto de vista operativo en el cumplimiento de las especificaciones de calidad, las reglamentaciones ambientales y las normas de seguridad.

Una herramienta fundamental para realizar el análisis de instrumentación son los modelos matemáticos en estado estacionario que representan el funcionamiento del proceso en condición de equilibrio. Estos fenómenos se modelan mediante sistemas de ecuaciones algebraicas asociadas a balances de masa y energía, relaciones termodinámicas y correlaciones experimentales. Dada una planta de procesos pueden existir varios modelos asociados, donde cada uno de ellos corresponde a distintos niveles de rigurosidad en la representación. Un modelo apropiado deberá contener todos los elementos necesarios que aseguren el nivel de exactitud requerida en la solución final, preservando al máximo su sencillez para facilitar su solución y análisis posterior.

Una vez establecido el modelo elegido para representar a un proceso, los algoritmos de clasificación de variables permiten por ejemplo establecer si la instrumentación existente en una planta es suficiente para conocer todas las variables de interés y establecer la locación de sensores (instrumentos de medición) adicionales para posibilitar la posterior verificación de la calidad de ciertas mediciones usando información del modelo del proceso. La categorización conduce a los siguientes tipos de variables (Romagnoli y Sanchez, 1999):

a) Variables no medidas: son aquellas cuyos valores no son sensados regularmente.

   Se subdivide en:

- Variables Observables: su valor puede obtenerse a partir de las variables medidas usando ecuaciones del modelo.
- Variables No Observables: su valor no puede calcularse a partir de las variables medidas mediante ecuaciones del modelo.

b) Variables medidas: son aquellas cuyos valores se conocen directamente a través de la instrumentación.

   Se las clasifica en:

- Variables Redundantes: variables medidas cuyos valores también pueden ser calculados a partir de los balances y del resto de las variables medidas.
- Variables No Redundantes: variables medidas cuyos valores no pueden computarse a partir de los balances y de las restantes variables medidas.

Por su parte, las ecuaciones del modelo pueden clasificarse en:

- Ecuaciones Asignadas: son aquellas que se utilizan para despejar variables observables.

- Ecuaciones Redundantes: son aquellas cuyas variables son todas observables o medidas, y no son empleadas para despejar otras variables observables.
- Ecuaciones No Asignadas: son aquellas que contienen al menos una variable no observable y por lo tanto no pueden emplearse para despejar variables observables.

La clasificación de variables no medidas se denomina análisis de observabilidad, mientras que la clasificación de variables medidas se conoce como detección de redundancias.

Con respecto a los objetivos específicos del análisis de observabilidad, usualmente la meta es reducir la cantidad de sensores. En este sentido, la mejor clasificación de variables no medidas es aquella que logra determinar el mayor número de variables observables para un conjunto de mediciones preestablecido.

La calidad de una metodología de clasificación puede evaluarse en relación con las siguientes propiedades:

- Eficacia: capacidad para determinar el mayor número de variables observables a partir de un conjunto preestablecido de mediciones.
- Eficiencia: potencial para obtener la clasificación en tiempos de ejecución razonablemente bajos para problemas industriales de gran dimensión.
- Aplicabilidad: posibilidad de emplear la metodología de clasificación para distintos tipos de plantas, independientemente de las características físicas del proceso y del tipo de modelo matemático usado para representar su funcionamiento.

3.2. MÉTODOS SECUENCIALES PARA REORDENAMIENTO ESTRUCTURAL DE MATRICES

Existen dos principales enfoques para realizar el análisis de observabilidad sobre una planta de procesos: uno ***orientado a topología*** y otro ***orientado a ecuaciones***. El primero (Kretsovalis y Mah, 1988) se basa en teoría de grafos y aplica reglas de análisis para clasificar las variables mediante la inspección de una secuencia de grafos derivados de la topología del proceso. Aunque estas técnicas son eficientes, su principal limitación es la pérdida de rigurosidad en el análisis cuando el modelo involucra relaciones fuertemente no lineales. El segundo enfoque (Romagnoli y Stephanopoulos, 1980) obtiene la clasificación por medio de un reordenamiento estructural de la matriz de ocurrencia del modelo. Dada su naturaleza, los métodos basados en reordenamientos estructurales permiten una mayor independencia del grado de no linealidad exhibido por el modelo matemático asociado al proceso.

Ponzoni y col. (1999) desarrollaron una estrategia estructural denominada GS-FLCN (global strategy with a first least-connected node algorithm) que es única en el sentido de que permite trabajar con modelos matemáticos fuertemente no lineales preservando la robustez. La principal desventaja de esta técnica es su costo computacional para problemas grandes, dado que su corazón algorítmico está basado en búsquedas en profundidad sobre los grafos asociados a los modelos. Por esta razón se consideró la paralelización de esta estrategia para solucionar el problema de eficiencia de la metodología y permitir el tratamiento de problemas grandes, tales como los asociados a plantas completas.

3.2.1. LA ESTRATEGIA GS-FLCN

El método GS-FLCN consiste básicamente en descomponer la matriz de ocurrencia ***M*** asociada al modelo del proceso químico mediante una búsqueda incremental por tamaño para encontrar el máximo número de subconjuntos de asignación de tamaño mínimo. Las filas de la matriz ***M*** corresponden a las ecuaciones del modelo de proceso y sus columnas están asociadas a las variables no medidas. Un subconjunto de asignación es un bloque asociado a un subsistema de ecuaciones algebraicas cuadrado que admite solución numérica. La Figura 3.1 muestra un esquema algorítmico de GS-FLCN. El objetivo final del análisis de observabilidad consiste en permutar ***M*** para lograr el patrón esquematizado en la Figura 3.2.

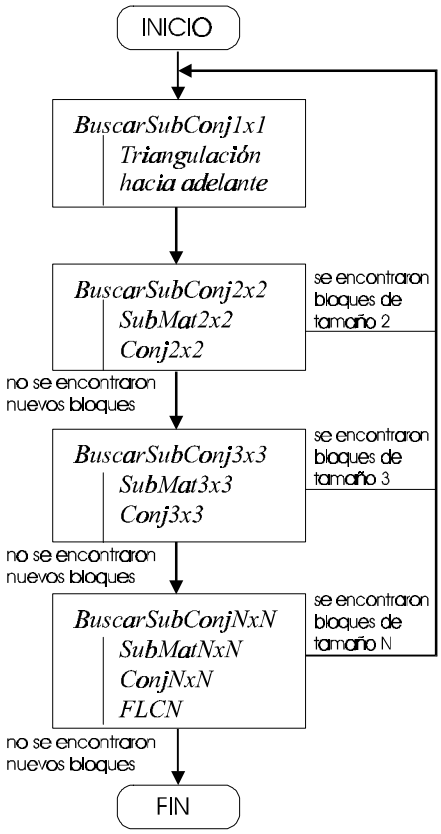

Figura 3.1: Esquema algorítmico de GS-FLCN

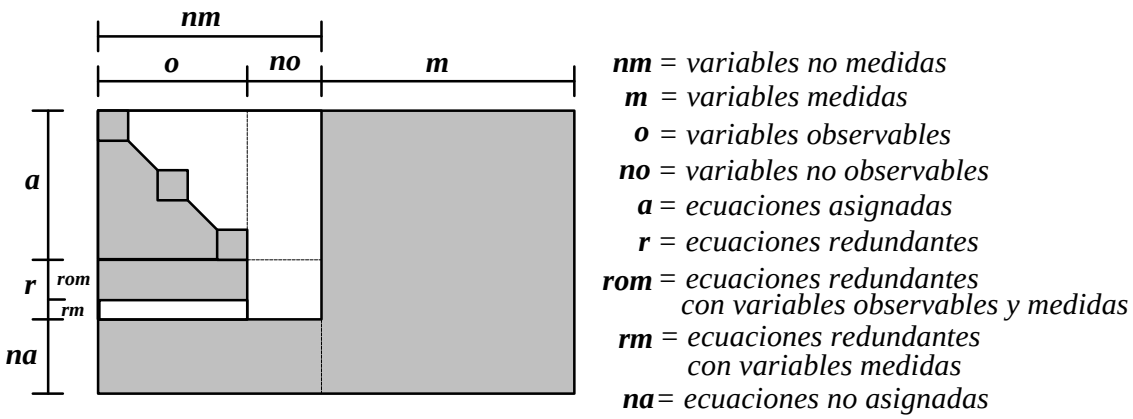

Figura 3.2: Matriz de ocurrencia reordenada

En primer lugar, se emplea el método clásico de triangulación hacia delante para detectar las variables calculables por despeje directo a partir de variables medidas (procedimiento *BuscarSubConj1x1*). Luego, GS-FLCN utiliza las rutinas *BuscarSubConj2x2* y *BuscarSubConj3x3* para detectar los subconjuntos de tamaño 2 y 3 respectivamente. Finalmente, el algoritmo FLCN (rutina *BuscarSubConjNxN*) se encarga de buscar los bloques de tamaño 4 en adelante. La distinción entre diferentes rutinas para detectar bloques de tamaño 2, 3 y más de 4 se debe a que en cada caso se aplican algoritmos específicamente diseñados. Cada una de las rutinas *BuscarSubConj2x2*, *BuscarSubConj3x3* y *BuscarSubConjNxN* utilizan dos procedimientos principales, *SubMatXxX* y *ConjXxX*. En primer lugar, *SubMatXxX* se encarga de construir la submatriz utilizada por *ConjXxX* para realizar la búsqueda. Cada vez que en un dado nivel se encuentra un subconjunto de asignación, la búsqueda se reinicia para asegurar la obtención de la mayor cantidad de bloques de tamaño mínimo.

El procedimiento *BuscarSubConjNxN* es el más costoso con respecto al tiempo de cómputo pues se encarga de detectar los subconjuntos de mayor tamaño. Cada subconjunto de asignación de tamaño *m* puede asociarse a un camino de longitud *m*-1 en *G*, donde los nodos corresponden a las variables observables. La rutina *FLCN* utiliza un

algoritmo de búsqueda DFS para encontrar todos los caminos de longitud *m*-1 sobre el grafo no dirigido *G* correspondiente a la matriz de ocurrencia M^TM. Por esta razón, la idea central es desarrollar una paralelización eficiente del algoritmo clásico DFS, para posteriormente incorporar este conocimiento en una nueva rutina *FLCN* (*pFLCN*) paralela.

3.3. PARALELIZACIÓN DE ALGORITMOS DE BÚSQUEDA EN PROFUNDIDAD SOBRE GRAFOS

3.3.1. SOBRE TEORÍA DE GRAFOS: ALGUNAS DEFINICIONES

Un ***grafo no dirigido*** *G* (o simplemente ***grafo)*** (Gibbons, 1994) puede representarse simbólicamente como *G* = (*N*, *E*), donde *N* es un conjunto de ***nodos*** y *E* es una colección (no necesariamente un conjunto) de pares no ordenados de nodos, que se denominan ***aristas***. Si *u* y *v* son nodos de *G* y existe un par no ordenado *a* = (*u*, *v*) en *E*, se dice que *a* une *u* y *v*, ó que existe una arista *a* entre *u* y *v*. En este caso, también se dice que *u* y *v* son ***incidentes*** en *a*, ó que *a* tiene ***incidencia*** en *u* y *v*. Por otra parte, diremos que *u* es ***adyacente*** a *v* y *v* es ***adyacente*** a *u*. Un grafo *G* = (*N*, *E*) se denomina ***grafo dirigido*** o ***digrafo*** cuando las aristas *e* = (*u*, *v*) de E son dirigidas. En este caso se dice que la arista *e* parte de *u* y llega a *v*, ó que *v* es adyacente a *u*.

Cuando dos o más aristas conectan un mismo par de nodos, estas se denominan ***aristas paralelas***. Una arista cuyos extremos coinciden se denomina ***bucle***. Se dice que ***G*** es un ***grafo simple***, si *G* no contiene aristas paralelas ni bucles. Además, un grafo simple *A* se dice un ***árbol*** si dados dos nodos cualesquiera *u* y *v* de *A*, existe un único

camino entre *u* y *v*. Por otra parte, un grafo $S = (N_S, E_S)$ es un ***subgrafo*** de $G = (N, E)$ si $N_S \subseteq N$ y $E_S \subseteq E$.

3.3.2. EL ALGORITMO SECUENCIAL DE BÚSQUEDA EN PROFUNDIDAD

La mayoría de los algoritmos de grafos requieren de un método sistemático para visitar sus nodos. En particular, la ***búsqueda en profundidad***, conocida en inglés como ***Depth-First Search*** (***DFS***), es una técnica ampliamente usada que posee características que contribuyen a obtener algoritmos muy eficientes.

Consideremos en primera instancia la DFS sobre grafos no dirigidos únicamente. Supongamos que el procedimiento está actualmente en el nodo *v*. Luego, el paso general en la búsqueda requiere visitar el próximo nodo adyacente a *v* que aún no haya sido alcanzado. Si no existe un nodo en tales condiciones, la búsqueda retorna al nodo *u*, el cual había sido visitado justo antes que *v*. El paso general de la búsqueda se repite hasta que todos los nodos de la componente del grafo que se está explorando hayan sido visitados. El Algoritmo 3.1 muestra el pseudo-código correspondiente a una búsqueda en profundidad.

```
Datos de entrada: G(N, E), v
Datos de entrada-salida: DFI, i
DFI(v) ← i.
i ← i+1.
Para cada arista (v, u) ∈ E hacer:
   Si DFI(u)=0 entonces
      DFS(G(N, E), u, DFI, i).
   fin-si
fin-para
```

Algoritmo 3.1: Búsqueda en profundidad sobre grafos

La entrada a este algoritmo es el grafo $G = (N, E)$ y la salida es un vector de etiquetas DFI(v) (depth-first index) para cada nodo v que inicialmente es 0 mientras que al finalizar el algoritmo indica el orden en el cual fue visitado el nodo v en el recorrido.

3.3.3. UN ALGORITMO PARALELO DE BÚSQUEDA EN PROFUNDIDAD

Los algoritmos de búsqueda en profundidad aparentan ser inherentemente secuenciales a partir de su definición (Chaudhuri, 1992). El problema comenzó a ser investigado en la década del ´70 por Eckstein y Alton (1977) y por Reghbati y Corniel (1978) sin mucho éxito. Un trabajo más reciente de Reif (1985) también enfatiza este argumento, demostrando que la búsqueda en profundidad en grafos no puede ser llevada a cabo en un tiempo paralelo polilogarítmico, es decir, en un tiempo $O(\log^c n)$ para cualquier constante $c \geq 1$.

3.3.3.1. Estrategia de paralelización

Dada la naturaleza distribuida del entorno de cómputo disponible se decidió implementar un algoritmo paralelo de búsqueda en profundidad basado en una descomposición del grafo de búsqueda utilizando la metodología *master-worker*. En este esquema existe una única tarea *master* que comienza a realizar una búsqueda en profundidad sobre el grafo, pero a determinado umbral (longitud de camino) delega la exploración del resto del subgrafo a un proceso *worker*. En el caso de grafos acíclicos, las búsquedas en los *workers* pueden realizarse de manera simultánea, lográndose así una paralelización efectiva. La Figura 3.3 muestra este esquema de distribución del procesamiento. En la Figura 3.4 se ejemplifica una instancia de la búsqueda, donde el umbral está ubicado a nivel 2; la tarea *master* alcanza en su recorrido al nodo c en el

umbral y envía a explorar a un worker el resto del subgrafo con raíz en c para completar la exploración de los caminos que comienzan con la secuencia a-b-c. El pseudo-código de estas dos rutinas se muestra en los Algoritmos 3.2 y 3.3.

```
Datos de entrada: G(N, E), v, prof
Datos de entrada-salida: DFI, i
DFI(v) ← i.
i ← i+1.
prof ← prof+1.
Para cada arista (v, u) ∈ E hacer:
   Si DFI(u)=0 entonces
      Si i < umbral entonces
          DFS(G(N, E), u, prof, DFI, i)
         Sino
          wDFS(G(N, E), u, prof, DFI, i)
      fin-si
   fin-si
fin-para
```

Algoritmo 3.2: mDFS: Algoritmo paralelo de búsqueda (*Master*)

```
Datos de entrada: G(N, E), v
Datos de entrada-salida: DFI, i
DFI(v) ← i.
i ← i+1.
Para cada arista (v, u) ∈ E hacer:
   Si DFI(u)=0 entonces
      wDFS(G(N, E), u, DFI, i).
   fin-si
fin-para
```

Algoritmo 3.3: wDFS: Algoritmo paralelo de búsqueda (*Worker*)

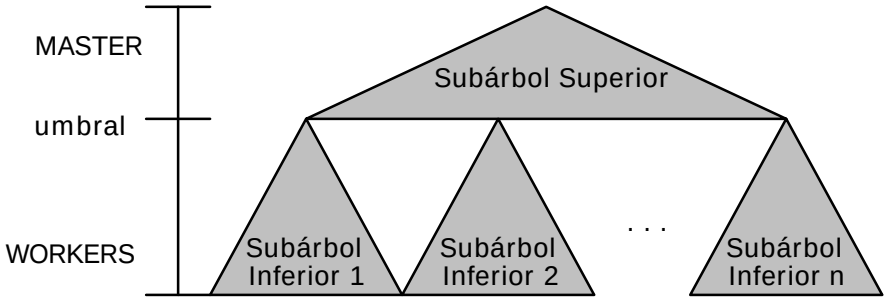

Figura 3.3: Descomposición del espacio de búsqueda.

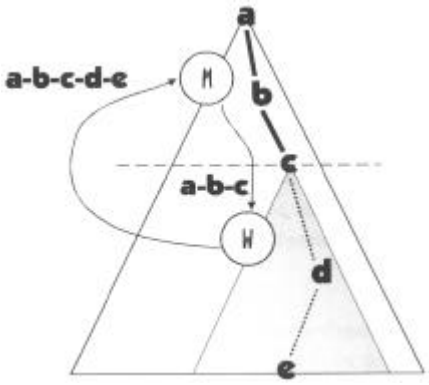

Figura 3.4: Ejemplo de una instancia de búsqueda

En un algoritmo simple de exploración, existen dos tipos de mensajes: procesar un determinado subárbol del árbol de búsqueda desde la tarea *master* a una tarea *worker* y finalizar la exploración del subárbol desde una tarea *worker* a la tarea *master*. Cabe aclarar que dependiendo de la aplicación particular en que se use este algoritmo, la semántica y/o la cantidad de los mensajes podrían variar.

3.3.3.2. Aspectos de implementación

La elección del umbral implica un compromiso entre el tamaño de los subárboles a explorar por parte de los *workers* y el número de mensajes que se utilizarán. Cuando el umbral aumenta hacia niveles más profundos, el tamaño de los subárboles es más

pequeño; en este sentido, un umbral grande sería más deseable. Sin embargo, es claro que la cantidad de subárboles inferiores crece exponencialmente, incluyendo la cantidad de mensajes que se necesitan para disparar sus búsquedas y reportar los resultados. Considerando este aspecto, un umbral más chico sería apropiado. En vista de estas características se realizaron experimentos utilizando 10 grafos de prueba representativos para determinar el umbral óptimo. Estos grafos fueron elegidos para imitar la cantidad de nodos y la densidad de los grafos asociados a modelos reales. Estas corridas paralelas revelaron que el valor óptimo es 2, mostrando que el costo dominante son los mensajes necesarios adicionales.

Dado que la cantidad de subárboles a explorar por los *workers* es normalmente mayor a la cantidad de procesadores en el sistema, es necesario repartir estas tareas en los nodos de cómputo de manera eficiente. El esquema de balanceo de carga elegido para esta implementación corresponde a una estrategia por demanda, es decir una distribución dinámica de tareas. Cada vez que el *master* alcanza el umbral y debe asignar el resto de la exploración a un *worker*, el *master* se fija si existe algún *worker* desocupado, en cuyo caso le envía un mensaje indicando que continúe la exploración con el subcamino actual. Si todos los *workers* están ocupados, el proceso *master* se bloquea hasta que recibe un mensaje de finalización de exploración por parte de un *worker*, el cual obviamente recibirá la próxima tarea para explorar. Esta estrategia de distribución de carga tiene muchas ventajas. En primer lugar, los subárboles que deben ser explorados por los *workers* son de diferentes tamaños y densidades, por lo tanto el tiempo requerido para recorrerlos no puede ser predicho. Además, el poder de cómputo de los nodos de procesamiento del cluster es compartido con otros usuarios, lo cual sobrecarga a las unidades de procesamiento en tiempo de ejecución. Como resultado, el tiempo empleado por un nodo para ejecutar cada tarea es desconocido. Una ventaja

adicional del esquema de balanceo de carga dinámico por demanda es su aplicabilidad en clusters con procesadores heterogéneos.

En un esquema de distribución de carga dinámico tradicional, mostrado en la Figura 3.5, el proceso *worker* ejecuta una tarea (tarea n), envía un mensaje al *master* con el resultado (a) y espera del master un nuevo requerimiento (b). Esto genera un tiempo de cómputo ocioso en los procesadores *workers* del sistema.

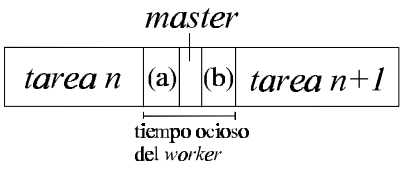

Figura 3.5: Esquema de distribución dinámica de carga

En esta trabajo de tesis se propone una modificación al esquema dinámico tradicional, permitiendo el envío de una tarea a un *worker* aún cuando éste se encuentre procesando un requerimiento. De esta manera, el *worker* no necesita esperar a que el *master* envíe otra tarea, sino que puede continuar realizando cómputo efectivo procesando la tarea que tiene en la cola.

3.4. GS-PFLCN: IMPLEMENTACIÓN Y RESULTADOS

La estructura del programa GS-FLCN sigue el esquema mostrado en la Figura 3.6. La búsqueda de subconjuntos de asignación de orden 4 en adelante es la sección que consume la mayor parte del tiempo de cómputo. Por tal motivo, se consideró conveniente paralelizar esta parte del código, mientras que los restantes procedimientos fueron preservados en sus versiones secuenciales.

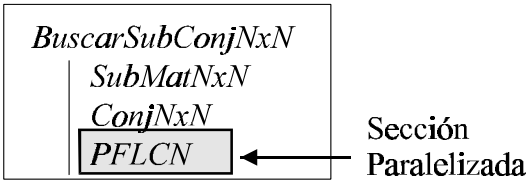

Figura 3.6: Procedimiento *SubConjNxN* que incluye a *pFLCN*

Como fue expuesto en la Sección 3.2.1, cada posible subconjunto de asignación de tamaño *m* puede asociarse a un camino de longitud *m*-1 en el grafo no dirigido correspondiente a la matriz de ocurrencia M^TM. Por lo tanto, se utilizó el algoritmo original de búsqueda en profundidad desarrollado en la Sección 3.3.3, pero acotando el recorrido hasta alcanzar dicha profundidad. Cada vez que se encuentra un camino de longitud *m*-1 se chequea si sus nodos conforman un subconjunto de asignación; en caso afirmativo, la tarea *worker* envía un mensaje al *master* para informar acerca del nuevo subconjunto encontrado, y el *master* envía mensajes al resto de los *workers* para que detengan la exploración. Cuando el chequeo es negativo, el proceso *worker* continúa la búsqueda hasta que todo el subgrafo ha sido explorado, en cuyo caso envía un mensaje al *master* para notificar que está disponible para recibir nuevos requerimientos. En el Apéndice A se describen los algoritmos Master-pFLCN y Worker-pFLCN que muestran la implementación utilizando PVM.

3.4.1. CASOS DE ESTUDIO

El desempeño del algoritmo GS-pFLCN fue establecido sobre la base del análisis de dos casos de estudio correspondientes a plantas industriales, comparándose los tiempos de cómputo paralelo con respecto al algoritmo secuencial existente. El primer ejemplo seleccionado corresponde una planta de síntesis de amoníaco (Bike, 1985) que produce 1500 toneladas diarias de amoníaco mediante el proceso de síntesis denominado

Haber-Bosch. El diagrama de flujo de la planta se muestra en la Figura 3.7. Para este estudio, la planta se representó mediante un modelo matemático de 560 ecuaciones algebraicas no lineales y 516 variables no medidas.

El segundo caso de estudio fue el modelo riguroso de una planta de producción de etano existente en nuestro medio. El proceso puede ser dividido en tres secciones principales: compresión de gas y deshidratación, separación criogénica y fraccionamiento. Se consideraron 87 unidades, 185 corrientes y 12 componentes, generándose un modelo con 1830 ecuaciones y 1425 variables no medidas. El diagrama de flujo de la planta se muestra en la Figura 3.8.

