Universidad Nacional Autónoma de México
Posgrado en Ciencia e Ingeniería de la Computación

Una Estrategia de Aglomeración para el Mapeo de Procesos Paralelos, basada en Propiedades de Corte y Flujo en Gráficas

que para obtener el grado de
Maestro en Ciencias (Computación)

Presenta:
Ing. Juan Carlos Catana Salazar

Tutor:
Dr. Jorge Luis Ortega Arjona
Facultad de Ciencias

México, D.F. Abril 2013

Resumen

Un sistema paralelo es un conjunto de procesos comunicantes que cooperan para lograr un objetivo en común. La eficiencia de un sistema paralelo, además de involucrar el algoritmo (paralelo o secuencial) y el desarrollo del mismo, también concierne a otros factores que deben ser considerados para lograr un mejor desempeño del sistema.

Debido a estos factores que afectan directamente el desempeño, un sistema paralelo puede tener un mejor/peor desempeño y variar en su tiempo de ejecución. Uno de estos factores es el mapeo, es decir, la asignación de los procesos paralelos en los núcleos donde se llevara a cabo su ejecución. Otro factor importante es la granularidad de los procesos, o la relación entre el tiempo de comunicación/cómputo que éstos realizan.

Si dos procesos estan físicamente cerca uno del otro, el tiempo de comunicación entre ellos puede ser relativamente pequeño; pero si hablamos de comunicaciones entre procesos distantes geográficamente, donde el mensaje pasa por una red, entonces el tiempo de comunicación puede crecer y consumir mucho más tiempo.

Considerando lo anterior, la tarea es proveer una asignación de procesos, la cual minimice los tiempos de comunicación que son agregados al tiempo de ejecución del sistema. En otras palabras, ¿puede un mismo sistema paralelo variar considerablemente su tiempo de ejecución, sólo modificando la forma en que se asignan los procesos a los procesadores en un sistema distribuido?

A mi familia y amigos, que siempre me han apoyado en cada uno de las decisiones que he tomado a lo largo de mi vida.

A mis abuelos por todas las enseñanzas y pasión que nos heredaron.

Sinceramente a mis queridos tíos de Puebla, Amalia y Rodolfo por todas las muestras de cariño y apoyo para que pudiera formar las bases de mi carrera.

A mis tíos Maria de Jesus y Jose Luis, por el apoyo y motivación que siempre encuentro en ustedes. A mis primos, Evelyn e Ivan, por adoptarme y hacerme parte de su nueva familia.

A mis queridas tías Felicidad y Rocío, por todo el apoyo incondicional que me han brindado desde el día que nací y hasta el ultimo día que la vida me permita seguir aprendiendo de ustedes, las admiro.

Finalmente a mis padres, Angela y Celerino, por todo el amor con que me hicieron crecer y por todas las enseñanzas y sacrificios que han hecho, por el compromiso de lograr una mejor vida para nosotros.

Agradecimientos

Quiero agradecer profundamente a mi tutor, el Dr. Jorge Luis Ortega Arjona, por su paciencia y compromiso con sus estudiantes, gracias por dirigir este trabajo, y aun más agradecido por la gran persona que es.

Al Dr. Héstor Benítez Péres, por todas las facilidades que me brindo y por todos sus comentarios y criticas, por la cotutoria que hizo con este trabajo.

A cada uno de los miembros del jurado, agradezco al Dr. Sergio Rajsbaum, ya que sin sus enseñanzas durante sus cursos la base de éste trabajo no se me hubiera ocurrido. A la Dra. María Elana Lárraga, por sus comentarios y el tiempo que dedico a mi trabajo, y al Dr. Javier Gomez quien amablemente acepto participar en este trabajo.

Finalmente y de manera infinita, reitero mi agradecimiento y compromiso con la Universidad Nacional Autónoma de México, la cual me recibió con los brazos abiertos, y de la cual he aprendido muchas más cosas que las que se enseñan en sus aulas, gracias por el compromiso de tener siempre un mejor futuro y un mejor país.

Índice general

ÍNDICE GENERAL

Índice de figuras

ÍNDICE DE FIGURAS

Índice de cuadros

ÍNDICE DE CUADROS

1

Introducción

1.1. El Contexto

Tradicionalmente, el software de las computadoras ha sido escrito de forma secuencial, es decir, existe un único flujo de instrucciones, donde cada una de ellas se ejecuta estrictamente una a la vez por una sola unidad de procesamiento [8]. Por otra parte, el cómputo paralelo es una forma en la cual muchas instrucciones se llevan a cabo simultáneamente [7].

La expectativa principal del cómputo paralelo es conseguir un mayor desempeño. En resumen, el cómputo paralelo se percibe como el potencial de mejorar el desempeño, costo-desempeño y productividad de los sistemas de software [7]. El paralelismo puede encontrarse en tres niveles diferentes: por bit, por instrucción o por tarea. El paralelismo de tareas es característico de un programa paralelo, donde cálculos enteros pueden llevarse a cabo con el mismo o diferentes conjunto de datos [5].