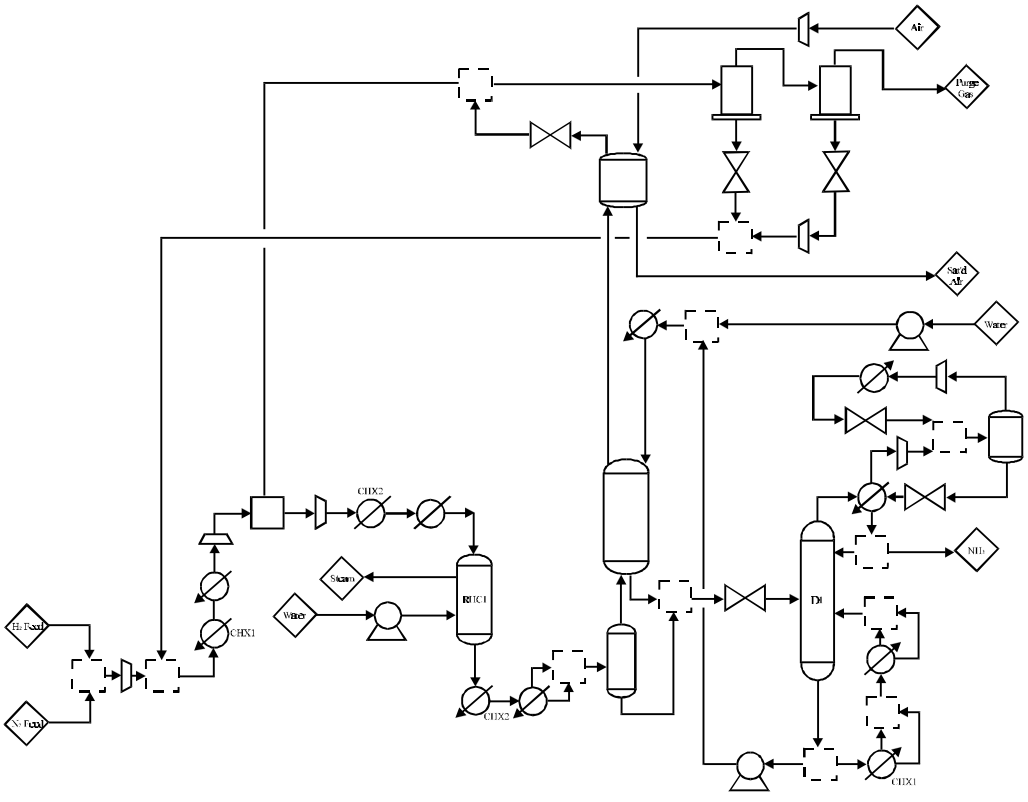

Figura 3.7: Esquema de la planta de síntesis de amoníaco

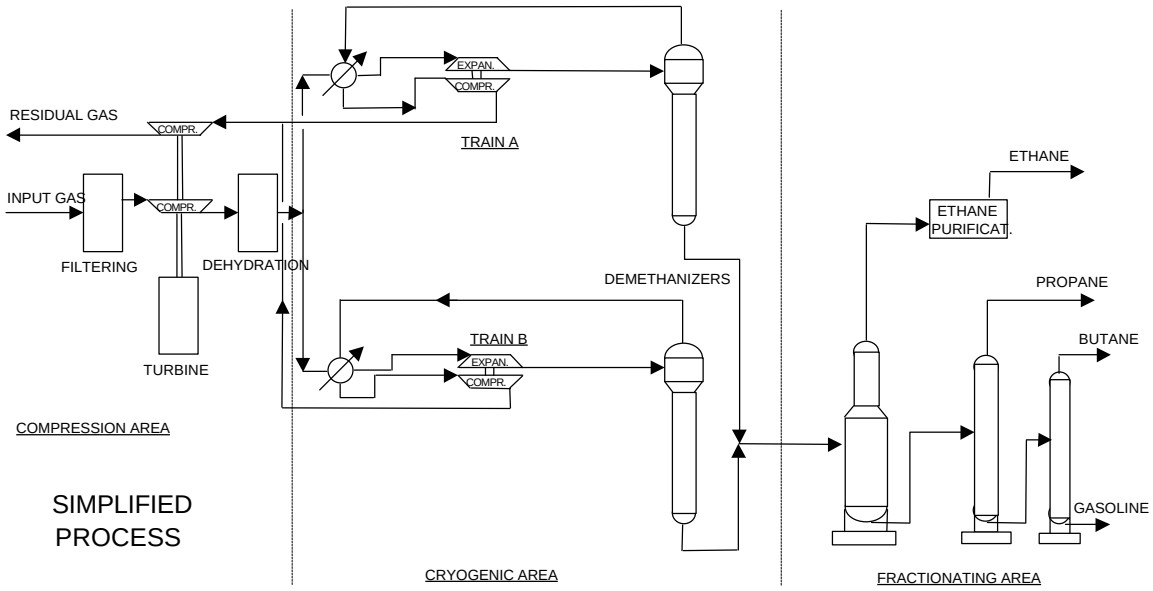

Figura 3.8: Esquema de la planta de producción de etano

Dada la magnitud y complejidad de estos problemas, tanto los sistemas de ecuaciones como las matrices de ocurrencia correspondientes al modelo y las restricciones necesarias para el test de admisión fueron generadas automáticamente mediante el software para construcción de modelos denominado ModGen (Vazquez y col., 2001), desarrollado especialmente en esta tesis. En el Apéndice B se comenta su implementación y la utilización para el desarrollo de estos casos de estudio.

3.4.2. RESULTADOS

Las corridas se realizaron sobre un sistema distribuido homogéneo conformado por 10 computadoras Pentium de 200 MHz con sistema operativo LINUX interconectadas mediante una red de área local Ethernet de 10Mbits/seg. Se comparó el desempeño de GS-pFLCN con su versión secuencial utilizando diferentes cantidades de

nodos de procesamiento, observándose importantes ganancias en tiempos de ejecución. Para alcanzar óptimos resultados la red fue aislada de otros procesos que pudieran interferir con las corridas.

La Tabla 3.1 muestra los tiempos consumidos por las corridas secuenciales y paralelas durante una etapa completa del análisis. Se aprecia que la paralelización permitió lograr una reducción significativa de tiempo para ambos casos de estudio. En la práctica debe tenerse en cuenta que las ganancias son aún más grandes, pues en el transcurso de una tarea de diseño el análisis de observabilidad se lleva a cabo varias veces hasta que se encuentra un conjunto final de variables medidas satisfactorio. En una situación típica, el ingeniero de procesos propone un conjunto inicial de mediciones, obtiene el patrón de observabilidad y analiza si la instrumentación es suficiente para tener un conocimiento del estado de la planta. Si algunas de las variables indeterminables son de interés, el ingeniero agrega o remueve mediciones en las ubicaciones más convenientes y repite el procedimiento. Cada vez que se introducen cambios debe ejecutarse una nueva corrida completa del algoritmo GS-pFLCN.

Las Figuras 3.9 y 3.10 muestran las mejoras en el desempeño logradas gracias a la implementación paralela en términos de speed-up y eficiencia. Las comparaciones de speed-up y eficiencia son justas debido a que en los dos casos de estudio se utilizaron como base los tiempos del algoritmo GS-FLCN secuencial. El uso de GS-pFLCN sobre un único procesador no hubiera sido apropiado como patrón de comparación pues las penalidades asociadas al manejo de tareas podrían conducir a sobrestimaciones artificiales del desempeño.

	Cantidad de procesadores (Algoritmo paralelo)					
	1 (Algoritmo secuencial)	2	4	6	8	10
Planta de Amoníaco	34:19	18:47	09:51	06:58	05:42	04:53
Planta de Etano	58:23	36:54	21.54	17:05	14:25	11:53

Tabla 3.1: GS-pFLCN: Tiempos de procesamiento secuencial y paralelo para los casos industriales

Número de Procesadores	Speed-Up / Eficiencia	
	Planta de Amoníaco	Planta de Etano
2	1.82 / 91%	1.58 / 79 %
4	3.42 / 87%	2.64 / 66 %
6	4.92 / 82%	3.41 / 56 %
8	6.02 / 75%	4.05 / 50 %
10	7.02 / 70%	4.91 / 49 %

Tabla 3.2: GS-pFLCN: Métricas de las corridas obtenidas

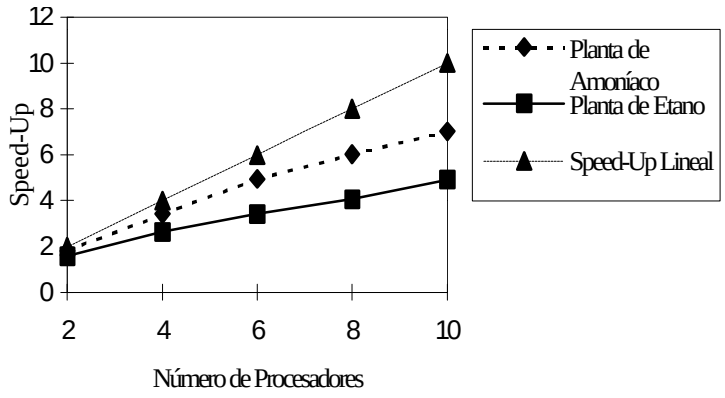

Figura 3.9: GS-pFLCN: Escalabilidad - Evolución del speed-up

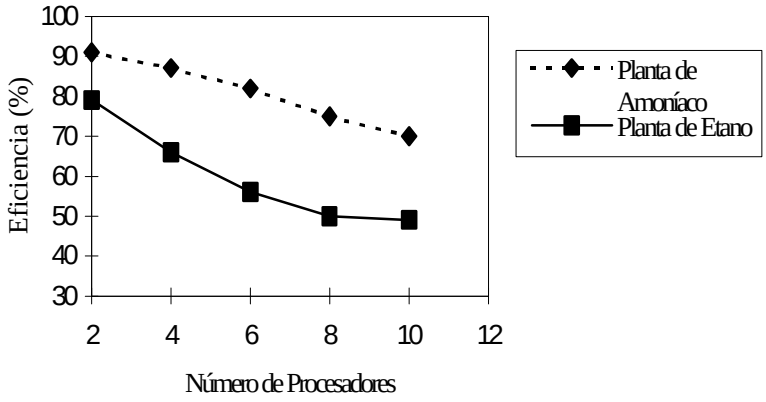

Figura 3.10: GS-pFLCN: Escalabilidad - Variación de la eficiencia

3.4.3. ANÁLISIS DEL DESEMPEÑO

El desempeño del algoritmo paralelo fue satisfactorio a pesar de que el sistema distribuido utilizaba una red de transmisión de datos Ethernet 10Mbits/seg. Deberían esperarse mejores resultados si se emplean redes con mayor ancho de banda y menor latencia.

Sin embargo, no es posible alcanzar una eficiencia del 100% debido a que GS-pFLCN contiene una sección de código secuencial (búsquedas a profundidad menor que 4) y a que existe una penalidad impuesta por el ambiente distribuido. Para las corridas mostradas en la Tabla 3.1, los tiempos insumidos en la sección de código secuencial

fueron de 32.45 y 401.37 segundos para la planta de amoníaco y etano, respectivamente. El primer caso siempre exhibió un mejor desempeño debido a que gran cantidad del tiempo total de cómputo se utilizaba en el procesamiento paralelo de los bloques de orden 4 y mayores. Con respecto a los costos de comunicación, se midió el efecto de la red de transmisión de datos utilizada. La Tabla 3.3 muestra el tiempo utilizado en comunicaciones para los dos casos de estudio con diferentes cantidades de estaciones de trabajo. Es claro que el grado de contención en la red es insignificante; los incrementos en los tiempos de comunicación a medida que se agregan más procesadores son pequeños, por lo tanto existe poca interferencia en la red de datos durante el pasaje de mensajes. Finalmente, es importante puntualizar que la elección de una estrategia dinámica de balanceo de carga por demanda minimiza los tiempos ociosos de los procesadores esclavos.

Cantidad de Procesadores	2	4	6	8	10
Tiempos de comunicación Planta de Amoníaco (seg)	32.15	32.25	32.50	33.10	33.97
Tiempos de comunicación Planta de Etano (seg)	101.1	102.01	103.76	105.12	107.3

Tabla 3.3: GS-pFLCN: Tiempos de comunicación para los casos de estudio

3.4.4. ANÁLISIS DE ROBUSTEZ

Dado que FLCN es de naturaleza combinatorial, los subconjuntos de asignación de mayor tamaño son computacionalmente costosos de detectar. En Ponzoni y col. (1999) se propuso el uso de factores de ramificación, una estrategia de aceleración que consiste en podar de manera inteligente algunas ramas del árbol de búsqueda para alcanzar niveles más profundos en menor tiempo. Estos factores de amplificación ponen una cota

para *k* en el análisis de complejidad, reduciendo así los tiempos de cómputo en detrimento de la robustez dado que algunos potenciales subconjuntos de asignación podrían ser eventualmente omitidos. Un beneficio adicional de la introducción del paralelismo en GS-FLCN es que permite explorar mayor cantidad de subárboles en tiempos razonables. De esta manera se requieren menos podas y se reduce el riesgo de excluir bloques de asignación. En la práctica, esto representa una mejora importante en la robustez de los resultados.

3.5. CONCLUSIONES

El análisis de observabilidad de las variables involucradas en los modelos matemáticos correspondientes a plantas industriales constituye una herramienta esencial para mejorar el diseño de instrumentación y por ende, el control de procesos industriales. En este sentido se consideró la paralelización de una técnica de reordenamiento estructural denominada GS-FLCN basada en una búsqueda sobre grafos. La principal ventaja del método GS-FLCN secuencial era su gran robustez para lograr la clasificación de variables. Este hecho, sumado a la naturaleza combinatoria del método, constituyen los aspectos fundamentales que motivaron su paralelización. Se desarrolló una versión paralela del algoritmo de búsqueda en profundidad que es el corazón de la metodología secuencial y se la incluyó en la versión paralela denominada GS-pFLCN. La implementación del método se efectuó utilizando la librería de pasaje de mensajes PVM (Parallel Virtual Machine). Para efectuar el análisis de desempeño, GS-pFLCN fue utilizado para desarrollar dos casos de estudio industriales reales, lográndose excelentes ganancias en lo que se refiere a tiempos de ejecución con respecto a su versión secuencial.

CAPÍTULO 4: UN ALGORITMO PARALELO DESCENTRALIZADO DE BÚSQUEDA EN GRAFOS

En este capítulo se presenta un nuevo algoritmo paralelo de búsqueda en profundidad sobre grafos. Los problemas sobre los cuales es aplicable este método son aquellos que involucran encontrar todos los caminos sin ciclos de una longitud determinada en un grafo de interés. El requerimiento de caminos sin ciclos significa que los caminos no pueden contener nodos repetidos.

Este algoritmo fue diseñado con el objetivo específico de mejorar el motor de búsqueda del algoritmo paralelo de observabilidad GS-pFLCN presentado en el Capítulo 3. Cabe destacar que naturalmente sus alcances son más amplios, pues el método puede ser utilizado en cualquier otra aplicación que requiera esta clase de búsquedas en grafos.

La principal ventaja de este algoritmo, en comparación con el método de búsqueda presentado en el capítulo anterior, es que evita recorrer caminos repetidos, generando de esta manera búsquedas más eficientes.

4.1. DESCRIPCIÓN DEL MÉTODO

El método de búsqueda propuesto se basa en una nueva forma de paralelización, la cual distribuye las tareas de cómputo sobre tres niveles de jerárquicos: el *master*, los *supervisores* y los *workers*. La tarea *master* organiza la distribución de los distintos subespacios de búsqueda entre los *supervisores*. Cada *supervisor* ordena la exploración

de los subcaminos correspondientes a su subespacio a un *worker* que tiene a su cargo. Este *worker* efectúa el recorrido del subespacio que le fue asignado y envía a su *supervisor* cada uno de los subcaminos hallados. Por último, el proceso *supervisor* se encarga de recombinar el subcamino obtenido de su *worker* con los subcaminos almacenados por otros *supervisores*. Finalmente, cada camino es chequeado para ver si satisface las condiciones requeridas para ser solución, las cuales están asociadas a cada problema en particular. En caso afirmativo, el camino aceptado es enviado al *master*. Desde un punto de vista general, esta propuesta apunta a una descentralización en la construcción de los caminos, incluyendo actividades de recombinación de subcaminos. La verificación que chequea si un camino es solución del problema también está descentralizada, dado que dicha tarea es llevada a cabo por los *supervisores*.

4.2. COMUNICACIÓN Y SINCRONIZACIÓN ENTRE TAREAS

La comunicación de cualquier tarea *worker* es bidireccional y se limita a la tarea *supervisor* asociada a ella. La tarea *master*, en cambio, tiene comunicación sólo con las tareas *supervisores*. Por último, cada tarea *supervisor* tiene canales de comunicación con la tarea *master*, la tarea *worker* subordinada a ella y cualquier otra tarea en el mismo nivel jerárquico, es decir, cualquier tarea *supervisor*.

En la Figura 4.1 se esquematiza la arquitectura de un sistema que implementa este método.

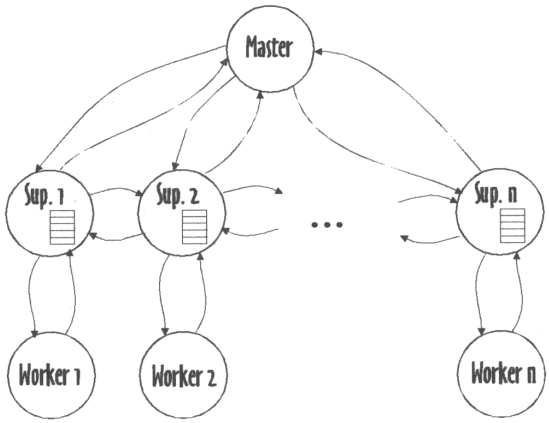

Figura 4.1: Arquitectura del sistema MSW

Cabe destacar que, dado que cada *worker* sólo intercambia información con el *supervisor* asociado, es conveniente usar el mismo nodo de procesamiento para ejecutar ambas tareas. Esto permite reducir sensiblemente las demoras de pasajes de mensajes, pues el flujo de comunicación entre un *supervisor* y su *worker* siempre es alto.

4.2.1. DISTRIBUCIÓN DE TAREAS PARA LA EXPLORACIÓN DE SUBÁRBOLES

La política de balanceo de carga es dinámica y por demanda. La tarea *master* mantiene información actualizada sobre los nodos raíces de todas las exploraciones actuales y terminadas, además de una cola con prioridad que contiene todos los subárboles pendientes que deben ser explorados. Los *supervisores* demandan al *master* un nuevo nodo raíz a medida que los *workers* finalizan la exploración asignada. Este orden de procesamiento es dinámico debido a que ni la secuencia ni el tiempo utilizado en la exploración se conocen de antemano. Estas características son propias de los clusters de estaciones de trabajo y se fundamentan en la heterogeneidad de los procesadores y nivel de utilización de los nodos de procesamiento por parte de otros

usuarios. En este marco, la técnica de balanceo de carga dinámica por demanda es la más apropiada.

4.2.2. CONSTRUCCIÓN DE CAMINOS

La tarea *master* es la encargada de la distribución de los subespacios de exploración entre los distintos *supervisores*, los cuales delegan la exploración efectiva del grafo a la tarea *worker* dependiente de ellos. Los mensajes intercambiados durante las exploraciones se muestran en la Figura 4.2. A medida que el *worker* W_1 encuentra un subcamino, esta información se reporta al *supervisor* W_1 asociado (1), donde se almacena en un registro interno. Este subcamino será la primera mitad de un camino completo, que se formará cuando S_1 lo envíe al *supervisor* S_2 que tiene almacenada la segunda mitad del camino (5). S_2 concatena ambas mitades para construir el camino de la longitud deseada (6). En otras palabras, cada subcamino cuya hoja es *c* es enviado al *supervisor* responsable de la exploración del subárbol cuyo nodo raíz es *c*. Una vez que se ha obtenido un camino completo, se procede a verificar, dentro de la misma tarea *supervisor* su viabilidad como solución para la clase de problemas que se está analizando.

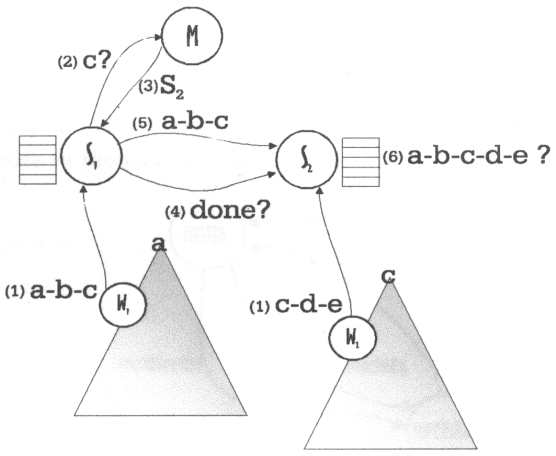

Figura 4.2: Construcción de caminos

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* cambia las prioridades de acuerdo con los mensajes que recibe de los *supervisores*, quienes lo consultan constantemente acerca del identificador de tarea (*task id*) del *supervisor* que tiene a cargo la exploración de la otra mitad del subcamino (flechas (2) y (3) en la Figura 4.2). Si en el momento de la consulta este nodo no ha sido asignado para ser explorado, 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 a explorar. Así, la búsqueda se optimiza porque se concentra en la exploración de subárboles asociados a muchos caminos completos.

4.2.3. PROTOCOLO DE FINALIZACIÓN

El algoritmo *master-supervisor-worker* (MSW) propuesto puede ser usado para encontrar el primer camino, o bien todos los caminos de una longitud fija que verifican una determinada propiedad en un grafo. En el primer caso, cuando se encuentra el primer camino, la tarea *master* recibe la correspondiente notificación y envía un mensaje de terminación a todos los *supervisores*; cada *supervisor* lo propaga hacia el *worker* subordinado y el algoritmo termina. En caso de no se encuentre ningún camino, o se haya alcanzado el final de un algoritmo que busca todos los caminos posibles, la terminación se lleva a cabo con un protocolo de finalización de dos etapas, tal como se ilustra en la Figura 4.3. En la primera, el *worker* que finalizó la exploración del subgrafo asignado reporta este evento a su *supervisor* (*0:WorkerDoneSignal*). Inmediatamente el *supervisor* requiere al *master* un nuevo nodo raíz para utilizar en una nueva exploración (*1:NextNodeQuery*). Cuando el *master* no tiene más nodos en su cola, informa al

supervisor de esta situación (*2:NoMoreNodes*). En este caso, el *supervisor* indica a su *worker* que finalice su ejecución (*3:StopSignal*) y se queda procesando su estructura de registros de camino interna hasta que todos los subcaminos han sido enviados a otros *supervisores* para recombinar. La segunda fase comienza tan pronto como el *supervisor* terminó de enviar todos los subcaminos de su registro interno. En ese momento el *supervisor* envía un mensaje de intención de finalización al *master* (*4: ReadyToCommit*). Cuando el *master* recibe el mismo mensaje de todos los *supervisores*, entonces les comunica el permiso de finalización (*5: CommitGranted*) y termina con su ejecución. Esta descripción corresponde a la interacción completa de un *supervisor*; el mismo protocolo se aplica para todos los *supervisores*.

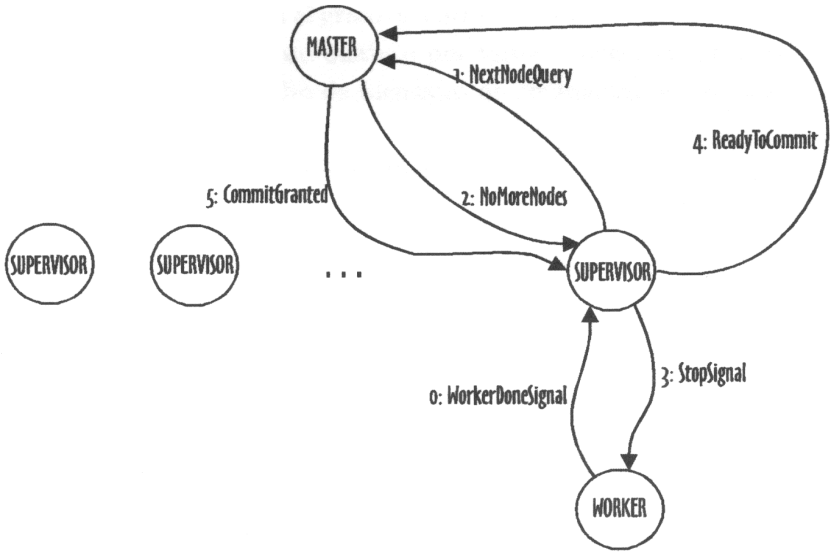

Figura 4.3: Protocolo de finalización

4.3. ESTRUCTURAS DE DATOS

Los requerimientos de memoria impuestos por el esquema MSW son mucho mayores que los del algoritmo presentado en la Sección 3.3 porque es necesario mantener información de los subcaminos explorados en forma descentralizada. Debido a la cantidad de información que utiliza la estrategia MSW, su algoritmo debe ser implementado con estructuras de datos eficientes.

La estructura más importante y compleja que se utiliza se encuentra en las tareas *supervisor* y es la que permite mantener el registro de los subcaminos encontrados. La estructura es básicamente un arreglo lineal de árboles, donde el *i*-ésimo componente contiene el subárbol explorado cuya raíz es *i*. Estos subárboles han sido implementados con la representación hijo extremo izquierdo - hermano derecho (HEI-HD) con punteros, de manera que la configuración pueda admitir cualquier cantidad de hijos por nodo. La elección de esta estructura de datos responde a un compromiso entre espacio de almacenamiento y velocidad de acceso.

La Figura 4.4 muestra esta estructura gráficamente para un grafo de *n* nodos. El *supervisor* representado en este ejemplo ha estado a cargo de la exploración de algunos subárboles, entre los cuales se encuentran los que comienzan con raíces 1 y *n*-1. Los componentes asociados a nodos raíces que son explorados por otros *supervisores* corresponden a subárboles vacíos en el registro de este supervisor.

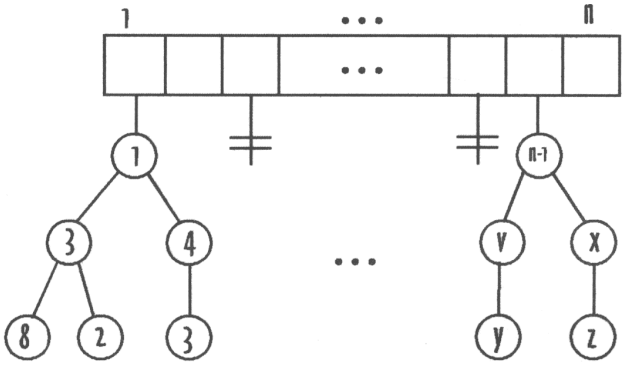

Figura 4.4: Estructura del registro de subcaminos

Además de la estructura para almacenar los subcaminos, existen otras estructuras de datos auxiliares más simples que son necesarias. En particular, el *supervisor* mantiene las siguientes:

- Contador de número de caminos finalizados en cada hoja: mantiene una cuenta del número de subcaminos en el registro interno que terminan en cada nodo. Este contador es útil para decidir qué subcaminos deben ser enviados para recombinar.

- Estructura de confirmaciones: su función principal es confirmar que todos los caminos enviados a otro *supervisor* para recombinar fueron procesados. Incluye un contador que lleva un registro de los mensajes de generación de subcaminos intercambiados entre cualquier par de *supervisores* y registra cada confirmación.

- Vector de alocación de tareas local: mantiene información actualizada sobre la distribución de los subespacios de búsqueda en forma empaquetada. La primera componente del par identifica la tarea, mientras que la segunda indica la terminación de la exploración. Cada *supervisor* mantiene su propio vector de alocación de tareas local.

La tarea *master* utiliza dos estructuras de datos:

- Cola con prioridad: contiene todos los nodos que aún no han sido explorados. Los nodos avanzan una posición en la cola cada vez que son requeridos por otros *supervisores*.
- Vector de alocación de tareas: tiene la misma función y esquema de almacenamiento que el vector de alocación de tareas local descripto para la tarea *supervisor*.

4.4. ANÁLISIS DE DESEMPEÑO

Para evaluar el desempeño del algoritmo de búsqueda MSW propuesto se lo implementó en C utilizando la librería de pasaje de mensajes PVM, comparándose su desempeño con el del algoritmo paralelo de búsqueda en profundidad presentado en el Capítulo 3. Dado que el objetivo primario era utilizarlo en problemas de observabilidad, se generaron diversos grafos de prueba de diferentes tamaños con un 5% de densidad de arcos, porque este es un valor típico en este tipo de problemas. La Tabla 4.1 muestra los tiempos de ejecución requeridos por los dos algoritmos corriendo en un cluster homogéneo compuesto por 4 PCs (Pentium 200Mhz y sistema operativo Linux) interconectadas por medio de una red Ethernet 10Mbit/seg. En ambos casos, el objetivo fue explorar los grafos para encontrar todos los caminos de longitud 10. La cantidad de nodos de cada uno de los grafos de prueba corresponde a casos de interés práctico. En tal sentido es importante destacar que el modelo matemático de una planta industrial pequeña normalmente conduce a grafos del orden de 500 nodos, mientras que los problemas típicos de tamaño mediano o grande tienen al menos 1000 nodos o más.

Cantidad de nodos del grafo	Tiempo de ejecución (seg.)		Ganancia (%)
Algoritmos	Master-Worker (MW)	Master – Supervisor – Worker (MSW)	
50	15	18	-2.0
75	43	39	9.3
100	73	61	16.4
500	418	249	40.4
1000	1167	676	42.1

Tabla 4.1: Tiempos de ejecución requeridos por las estrategias MW y MSW

La primera columna de la Tabla 4.1 indica el tamaño del grafo utilizado para la corrida paralela. La segunda muestra el tiempo de ejecución utilizando el algoritmo de búsqueda presentado en la Sección 3.3, al cual hemos denominado MW dada su jerarquía Master-Worker. Análogamente, la tercera columna corresponde a los tiempos de ejecución de algoritmo MSW. La última columna muestra la ganancia en tiempo del algoritmo MSW sobre el MW expresada en porcentajes. Con respecto a la calidad de los resultados, cabe aclarar que la comparación es justa pues las dos implementaciones encontraron los mismos conjuntos de caminos solución para cada instancia de problema.

Exceptuando los grafos de tamaño pequeño, la estrategia MSW fue siempre más eficiente en términos de tiempo de ejecución. A partir de la Tabla 4.1 es claro que las ganancias son más significativas a medida que el tamaño de los problemas aumenta. Por esta razón, el método es especialmente atractivo para ser incorporado en paquetes de diseño de instrumentación. Las razones por las cuales no existen ganancias para grafos pequeños es que estos grafos contienen pocos nodos repetidos. Así, la sobrecarga de inicialización de los procesos y pasajes de mensajes prevalece sobre las ganancias en el tiempo efectivo de búsqueda. De todos modos, los problemas asociados a grafos chicos

no son de interés práctico, pues los problemas de observabilidad clásicos que se resuelven utilizando computadoras nunca resultan tan pequeños.

Con respecto al efecto de la aplicación de la estrategia descentralizada en búsquedas sobre grafos con densidad de arcos mayores, es natural prever que se lograrán mayores ganancias con respecto a la estrategia MW. Esto se debe a que existe una mayor cantidad de nodos repetidos en los caminos a medida que la densidad de árbol aumenta.

4.5. CONCLUSIONES

En este capítulo se presentó un nuevo algoritmo para encontrar todos los caminos de una longitud determinada en un grafo no dirigido mediante búsquedas en profundidad. La ventaja más importante de este enfoque con respecto al método paralelo de búsqueda descripto en la Sección 3.3 es la significativa reducción en los tiempos requeridos para realizar la búsqueda. Esto se logra gracias a que el algoritmo evita múltiples exploraciones de un mismo subgrafo mediante el registro de cada subcamino encontrado la primera vez que se explora, incluyendo además actividades de recombinación de los subcaminos registrados. También se obtiene una reducción sustancial en los tiempos de exploración de cada subgrafo, ya que cada tarea worker considera caminos de la mitad de longitud que la de los caminos originales.

Aunque la arquitectura es simple con respecto al balanceo de carga, los requerimientos de coordinación son más exigentes. Los protocolos de comunicación y sincronización son más complejos, razón por la cual se debió diseñar con especial cuidado el protocolo de finalización y la estructura de datos para el almacenamiento de los subcaminos.

Si bien el objetivo original fue desarrollar un nuevo algoritmo para encontrar caminos de una determinada longitud en grafos no dirigidos con el propósito de mejorar la implementación de GS-pFLCN, debe notarse que la utilidad de esta nueva técnica no está limitada al campo del diseño de instrumentación de plantas de procesos. Dado que una gran variedad de problemas que surgen en diferentes ámbitos puede ser modelada mediante grafos, el algoritmo MSW presentado podría ser utilizado en esas aplicaciones, previéndose mejoras de similar magnitud.

CAPÍTULO 5: ALGORITMOS PARALELOS PARA OPTIMIZACION NUMERICA

Un problema de ***optimización*** consiste básicamente en encontrar un conjunto de valores apropiados que obtienen el máximo rendimiento para un objetivo definido en un problema particular, cumpliendo al mismo tiempo un conjunto de condiciones establecidas. La optimización de modelos es utilizada en diversas áreas tales como economía, biología, ingeniería, medicina, etc. Dentro del campo de la ingeniería de procesos se aplica en el diseño de equipos, diseño de plantas completas, síntesis de procesos, control óptimo, etc. Los propósitos de la optimización pueden ser económicos (maximizar el rendimiento de la producción, reducir los costos), de producción (obtener mayor cantidad de productos procesados), de seguridad (minimizar el riesgo de operación), etc.

A menudo, los problemas de optimización que surgen en aplicaciones reales insumen una gran cantidad de tiempo de cómputo. En determinadas ocasiones la demora para obtener la solución no es aceptable; un ejemplo de esto son los problemas de optimización que son utilizados en aplicaciones *on-line*. Por esta razón el desarrollo de algoritmos paralelos de optimización constituye una actividad atractiva tanto en el ámbito científico como en la producción.

En este capítulo se presenta un nuevo algoritmo paralelo de optimización basado en descomposición de dominios y se comprueba su efectividad aplicándolo a diversos problemas representativos en ingeniería.

5.1. EL PROBLEMA DE OPTIMIZACIÓN

Desde un punto de vista más formal, un problema general de optimización puede ser expresado como la búsqueda de un mínimo (o máximo) de una función real $f(\boldsymbol{x})$ con argumento $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, cuyos valores deben satisfacer un conjunto de ecuaciones $h(\boldsymbol{x})_i = 0$, un conjunto de inecuaciones $g(\boldsymbol{x})_j \geq 0$ y los límites $\boldsymbol{x}_k^U \geq \boldsymbol{x}_k \geq \boldsymbol{x}_k^L$, $\boldsymbol{x}_k^U \in \Re$ y $\boldsymbol{x}_k^L \in \Re$. Tradicionalmente, la función $f(\boldsymbol{x})$ se denomina función objetivo y las componentes del vector $\boldsymbol{x}$ se conocen como variables de optimización o de decisión, mientras que las expresiones $h(\boldsymbol{x})_i = 0$ y $g(\boldsymbol{x})_j \geq 0$ se denominan restricciones de igualdad y desigualdad respectivamente. Si bien los límites de las variables de optimización $\boldsymbol{x}_k^U$ y $\boldsymbol{x}_k^L$ son casos especiales de restricciones de desigualdad, la diferencia conceptual en el modelamiento y su tratamiento especial en los distintos algoritmos de optimización permiten que sean tratados como una clasificación propia.

La notación matemática para este problema general es:

$$\begin{aligned} &Min \quad f(\boldsymbol{x}) \\ &sujeto\ a \\ &\quad h_i(\boldsymbol{x}) = 0 \qquad i = 1, ..., ri \\ &\quad g_j(\boldsymbol{x}) \geq 0 \qquad j = 1, ..., rd \\ &\quad \boldsymbol{x}_k^U \geq \boldsymbol{x}_k \geq \boldsymbol{x}_k^L \qquad k = 1, ..., n \end{aligned} \tag{5.1}$$

donde *n*, *ri* y *rd* son la cantidad de variables de optimización, de restricciones de igualdad y de restricciones de desigualdad respectivamente.

5.2. CLASIFICACIÓN DE PROBLEMAS DE OPTIMIZACIÓN

Si bien la expresión (5.1) modela de manera general a los problemas de optimización, éstos pueden clasificarse en diferentes categorías dependiendo del número de restricciones involucradas y del tipo de funciones usadas en el modelo.

La clasificación más general corresponde a la cantidad de restricciones involucradas. De esta manera, el problema (5.1) se denomina problema de optimización *con restricciones* (o *restringido*). Si en la expresión (5.1) no hay restricciones, es decir *ri* = *rd* = 0 y $\left|\boldsymbol{x}_k^U\right| = \left|\boldsymbol{x}_k^L\right| = \infty, k = 1,..., n$, el problema se denomina *sin restricciones* (o *no restringido*).

Luego podemos clasificar a los problemas de optimización según la estructura de las funciones f, h_i y g_j, y según la dimensión de $\boldsymbol{x}$. Si tenemos un problema de optimización sin restricciones donde el vector *x* tiene una sola componente, es decir $\boldsymbol{x} \in \Re$, entonces el problema se denomina *univariable* y constituye la subclase más simple, aunque no menos importante. Los problemas con restricciones donde todas las funciones h_i y g_j son lineales se denominan *con restricciones lineales* (o *linealmente restringidos*); cuando además la función objetivo *f* es lineal entonces la expresión (5.1) se denomina *problema de optimización lineal*.

La clasificación dada anteriormente asume que las variables del problema son números reales. Existen subsecuentes clasificaciones que consideran otros dominios para $\boldsymbol{x}$ (por ejemplo enteros, valores booleanos, etc.) y que no se mencionan por estar fuera del alcance de esta tesis. Para información adicional referirse a Reklaitis y col. (1983) .

5.3. ALGORITMOS PARALELOS DE OPTIMIZACIÓN

Con respecto a los algoritmos de optimización paralelos, existen dos tendencias diferenciadas:

(a) Algoritmos de optimización intrínsecamente paralelos.

(b) Algoritmos de optimización secuenciales que son transformados en paralelos.

En el caso (a) la estructura algorítmica es directamente paralelizable. Esto se logra cuando un problema posee una estructura especial o mediante una transformación que permite su tratamiento en forma paralela. En cualquier caso, el objetivo final es realizar el procesamiento sobre diferentes conjuntos de datos al mismo tiempo. Mangasarian (1995) y Mittelman (1996) presentaron una metodología que consiste en particionar el vector de variables de optimización en p bloques y definiendo p subproblemas asociados. Cada uno de ellos se resuelve en paralelo con una técnica tipo Jacobi y estos resultados intermedios se utilizan para producir una mejor solución, repitiéndose este procedimiento hasta lograr la convergencia. Ferris y Mangasarian (1994) continuaron con esta línea definiendo el enfoque de Distribución Paralela de Variables (DPV). A diferencia de los métodos mencionados anteriormente, donde los bloques de variables manejados por los otros procesadores se mantienen constantes durante la ejecución de cada subproblema en paralelo, DPV permite que estos bloques se muevan en direcciones arbitrarias. En todos estos trabajos se consideraron problemas sin restricciones.

En el caso (b), el algoritmo paralelo se desarrolla modificando el código secuencial original, introduciendo el paralelismo en diferentes secciones de cómputo intensivo. Dennis y Torczon (1991) diseñaron un método paralelo para búsqueda multidireccional, que luego evolucionó hacia técnicas de búsquedas con patrones

(Hough y col., 2001). La principal desventaja de estos métodos es el desaprovechamiento de información brindada por las derivadas de las función objetivo y las restricciones para encontrar el óptimo de manera eficiente. Además, estos trabajos han sido desarrollados como herramientas de propósito general, no habiéndose reportado experiencias sobre aplicaciones específicas en ingeniería. En cuanto a los métodos tipo gradiente, High y LaRoche (1995) introdujeron el paralelismo en un método de optimización SQP (Successive Quadratic Programming) mediante la evaluación simultánea de la función objetivo y las restricciones. Nuevamente, todas las implementaciones citadas en este párrafo fueron diseñadas especialmente para correr sobre computadoras paralelas, existiendo muy pocos antecedentes de algoritmos paralelos concebidos para su ejecución sobre clusters.

Es claro que los dos enfoques poseen aspectos favorables. Por un lado, los métodos de optimización intrínsecamente paralelos tienen la ventaja poseer una estructura algorítmica pensada en términos del paradigma paralelo y por lo tanto su implementación sobre sistemas de cómputo paralelos es más “elegante”. Por otro lado, los algoritmos secuenciales transformados en paralelos tienen la ventaja de estar basados en conceptos matemáticos robustos y probados.

En vista de las características salientes de ambos enfoques, en este capítulo se desarrollará en profundidad el estudio de los algoritmos intrínsecamente paralelos, proponiéndose un nueva metodología para el particionamiento del dominio y el tratamiento de problemas no lineales, mientras que en el Capítulo 6 se abordarán los aspectos de paralelización de algoritmos secuenciales.

5.4. ALGORITMOS INTRINSECAMENTE PARALELOS

Una de las formas de resolver problemas de optimización en forma paralela es aprovechando estructuras especiales del modelo matemático. En muy pocos casos el conjunto de restricciones de la formulación (5.1) puede descomponerse en *p* subconjuntos independientes, donde cada uno de ellos contienen variables que pertenecen al bloque correspondiente en la forma particionada $\boldsymbol{x}^T=[\boldsymbol{x_1}^T|\boldsymbol{x_2}^T|...|\boldsymbol{x_l}^T|...|\boldsymbol{x_p}^T]$. Si además la función objetivo *f* puede ser expresada como $\sum_{i=1}^{p} f_i(\boldsymbol{x_i})$ en (5.1), el problema se denomina separable en bloques y los subproblemas de optimización resultantes:

$$\begin{aligned} &\textit{Min} && f_i(\boldsymbol{x}) \\ &\textit{sujeto a} && \\ & && h_*^i(\boldsymbol{x_i}) = 0 \qquad i := 1,..., p \\ & && g_*^i(\boldsymbol{x_i}) \geq 0 \\ & && \boldsymbol{x_*^{Ui}} \geq \boldsymbol{x_*^i} \geq \boldsymbol{x_*^{Li}} \end{aligned} \tag{5.2}$$

pueden ser resueltos en paralelo de manera independiente en *p* procesadores. Si bien la paralelización en este caso es muy efectiva, la principal desventaja es que en la práctica no existen problemas que puedan ser modelados de forma separable en bloques.

Otro caso surge cuando la función objetivo es lineal o cuadrática separable y las restricciones son lineales. Estos problemas pueden resolverse usando un algoritmo general de punto interior de programación lineal o cuadrática (Censor y Zenios, 1997). El mayor costo computacional de estos algoritmos aparece en la factorización y solución del sistema de ecuaciones para el vector de incógnitas D$\boldsymbol{y}$ de la forma:

$$(\boldsymbol{A}\mathrm{q}\boldsymbol{A}^T)\mathrm{D}\boldsymbol{y} = \mathrm{y} \tag{5.3}$$

donde $\boldsymbol{A}$ es la matriz de restricciones del problema, **y** es un vector, **q** es una matriz diagonal y D**y** es una dirección del algoritmo de punto interior. Un caso especial de interés es cuando $\boldsymbol{A}$ posee bloques en la diagonal y bloques identidad en la fila inferior (5.4); esta disposición corresponde a aplicaciones de optimización de redes de costos. Otro caso es cuando $\boldsymbol{A}$ posee bloques en la diagonal y en la primer columna (5.5) y surge en el tratamiento de problemas de programación estocástica de dos fases. Para estos casos particulares es posible aplicar métodos específicos para resolver (5.3) utilizando las matrices de tamaño reducido $\boldsymbol{A_1}, \ldots, \boldsymbol{A_K}$, o $\boldsymbol{A_0}, \boldsymbol{W_1}, \ldots, \boldsymbol{W_N}, \boldsymbol{T_1}, \ldots, \boldsymbol{T_N}$,.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A_1} & & & \\ & \boldsymbol{A_2} & & \\ \vdots & & \ddots & \\ & & & \boldsymbol{A_K} \\ \boldsymbol{I} & \boldsymbol{I} & \ldots & \boldsymbol{I} \end{pmatrix} \tag{5.4}$$

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A_0} & & & \\ \boldsymbol{T_1} & \boldsymbol{W_1} & & \\ \vdots & & \ddots & \\ \boldsymbol{T_N} & & & \boldsymbol{W_N} \end{pmatrix} \tag{5.5}$$

Estos subsistemas son independientes y pueden ser resueltos en paralelo. Así, el paralelismo no se encuentra explícitamente en el algoritmo de punto interior sino en los cálculos algebraicos que se realizan cuando se aplica a estos problemas con estructuras especiales. Naturalmente el principal inconveniente de esta metodología es su escaso rango de aplicación.

Otro método intrínsecamente paralelo mencionado es el de descomposición de dominios y es aplicable a problemas no lineales sin restricciones (Bertsekas y Tsitsiklis, 1997). La característica principal de esta metodología es que resuelve subproblemas de

menor dimensión mediante el particionamiento del dominio de búsqueda, mientras que un proceso maestro utiliza estos resultados para estimar el óptimo del problema original utilizando un esquema de actualización tipo Jacobi. Esta solución parcial permite definir nuevos subproblemas y el proceso se repite iterativamente hasta lograr la convergencia. Los subproblemas y el proceso maestro son de menor dimensión que el problema original; de esta manera, aún si son necesarias varias iteraciones para alcanzar la solución, se espera que el algoritmo de descomposición sea más rápido con respecto a un algoritmo que ataca directamente el problema original.

Como ha sido expuesto, todas las metodologías poseen limitaciones para el tratamiento de problemas no lineales generales que surgen en ingeniería. Para alcanzar este objetivo, en este trabajo se propone el uso de un algoritmo de descomposición de dominios basado en la técnica DPV (Ferris y Mangasarian, 1994), ampliando su uso para problemas generales (5.1) mediante el uso de funciones de penalidad y una nueva estrategia para el particionamiento de los bloques.

5.5. UN ALGORITMO DE DESCOMPOSICION DE DOMINO - DISTRIBUCION PARALELA DE VARIABLES.

Consideremos primeramente el siguiente problema de programación:

$$\begin{array}{ll} Min & f(\boldsymbol{x}) \\ sujeto\ a & \\ & \boldsymbol{x} \in X \end{array} \tag{5.6}$$

donde $X \hat{I} \Re^n$ es un conjunto convexo cerrado y $f : \Re^n \to \Re$ tiene derivadas parciales primeras continuas en $\Re^n$. El método de distribución paralela de variables (DPV) fue propuesto originalmente para resolver problemas de optimización sin restricciones

(Ferris y Mangasarian, 1994). Su principal característica es que el vector de variables de optimización es particionado en *p* bloques $\boldsymbol{x}^T = [\boldsymbol{x_1}^T | \boldsymbol{x_2}^T | \ldots | \boldsymbol{x_l}^T | \ldots | \boldsymbol{x_p}^T]$, los cuales se distribuyen en *p* procesadores, donde $\boldsymbol{x_l} \in \Re^{n_l}, \sum_{l=1}^{p} n_l = n$. Dada una iteración $\boldsymbol{x^i} \in \Re^n$, el procesador *l* es el encargado de actualizar el bloque de variables $\boldsymbol{x^i_l} \in \Re^{nl}$ resolviendo el problema:

$$\begin{aligned} (\boldsymbol{y^i_l}, \boldsymbol{\quad}^{i}_{\bar{l}}) \in Min_{x_l,\ \ddot{e}} \quad & \boldsymbol{f}(\boldsymbol{x_l}, \boldsymbol{\quad}^{i}_{\bar{l}} \boldsymbol{D} \boldsymbol{\quad} \ddot{\boldsymbol{e}}^{i}_{\bar{l}} \boldsymbol{\quad}_{\bar{l}}) \\ \textit{sujeto a} \quad & (\boldsymbol{x_l}, \boldsymbol{x^i}_{\bar{l}} + \boldsymbol{D^i}_{\bar{l}} \boldsymbol{\quad}_{\bar{l}} \ddot{\boldsymbol{e}} \boldsymbol{\quad}) \in X \\ & \boldsymbol{x^{i_l}} := (\boldsymbol{y^i}_l, \boldsymbol{x^i}_{\bar{l}} + \boldsymbol{D^i}_{\bar{l}} \boldsymbol{\quad}_{\bar{l}} \ddot{\boldsymbol{e}} \boldsymbol{\quad}) \end{aligned} \tag{5.7}$$

Aquí $\bar{l}$ denota el complemento de *l* en $\{1,..., p\}$, $\ddot{\boldsymbol{e}}_{\bar{l}} \in \Re^{p-1}$. La matriz $\boldsymbol{D^i}_{\bar{l}}$ es una matriz diagonal en bloques de $n_{\bar{l}} \times n_{\bar{l}}$ que se construye tomando direcciones arbitrarias $\boldsymbol{d^i} \in \Re^n$ y separándola en bloques de $\boldsymbol{d^i_s} \in \Re^{ns}$, s = 1,...,*p*. Estos vectores forman la estructura diagonal de $\boldsymbol{D^i}_{\bar{l}}$ de la siguiente manera:

$$\boldsymbol{D^i}_{\bar{l}} := \begin{bmatrix} \boldsymbol{d^i}_1 & & & & & & \\ & \ddots & & & & & \\ & & \boldsymbol{d^i}_{i-1} & & & & \\ & & & \boldsymbol{d^i}_{i+1} & & & \\ & & & & \ddots & & \\ & & & & & \boldsymbol{d^i}_p & \end{bmatrix} \tag{5.8}$$

Una vez que las soluciones intermedias $\boldsymbol{x^{il}}$, *l* = 1,..., *p* han sido computadas por los procesadores en paralelo, el conjunto de puntos $\{\boldsymbol{x^i}, \boldsymbol{x^{il}}, ..., \boldsymbol{x^{ip}}\}$ es explorado para encontrar una aproximación a la solución mejorada mediante una etapa de sincronización que puede ser expresada como:

$$
\begin{aligned}
&\boldsymbol{x}^{i+1} = m^{i}{}_{0}\boldsymbol{x}^{i} + \sum_{k=1}^{p} m^{i}{}_{k}\boldsymbol{x}^{i_k} \\
&(m^{i}{}_{0}, m^{i}{}_{1}, \ldots, m^{i}{}_{p}) \in Min_{m^{i}{}_{0}, m^{i}{}_{1}, \ldots, m^{i}{}_{p}} \quad f(m^{i}{}_{0}\boldsymbol{x}^{i} + \sum_{k=1}^{p} m^{i}{}_{k}\boldsymbol{x}^{i_k}) \\
&\qquad \textit{sujeto a} \quad m^{i}{}_{0}\boldsymbol{x}^{i} + \sum_{k=1}^{p} m^{i}{}_{k}\boldsymbol{x}^{i_k} \in X \\
&\qquad\qquad m_0 + \sum_{k=1}^{p} m_k = 1
\end{aligned}
\tag{5.9}
$$

Las propiedades de convergencia del método sólo requieren que la etapa de sincronización genere un punto cuyo valor en la función objetivo no es peor que alguno de los ya generados por los procesadores. A diferencia de otras técnicas paralelas para optimización tales como multisplitting (Mittelmann, 1996) o distribución paralela del gradiente (Mangasarian, 1995), la característica distintiva de DPV es la presencia del término $\boldsymbol{x}^{i}_{\bar{I}} + \boldsymbol{D}^{i}_{\bar{I}}{}_{I}\dot{\boldsymbol{\epsilon}}$ en (5.7). Así, en lugar de mantener constantes los valores de los bloques de variables $\boldsymbol{x}_{\bar{I}}$, los subproblemas de sincronización permiten a estos bloques moverse en direcciones arbitrarias $\boldsymbol{D}^{i}_{\bar{I}}$. Estas direcciones pueden ser generadas, por ejemplo, usando los métodos clásicos de descenso más abrupto o quasi-Newton (Reklaitis y col., 1983) en el espacio de estas variables.