Este trabajo aborda el paralelismo por tarea, donde un problema se divide en partes de forma que pueda resolverse concurrentemente. Cada parte está compuesta por una serie de instrucciones (y datos), cuya ejecución es simultánea sobre diferentes componentes de procesamiento [8].

El problema y/o los datos pueden descomponerse en tres formas diferentes: cuando los datos se dividen entre el número de procesos paralelos, se conoce como una descomposición de dominio; cuando el algoritmo se descompone en procesos paralelos, se dice que es una descomposición funcional; y cuando sucede una combinación de las anteriores, se conoce como paralelismo de actividad [1,8].

1. INTRODUCCIÓN

Las computadoras paralelas ofrecen el potencial de concentrar recursos computacionales para resolver problemas computacionales [1]. Una computadora paralela es un conjunto de elementos de procesamiento que son capaces de comunicarse y trabajar cooperativamente para resolver un problema computacional [1,7]. Esta definición es lo bastante amplia para incluir súper computadoras paralelas, con cientos o miles de procesadores, redes de estaciones de trabajo, estaciones de trabajo con múltiples procesadores y sistemas embebidos [1].

Un programa o sistema paralelo no solo tiene múltiples flujos de instrucciones en ejecución al mismo tiempo, si no también existen múltiples flujos de comunicación entre procesos [8]. Un sistema paralelo es mas difícil de escribir que su programa secuencial correspondiente, ya que la concurrencia agrega un nivel mas de complejidad [1,8,12], así como problemas de comunicación, sincronización y condiciones de competencia entre procesos, aunado al proceso de mapeo (calendarización de procesos) y balance en la carga de trabajo de las unidades de procesamiento. Todos los factores anteriores afectan directamente el desempeño del sistema paralelo [8].

El balance de carga se refiere a la distribución uniforme de trabajo entre los procesadores del sistema, tal que cada procesador realice un trabajo similar y se mantenga ocupado durante la mayor parte de tiempo. También puede ser considerado como una minimización del tiempo inactivo [8]. Las estrategias de mapeo proveen un balance de carga (que no siempre es el deseado) [8].

Comúnmente un programa paralelo puede clasificarse dependiendo de sus necesidades de comunicación y/o sincronización. La granularidad $G(p_i)$ de un proceso p_i es la métrica cualitativa que define la relación entre el tiempo de procesamiento t_{proc} respecto al tiempo de comunicación t_{com} [5,8].

Típicamente los periodos de cómputo de un proceso son separados por periodos de comunicación dados por eventos de sincronización [8].

$$G(p_i) = \frac{t_{proc}}{t_{com}}$$

Ésta relación entre procesamiento y comunicación deriva tres tipos de granularidad:

1. **Fina:** Se dice que un programa es de granularidad fina si el tiempo de comunicación es mayor al tiempo de procesamiento.

$$t_{com} > t_{proc}$$

2. **Media:** Se define como granularidad media cuando el tiempo de comunicación es aproximadamente igual al tiempo de procesamiento.

$$t_{com} \simeq t_{proc}$$

3. **Gruesa:** Se define cuando el tiempo de comunicación es menor al tiempo de procesamiento.

$$t_{com} < t_{proc}$$

La granularidad más eficiente depende del algoritmo y de la arquitectura de hardware en donde el programa sea ejecutado [8].

Cualquier sistema multiprocesador o máquina paralela debe hacer uso de una o más memorias para implementar comunicación. Existen diferentes arquitecturas con diferentes aproximaciones que difieren en el método usado para la comunicación de datos entre procesos [12]. A continuación se enuncian tres de ellas:

1. **Memoria compartida:** Todos los procesos tiene acceso directo (normalmente mediante un bus) a memoria física. Cada proceso tienen la misma "foto"de la memoria, y puede direccionar y accesar a la misma localidad de memoria lógica sin tener en cuenta físicamente dónde se encuentre. Los múltiples procesadores operan de forma independiente, pero comparten el mismo recurso de memoria. Los cambios en la memoria local que efectúa un proceso son visibles para cualquier otro proceso [8].

   La arquitectura de memoria compartida puede clasificarse en dos clases principales de acuerdo al tiempo de acceso [8,12]:

   - Acceso Uniforme a Memoria (UMA): Los tiempos de accesos son uniformes para cada proceso. Comúnmente representado por maquinas SMP [1].

[1] Múltiples procesadores simétricos (SMP por sus siglas en ingles): arquitectura de Hardware donde múltiples procesadores comparten un espacio de memoria global y acceso a todos los recursos; cómputo de memoria compartida [8].