Otra característica interesante de DPV es su implementación sencilla. DPV refleja el paradigma clásico maestro-trabajador de manera directa. Los trabajadores, que se ejecutan en diferentes procesadores, resuelven los subproblemas (5.7) en paralelo. Los resultados son enviados a un proceso maestro, que genera una nueva aproximación a la solución utilizando la etapa de sincronización (5.9) y chequea el criterio de convergencia.

5.6. TRATAMIENTO DE PROBLEMAS CON FUNCION OBJETIVO Y RESTRICCIONES NO LINEALES.

La extensión de la formulación básica de este método para el tratamiento de problemas no lineales no es directa. En muy pocos casos el conjunto de restricciones de la formulación (5.1) es separable en bloques. Para resolver este problema, una alternativa que se propone en esta tesis es transformar la formulación no lineal general en un problema de optimización sin restricciones mediante un modificador para luego resolver el problema simplificado en forma paralela.

Uno de los modificadores más conocidos es la función de penalidad (Reklaitis y col., 1983):

$$\begin{aligned} &Min \;\; P(\boldsymbol{x}, R) = f(\boldsymbol{x}) + \Omega(R, g(\boldsymbol{x}), h(\boldsymbol{x})) \\ &sujeto\ a \\ &\qquad \boldsymbol{x}_k^U \geq \boldsymbol{x}_k \geq \boldsymbol{x}_k^L \qquad k = 1 \ldots n. \end{aligned} \tag{5.10}$$

donde R es un conjunto de parámetros de penalidad y Ω es el término de penalidad, que es una función de R y de las restricciones. En este trabajo elegimos el operador $\langle\ \rangle$ y de términos parabólicos $(\ )$ como penalidades para tratar las restricciones de desigualdad e igualdad respectivamente. La fórmula correspondiente es:

$$\begin{aligned} &\Omega = R\langle g(\boldsymbol{x})\rangle^2 + R\big(h(\boldsymbol{x})\big)^2 \\ &\langle a \rangle = \begin{cases} a & \text{if } a \leq 0 \\ 0 & \text{if } a > 0 \end{cases} \end{aligned} \tag{5.11}$$

Con respecto a las cotas $\boldsymbol{x}_k^U$ y $\boldsymbol{x}_k^L$ en las variables de optimización es claro que son separables. De esta manera el algoritmo DPV puede manejarlas de forma explícita.

5.7. ANÁLISIS DE SENSIBILIDAD

En los problemas de optimización lineal, el análisis de sensibilidad permite obtener los denominados precios sombra. Estos indicadores miden los cambios en el valor óptimo de la función objetivo con respecto a perturbaciones en los lados derechos de las restricciones. En contraste, los problemas de optimización no lineales utilizan los valores de los multiplicadores de Lagrange asociados a las restricciones para el mismo propósito. Para una restricción de igualdad $h_k(\boldsymbol{x}) = \boldsymbol{b}_k$, el multiplicador $\boldsymbol{v}_k$ asociado a la función se define como:

$$\boldsymbol{v}_k = \frac{\partial f}{\partial \boldsymbol{b}_k}, \quad k = 1,..., ri \tag{5.12}$$

Por lo tanto, una aproximación de primer orden para la variación del valor óptimo de la función objetivo conduce a la expresión siguiente, que es una función compuesta por las variaciones en los lados derechos:

$$f(\boldsymbol{x}) - f(\boldsymbol{x}^*) = \sum_{k=1}^{ec} \left(\frac{\partial f}{\partial \boldsymbol{b}_k} \right) \Delta \boldsymbol{b}_k = \sum_{k=1}^{ec} \boldsymbol{v}_k \Delta \boldsymbol{b}_k \tag{5.13}$$

En este trabajo se propone usar a los multiplicadores de Lagrange del problema de optimización original (5.1) para determinar el grado de sensibilidad que tiene la función objetivo *f* en (5.7) con respecto a cada variable de optimización. Estos valores servirán como una guía para realizar el particionamiento del vector de variables $\boldsymbol{x}^T = [\boldsymbol{x_1}^T | \boldsymbol{x_2}^T | \ldots | \boldsymbol{x_l}^T | \ldots | \boldsymbol{x_p}^T]$.

5.7.1. ESTRATEGIAS DE PARTICIONAMIENTO DEL DOMINIO.

Cuando el algoritmo DPV es implementado sin una política inteligente para realizar el particionamiento del vector de variables de optimización, el desempeño del mismo es muy pobre en términos de tiempo de ejecución. Los algoritmos existentes que proponen descomposición de dominio asumen una alocación contigua de las variables en los bloques.

En la expresión (5.7) el procesador l actualiza las variables en el bloque $\boldsymbol{Y}_l$ mientras el resto de los procesadores actualizan simultáneamente sus bloques correspondientes, es decir el complemento $\boldsymbol{Y}_{\bar{l}}$. El objetivo de la estrategia de particionamiento es distribuir las variables en los bloques en forma adecuada, de manera que $\boldsymbol{Y}_{\bar{l}}$ tenga un impacto mínimo en la optimización de las variables de $\boldsymbol{Y}_l$.

Los beneficios específicos que se obtienen por realizar un correcto particionamiento son varios. En primer lugar se reduce el número total de iteraciones del algoritmo DPV pues las variables de los distintos bloques convergen más rápidamente a la solución. Como consecuencia, se reduce el número de mensajes que deben manejarse en la red de estaciones de trabajo, pues cada iteración del algoritmo involucra el intercambio de $2p$ mensajes (en un cluster de p procesadores). El maestro envía a los trabajadores p mensajes conteniendo el punto inicial. Luego, otros p mensajes con los resultados intermedios son devueltos al maestro, donde son combinados para generar la nueva aproximación a la solución. Además, el número evaluaciones de funciones también disminuye, permitiendo significativos ahorros de tiempo especialmente para problemas donde la evaluación de las funciones son costosas. Estos beneficios no están restringidos a implementaciones basadas en pasaje de mensajes en redes de estaciones de

trabajo; también pueden ser aplicadas al implementar esta clase de algoritmos en computadoras paralelas con acceso global a la memoria.

Un criterio de particionamiento de dominio propuesto inicialmente en esta tesis consiste en usar la información provista por el gradiente de la función objetivo (Vazquez y col., 2002). Las variables cuyas derivadas tienen magnitudes similares, dentro de una determinada tolerancia, son puestas juntas en un mismo bloque. El resto de la variables se distribuyen libremente en los bloques restantes. Esta regla heurística pretende mantener juntas a las variables que van a tener un comportamiento similar en un bloque.

Otra propuesta de particionamiento propuesta se basa en el análisis de sensibilidad (Vazquez y Brignole, 2002). La regla heurística consiste en analizar los elementos de $\boldsymbol{Y}_{\bar{I}}$ como si fueran parámetros del problema. Esto equivale a expresar el problema (5.7) como:

$$\begin{aligned} & Min\ f\ (\boldsymbol{x}) \\ sujeto\ a & \\ & Y_{\bar{I}}(\boldsymbol{x}) = \boldsymbol{c} \end{aligned} \tag{5.14}$$

donde $\boldsymbol{c}$ es el vector que contiene los valores constantes asignados a $\boldsymbol{x}_I^i$ para la iteración global en ejecución. En realidad el método no propone resolver (5.14); la relevancia de este problema en este contexto se relaciona con los multiplicadores de Lagrange $\boldsymbol{v}_{\bar{I}}$ asociado a cada restricción de igualdad porque representan las medidas de sensibilidad para el bloque de variables libres $\boldsymbol{Y}_I$. Como se menciona en Reklaitis y col. (1983), los multiplicadores pueden encontrarse resolviendo el sistema lineal asociado:

$$\nabla f(\boldsymbol{x}^*) = \sum \boldsymbol{v}_{\bar{I}} (\nabla \boldsymbol{Y}_{\bar{I}} - \boldsymbol{c}) \tag{5.15}$$

En un cluster de *p* procesadores se resuelven *p* sistemas lineales para distintos conjuntos de parámetros. Los multiplicadores con magnitudes muy grandes indican que la función objetivo es sensible a cambios en la variable de optimización correspondiente. De esta manera, las variables que tienen multiplicadores altos para un subproblema dado se eligen como variables libres en su correspondiente procesador. El resultado final es una distribución de los bloques que agrupan a las variables que tienen una influencia similar sobre la función objetivo.

5.7.2. ANÁLISIS DE DESEMPEÑO

Con el objetivo de medir las ganancias en el número de iteraciones y evaluaciones de funciones que se logran lograrse usando las estrategias de particionamiento propuestas, los siguientes problemas optimización no lineal fueron seleccionados (Hock y Schittkowski, 1981):

A - El problema de alkylación propuesto pot Bracken y McCormick (1968),

B - Un problema de separación de membrana de 3 etapas dado en Dembo (1976),

C - El problema de diseño óptimo de un reactor, Rijckaert (1973),

D - Un problema de equilibrio químico estudiado por Bracken y McCormick (1968),

E - Un problema de diseño de un intercambiador de calor presentado por Avriel y Williams (1971).

F - Un problema de distribución de potencia estático propuesto por Bartholomew-Biggs (1976).

La Tabla 5.1 muestra el tamaño de cada caso de estudio. Se usaron como puntos iniciales los sugeridos en la formulación del problema original (Hock y Schittkowski, 1981).

Problema	Variables de optimización	Restricciones de desigualdad	Restricciones de igualdad	Cotas
A	10	8	3	20
B	13	15	0	26
C	8	6	0	16
D	10	0	3	10
E	8	6	0	16
F	9	0	6	8

Tabla 5.1: Cantidad de variables y ecuaciones para cada problema.

En las Tablas 5.2 y 5.3 se reportan el número de iteraciones y de evaluaciones de funciones requeridas para resolver estos problemas mediante el algoritmo DPV con y sin particionamiento respectivamente. El vector de variables de optimización *x* fue dividido en 2 y 4 bloque, considerando a los mismos de igual o similar tamaño para asegurar una distribución de tareas correcta entre los procesadores. La matriz de direcciones *D* se definió mediante el método de descenso más abrupto. Se utilizó la siguiente condición de terminación testeada al finalizar cada etapa de sincronización:

$$| \nabla f(\boldsymbol{X}^{k}) | \leq 10^{-4}$$

Cantidad de Procesadores	2			4		
Particionamiento	No	Basado en Gradiente	Basado en Sensibilidad	No	Basado en Gradiente	Basado en Sensibilidad
Problema	Cantidad de iteraciones					
A	36	27	23	43	32	30
B	47	42	38	54	50	48
C	32	27	27	38	32	31
D	39	29	28	42	31	30
E	29	25	23	35	29	27
F	41	37	36	44	40	37

Tabla 5.2: Cantidad de iteraciones

Cantidad de Procesadores	2			4		
Particionamiento	No	Basado en Gradiente	Basado en Sensibilidad	No	Basado en Gradiente	Basado en Sensibilidad
Problema	Cantidad de iteraciones					
A	167	149	129	186	158	150
B	369	328	299	398	360	338
C	166	143	143	183	179	171
D	295	263	255	325	281	273
E	190	175	160	217	179	165
F	335	305	295	366	329	306

Tabla 5.3: Número total de evaluaciones de funciones

Las estrategias de particionamiento mostraron su efectividad tanto en la reducción de la cantidad de iteraciones necesarias en el algoritmo DPV como en la cantidad de evaluaciones requeridas para resolver los problemas. Estas ganancias son más significativas cuando los bloques se generan utilizando el criterio de particionamiento

basado en el análisis de sensibilidad. Las funciones objetivo y/o las restricciones que surgen al modelar problemas de ingeniería son costosas de evaluar. De esta manera, aún reducciones menores en la cantidad de evaluaciones de funciones permiten reducciones de tiempo de cómputo significativas.

5.8. CASO DE ESTUDIO: UNA PLANTA DE EXPANSIÓN

A continuación se presenta un caso de estudio real con el objetivo de cuantificar las mejoras introducidas con la nueva propuesta de paralelización.

5.8.1. DESCRIPCIÓN

En una planta típica de etano, el gas de entrada es filtrado y comprimido en tres compresores paralelos. Luego se lo enfría mediante aire y se lo deshidrata para evitar la formación de hielo e hidratos. Después este tratamiento, el gas que ingresa se divide en dos corrientes iguales, cada una de las cuales se envía a un tren de paralelización criogénico distinto. El sector criogénico constituye la base de la planta de expansión de baja temperatura. El producto de la corriente inferior de las dos columnas demetanizadoras es mezclado y enviado a un proceso de separación convencional para obtener etano puro, propano, butano y gasolina natural. Después del intercambio de calor, el producto de tope de la demetanizadora (gas residual) es comprimido en un booster impulsado por el expansor y finalmente recomprimido a la presión de la cañería para ser enviado como gas para la venta. La recompresión se realiza en compresores paralelos que funcionan con turbinas a gas.

Un proceso típico de turboexpansión, conocido también como proceso industrial estándar de una etapa, se muestra en la Figura 5.1.

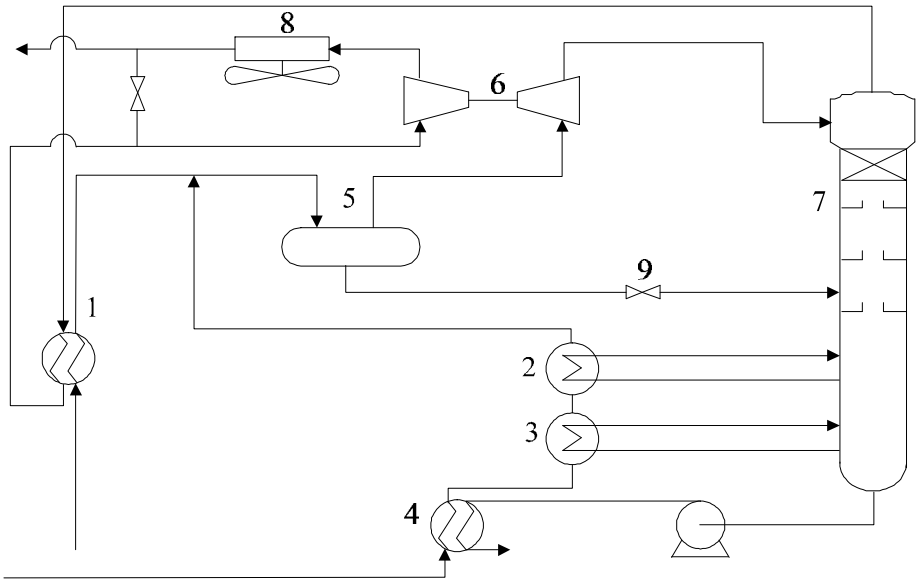

Figura 5.1: Sector criogénico de la planta de turboexpansión: 1) intercambiador de calor gas-gas 2) sector de rebullidores 3) rebullidor inferior 4) intercambiador de calor del producto 5) tanque frío 6) turboexpansor 7) demetanizadora 8) enfriadores de aire 9) válvulas Joule-Thomson.

El gas de entrada es enfriado por intercambio de calor con el gas residual y los rebullidores ubicados en los laterales y en el fondo de la demetanizadora y, eventualmente, mediante refrigeración externa. El gas de entrada parcialmente condensado se envía a un separador vapor-líquido (tanque frío). El vapor es expandido a través del turboexpansor para obtener la baja temperatura requerida para la recuperación de etano antes de ser alimentado al tope de la columna demetanizadora. El líquido del tanque frío pasa a través de una válvula Joule-Thomson e ingresa a la sección inferior de la demetanizadora, columna de destilación de baja temperatura que realiza una separación entre metano y etano. El metano y los componentes más livianos, tal como el nitrógeno, constituyen el producto de tope mientras que el etano y los hidrocarburos más pesados constituyen los principales productos de fondo de la demetaniadora. El dióxido

de carbono, que tiene una volatilidad intermedia entre el metano y el etano, se distribuye entre las corrientes de tope y fondo. Como la salida del expansor usualmente tiene dos fases, la líquida es usada como reflujo para la columna. Después de enfriar el gas entrada, el producto de tope o gas residual es recomprimido a la presión de cañería para su venta. El producto de fondo de la demetanizadora se fracciona para producir etano puro, propano, butano y gasolina natural. Dado que la fracción de C2+ en el gas de ingreso es relativamente baja, esta planta puede ser operada sin refrigeración externa.

5.8.2. FORMULACIÓN DEL PROBLEMA

Se estudió una planta de turboexpansión a gran escala con el objeto de maximizar la producción de etano. El modelo puede ser formulado como un problema de programación no lineal (NLP), donde el vector *x* representa las variables continuas de optimización tales como presión, temperatura y caudales. La optimización se realizó integrando un simulador riguroso del tipo secuencial-modular (Seider y col., 2000) al programa de optimización. La función objetivo es la producción de etano y las restricciones no lineales de desigualdad representan restricciones que aparecen en las especificaciones del proceso y los límites en las capacidades de los equipos. El conjunto de ecuaciones no lineales constituye el modelo matemático de la planta, que fue resuelto en un simulador ad-hoc.

La planta fue simulada rigurosamente con el simulador secuencial-modular PROSYD. El modelo fue ajustado cuidadosamente a partir de datos reales de una planta a gran escala, actualmente en operación. La simulación comienza en el tanque frío. La columna demetanizadora fue simulada rigurosamente mediante el método Naphtali-Sandholm, especificando la presión de tope y el caudal del fondo. Los datos termodinámicos de fueron obtenidos mediante las ecuaciones de estado SRK (Soave-

Redlich-Kuong) (Smith y col., 2000). Dado que la planta no tiene refrigeración externa, las corrientes de proceso fueron integradas energéticamente para lograr mejores condiciones. Las composiciones de las corrientes de entrada se muestran en la Tabla 5.4.

Componente	% molar
Nitrógeno	1.44
CO_2	0.65
Metano	90.43
Etano	4.71
Propano y componentes mas pesados	2.77

Tabla 5.4: Composición en la alimentación

Las principales variables de optimización son la temperatura y la presión del tanque frío, la presión de la columna demetanizadora y el caudal de fondo. Los límites en estas variables se muestran en la Tabla 5.4. La cota inferior para la temperatura del tanque frío fue determinada mediante el diagrama de envolvente de fase para el gas de entrada con el objeto de asegurar la existencia de dos fases. Los límites en la presión del tanque frío fueron seleccionados mediante un cálculo de envolvente de fase para evitar la región retrógrada y se muestran en la Tabla 5.5.

Variable de Optimización	T_{ct} (K)	P_{dem}(bar)	B_{dem}(kmol/h)	P_{ct}(bar)
Límite inferior	208.15	18.00	900.00	55.00
Límite superior	228.15	24.00	1600.00	60.00

Tabla 5.5: Límites en las variables de optimización

Las especificaciones del proceso y limitaciones del equipamiento se definieron mediante ocho restricciones no lineales que contemplan los siguientes aspectos:

- Con respecto a la capacidad de calentamiento en los intercambiadores de calor de gas de entrada (1), el producto de los coeficientes de transferencia de calor y el área de transferencia deben estar dentro de los límites del equipo.
- El calor producido en el rebullidor de fondo de la demetanizadora (3) debe ser menor que una cota superior.
- El calor producido en el intercambiador de calor de fondo aguas debajo de la demetanizadora (4) debe ser menor que una cota superior.
- Se debe mantener una dada relación metano/etano en el fondo de la demetanizadora (7).
- Los caudales de líquido y vapor circulantes entre los platos de la demetanizadora (7) deben ser menores que los valores de inundación.
- Se deben imponer condiciones para evitar la precipitación de dióxido de carbono.

Los números entre paréntesis corresponde a los equipos indicados en la Figura 5.1.

5.8.3. IMPLEMENTACIÓN DEL CÓDIGO PARALELO Y RESULTADOS

En primer lugar, se resolvió el modelo de la planta de expansión secuencialmente usando la formulación de penalidad descripta en la Sección (5.6) para realizar una comparación justa. La Tabla 5.6 muestra el caso base (punto inicial) y el óptimo obtenido. Cabe destacar que sólo hay una restricción activa en el óptimo, que corresponde a la máxima relación metano/etano en el fondo de la demetanizadora. La función objetivo (producción de etano) es 736.17 kmol/h y el punto óptimo muestra un incremento de 7.06% en la recuperación del etano. El tiempo de cómputo secuencial

medido fue de 7min. 16seg, corriendo en una computadora PC Pentium 133 Mhz. con sistema operativo LINUX y GNU FORTRAN.

Con respecto a la formulación paralela del problema, se particionó el modelo de la planta de expansión en dos bloques que contienen dos variables de optimización cada uno. Los valores para el criterio de terminación y el cálculo de la matriz de dirección ***D*** en el código DPV implementado son los mismos de los ejemplos de prueba de la Sección 5.7.2. El algoritmo paralelo fue inicializado con el mismo punto del secuencial y obtuvo el mismo óptimo como el mostrado en la Tabla 5.6. El tiempo de cómputo medido fue de 4min. 41seg., utilizando dos computadoras PC Pentium 133 Mhz interconectadas por una red Ethernet de 10 Mb/seg y utilizando el sistema operativo LINUX y GNU FORTRAN.

Variable	Lower bound	Upper bound	Base case (starting point)	Operating optimum
Tcold tank (K)	208.15	233.15	211.15	210.78
Pdem (bar)	18.00	24.00	19.00	18.00
Bdem (kmol/h)	900.00	1700.00	1300.00	1341.63
Pcold tank(bar)	54.00	60.00	57.00	59.55
Q4 (kW)			2095.45	2286.05
Q5 (kW)			460.23	915.81
Ethane rec.(%)			76.11	81.49
Prop. Rec.(%)			98.40	98.69
c1/c2 bottoms			0.008	0.04

Tabla 5.6: Comparación entre el punto de operación inicial y el punto de operación óptimo

Los tiempos de cómputo registrados mostraron un speed-up de 1.55 y una eficiencia de 77.5%. En todo los casos, el código de optimización usado fue TNBC (Nash, 1984), tanto para resolver el modelo en forma secuencial de cómo los subproblemas del algoritmo paralelo.

5.9. CONCLUSIONES

En este capítulo se presentó un nuevo algoritmo basado en la técnica de descomposición de dominios para resolver problemas de optimización no lineal que surgen en el área de ingeniería de procesos. Con este objetivo se propuso el uso de funciones de penalidad para permitir el tratamiento de modelos con restricciones no lineales. Además se propusieron dos políticas de particionamiento de dominio que permiten una ejecución más eficiente del algoritmo iterativo, ya sea reduciendo la cantidad de iteraciones del mismo o las evaluaciones de funciones que deben realizarse.

CAPÍTULO 6: PARALELIZACIÓN DE ALGORITMOS SECUENCIALES PARA OPTIMIZACION NUMERICA

El desarrollo de algoritmos de optimización paralelos puede ser abordado mediante la paralelización de los métodos secuenciales existentes. Una forma directa es utilizar un compilador que genere código paralelo a partir de una implementación secuencial (Harrison, 1990) (Chimowitz y Bielinis, 1987). Sin embargo este enfoque requiere de computadoras paralelas, preferentemente con acceso a memoria global y la eficiencia en el uso de los procesadores es muy pobre.

En este capítulo se presentan estrategias para la paralelización de dos algoritmos de optimización ampliamente usados: SQP Y GRG y se comprueba su efectividad aplicándolo a diversos problemas representativos en ingeniería.

6.1. ALGUNOS ALGORITMOS SECUENCIALES CLÁSICOS

Si bien el trabajo ha sido enfocado a un método de optimización específico, muchos aspectos de la paralización y las estrategias para evaluación de desempeño discutidas puedan ser generalizadas, y así ser útiles para el desarrollo de otros códigos de optimización paralelo-distribuidos.

6.2. EL MÉTODO DE OPTIMIZACIÓN SQP

El método de optimización conocido como Programación Cuadrática Sucesiva (Successive Quadratic Programming, SQP) (Powell, 1978a; Powell, 1978b) es la técnica más poderosa conocida hasta la fecha para resolver problemas de optimización no lineal de la forma (5.1). Debido a su exitoso en comparación con otros métodos (Schittkowski, 1980) en la actualidad se utiliza en diversas áreas de la ingeniería, tales como sistemas de procesos, optimización mecánica estructural (Schittkowski y col., 1994), ajuste de datos y control óptimo de sistemas farmacéuticos (Boderke y col., 2000), cómputo de caudales de ingreso óptimos para reactores tubulares (Birk y col., 1999), deshidratación de alimentos en hornos convectores (Frias y col., 2001) y diseño óptimo de radiadores para satélites de comunicación (Hartwanger y col., 2000).

La estrategia del método consiste en formular y resolver en cada iteración un subproblema de optimización de tipo QP (Quadratic Programming), el cual se obtiene linealizando las restricciones y tomando como función objetivo una aproximación cuadrática al Lagrangiano del modelo original:

$$L(\boldsymbol{x},\boldsymbol{u},\boldsymbol{v}) = f(\boldsymbol{x}) - \sum_{i}^{ri} \boldsymbol{v}_i h_i(\boldsymbol{x}) - \sum_{i}^{rd} \boldsymbol{u}_i g_i(\boldsymbol{x}) \tag{6.1}$$

donde los vectores $\boldsymbol{v} \in \Re^{ri}$ y $\boldsymbol{u} \in \Re^{rd}$ son los multiplicadores de Lagrange asociados a las restricciones de igualdad y desigualdad, respectivamente. Por lo tanto, en la iteración *k*-ésima el problema de optimización QP resultante es:

$$\begin{aligned} &Min \quad \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k + \tfrac{1}{2} \boldsymbol{d}_k{}^T \boldsymbol{H}_k \boldsymbol{d}_k \\ &sujeto \ \ a \\ &\qquad g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{d}_k \geq 0 \quad i = 1,..,ri \\ &\qquad h_j(\boldsymbol{x}_k) + \nabla h_j(\boldsymbol{x}_k)^T \boldsymbol{d}_k = 0 \quad j = 1,..,rd \end{aligned} \tag{6.2}$$

donde $\boldsymbol{x}_k$ es una aproximación a la solución, y $\boldsymbol{H}_k$ es una aproximación al Hessiano del Lagrangiano (6.1). Puede ocurrir que el problema (6.2) resulte inconsistente, razón por la cual se introduce en el modelo un parámetro de relajación. Para información adicional sobre este tópico referirse a (Schittkowski, 1983).

El resultado del problema (6.2) es un vector de dirección de búsqueda $\boldsymbol{d}_k$. El nuevo valor para la iteración k+1 se obtiene mediante una búsqueda en línea en la dirección $\boldsymbol{d}_k$. La longitud de paso α_k elegida debería garantizar el descenso de una función de mérito $P(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k)$. Una posible elección es minimizar la función de penalidad:

$$P(\boldsymbol{x}, \boldsymbol{\rho}) = f(\boldsymbol{x}) + \sum_{i=1}^{ri} \boldsymbol{\eta}_i \left| h_i(\boldsymbol{x}) \right| + \sum_{j=1}^{rd} \boldsymbol{\rho}_j \min(0, g_j(\boldsymbol{x})) \tag{6.3}$$

donde $\boldsymbol{\rho}$ y $\boldsymbol{\eta}$ son parámetros que controlan el grado de violación de las restricciones y deben ser elegidos de manera apropiada para garantizar una dirección descendente de la función de mérito (Powell, 1978a). Una vez determinada la longitud de paso óptima, el nuevo vector solución se calcula como

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k . \tag{6.4}$$

En cuanto a la aproximación cuadrática del Hessiano $\boldsymbol{H}_k$ utilizada en (6.2), Powell (1978b) sugiere actualizarla por medio de la fórmula (6.5), con $\boldsymbol{H}_0$=$\boldsymbol{I}$:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k - \frac{\boldsymbol{H}_k \boldsymbol{z}\boldsymbol{z}^T \boldsymbol{H}_k}{\boldsymbol{z}^T \boldsymbol{H}_k \boldsymbol{z}^T} + \frac{\boldsymbol{w}\boldsymbol{w}^T}{\boldsymbol{z}^T \boldsymbol{w}} \tag{6.5}$$

$$\boldsymbol{y} = \nabla_x L(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}, \boldsymbol{v}_{k+1}) - \nabla_x L(\boldsymbol{x}_k, \boldsymbol{u}_{k+1}, \boldsymbol{v}_{k+1}) \tag{6.6}$$

$$\boldsymbol{z} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k , \quad \boldsymbol{w} = q\,\boldsymbol{y} + (1-q)\boldsymbol{H}_k \boldsymbol{z} \tag{6.7}$$

$$q = \begin{cases} 1 & \text{si } \boldsymbol{z}^T \boldsymbol{y} \geq 0.2 \boldsymbol{z}^T \boldsymbol{H}_k \boldsymbol{z} \\ \dfrac{0.8 \boldsymbol{z}^T \boldsymbol{H}_k \boldsymbol{z}}{\boldsymbol{z}^T \boldsymbol{H}_k \boldsymbol{z} - \boldsymbol{z}^T \boldsymbol{y}} & \text{caso contrario} \end{cases} \tag{6.8}$$

Esta fórmula, conocida como BFS, permite que $\boldsymbol{H}$ sea definida positiva en todas las iteraciones, propiedad conveniente desde el punto de vista numérico.

La convergencia del algoritmo se implementa típicamente chequeando que las normas de la expresión (6.1) y del vector de restricciones activas sean menores que ciertos valores de tolerancia especificados por el usuario. Por definición las restricciones activas son aquellas restricciones de igualdad ó desigualdad que han sido violadas o son iguales a cero.

Finalmente, el algoritmo SQP puede expresarse de la siguiente forma:

Datos de entrada: $\boldsymbol{x}$
Datos de salida: $\boldsymbol{x}$, k, {multiplicadores,...}
$k \leftarrow 0$.
$\boldsymbol{H} \leftarrow$ *MatrizIdentidad*.
Repetir
    Resolver el problema QP (6.2) para calcular $\boldsymbol{d}_k$
    Calcular long. paso α_k minimizando (6.3)
    Calcular nuevo vector solución $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$
    Actualizar $\boldsymbol{H}$ según (6.5)-(6.8)
    k=k+1;
Hasta convergencia

Algoritmo 6.1: Esquema del método SQP

6.3. PARALELIZACIÓN DEL MÉTODO SQP

Como se mencionó anteriormente, el alto costo en tiempos de cómputo requerido para la evaluación de las funciones del modelo constituye una característica distintiva de numerosos problemas de optimización en el campo de la ingeniería. Las tareas de evaluación típicamente requieren mucho más tiempo que el resto de los cálculos. En el campo de la ingeniería de sistemas de procesos en particular, esto suele suceder porque cada evaluación requiere la simulación numérica del modelo completo de la planta de procesos a partir de un conjunto de condiciones de entrada que cambian en cada iteración principal del algoritmo. Esto puede implicar la necesidad de resolver cientos de ecuaciones. Como resultado, las secciones del algoritmo que llevan a cabo estas evaluaciones repetitivamente son consideradas secciones 'calientes', pues dominan el tiempo total de ejecución. Los procedimientos de evaluación para el gradiente y el Hessiano, junto a la etapa de búsqueda en línea, constituyen puntos calientes inevitables que aparecen en todas las principales estrategias de optimización no lineal.

En el caso particular de un algoritmo SQP, tanto la búsqueda en línea como la actualización de la matriz $\boldsymbol{H}$ y los cálculos de derivadas para chequear la convergencia son directamente paralelizables. Con respecto al problema QP (6.2), éste no puede ser paralelizado en forma directa debido a su complejidad. Sin embargo el tiempo que consume la resolución de este problema es despreciable cuando se lo compara con los tiempos requeridos para la evaluación del gradiente de las restricciones, el proceso de búsqueda en línea y la actualización de la matriz $\boldsymbol{H}$.

En las subsecciones siguientes se propone una estrategia a seguir para la evaluación simultánea de funciones y su aplicación a las diferentes secciones de un algoritmo SQP .

6.3.1. EVALUACIÓN PARALELA DE FUNCIONES EN ENTORNOS DISTRIBUIDOS

Una manera natural de implementar la evaluación simultánea de funciones es mediante un esquema clásico maestro-trabajador, donde el maestro distribuye y recolecta las evaluaciones de las funciones hechas por los procesos trabajadores. Es necesario una correcta técnica de balanceo de carga para proveer una división justa del esfuerzo computacional en todos los procesadores. Sin embargo, una implementación paralela en un entorno distribuido heterogéneo debe considerar los aspectos distintivos de un cluster heterogéneo de estaciones de trabajo, donde el desempeño de cómputo de cada estación de trabajo puede ser diferente o donde las evaluaciones deben competir con otros procesos de usuarios que se ejecutan en los nodos, lo cual resulta en niveles de desempeño impredecibles en tiempo de ejecución. Asimismo, el tiempo requerido para evaluar una misma función puede diferir dependiendo de su argumento.

De la misma manera que la aplicación de reordenamiento estructural presentada en el Capítulo 3, el modelo modificado de distribución dinámico por demanda diseñado en el marco de esta tesis se ajusta perfectamente a esta situación. Esta técnica permite que no existan procesadores ociosos y elimina el retardo producido por la red de datos entre una evaluación de función y otra.

6.3.2. PARALELIZACIÓN DE LA BÚSQUEDA EN LÍNEA

La estrategia secuencial adoptada consiste en realizar una búsqueda en línea en la dirección $\boldsymbol{d}_k$ buscando un descenso en la función de mérito (6.3). Más específicamente esto implica resolver:

$$\underset{a_k}{Min} \;\; P(\boldsymbol{x}_k + a_k + \boldsymbol{d}_k) \tag{6.9}$$

de manera que el nuevo vector $\boldsymbol{x}$ en la iteración k+1 del algoritmo será $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + a_k \boldsymbol{d}_k$.

La implementación de la búsqueda en línea es un aspecto crítico en una estrategia SQP y tiene un impacto importante en la eficiencia del código resultante. Si bien la búsqueda en línea permite estabilizar al algoritmo y hacerlo progresar, no es deseable realizar tantas evaluaciones de la función de mérito P cuando se resuelve (6.9) debido al alto costo computacional. Mas aún, el comportamiento de P es bastante irregular debido a la presencia de los términos de penalidad que modelan a las restricciones del problema. Por esta razón ninguna implementación realiza una minimización exacta de (6.9), aceptándose sólo una aproximación del a_k óptimo, que llamaremos $\tilde{a}$.

En su fase inicial, la estrategia secuencial para encontrar la longitud de paso $\tilde{a}$ involucra acotar este valor en un intervalo utilizando un patrón de búsqueda heurístico. Un caso particular es el método de Swann (Reklaitis y col., 1983) que consiste en generar $w+1$ puntos de prueba $\tilde{\boldsymbol{x}}$ mediante la fórmula:

$$\tilde{\boldsymbol{x}}_q = \boldsymbol{x}_k + 2^q b \boldsymbol{d}_k \tag{6.10}$$

donde q = 0, 1, ..,w , y b un tamaño del paso predeterminado. La búsqueda finaliza cuando se encuentra un $\tilde{\boldsymbol{x}}$ tal que:

$$P(\tilde{\boldsymbol{x}}_{q-2}) \geq P(\tilde{\boldsymbol{x}}_{q-1}) \leq P(\tilde{\boldsymbol{x}}_q) \tag{6.11}$$

para $q \geq 2$, de manera que un primer valor aproximado para $\tilde{a}$ es $2^{q-1} b$, según (6.10). De la misma manera, adoptando una estrategia similar a la búsqueda por bisección, este

procedimiento podría repetirse para obtener una mejor estimación de $\tilde{a}$, pero el nuevo intervalo de búsqueda tendría los límites $\tilde{x}_{q-2}$ y $\tilde{x}_q$.

La implementación paralela de la estrategia secuencial descripta es directa. Si se dispone de $w+1$ procesadores en el cluster se pueden realizar en forma simultánea las $w+1$ evaluaciones $P(\tilde{x}_q)$, siendo $\tilde{x}_q$ el definido en (6.10). El algoritmo paralelo puede expresarse de la siguiente forma:

Datos de entrada: x_k, b
Datos de salida: a
Para cada $\tilde{x}_q$ definido en (6.10) **hacer en paralelo:**
    Evaluar $P(\tilde{x}_q)$.
fin-para
$a \leftarrow \tilde{x}_{q-1}$ tal que $P(\tilde{x}_{q-2}) \geq P(\tilde{x}_{q-1}) \leq P(\tilde{x}_q)$

Algoritmo 6.2: Búsqueda en línea en el método SQP

Si la cantidad de procesadores *p* disponibles en el cluster es pequeña, una solución es particionar el intervalo de búsqueda en $\lceil (w+1)/p \rceil$ secciones, evaluando en paralelo *p* funciones $P(\tilde{x}_q)$ en cada iteración. Las iteraciones se detienen naturalmente cuando se verifica la propiedad (6.11).

6.3.3. EVALUACIÓN DE GRADIENTES EN PARALELO

Todas las técnicas de optimización basadas en gradiente emplean información sobre las derivadas de las funciones involucradas. En el caso particular del algoritmo SQP, esto se utiliza tanto en el chequeo de convergencia del algoritmo como también en la actualización de la matriz ***H***. Generalmente, en situaciones reales, las funciones no

pueden ser escritas o derivadas de forma analítica. Por lo tanto, las derivadas en general se calculan por diferenciación numérica. Existen varias fórmulas que aproximan a la derivada primera. En este trabajo en particular se empleó la conocida aproximación por diferencias hacia adelante, que representa un cociente incremental y por lo tanto, surge en forma natural de la definición de derivada:

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}_i} = \frac{f(\boldsymbol{x} + \boldsymbol{\alpha}_i \boldsymbol{e}_i) - f(\boldsymbol{x})}{\boldsymbol{\alpha}_i} + O(\boldsymbol{\alpha}_i), \quad i = 1,\ldots, n \tag{6.12}$$

donde $\boldsymbol{\alpha}_i$ es una constante pequeña, $\boldsymbol{e}_i$ es el versor *i*-esimo y el último término corresponde a la magnitud del error de aproximación, que es proporcional a $\boldsymbol{\alpha}_i$. Todas las evaluaciones de funciones involucradas en estos cálculos pueden realizarse eficientemente usando procesamiento paralelo. Esta formulación requiere evaluar una única función en cada punto perturbado $(\boldsymbol{x} + \boldsymbol{\alpha}_i \boldsymbol{e}_i)$ además de la evaluación de $f(\boldsymbol{x})$. Así, para un problema de dimensión n se necesitan $n+1$ tareas para calcular un gradiente.

Las evaluaciones de funciones pueden realizarse en forma simultánea, pues no existen dependencias entre las mismas. De esta manera, si en el cluster se dispone de al menos $n+1$ procesadores, el proceso requiere una sola iteración. En caso contrario, las evaluaciones pueden mantenerse en un pool de tareas pendientes y utilizar un esquema maestro-esclavos para realizar el cómputo de las mismas. Nuevamente, la elección ideal es utilizar un esquema de distribución de carga dinámico por demanda, siguiendo las direcciones de implementación comentadas en el Capítulo 3.

6.3.4. IMPLEMENTACIÓN

En primer lugar, se implementaron un algoritmo de optimización SQP secuencial y otro paralelo utilizando los concepto descriptos en la Sección 6.3. Para esto se utilizó el lenguaje de programación FORTRAN y la librería de pasaje de mensajes PVM. Para resolver el problema QP se utilizó el código QL0001 v1.4 (Powell, 1983) de distribución libre.

Con el objeto de evaluar las mejoras logradas mediante la paralelización, se compararon los resultados obtenidos en forma secuencial y paralela de cuatro problemas de optimización no lineales tomados de los casos de estudio de Hock y Schittkowski (1981). La Tabla 6.1 muestra el tamaño de cada caso de estudio. Se usaron como puntos iniciales los sugeridos en la formulación original del problema. En este trabajo, las funciones objetivo correspondientes fueron sobrecargadas artificialmente en su tiempo de cómputo para simular modelos con evaluación de función realmente costosas.

Problema # Hock y Schitt-kowski (1981)	Variables de optimización	Restricciones de desigualdad	Restricciones de igualdad	Cotas
78	5	0	3	0
113	10	8	0	0
114	10	8	3	20
117	15	5	0	15

Tabla 6.1: Especificación de los problemas de prueba

Las corridas fueron llevadas a cabo usando un cluster de estaciones de trabajo heterogéneas que incluye una PC PentiumII 400Mhz (WS1), una PC Pentium 133Mhz

(WS2) y una DEC Alpha 150Mhz (WS3) interconectadas mediante una red Ethernet de 10Mb/sec. Las estaciones de trabajo basadas en procesadores Intel utilizan el sistema operativo LINUX, mientras que la DEC Alpha usa OSF/1. La estación de trabajo WS1 corrió el proceso maestro y las tres máquinas ejecutaron los procesos trabajadores.

Debido a la diferencia en el desempeño de los procesadores, en este trabajo se ha desarrollado una nueva métrica de evaluación denominada speed-up ponderado, la cual se describe en el Apéndice D (Vazquez y Brignole, 1999a). La Tabla 6.2 muestra el número de iteraciones necesarias para la convergencia en cada caso de estudio, seguido por los tiempos secuenciales requeridos en cada estación de trabajo. Las dos últimas columnas contienen el tiempo paralelo y el speed-up poderado obtenidos.

Problema # Hock y Schitt-kowski (1981)	Número de iteraciones	Tiempo secuencial WS1	Tiempo secuencial WS2	Tiempo secuencial WS3	Tiempo paralelo	Speed-up ponderado
78	5	16.3s	50.3s	21.4s	10.5s	2.33
113	13	59.6s	3:10.1s	1:17.23s	38.2s	2.22
114	6	30.8s	1:36.0s	36.3s	18.1s	2.35
117	20	2:0.2s	6:5.2s	2:36.9s	1:16.5s	2.24

Tabla 6.2: Desempeño paralelo del algoritmo SQP

6.3.5. RESULTADOS

Las medidas correspondientes al speed-up ponderado son satisfactorias, con una eficiencia de más del 90% de utilización. Luego de la paralelización, la única etapa secuencial fue el resolvedor QP y las operaciones algebraicas básicas. Con respecto a los tiempos de cómputo, estas etapas no son afectadas por la complejidad del procedimiento

de evaluación. En situaciones donde las funciones del problema sean más costosas de evaluar se espera un incremento del speed-up. Por el contrario, las ganancias obtenidas cuando las funciones objetivo y restricciones son más simples serán bajas debido a la preponderancia de los costos de comunicación.

6.4. EL MÉTODO DE OPTIMIZACIÓN GRG

Otro algoritmo de optimización no lineal clásico es el método del Gradiente Reducido Generalizado (GRG) (Abadie y Carpentier, 1969).

Por simplicidad consideremos el problema general de optimización no lineal con restricciones de igualdad:

$$\begin{aligned} &Min \quad f(\boldsymbol{x}) \\ &sujeto\ a \\ &h_k(\boldsymbol{x}) = 0 \quad k = 1,...,ri \end{aligned} \tag{6.13}$$

donde $ri < n$. Supongamos una aproximación lineal del sistema $h_k(\boldsymbol{x}) = 0,\ k = 1,...,ri$ en torno a un punto $\boldsymbol{x_1}$. El resultado es:

$$\tilde{h}_k(\boldsymbol{x};\boldsymbol{x_1}) \equiv h_k(\boldsymbol{x_1}) + \nabla h_k(\boldsymbol{x_1})(\boldsymbol{x} - \boldsymbol{x_1}), \quad k = 1,...,ri\,. \tag{6.14}$$

Si utilizamos estas ecuaciones linealizadas para predecir otro punto factible, este debería satisfacer:

$$\tilde{h}_k(\boldsymbol{x};\boldsymbol{x_1}) = 0, \quad k = 1,...,ri\,. \tag{6.15}$$

Como $h_k(\boldsymbol{x_1}) = 0,\ k = 1,...,ri$, este punto también debe satisfacer el sistema de ecuaciones:

$$\nabla h_k(\boldsymbol{x}_1)(\boldsymbol{x}-\boldsymbol{x}_1)=0, \quad k=1,...,ri\,. \tag{6.16}$$

Como este sistema tiene más incógnitas que ecuaciones, su solución en general no es única. Si el sistema es de rango lleno, entonces puede resolverse para *ri* de las *n* variables en términos de las $n-ri$ restantes. Para esto se particiona al vector $\boldsymbol{x}$ en dos conjuntos de variables: $\hat{\boldsymbol{x}}\in\Re^{ri}$, que contiene las variables denominadas *básicas*, y $\bar{\boldsymbol{x}}\in\Re^{n-ri}$ llamadas *no básicas*. De la misma manera podemos particionar a $\nabla h(\boldsymbol{x})$ en $\nabla\hat{h}(\boldsymbol{x})$ y $\nabla\bar{h}(\boldsymbol{x})$, esto es $\nabla h_k(\boldsymbol{x})=\left(\boldsymbol{J}(\boldsymbol{x})\vdots\boldsymbol{C}(\boldsymbol{x})\right)$, donde:

$$\boldsymbol{J}(\boldsymbol{x})=\begin{pmatrix}\nabla\hat{h}_1(\boldsymbol{x})\\ \nabla\hat{h}_2(\boldsymbol{x})\\ \vdots\\ \nabla\hat{h}_{ri}(\boldsymbol{x})\end{pmatrix}=\nabla\hat{h}(\boldsymbol{x})\,; \qquad \boldsymbol{C}(\boldsymbol{x})=\begin{pmatrix}\nabla\bar{h}_1(\boldsymbol{x})\\ \nabla\bar{h}_2(\boldsymbol{x})\\ \vdots\\ \nabla\bar{h}_{ri}(\boldsymbol{x})\end{pmatrix}=\nabla\bar{h}(\boldsymbol{x})\,. \tag{6.17}$$

Ahora, el sistema de ecuaciones (6.16) puede ser reescrito como:

$$\boldsymbol{J}(\hat{\boldsymbol{x}}-\hat{\boldsymbol{x}}_1)+\boldsymbol{C}(\bar{\boldsymbol{x}}-\bar{\boldsymbol{x}}_1)=0 \tag{6.18}$$

Suponiendo que se eligió una partición de $\boldsymbol{x}$ tal que $\boldsymbol{J}(\boldsymbol{x})$ es no singular, se puede obtener el valor de la partición de variables básicas en función de las no básicas:

$$\hat{\boldsymbol{x}}-\hat{\boldsymbol{x}}_1=-\boldsymbol{J}^{-1}(\hat{\boldsymbol{x}}_1)\boldsymbol{C}(\bar{\boldsymbol{x}}_1)(\bar{\boldsymbol{x}}-\bar{\boldsymbol{x}}_1) \tag{6.19}$$

De esta manera es posible eliminar variables las variables básicas $\hat{\boldsymbol{x}}$ de la función *f* que se desea minimizar, resultando:

$$\tilde{f}(\bar{\boldsymbol{x}}\vdots\hat{\boldsymbol{x}})=f\left(\boldsymbol{x}_1-\boldsymbol{J}^{-1}(\hat{\boldsymbol{x}}_1)\boldsymbol{C}(\bar{\boldsymbol{x}}_1)(\bar{\boldsymbol{x}}-\bar{\boldsymbol{x}}_1);\bar{\boldsymbol{x}}\right). \tag{6.20}$$

Si se plantean las condiciones de optimalidad, la condición necesaria de primer orden es:

$$\nabla \tilde{f}(\boldsymbol{x}_1) = \nabla \bar{f}(\boldsymbol{x}_1) - \nabla \hat{f}(\boldsymbol{x}_1)\boldsymbol{J}^{-1}(\hat{\boldsymbol{x}}_1)C(\bar{\boldsymbol{x}}_1); \tag{6.21}$$

$$\nabla \hat{f} = (\partial f / \partial \hat{\boldsymbol{x}}); \tag{6.22}$$

$$\nabla \bar{f} = (\partial f / \partial \bar{\boldsymbol{x}}); \tag{6.23}$$

donde el vector $\nabla \hat{f}$ definido en (6.22) se denomina gradiente reducido.