- Acceso No Uniforme a Memoria (NUMA): Frecuentemente, realizado con enlaces de dos o mas SMPs, donde un SMP puede acceder directamente a la memoria de otro SMP en el sistema. No todos los procesadores tienen tiempo acceso uniforme a todas las memorias. Los accesos a memoria a través de el enlace son lentos.

2. **Memoria distribuida:** Son componentes de memoria independiente que requieren de una red de comunicaciones para establecer interconexión entre procesos. Los procesos pueden "ver"lógicamente la memoria local, y deben usar algún mecanismo de comunicación para accesar a la memoria de otras máquinas donde otros procesos se están ejecutando. Cada procesador tiene una memoria local propia. No existe un espacio de memoria global entre todos los procesadores del sistema. Todos los cambios que ocurran en la memoria local son relevantes únicamente para los procesos locales [8,12].

3. **Memoria compartida/distribuida:** Son un conjunto de máquinas conectadas por un medio de red, con múltiples procesadores compartiendo un espacio de direcciones de memoria distribuida. Cada procesador en una máquina sólo conoce su propia memoria local [8].

Aun cuando cada esquema presenta ventajas bajo ambientes propios, la diferencia principal entre ellos radica en los tiempos de comunicación. Bajo memoria compartida, se tiene que la comunicación es rápida y uniforme en tiempo de acceso, debido a la cercanía de la memoria con las unidades de procesamiento [8]. Por otra parte, sobre el esquema de memoria distribuida, los tiempos de comunicación son variables dependientes de las características del medio de red, protocolos de comunicación, ancho de banda y demás factores externos que ocasionan accesos no uniformes a memoria [8].

Por cada comunicación c entre dos procesos p_i y p_j, tales que ambos procesos están alojados en una misma memoria física, se agrega al tiempo de ejecución global TEG un tiempo de comunicación $t_{constante}$; en otro caso, por cada comunicación c entre dos procesos p_i y p_j, tales que ambos procesos están alojados disjuntos (en diferentes memorias físicas), se agrega el tiempo de ejecución global TEG un tiempo de comunicación $t_{variable}$, donde:

$$t_{constante} << t_{variable}$$

1.2. El Problema

El tiempo de ejecución global $TEG(S)$ de un sistema paralelo S, además de incluir el tiempo de cómputo $TC(S)$, es decir, el tiempo que toma calcular el resultado, incluye también los tiempos que inherentemente agregan las comunicaciones entre procesos del mismo sistema paralelo.

Bajo cualquier esquema de memoria, el tiempo que agregan las comunicaciones entre procesos impacta en el tiempo de ejecución global $TEG(S)$ del sistema paralelo. El impacto es proporcional a la cantidad de comunicaciones que el sistema paralelo efectúe y al tipo de comunicación que lleve a cabo.

Sean $c_l(S)$ la cantidad de comunicaciones locales (por memoria compartida) del sistema paralelo S, y sea $c_r(S)$ la cantidad de comunicaciones remotas del mismo sistema paralelo. Entonces al tiempo de ejcución global $TEG(S)$ del sistema paralelo le es agregado $c_l(S)$ comunicaciones de tiempo constante $t_{constante}$ más $c_r(S)$ comunicaciones de tiempo variable $t_{variable}$, con lo que se obtiene:

$$TEG(S) = TC(S) + c_l(S) * t_{constante} + c_r(S) * t_{variable}$$

Considerando lo anterior, lo deseable es maximizar la cantidad de comunicaciones locales $c_l(S)$ y minimizar (hasta donde sea posible) la cantidad de comunicaciones por paso de mensajes sobre la red $c_r(S)$, para agregar el menor tiempo de comunicaciones posible, evitando así que éstos no tengan un impacto importante sobre el tiempo de ejecución global del sistema paralelo.

Un escenario que puede agravarse considera procesos de granularidad fina en una ejecución sobre una arquitectura de memoria distribuida, es decir, un conjunto de procesos que se comunican frecuentemente (en relación al cómputo que llevan a cabo) por medio de una red. Un sistema de éstas características logra generar una gran cantidad de comunicaciones las cuales pueden saturar el medio de red y causar problemas, como pérdida de paquetes y la denegación de servicios de la red, entre otros.

1.3. La Hipótesis

Dado un conjunto de procesos, donde existe comunicación entre algunos pares de ellos, ¿hay una forma de aglomerar dicho conjunto, tal que al mapear las aglomeraciones

sobre un sistema distribuido se agregue un tiempo mínimo de comunicaciones al tiempo de ejecución global?

Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto de n procesos paralelos. Sea $C = \{q_1, q_2, ..., q_m\}$ tal que $q_g = (p_i, p_j)$, un conjunto de m parejas de procesos comunicantes. Sea $k \in \mathbb{Z}$ el número de aglomeraciones (particiones) de $P = \{P_1 