6.4.1. ALGORITMO GRG

A partir del desarrollo teórico del método, el Algoritmo 6.3 muestra el código correspondiente al método GRG secuencial.

Datos de entrada: $\boldsymbol{x}_0$, ε_1

Datos de salida: $\boldsymbol{x}$ solución

// $\boldsymbol{x}_0$: punto inicial
// ε_1: tolerancia para la terminación del algoritmo

mientras $\left\|\nabla \tilde{f}(\boldsymbol{x}_k)\right\| < \varepsilon_1$ **hacer**

Buscar una partición $\boldsymbol{x}_k = (\hat{\boldsymbol{x}}_k : \bar{\boldsymbol{x}}_k)$ tal que $\boldsymbol{J}(\boldsymbol{x}_k)$ sea no singular.
Construir una dirección de búsqueda:
$\bar{\boldsymbol{d}} = -(\nabla \tilde{f}(\boldsymbol{x}_k))^T$; $\hat{\boldsymbol{d}} = -\boldsymbol{J}^{-1}(\boldsymbol{x}_k)\boldsymbol{C}(\boldsymbol{x}_k)\bar{\boldsymbol{d}}$;
$\boldsymbol{d} = (\hat{\boldsymbol{d}} : \bar{\boldsymbol{d}})^T$.
$\alpha_k \leftarrow$ longitud de paso de la búsqueda en línea
$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_{k+1} + \alpha_k \boldsymbol{d}$

fin-mientras

Algoritmo 6.3: Algoritmo de optimización GRG

Si bien el vector $\boldsymbol{d}$ genera una dirección descendente, la búsqueda en línea puede conducir a puntos no factibles. Este se debe fundamentalmente a la linealización de las restricciones usada en el cálculo del vector de dirección. Por esta razón, en lugar de

realizar una búsqueda en línea tradicional en la dirección $\boldsymbol{d}$, la estrategia es calcular $\bar{\boldsymbol{d}}$ mediante una linealización y proyectarlo en una superficie de restricciones. Esto implica minimizar $f(\boldsymbol{x})$ sobre la curva definida de manera implícita por el conjunto de valores a y $\hat{\boldsymbol{x}}$ que satisfacen la ecuación:

$$h(\bar{\boldsymbol{x}}+a\boldsymbol{d}\,\vdots\,\hat{\boldsymbol{x}})=0\,. \tag{6.24}$$

Por lo tanto, la búsqueda en línea del Algoritmo 6.3 es reemplazada por el procedimiento descripto anteriormente. Su implementación se muestra en el Algoritmo 6.4.

Datos de entrada: $\boldsymbol{x}_k$, $\boldsymbol{d}$

Datos de salida: $\boldsymbol{x}_{k+1}$

```
// x_k : punto inicial
// å_2: tolerancia para restricción activa
// a_0 : longitud de paso predeterminada
// g : parámetro de reducción de longitud de paso

 i ← 0.
 a ← a_0 .  // long.paso inicial.
 repetir
    v_i = x_k + a d .
    si |h_k(v_i)| ≤ e_2, k = 1,..., ri    // v_i es factible
       si  f(x_k) ≤ f(v_i)
             // es necesario achicar la longitud de paso
             a ← g a .
          sino
             // se obtuvo una mejora en la solución
             x_{k+1} ← v_i .
       fin-si
     sino
       repetir
             H(v_i) ← (h_1(v_i), h_2(v_i),..., h_ri(v_i)) .
             v̂_{i+1} ← v̂_i − J^{-1}(v_i) H(v_i) .
             v̄_{i+1} ← v̄_i .
```

```
      hasta  ||v̂_{i+1} − v̂_i|| ≤ e_3
      si  |h_k(v_i)| > e_2, k = 1,..., ri     // v_i es factible
            a ← ga .
      fin-si
   fin-si
hasta  f(x_k) > f(v_i)
```

Algoritmo 6.4: Búsqueda en línea en el método GRG

6.4.2. PARALELIZACIÓN DEL MÉTODO GRG

Siguiendo el mismo razonamiento descripto en la Sección 6.3 para el método SQP, la paralelización de las evaluaciones de funciones involucradas en el algoritmo nuevamente conducen a significativas ganancias en tiempo de cómputo cuando éstas son costosas en términos de tiempo de ejecución.

En el caso particular del algoritmo GRG, el paralelismo puede ser introducido en cada iteración para evaluar las matrices $\boldsymbol{J}$ y $\boldsymbol{C}$ (pues involucran calcular el gradiente de las restricciones), chequear la condición de terminación (a través del cálculo del gradiente reducido $\nabla\tilde{f}$) y en el procedimiento de búsqueda descripto en el Algoritmo 6.4. En este caso el principal costo computacional reside en el ciclo que evalúa $\hat{\boldsymbol{v}}_{i+1} \leftarrow \hat{\boldsymbol{v}}_i - \boldsymbol{J}^{-1}(\boldsymbol{v}_i)\boldsymbol{H}(\boldsymbol{v}_i)$. En la implementación, esto se lleva a cabo planteando el sistema de ecuaciones no lineales:

$$\boldsymbol{J}(\boldsymbol{v}_i)\hat{\boldsymbol{v}}_{i+1} = \boldsymbol{J}(\boldsymbol{v}_i)\hat{\boldsymbol{v}}_i - \boldsymbol{H}(\boldsymbol{v}_i) \tag{6.25}$$

que se resuelve mediante el método Newton. Este procedimiento queda paralelizado a través de la evaluaciones de los gradientes de $\boldsymbol{J}$.

Todas estas evaluaciones de funciones pueden hacerse de la misma manera que se hizo en el algoritmo paralelo SQP, utilizando el esquema de distribución de tareas dinámico desarrollado en el Capítulo 3.

6.4.3. RESULTADOS

Los algoritmos del método GRG paralelo propuesto se implementaron usando el lenguaje de programación ANSI C y la librería de pasaje de mensajes PVM. Para el análisis de desempeño se utilizó un problema que incluye sólo restricciones de igualdad pues en este trabajo, por razones de simplicidad, sólo se consideró esta clase de problemas. El objetivo consiste en minimizar la función:

$$Min \quad \sum_{t=1}^{T} \boldsymbol{x}_t^{\,2}$$

sujeto a

$$h_t(\boldsymbol{x}) = (\boldsymbol{x}_0 - \boldsymbol{x}_{100+t})^2 + \boldsymbol{x}^2_{\,t+1} - \boldsymbol{x}^2_{\,100+t}) = 0$$

donde t = 100. Es fácil ver que este problema tiene el mínimo en $\boldsymbol{x} = (0,0,...,0)^T$. Las corridas fueron llevadas a cabo usando el cluster de estaciones de trabajo utilizado en la Sección 6.3.4. Debido a la diferencia en el desempeño de los procesadores, se utilizó nuevamente la métrica de speed-up ponderado descripta en el Apéndice D. Los resultados se muestran en la Tabla 6.3.

Tamaño del problema	Tiempo secuencial WS1	Tiempo secuencial WS2	Tiempo secuencial WS3	Tiempo paralelo	Speed-up ponderado
t = 50	8.6s	26.9s	10.3s	6.5s	1.84
t = 100	14.0s	40.3s	16.5s	10.5s	1.82

Tabla 6.3: Desempeño paralelo del algoritmo GRG

6.5. CONCLUSIONES SOBRE LA PARALELIZACIÓN DE ALGORITMOS SECUENCIALES DE OPTIMIZACIÓN

En este capítulo se presentó un análisis sobre la paralelización de algoritmos secuenciales existentes. El aspecto positivo más importante de este enfoque es que el algoritmo paralelo resultante hereda todas las propiedades matemáticas de la metodología secuencial, tanto en la robustez, velocidad de convergencia, etc. Esto se debe a que los métodos de optimización que son paralelizados usados han sido desarrollados y probados en forma exhaustiva durante años y hasta décadas.

En particular, en este capítulo se desarrollaron, implementaron y chequearon las versiones paralelas de las dos principales metodologías de optimización basadas en gradiente para el tratamiento de problemas no lineales restringidos: los métodos SQP y GRG. En cada caso se analizó la metodología básica, se identificaron las secciones de código que generan el mayor costo computacional y se desarrolló la estrategia para su paralelización. Finalmente se implementaron los dos algoritmos paralelos para verificar su desempeño. En el caso del algoritmo paralelo SQP se utilizaron diferentes casos de estudio académicos cuyas funciones objetivo y restricciones fueron recargadas en su tiempo de cómputo para simular el comportamiento de problemas de optimización más complejos.

Dado que por razones de simplicidad la versión paralela del método GRG sólo trataba casos con restricciones de igualdad, no se pudo realizar una comparación directa entre SQP y GRG. Sin embargo, el objetivo en este capítulo no fue presentar el “mejor” algoritmo de optimización paralelo sino analizar las estrategias que pueden ser aplicadas, ya sea para estos métodos u otros. En este sentido el cálculo de gradientes, la

actualización de Hessianos y el procedimiento de búsqueda en línea son partes fundamentales en otros algoritmos de optimización, como por ejemplo los métodos no lineales sin restricciones.

CAPÍTULO 7: CONCLUSIONES E INVESTIGACIONES FUTURAS

El objetivo de esta tesis ha sido explorar el campo del procesamiento paralelo distribuido utilizando redes de estaciones de trabajo para resolver problemas estructurales y numéricos que surgen en el campo de la ingeniería de sistemas de procesos. En el primer caso, se estudió específicamente el área del análisis de observabilidad en plantas de procesos aplicando búsquedas en profundidad de grafos y en el segundo caso, se abordó el problema de optimización aplicado a modelos con función objetivo y restricciones no lineales.

Como resultado de estas investigaciones se ha logrado el desarrollo de nuevas técnicas eficientes especialmente diseñadas para la clasificación de variables en procesos industriales y el desarrollo de metodologías novedosas para la solución paralela de problemas de optimización. El conocimiento desarrollado puede extenderse a otras áreas. En tal sentido, estamos desarrollando un nuevo esquema de distribución de tareas en algoritmos paralelos que es de aplicabilidad general.

Como corolario de esta tesis, en este capítulo se resumen los principales resultados obtenidos con cada una de las técnicas desarrolladas, el alcance de los nuevos descubrimientos y los lineamientos generales para futuras investigaciones en el tema. También se incluyen sugerencias sobre extensiones naturales del trabajo desarrollado en el marco de esta tesis.

7.1. CONCLUSIONES

7.1.1. ANÁLISIS DE OBSERVABILIDAD

El análisis de observabilidad de las variables involucradas en los modelos matemáticos correspondientes a plantas industriales constituye una herramienta esencial para mejorar el diseño de instrumentación y por ende, el control de procesos industriales. En esta tesis se consideró la paralelización de una técnica de reordenamiento estructural denominada GS-FLCN, que está basada en búsquedas sobre grafos especialmente adaptadas. La principal ventaja del método es su gran robustez para lograr la clasificación de variables; al mismo tiempo la naturaleza combinatoria del método constituye el aspecto fundamental que motivó su paralelización. Más específicamente, se desarrolló un algoritmo paralelo mediante la exploración simultánea de subgrafos en profundidad. La implementación del nuevo algoritmo, denominado GS-pFLCN, fue utilizada para el análisis de dos casos de estudio industriales reales, lográndose excelentes ganancias en cuanto a tiempos de ejecución con respecto a su versión secuencial. Estas ganancias se vieron reflejadas en un rango de utilización de 60-90% en los procesadores del cluster.

Debido a la importancia de la búsqueda en grafos tanto en el área de ingeniería de procesos como en otras disciplinas científicas, se desarrolló un nuevo algoritmo paralelo de búsqueda con una jerarquía de tareas master-supervisor-worker. Este método es aplicable a problemas que involucran encontrar todos los caminos sin ciclos de una longitud determinada en un grafo de interés. La principal ventaja de este algoritmo, en comparación con el método de búsqueda utilizado en GS-pFLCN, es que evita recorrer caminos repetidos, generando de esta manera búsquedas más eficientes. Se implementó

una versión paralela del método y se chequeó su desempeño con grafos que simulan en tamaño y densidad a aquellos que modelan a las plantas de proceso de tamaño medio o grandes. Se observó un excelente desempeño del mismo, debido fundamentalmente a que este algoritmo evita recorrer caminos repetidos. En vista de las características de diseño del algoritmo se pueden preveer ganancias aún mejores cuanto mayor sea el tamaño o densidad de los grafos.

Asimismo, como complemento indispensable para poder evaluar el desempeño de los algoritmos de observabilidad para casos de estudio reales grandes y/o complejos, se desarrolló un generador de modelos matemáticos de plantas de procesos especialmente adaptado para soportar paquetes de diseño de instrumentación. La herramienta, denominada ModGen, fue construida siguiendo un paradigma de desarrollo específicamente diseñado para aprovechar los rasgos típicos de estas aplicaciones, entre los cuales se destaca el alto nivel de especialización y la necesidad de manejar grandes volúmenes de información heterogénea. El paquete, que permite el tratamiento de problemas de grandes dimensiones, es modular y amigable. Agiliza y simplifica la labor del ingeniero por cuanto lo ayuda a construir y analizar modelos de procesos en forma sencilla, eficiente y confiable. En el contexto de esta tesis, su empleo era imperioso para posibilitar el planteo de los casos de estudio industriales sobre los cuales se verificó el desempeño de los nuevos algoritmos paralelos. Sin embargo, cabe destacar que el paquete constituye un módulo completo para acompañar cualquier algoritmo estructural de clasificación de variables en problemas de instrumentación. Más aún, el diseño de ModGen fue concebido previendo su futura expansión para posibilitar su uso en otras ramas de la ingeniería de procesos que también se basan en el análisis de modelos, tales como simulación dinámica ó diseño de procesos.

7.1.2. OPTIMIZACIÓN

Con respecto a la optimización numérica, es bien conocida su gran importancia en los ambientes tecnológicos y de producción actuales, tanto para el diseño como la simulación de procesos industriales. En esta tesis se determinaron dos estrategias básicas para la paralelización de problemas de optimización. Por un lado, se consideraron algoritmos inherentemente paralelos, es decir, algoritmos que son paralelos desde su propia concepción teórica. Por otro lado se exploró la paralelización de algoritmos de optimización clásicos, que fueron paralelizados transformando las secciones críticas de su código.

En el área de los métodos inherentemente paralelos utilizamos una técnica denominada DPV (Distribución Paralela de Variables), la cual estaba originalmente limitada a problemas sin restricciones. En el marco de esta tesis se trabajó en la extensión de su rango de aplicación a problemas con función objetivo y restricciones no lineales mediante el uso de modificadores. Además, se desarrollaron dos técnicas para tratar el problema de distribución de las variables en los bloques, aspecto nunca había sido tratado en la literatura. Estas propuestas permiten un mejor desempeño del algoritmo de optimización pues reducen la cantidad de iteraciones del algoritmo y el número de evaluaciones de funciones. Se analizó la efectividad de estas estrategias utilizando diferentes problemas de prueba académicos y un caso de estudio real basado en una planta de turboexpansión. En todos los casos se obtuvieron significativas mejoras de los tiempos de ejecución. Se observó que su empleo puede ser efectivo cuando la función objetivo y las restricciones del problema son computacionalmente costosas. En caso contrario, los costos involucrados en el pasaje de mensajes dentro del cluster

pueden tener preponderancia sobre el tiempo total del cómputo paralelo, en cuyo caso no se justifica su aplicación.

Con respecto al desarrollo de algoritmos paralelos a partir de versiones secuenciales, debe tenerse en cuenta que el algoritmo paralelo resultante hereda todas las propiedades matemáticas de la metodología secuencial, tanto en aspectos asociados a robustez como a velocidad de convergencia. Más específicamente, en el marco de esta tesis se analizó y desarrolló la paralelización de las dos principales técnicas de optimización basadas en gradiente para problemas no lineales con restricciones: SQP (Successive Quadratic Programming) y GRG (Generalized Reduced Gradient). Para estos casos se determinaron las secciones de código factibles de paralelizar y se propusieron las modificaciones correspondientes, midiéndose su efectividad a través de diferentes casos de prueba académicos. Las conclusiones obtenidas son similares al caso de los algoritmos de optimización inherentemente paralelos e indican que su aplicación a casos industriales reales puede ser muy efectiva cuando las funciones objetivo y restricciones son costosas en términos del tiempo de cómputo.

7.2. INVESTIGACIONES FUTURAS

A futuro se vislumbran interesantes perspectivas para continuar las investigaciones y resultados alcanzados en este trabajo de tesis. Un objetivo a corto plazo es incluir el algoritmo GS-pFLCN como una opción dentro de una herramienta específica para el análisis de observabilidad. Por otro lado, sería factible la utilización de las búsquedas paralelas en grafos desarrolladas en los Capítulos 3 y 4 en nuevas aplicaciones que se basan en estas búsquedas. Un caso específico sería su empleo en otros problemas donde se necesite un reordenamiento estructural diferente al requerido

por el análisis de observabilidad. Otra línea de investigación de gran interés sería extender los conceptos utilizados a otras formas de búsquedas, tales como las efectuadas a lo ancho y las basadas en patrones.

Con respecto al generador de modelos ModGen, si bien es completo en sí mismo y puede ser utilizado independientemente, el software fue diseñado con vistas a su futura incorporación al sistema de soporte de decisión completo para diseño de instrumentación que estamos desarrollando en nuestro grupo de trabajo. Básicamente, el DSS involucra nuevos módulos para análisis de observabilidad, detección de mediciones redundantes y reconciliación de datos de planta, así como también una variedad de herramientas de ayuda en la toma de decisiones. La etapa de observabilidad permitirá optar por diversos algoritmos secuenciales ó por la implementación paralelo descentralizada de GS-FLCN presentada en esta tesis. Se está trabajando actualmente en la implementación definitiva de un nuevo algoritmo para análisis de redundancias y se proyecta ocuparse en breve del ensamble definitivo de todos los módulos.

Por otra parte, en el área de optimización, el método de distribución paralela de variables podría ser extendido como herramienta para la resolución de sistemas de ecuaciones no lineales en paralelo. Sería posible particionar el vector de variables del sistema y utilizar un esquema tipo Jacobi en bloques para encontrar la solución del mismo. En este caso cada procesador sería responsable de actualizar las variables que le corresponden a su bloque. En este sentido vale destacar que no existen trabajos científicos conocidos en este tema.

En cuanto a la paralelización de algoritmos de optimización secuenciales un aspecto a desarrollar es el tratamiento de casos denominados MINLP (problemas no lineales con mezcla entera). Esta clase de problemas, que involucran variables discretas

en su formulación, pueden ser extremadamente costosos para resolver y son ampliamente usados, por ejemplo, en ingeniería de sistemas de procesos.

En definitiva, consideramos que el impacto de las investigaciones y resultados descriptos en esta tesis puede ser muy significativo, principalmente teniendo en cuenta sus alcances ya que son muchas las posibilidades de extender el rango de aplicabilidad de los métodos que fueron analizados en este trabajo.

APÉNDICE A: Algoritmos de búsqueda en profundidad Master-Worker

A.1 Algoritmo Master-pFLCN

```
      Datos de entrada: G(N, E)
      Datos de entrada-salida:

crear t tareas worker (tids)
// t: cantidad de procesadores disponibles
enviar G=(N, E) a todo tids
encolar todo tids en cola Q
crear pila  pilaMaster de recorrido en profundidad

Para cada nodo ∈ N hacer:
    apilar todo nodo en pilaMaster
    Mientras pilaMaster no está vacía
         nodo ← tope(pilaMaster)
         ady ← siguiente_adyacente(nodo)
         Si no tiene más adyacentes
              apilar ady en pilaMaster
              profundidadm ← profundidadm-1
              Si profundidadm=0
                   Si Q está vacía
                        MIENTRAS Q está vacía
                        recibir mensaje m(tid, tag) bloqueante
                        Según sea tag

                        caso: TotalPathCheck
                             desempaquetar camino
                             Si camino es solución
                                  broadcast StopSignal
                                  terminar ejecución
                             fin-si
                             terminar ejecución
```

```
                    caso: WorkerDoneSignal
                    encolar tid en Q
                    fin-mientras

               fin-si

        ... continúa en el algoritmo \ref mw-master2 ...

            fin-si
        fin-si
    fin-mientras
fin-para

               ... viene del algoritmo \ref mw-master ...

               tid ← tope(Q)
               enviar pilaMaster al worker tid
               desapilar pilaMaster
               profundidadm ← profundidadm+1
            fin-si
        sino
            desapilar pilaMaster
            profundidadm ← profundidadm+1$
        fin-si
    fin-mientras
fin-para

Mientras Q está vacía

    recibir mensaje m(tid, tag) bloqueante
    Según sea tag

    Caso: TotalPathCheck}
    desempaquetar camino
        Si camino es solución
            broadcast StopSignal
        terminar ejecución
        fin-si
    terminar ejecución
    Caso: WorkerDoneSignal
        encolar tid en Q
fin-mientras
```

A.2 Algoritmo Worker-pFLCN

```
      Datos de entrada: G(N, E)
      Datos de entrada-salida:

recibir grafo G
crear pila P de recorrido en profundidad
repetir

    recibir mensaje m(tid, tag) bloqueante
    Según sea tag

    Caso: ProcSubTreeSignal
    desempaquetar nodo, profundidadw
    desempaquetar pilaMaster
    apilar todo nodo ∈ pilaMaster en P
    Mientras P no está vacía
         nodo ← tope(P)
         ady ← siguiente_adyacente(nodo)
         Si no tiene más adyacentes
              apilar ady en P
              profundidadw ← profundidadw-1
              Si profundidadw=0
                   enviar P al master tid
                   desapilar P
                   profundidadw ← profundidadw+1
                   recibir m(tid, StopSignal) bloqueante
                   Si hay mensaje
                        terminar ejecución
                   fin-si
              fin-si
         sino
              desapilar P
              profundidadw ← profundidadw+1
         fin-si
    fin-mientras
    enviar WorkerDoneSignal a tid

    Caso: StopSignal
    terminar ejecución
fin-repetir
```

APÉNDICE B: ModGen: Un Generador de Modelos para Diseño de Instrumentación

B.1 INTRODUCCIÓN

Un modelo matemático de un proceso es un conjunto de ecuaciones algebraicas y/o diferenciales que representan el comportamiento real de la planta. La generación de modelos de procesos apropiados es una actividad compleja que requiere tiempo y habilidad. Las herramientas de modelado comúnmente reportadas en la literatura son generadores de modelos dinámicos cuyo objetivo es asistir a modeladores con una amplia gama de necesidades diversas. Marquardt1 presentó una revisión sobre modelamiento con ayuda de la computadora. Posteriormente, Bogusch y Marquardt2 destacaron la importancia de estructurar adecuadamente los modelos y desarrollaron VeDa, un modelo orientado a objetos. Según Vázquez-Román et al.3, las técnicas de modelamiento pueden clasificarse en dos categorías básicas. La primera corresponde a paquetes, tales como SpeedUp4 o gPROMS5, que contienen una librería de modelos para la descripción de unidades individuales. La segunda categoría abarca aquellos generadores basados en descripciones físicas del proceso mediante leyes fundamentales. Siguiendo este enfoque, Vázquez-Román et al.3 construyeron KBMoSS, un sistema de soporte de modelado con una interface de usuarios flexible implementada usando el paradigma orientado a objetos. Por su parte, Dieterich y Eigenberger6 desarrollaron Bimap, que emplea computación simbólica para manipular un conjunto de ecuaciones básicas una vez que el modelador ha definido las suposiciones y simplificaciones deseadas. Es importante destacar que una buena interface de usuario se vuelve una parte indispensable de un generador de modelos cuando los modelos de los procesos son

largos y complejos. Berrais7 discutió en detalle las consideraciones generales que deben tenerse en cuenta para lograr una interface de buena calidad en cuanto a criterios de performance del sistema e interacción con el usuario.

Aunque los principios que gobiernan los procesos son bastante generales, cada aplicación específica (simulación, optimización, etc.) exige demandas especiales, tales como distintos niveles de detalle y variadas facilidades de traducción para transformar modelos y vincularlos a los módulos resolvedores. En particular, el objetivo de este trabajo fue desarrollar un generador de modelos especialmente adaptado para propósitos de diseño de instrumentación, lo cual proporcionó un entorno adecuado para verificar el desempeño de nuevas estrategias de clasificación de variables. Los siguientes paquetes comerciales para reconciliación de datos de planta están disponibles en el mercado: DATACON, VALI II y SIGMA Fine97‡. Contienen interfaces guiadas por menúes que facilitan la definición de la topología del proceso y generan un conjunto de ecuaciones de modelo en forma automática. El software puede ser usado interactivamente para seleccionar nuevas mediciones con el objeto de cumplir requerimientos de redundancia. Sin embargo, la implementación de un generador de modelos es necesaria con fines de investigación porque los paquetes comerciales existentes constituyen verdaderas “cajas negras” pues las rutinas que los componen no pueden ser aisladas, extraídas o modificadas para ser adaptadas a necesidades especiales. Una capacidad deseable que posee el generador de modelos que se presenta en este apéndice es su flexibilidad para incluir ecuaciones desarrolladas por los usuarios para relacionar variables del proceso.

Teniendo en cuenta las consideraciones arriba mencionadas, se desarrolló una interface gráfica de usuario (GUI) basada en ventanas denominada ModGen (Vazquez et al, 2000) para la generación de modelos matemáticos de estado estacionario de plantas

de procesos. La herramienta fue diseñada específicamente para estudios de instrumentación, con herramientas para lograr una interacción sencilla con algoritmos que efectúan la clasificación de variables no medidas (análisis de observabilidad) sobre la base de técnicas estructurales. Esta interface hace posible efectuar el análisis riguroso de instrumentación de problemas grandes que abarcan plantas completas, tales como la planta de etano y la planta de amoníaco que se incluyen como casos de estudio en esta tesis. La posibilidad de incluir formulaciones no lineales y correlaciones especializadas, unida a algoritmos de análisis rigurosos y robustos condujo a resultados realistas altamente confiables. En estos casos, dada la magnitud y complejidad de los modelos correspondientes, tanto el ingreso de datos como los reordenamientos requeridos deben realizarse en forma automática. De lo contrario, la tarea de manipulación de la información resulta larga y tediosa, proclive a los errores humanos.

B.2 RASGOS PRINCIPALES

En esta sección se describen las características y funcionalidades más importantes de ModGen, incluyendo algunas pantallas que muestran cómo luce la interface. El entorno es amigable, permitiendo la interacción con gráficas de los equipos, instrumentos y tablas de corrientes en un estilo de presentación que resulta familiar a los ingenieros de procesos. De acuerdo con los estándares típicos de las GUI, hay barras de herramientas y menúes desplegables. El usuario puede ingresar, modificar y eliminar unidades fácilmente, elegir el nivel deseado de complejidad para las ecuaciones (balances de masa totales y por componentes, balances de energía y relaciones termodinámicas), introducir ecuaciones adicionales en términos simbólicos y definir las variables medidas.

Siempre que se ingresa un nuevo modelo, deben inicializarse algunos parámetros de aplicabilidad general. La Figura B.1 muestra la ventana donde el usuario decide qué balances se generarán por defecto para cada unidad.

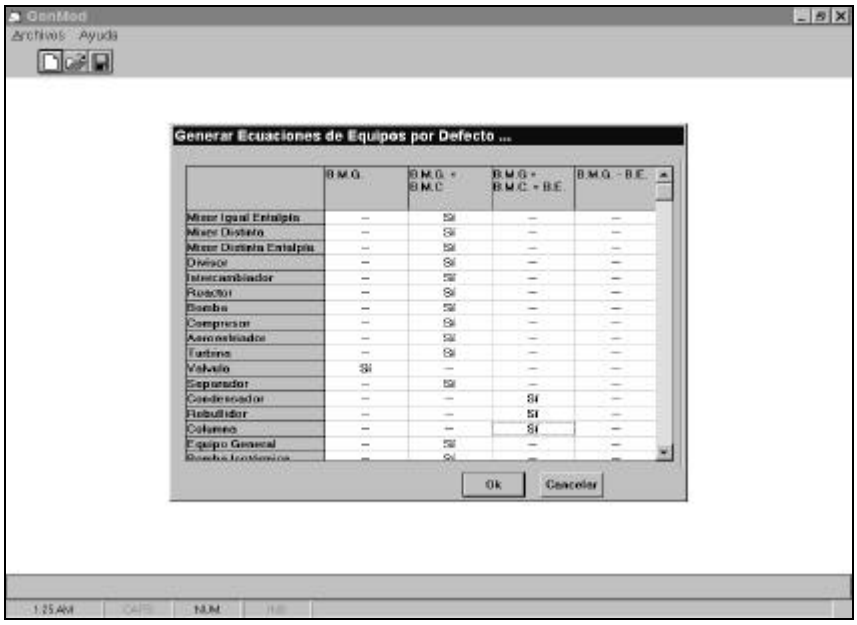

Figura B.1: Especificación de las ecuaciones de balance por defecto

La Figura B.2 corresponde a la selección del grado de complejidad de las ecuaciones termodinámicas para densidades, entalpías y constantes de equilibrio, mientras que la pantalla que se muestra en la Figura B.3 permite la especificación de los componentes de la planta. Los bloques sombreados resaltan algunos de los componentes elegidos para el caso de estudio específico que se está ingresando. Esta facilidad es práctica cuando se editan corrientes porque el usuario sólo tiene acceso a un conjunto reducido que contiene sólo los componentes relevantes. De este modo, esta información es más fácil de manipular.

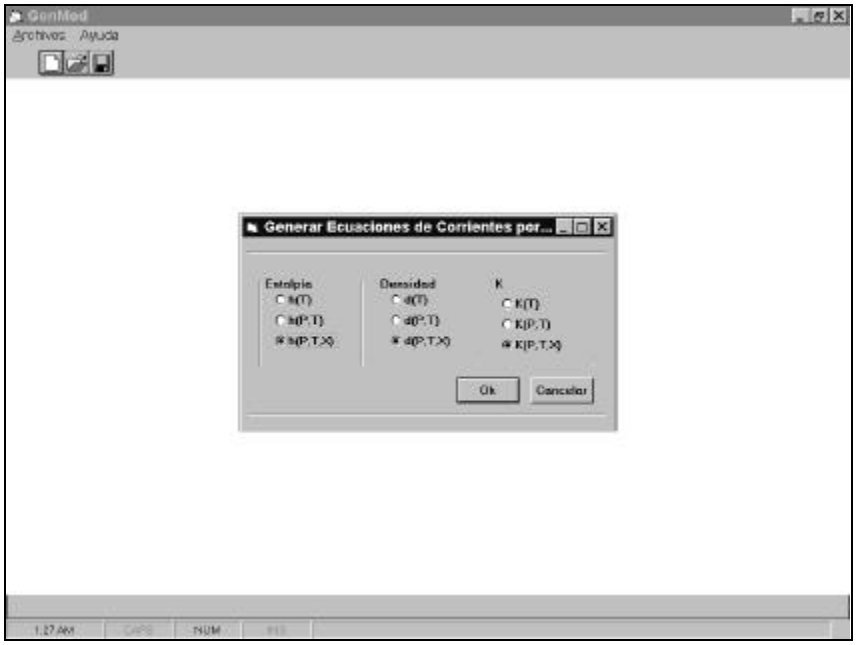

Figura B.2: Especificaciones por defecto de las funcionalidades para propiedades termodinámicas

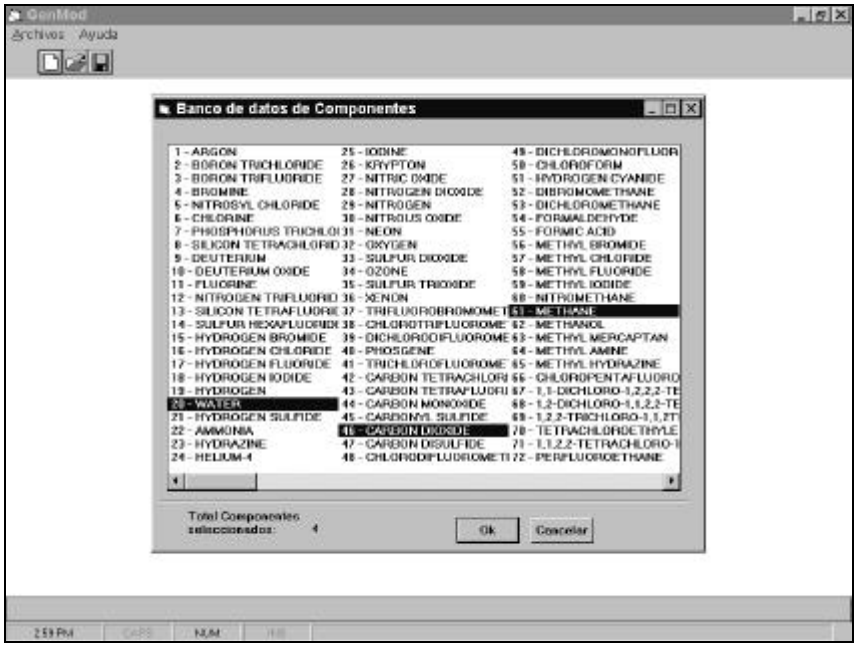

Figura B.3: Definición de los componentes por defecto

Después de que se ha cargado la información básica, aparecen los comandos principales de edición y visualización. Es importante puntualizar que todas las opciones por defecto pueden ser modificadas fácilmente para poder modelar cada unidad y corriente con el nivel de complejidad deseado La Figura B.4 muestra las ventanas asociadas a las unidades de procesos. Se observa una lista de unidades con los nombres que las identifican. Las representaciones gráficas como la mostrada en la figura permiten efectuar links desde cualquier equipo hacia las distintas corrientes asociadas y obtener detalles de la instrumentación.

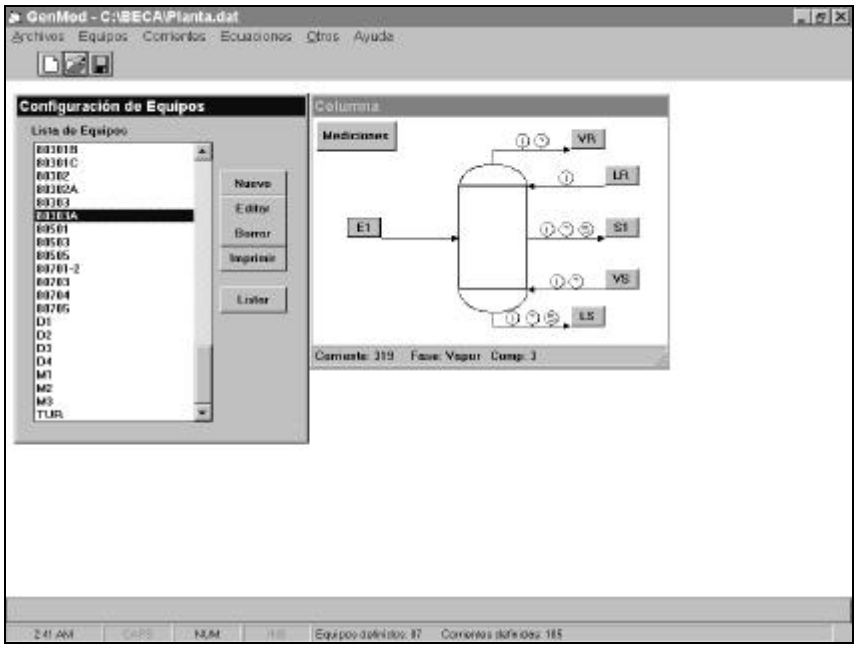

Figura B.4: Ventana principal para manipulación de equipos

Las Figuras B.5, B.6 y B.7 muestran algunas facilidades de edición. El botón de “Ecuaciones” despliega la ventana que se muestra en la Figura 5. Esta opción permite modificar el conjunto de ecuaciones definidas por defecto para un dado equipo, para que el usuario pueda adaptarlo a sus necesidades. En tal sentido, es posible agregar ecuaciones definidas por el usuario y también borrar ecuaciones existentes. Todo cambio se refleja inmediatamente en la ventana de visualización del sistema de ecuaciones,

donde todas las variables medidas aparecen subrayadas. Estos rasgos permiten individualizar aspectos clave del diseño en forma rápida y sencilla, lo cual simplifica la toma de decisiones por parte del usuario en la etapa de establecer si una solución intermedia es satisfactoria. Asimismo, se agiliza la tarea de identificar las modificaciones más apropiadas a efectuar en la instrumentación propuesta, con vistas a lograr la configuración óptima con una mínima cantidad de iteraciones globales.

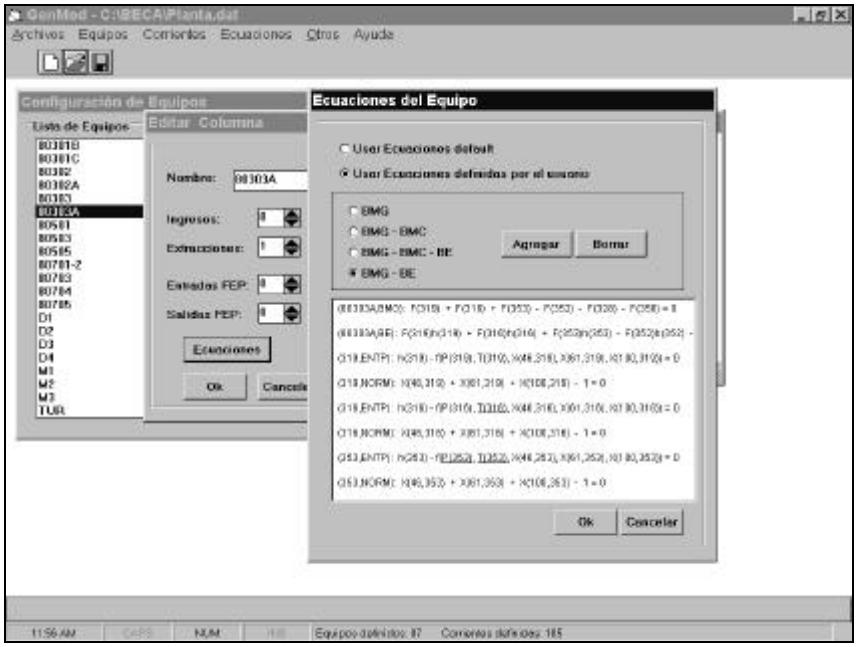

Figura B.5: Definición de ecuaciones de balance en forma individual

También es posible modificar las características de las corrientes, tales como composiciones o funcionalidades, en forma individual (Figura B.6) ó conjunta (Figura B.7).

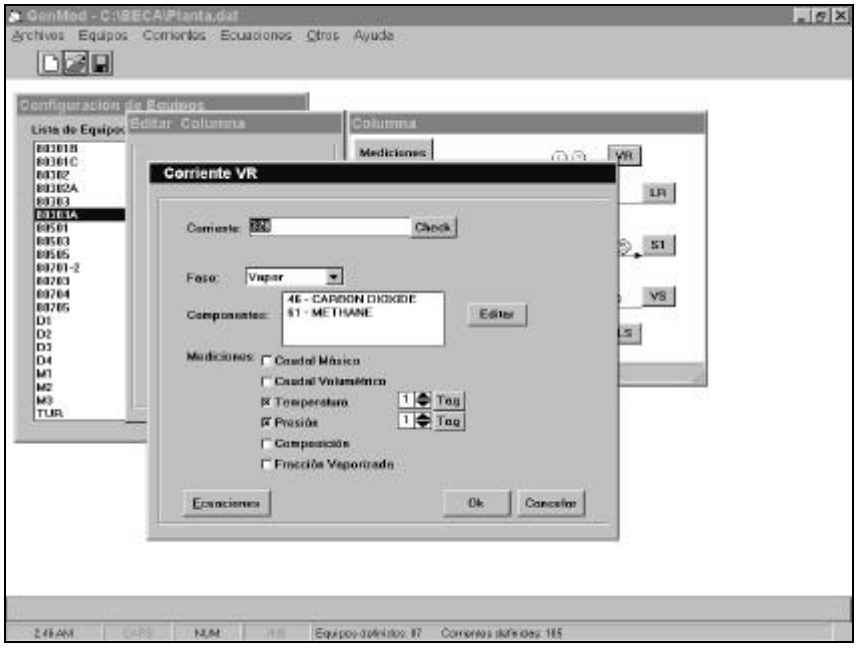

Figura B.6: Edición de una corriente

Para cambios grupales, el software incluye operadores de selección que permiten elegir ágilmente el conjunto de corrientes a modificar. Por ejemplo, la planta de etano (cita) tenía muchas corrientes de líquido cuyas propiedades termodinámicas no variaban significativamente con la presión. Por lo tanto, se decidió redefinir las funcionalidades para densidades y entalpías en forma acorde. La Figura B.7 muestra cómo se llevó a cabo esta modificación, eligiendo automáticamente a todas las corrientes líquidas a la vez mediante el operador de selección.

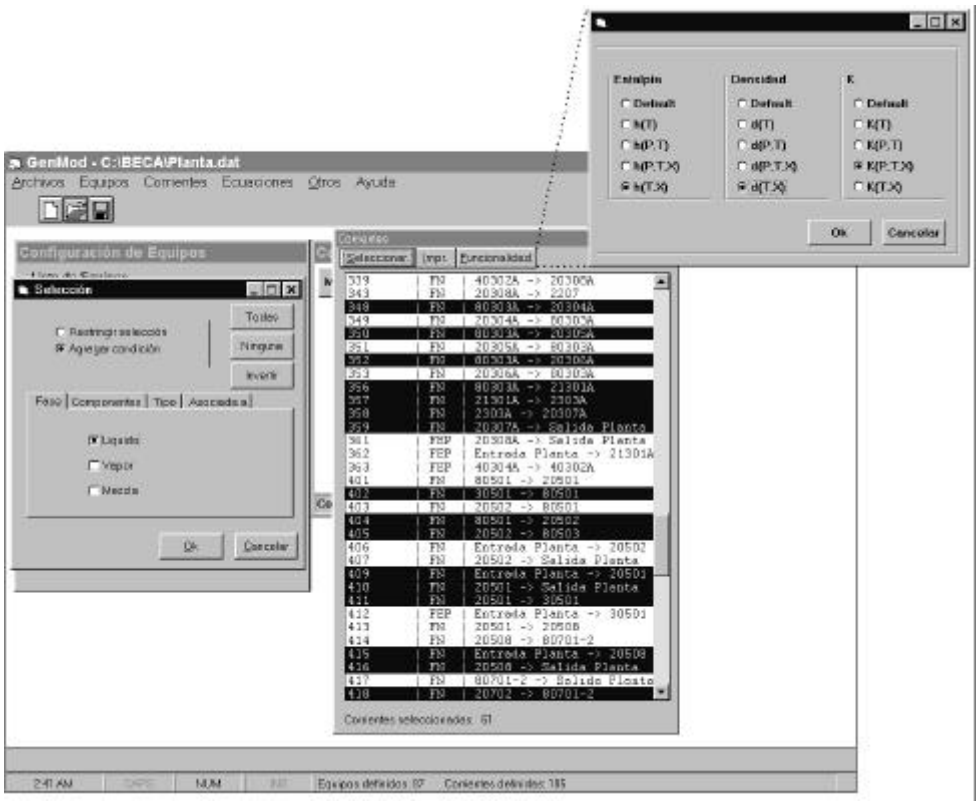

Figura B.7: Modificación simultánea de propiedades en diferentes corrientes

Cabe destacar que el operador de selección también facilita el análisis porque permite visualizar todas las corrientes que exhiben ciertas características solicitadas a través de la condición de selección. Por ejemplo, es posible visualizar rápidamente los sectores de planta que contienen un dado compuesto de interés.

La interface es inteligente por cuanto incorpora cierto conocimiento asociado a la fisicoquímica de los problemas. Más específicamente, la interface chequea la consistencia de los datos y define automáticamente aquellas características de las salidas que han quedado impuestas por los datos de entrada, tales como el estado ó la composición de algunas corrientes. De este modo, el software detecta muchos errores humanos que pueden deslizarse durante la entrada de datos, preserva la coherencia de la información y reduce significativamente la cantidad de tiempo requerida para ingresar los datos. Asimismo, ModGen permite construir modelos alternativos de distintos

tamaños y niveles de complejidad para la misma planta como modificaciones a partir del modelo previamente definido, preservando siempre la consistencia de los datos. Esto se ejecuta mediante actualizaciones automáticas. Por ejemplo, la planta de etano fue definida originalmente con 12 componentes. Posteriormente, para algunas corridas, se redefinió el conjunto de componentes para lograr un modelo simplificado que involucrara sólo los 4 componentes principales de la planta.

B.3 MODGEN COMO UNA HERRAMIENTA PARA DISEÑO DE INSTRUMENTACIÓN

Con respecto al análisis de los modelos, ModGen tiene rasgos únicos especialmente diseñados para interactuar con los algoritmos de diseño de instrumentación. La Figura B.8 muestra la interacción entre ModGen, el usuario y las herramientas de análisis de observabilidad. Toda la información requerida por la rutina de clasificación es provista por el generador de modelos. El usuario primero especifica la topología del proceso en forma interactiva empleando la interface. La GUI chequea la consistencia de los datos para asegurar la correctitud del modelo. Una vez que el modelo ha sido definido, es posible generar el sistema de ecuaciones algebraicas correspondiente y visualizarlo simbólicamente. Todas las variables seleccionadas como medidas aparecen subrayadas.

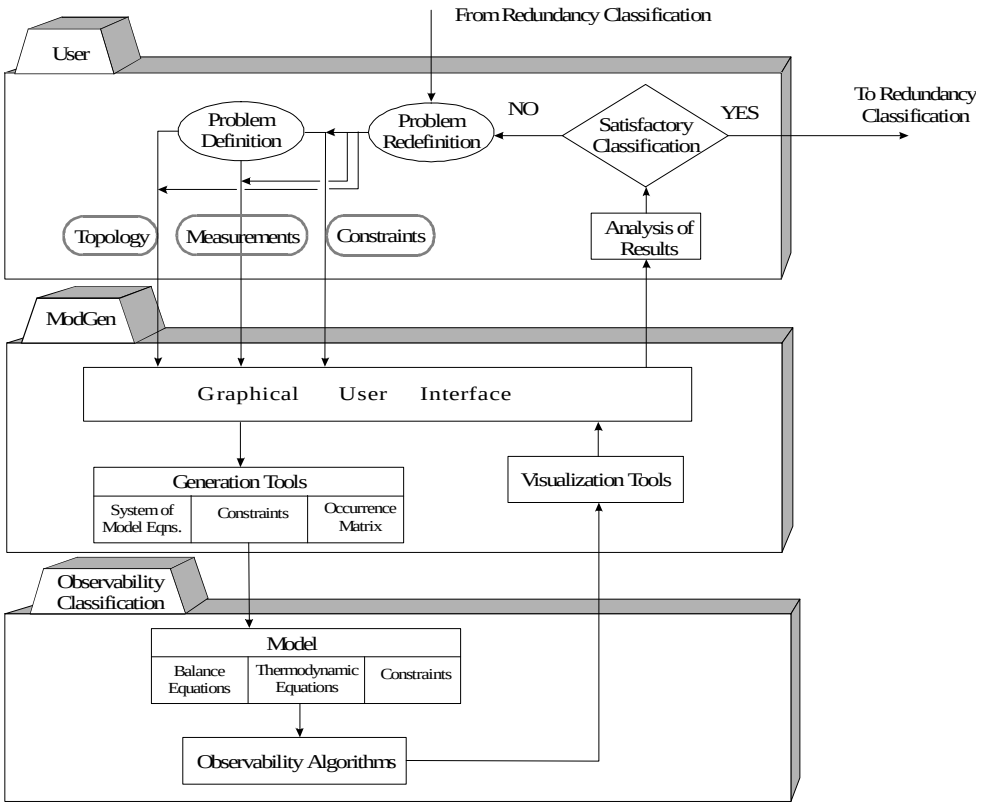

Figura B.8: El rol de ModGen en un entorno para análisis de observabilidad

Con respecto al análisis de los modelos, ModGen construye la matriz de ocurrencia cuyas columnas corresponden a las variables no medidas de acuerdo con el formato requerido por los algoritmos de observabilidad. Otro rasgo útil es la posibilidad de generar algunas restricciones en forma automática, con el objeto de prohibir la asignación de ciertas variables a una dada ecuación. Esto es necesario, por ejemplo, cuando se desea evitar despejes complejos ó procedimientos de evaluación de funciones numéricamente mal condicionados, los cuales resultan indeseables en las siguientes etapas del diseño de instrumentación (análisis de redundancias y reconciliación de mediciones). Además, ModGen genera información acerca de subconjuntos de asignación no permitidos porque representan corrientes paralelas ó contienen funciones implícitas. Estas prohibiciones son indispensables para lograr resultados aceptables de observabilidad que permitan concluir el diseño completo sin problemas numéricos

posteriores. Las corrientes paralelas deben evitarse porque están asociadas a subsistemas numéricamente singulares, mientras que el cálculo de variables no medidas mediante funciones implícitas resulta computacionalmente costoso pues requiere emplear un procedimiento iterativo para resolver ecuaciones no lineales. Finalmente, ModGen proporciona archivos auxiliares que traducen adecuadamente los resultados de los algoritmos de observabilidad y muestran la información de salida en un formato amigable.

La Figura B.9 muestra la ventana que exhibe un resumen de los principales resultados generales de una clasificación de variables para la planta de etano. La información incliye las dimensiones del modelo, el porcentaje de variables asignadas y la cantidad de subconjuntos de asignación encontrados para cada tamaño de bloque.

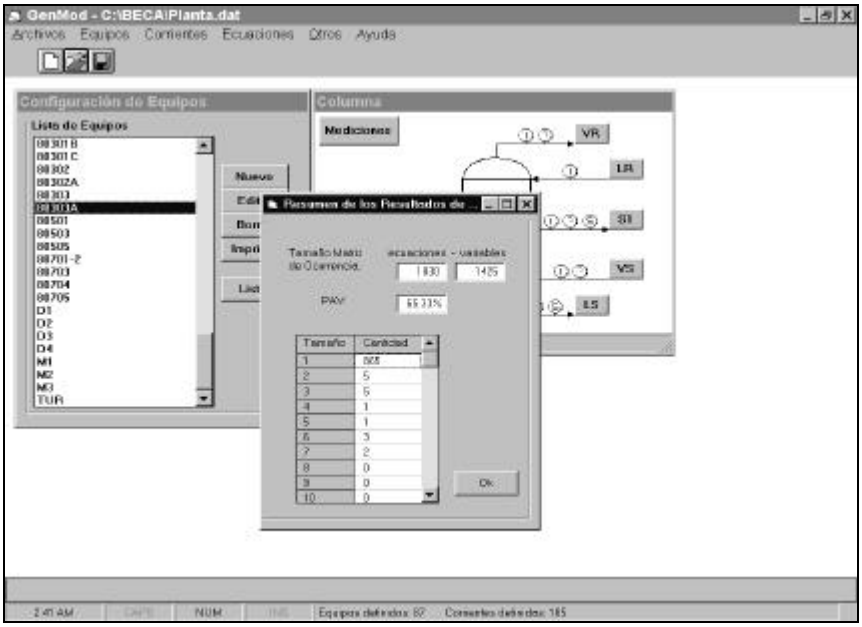

Figura B.9: Reporte de resultados de clasificación

ModGen también facilita la interacción entre el ingeniero y los algoritmos, ayudando en la interpretación y comparación de resultados. Por ejemplo, un punto clave es la admisibilidad de los subconjuntos de asignación hallados por el algoritmo de observabilidad. Si bien existen criterios automáticos para prohibición de conjuntos, suele resultar necesario que el ingeniero examine los subconjuntos remanentes para detectar problemas específicos no generalizables. Para facilitar esta tarea, los subconjuntos de asignación pueden ser examinados en forma individual como se muestra en la Figura B.10.

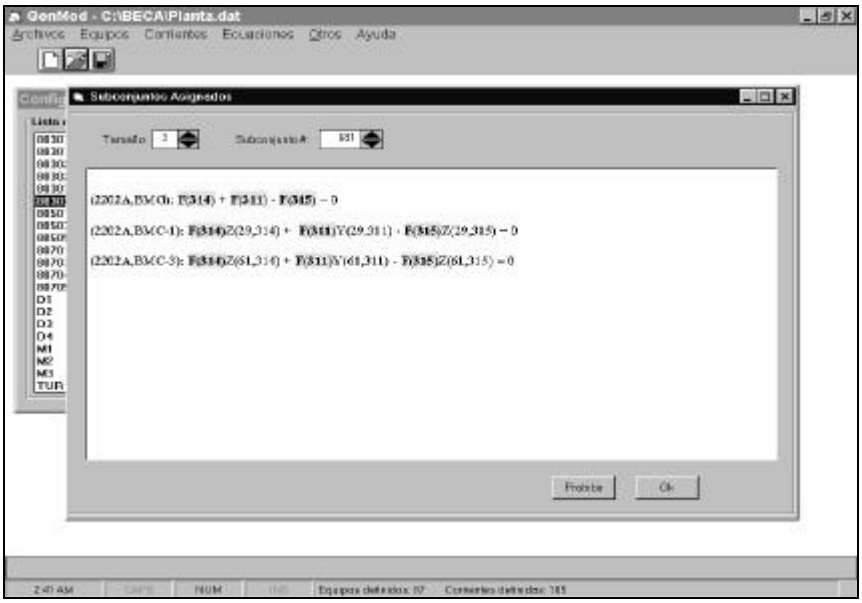

Figura B.10: Ventana con información sobre un subconjunto de asignación en particular

De este modo, ModGen ayuda al diseñador acelerando las tareas de generación de modelos, modificación de datos de entrada y visualización de resultados. Esto es sumamente ventajoso teniendo en cuenta no sólo la dimensión y complejidad de los problemas tamaño planta sino también el hecho de que el procedimiento completo de análisis de observabilidad involucra varias iteraciones globales. En primer lugar, el

ingeniero obtiene una clasificación inicial. Luego, debe evaluar las variables indeterminables remanentes, con el objeto de decidir qué mediciones deberían ser adicionadas ó eliminadas. La siguiente etapa es introducir unos pocos cambios en la instrumentación propuesta y recomenzar el procedimiento de clasificación. Cabe destacar que el agregado de cada medición puede disparar el desacople de otras ecuaciones, generando así un cierta cantidad de nuevas variables observables. Por lo tanto, no es conveniente introducir simultáneamente todas las variables indeterminables de interés como nuevas mediciones, ya que esto conduciría a un diseño con excesiva cantidad de sensores. El procedimiento completo se repite varias veces hasta que el usuario está completamente satisfecho con la clasificación obtenida, es decir cuando no quedan variables indeterminables de interés.

B.4 IMPLEMENTACIÓN

Este proyecto exhibe las siguientes características que influyeron en la política de implementación:

El software deseado era altamente dedicado porque el contexto en que se lo emplea se limita a aplicaciones específicas del área de la ingeniería de sistemas de procesos. La interface debía ser versátil y amigable en vista de la enorme cantidad de equipos, corrientes y mediciones involucrados en cualquier planta de procesos.

Debido al gran volumen de información heterogénea que debía manejarse, resultó necesario estudiar y definir las estrcturas de datos óptimas para lograr una manipulación eficiente.

Si bien el objetivo específico de este proyecto era contar con un generador de modelos de plantas en estado estacionario diversos grados de complejidad para resolver

problemas de diseño de instrumentación, muchas otras aplicaciones- tales como simulación y optimización de plantas- hacen uso de estas representaciones matemáticas. Por lo tanto, el sistema de software a desarrollar debía ser modular con vistas a simplificar su futura extensión para su uso en otras aplicaciones de interés en ingeniería química.

Los usuarios finales de la GUI siempre se encontraban disponibles para guiar al desarrollador del software en forma continua hacia un producto que satisfaga completamente todos sus requerimientos.

En vista de estas características, se definió un paradigma de software apropiado y se seleccionó un lenguaje de implementación que soportara dicho paradigma. Esto permitió una implementación eficiente, satisfaciendo las expectativas sobre calidad de productos y plazos de ejecución exigidas por los usuarios finales.

APÉNDICE C: Algoritmo de Busqueda MSW - Detalle algorítmico

A continuación se presentarán las referencias a los algoritmos *master*, *supervisor* y *worker* del método propuesto, junto con detalles de los mensajes que utilizan dichas tareas.

C.1 TAREA MASTER

El algoritmo *master* recibe como entrada el grafo sobre el cual se va a realizar la exploración. Cuando comienza su ejecución, crea las tareas *supervisor*, a las cuales envía el grafo de entrada y una orden de recorrer el árbol en profundidad a partir de un nodo en particular, que es distinto para cada tarea. Luego, la tarea *master* entra en un ciclo cerrado en donde espera el arribo de mensajes de cualquiera de los *supervisores*. Los mensajes recibidos pueden ser los siguientes:

- *ProcSubTreeQuery*: consulta de un *supervisor* acerca del *supervisor* encargado de la exploración de determinado nodo. Si dicho nodo aún no ha sido explorado, se prioriza este último sobre el resto de los nodos no explorados.
- *NextNodeQuery*: una consulta de un *supervisor* acerca del próximo nodo a explorar. Si ya se han explorado todos los nodos, el *master* le envía una indicación de dicha situación (*NoMoreNodes*).
- *TotalPathCheck*: indicación de que se ha encontrado un camino que constituye una solución. Como consecuencia, se envía a todas las tareas *worker* una indicación de finalización de ejecución (*Stop Signal*).

- *ReadyToCommit*: indicación de una tarea *supervisor* acerca de su intención de finalizar. Si todos los *supervisores* han enviado la misma intención, se les otorga el permiso de finalizar (*CommitGranted*).

Algoritmo MSW-Master

```
        Datos de entrada: G(N, E)
        Datos de entrada-salida:

crear t tareas supervisor (tids)
enviar G a todo tids
enviar ProcSubTreeSignal a tids
Repetir
      recibir mensaje m(tid,tag) en forma bloqueante
      Según sea tag
            Caso: ProcSubTreeQuery
                  desempaquetar nodo
      Si no se ha explorado nodo
            priorizar la exploración de nodo
                   sino
            enviar a tid el supervisor encargado de nodo

            fin-si
            Caso: NextNodeQuery
            Si hay nodos sin explorar
                  nodo ← nodo más pedido
            enviar ProcSubTreeSignal, nodo a tid
              sino
                  enviar NoMoreNodes a tid
            fin-si

            Caso: TotalPathCheck
            desempaquetar camino
            broadcast StopSignal
            terminar ejecución
```

```
            Caso: ReadyToCommit

            Si todo supervisor está listo para terminar
                  enviar CommitPermission a tids
                  terminar ejecución
            fin-si

fin-repetir
```

C.2 TAREA SUPERVISOR

La tarea *supervisor* es, sin duda alguna, la más compleja de las tres. Creada por la tarea *master*, de quien recibe el grafo de entrada, cada tarea *supervisor* a su vez crea una tarea *worker* y le envía el grafo recibido del *master*. Luego, el *supervisor* queda a la espera de mensajes, o bien realiza recorridos en sus estructuras internas, como se detalla a continuación.

Los mensajes recibidos pueden provenir de diversas fuentes dada la gran interacción entre cada *supervisor* y el resto de las tareas del sistema. Separaremos entonces la descripción de cada uno de los mensajes discriminando según su fuente. Los siguientes mensajes provienen del *master*:

- *ProcSubTreeIdentif*: respuesta a una consulta de identificación de la tarea encargada de procesar un determinado subgrafo.
- *ProcSubTreeSignal*: indicación de procesar un subgrafo, la cual se traslada a la tarea *worker* asociada.
- *NoMoreNodes*: indicación de finalización de exploración de todo el grafo. Como consecuencia, se indica a la tarea *worker* que finalice su ejecución.

- *StopSignal*: indicación de finalización de ejecución inmediata, la cual se traslada a la tarea *worker* y se finaliza la propia ejecución.
- *CommitGranted*: indicación de finalización debido a que se ha explorado exhaustivamente el grafo sin éxito. Con este mensaje finaliza la segunda fase del protocolo de terminación del algoritmo.

Los siguientes mensajes pueden ser recibidos desde otros *supervisores*:

- *GenerateAllPaths*: una indicación de otro *supervisor* que ordena iniciar la tarea de recombinación de caminos con un camino recientemente enviado. Este mensaje puede generar el envío de una señal de solución encontrada (*TotalPathCheck*), si es que alguno de los caminos resultantes de la recombinación satisface los requerimientos impuestos para establecerse como solución.
- *SubTreeEndQuery*: consulta acerca de la finalización de exploración de un determinado subgrafo. En caso afirmativo, puede generar una respuesta afirmativa (*SubTreeEnd*) al *supervisor* que realizó la consulta.
- *SubTreeEnd*: íntimamente ligado al mensaje anterior, constituye la respuesta a una consulta *SubTreeEndQuery* previa.
- *AckGAPMsg*: reconocimiento de otro *supervisor* de haber recibido y recombinado un camino previamente enviado.

Finalmente, los mensajes recibidos desde una tarea *worker* asociada a él pueden ser los siguientes:

- *SubPath*: mensaje que incluye un subcamino encontrado por la tarea *worker*. Produce la inserción de dicho camino en la estructura de registro de subcaminos local a la tarea *supervisor*.

- *WorkerDoneSignal*: indicación de la tarea *worker* de que éste ha finalizado de explorar un determinado subgrafo. Como consecuencia de ello, se envía a la tarea *master* un requerimiento de nuevos nodos a procesar (*NextNodeQuery*).

Cuando su worker está trabajando, la tarea *supervisor* recorre su estructura de registro de subcaminos, intentando obtener las mitades faltantes de los subcaminos aún no recombinados. En este recorrido se puede determinar, en base a la hoja de cada subcamino, cuál de las siguientes situaciones vale:

1. No se conoce la identidad de la tarea encargada de explorar dicho subgrafo.
2. Se conoce su identidad, pero no se sabe si se ha finalizado con la exploración.
3. El subcamino fue solicitado y está en condiciones de ser enviado para su recombinación con los subcaminos del registro de la otra tarea *supervisor*.

En los primeros dos casos, se generan los mensajes de consulta correspondiente para poder actualizar las estructuras de estado locales. En el último caso, se envían a recombinar los caminos terminados en esa hoja. A continuación se presenta el algoritmo en pseudo-código de la tarea *supervisor* (MSW-Supervisor).

C.3 TAREA WORKER

Cuando el algoritmo *worker* es iniciado por una tarea *supervisor*, éste recibe el grafo de entrada. Luego, el algoritmo entra en un ciclo cerrado a la espera de mensajes del *supervisor* asociado con él. Puede recibir dos tipos de mensajes:

- *ProcSubTreeSignal*: orden de explorar el grafo a partir de un nodo *n* con profundidad *prof*. Realiza un recorrido en profundidad clásico, acotado por *prof*, enviando cada camino desde *n* hacia una hoja a su correspondiente *supervisor* (*SubPath*). Cuando se agota la exploración del subgrafo actual, el *worker* envía una indicación de esta situación (*WorkerDoneSignal*).

- *StopSignal*: una orden de terminar la ejecución, ya sea porque se ha encontrado una solución, o bien se porque se ha explorado todo el grafo.

```
       Datos de entrada: G(N, E)
       Datos de entrada-salida:

recibir grafo G
crear pila P de recorrido en profundidad

Repetir
    recibir mensaje $m(tid,tag)$ en forma bloqueante
    Según sea tag

    Caso: ProcSubTreeSignal
          desempaquetar nodo, profundidad
          apilar nodo en P
          mientras la pila P no está vacía
               nodo ← tope(P)
               ady ← adyacente(nodo)
               Si no tiene más adyacentes
                    apilar ady en P
                    profundidad ← profundidad-1
                    Si profundidad = 0
                         enviar P a supervisor tid
                         desapilar P
                         profundidad ← profundidad+1
                         recibir m(tid,StopSignal)no bloqueante
```

```
                        Si hay mensaje
                            terminar ejecución
                        fin-si
                    fin-si

                sino
                    desapilar P
                    profundidad ← profundidad+1
                fin-si
            fin-mientras
            enviar workerDoneSignal a tid

    Caso: StopSignal
            terminar ejecución
fin-repetir
fin
```

APÉNDICE D: Speed-up ponderado

Uno de los aspectos a considerar en un sistema paralelo con procesadores heterogéneos es la redefinición de las métricas estándar definidas para computadoras paralelas. Debido a la diferencia en el desempeño de los procesadores, las medidas de speed-up o eficiencia, por ejemplo, no pueden ser aplicadas en forma directa. En este trabajo se ha desarrollado una nueva métrica de evaluación denominada speed-up ponderado para clusters de estaciones de trabajo heterogéneas, que tiene en cuenta la potencia de cálculo de cada máquina. Para esto definimos speed-up ponderado *WSU* (weighted speed-up) como:

$$WSU = \frac{\sum_{i=1}^{p} ST_i * CPF_i}{TM}, CPF_i = \frac{WCP_i}{\sum_{j=1}^{p} WCP_j}, WCP_i = \frac{\sum_{j=1}^{p} ST_j}{ST_i}$$

donde p es el número de procesadores, TM es el tiempo de cómputo paralelo total del algoritmo y WST_i, ST_i, CPF_i, WCP_i son el tiempo secuencial ponderado, el tiempo secuencial, el factor de capacidad de cómputo y la capacidad de cómputo ponderada para el procesador i respectivamente. De esta manera es posible considerar en la fórmula final el aporte individual de cada procesador, teniendo en cuenta el poder de cada uno de ellos a través del factor de capacidad de cómputo *CPF* y la y la capacidad de cómputo ponderada *WCP*.

REFERENCIAS

Abadie,J. & Carpentier,J. *Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints.* Academic Press, New York. (1969)

Amdahl, G. Validity of the Single-Processor Approach to Achieve Large-Scale Computing Capabilities. 30, 483. 1967. Proc. 1967 AFIPS Conf.

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

Bike,S. *Design of an Ammonia Synthesis Plant*. Carnegie Mellon University. (1985)

Birk,J., Liepelt,M. & Vogel,F. Computation of Optimal Feed Rates and Operation Intervals for Tubular Reactors *Journal of Process Control* **9,** 325-336. (1999)

Blackford,L.S., Choi,J., Cleary,A., D'Azevedo,E. & Demmel,J. *Scalapack Users' Guide*. SIAM. (1997)

Boderke,P., Wolf,M. & Merkle,H.P. Modeling of Diffusion and Concurrent Metabolism in Cutaneous Tissue *Journal on Theoretical Biology* **204,** 393-407. (2000)

Censor,Y. & Zenios,S. *Parallel Optimization: Theory, Algorithms, and Application*. Oxford University Press. (1997)

Chaudhuri,P. *Parallel Algorithms. Design and Analisys*. Prentice Hall, Sydney. (1992)

Chimowitz, E. H. and Bielinis, R. Z. Analysis of Parallelism in Modular Flowsheet Calculations. AIChE Journal 33[6], 976-986. (1987).

Dennis,J. & Torczon,V. Direct-Search Methods on Parallel Machines *SIAM Journal on Optimization* 937-954. (1991)

Dowd,K. & Severance,C. *High Performance Computing*. O´Reilly. (1998)

Duff,I.S., Erisman,A.M. & Reid,J.K. *Direct Methods for Sparse Matrices*. Oxford University Press. (1997)

Eckstein, D. M. and Alton, D. A. Parallel Graph Processing Using Depth-First Search. 21-29. Waterloo, Ont. Proceedings of the Conference on Theoretical Computer Science. (1977)

Ferris,M.C. & Mangasarian,O.L. Parallel Variable Distribution *SIAM Journal on Optimization* **4,** 815-832. (1994)

Frias,J.M., Oliveira,J.C. & Schittkowski,K. Modelling of Maltodextrin DE12 Drying Process in a Convection Oven *Applied Mathematical Modelling* **(en prensa)**.

Gau, C. Y. and Stadtherr, M. A. Parallel Interval Analysis for Chemical Process Modeling. First SIAM Conference on Computations in Science and Engineering, Washington DC. (2002)

Geist,A., Beguelin,A., Dongarra,J., Jiang,W., Manchek,R. & Sunderam,V. *PVM: Paralle Virtual Machine. A Users Guide and Tutorial for Network Parallel Computing*. MIT Press. (1994)

Gibbons,A. *Algorithmic Graph Theory*. Cambridge University Press, Newcastle. (1994)

Harrison, B. K. Exploiting Parallelism in Chemical Engineering Computations. AIChE Journal 36[2], 291-292. (1990).

Hartwanger,C., Schittkowski,K. & Wolf,H. Computer Aided Optimal Design of Horn Radiators for Satellite Communication *Engineering Optimization* **33,** 221-244. (2000)

High,K.A. & LaRoche,R.D. Parallel Nonlinear Optimization Techniques for Chemical Process Design Problems *Comp. & Chem. Engng.* **19,** 807-825. (1995)

Hock,W. & Schittkowski,K. *Test Examples for Nonlinear Programming Codes*. Springer-Verlag, New York. (1981)

Hough, P., Kolda, T. G., and Torczon, V. Asynchronous parallel pattern search for nonlinear optimization. SIAM Journal on Scientific Computing 23[1], 134-156. (2001).

Ingle, N. K. and Mountziaris, T. J. A Multifrontal Algorithm for the Solution of Large Systems of Equations using Network-Based Parallel Computing. Comp.& Chem.Engng. 19[6/7], 671-681. (1995).

Koelbel,C.H. & Zosel,M.E. *The High Performance Fortran Handbook*. MIT Press. (1993)

Kretsovalis,A. & Mah,R.S.H. Observability and Redundancy Classification in Generalized Process Networks - I. Theorems *Comp. & Chem. Engng.* **12,** 671-687. (1988)

Larsen, M. and Madsen, P. A Scalable Parallel Gauss-Seidel and Jacobi Solver for Animal Genetics. Lecture Notes in Computer Science 1697, 356-363. (1999).

Mangasarian,O.L. Parallel Gradient Distribution in Unconstrained Optimisation *SIAM Journal on Control and Optimization* **33,** 1916-1925. (1995)

Mittelmann,H. Parallel Multisplitting for Constrained Optimization *Parallel Algor. Appl.* **9,** 91-99. (1996)

Nash,S.G. Newton-type Minimization via the Lanczos Method *SIAM Journal on Numerical Analysis* **21,** 770-778. (1984)

NCSA. Netfinity Supercluster. http://archive.ncsa.uiuc.edu/SCD/Hardware/NTCluster/ . (2002).

Pacheco,P. *Parallel Programming With MPI*. Morgan Kaufmann Publishers. (1996)

Ponzoni,I., Sanchez,M.C. & Brignole,N.B. A New Structural Algorithm for Observability Classification *Ind. Eng. Chem. Res.* **38,** 3027-3035. (1999)

Powell,M.J.D. A Fast Algorithm for Nonlinearly Constraint Optimization Calculations *Lecture Notes in Mathematics* **630**. (1978)

Powell,M.J.D. The Convergence of Variable Metric Methods for Nonlinearly Constrained Optimization Calculations In: *Nonlinear Programming* (Mangasarian,O.L., Meyer,R.R. & Robinson,S.R., eds.), Academic Press. (1978)

Powell, M.J.D. On the quadratic programming algorithm of Goldfarb and Idnani. DAMTP 1983/Na 19. University of Cambridge. (1983)

Reghbati,E. & Corniel,D.G. Parallel Computations in Graph Theory *SIAM Journal on Computing* **2,** 230-237. (1978)

Reif,J.H. Depth-First Search is Inherently Sequential *Inform. Process. Lett.* **20,** 229-234. (1985)

Reklaitis,G., Ravindran,A. & Ragsdell,K.M. *Engineering Optimization*. John Wiley & Sons. (1983)

Romagnoli,J.A. & Sanchez,M.C. *Data Processing and Reconciliation for Chemical Process Operations*. Academic Press Inc., San Diego. (1999)

Romagnoli,J.A. & Stephanopoulos,G. On the Rectification of Measurement Errors for Complex Chemical Plants *Chem. Engng. Sci.* **35,** 1067-1081. (1980)

Sandia National Laboratories. The Computational Plant. http://www.cs.sandia.gov/cplant/ . (2002).

Scalapack. Scalapack Examples. http://www.netlib.org/scalapack/examples.html . 2002.

Schittkowski,K. *Nonlinear Programming Codes*. Springer. (1980)

Schittkowski,K. On the Convergence of a Secuential Quadratic Programming Method with an Augmented Lagrangian Search Direction *Mathematische Operationsforschung und Statistik* **14,** 197-216. (1983)

Schittkowski,K., Zillober,C. & Zotemantel,R. (1994) Numerical Comparison of Nonlinear Programming Algorithms for Structural Optimization *Structural Optimization* **7,** 1-28. (1994)

Scott, J. A. The Design of a Portable Parallel Frontal Solver for Chemical Engineering Problems. Comp.& Chem.Engng. 25[11/12], 1699-1709. (2001).

Seider,W.D., Seader,J.D. & Lewin,D.R. *Process Design Principles Synthesis, Analysis and Evaluation, Simulation of Process Flowsheets.* John Wiley & Sons. (2000)

Smith,J.M., Van Hess,H.C., Abbott,M.M. & Van Ness,H. *Introduction to Chemical Engineering Thermodynamics*. McGraw Hill College Div. (2000)

Vazquez, G. E. and Brignole, N. B. Parallel NLP Strategies using PVM on Heterogeneous Distributed Environments. Lecture Notes in Computer Science: Recent Advances in Parallel Virtual Machine and Message Passing Interface 1697, 533-540. Springer. (1999)

Vazquez, G. E. and Brignole, N. B. A Sensitivity-Analysis Approach for Domain Partitioning in Multisplitting Algorithms. Mang, H. A., Rammerstorfer, F. G., and Eberhardsteiner, J. Austria. Fifth World Congress on Computational Mechanics. (2002).

Vazquez,G.E., Diaz,S., Brignole,N.B. & Bandoni,A. Optimisation of Industrial Problems using Parallel Processing under Distributed Environments *Chemical Engineering Communications* **189,** 643-657. (2002)

Vazquez,G.E., Ponzoni,I., Sanchez,M.C. & Brignole,N.B. ModGen: A Model Generator for Instrumentation Analysis *Advances in Engineering Software* **32,** 37-48. (2001)

Yellen,J. & Gross,J.L. *Graph Theory and Its Applications - Discrete Mathematical and Applications Series*. CRC Press. (1998)

INDICE DE ALGORITMOS

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE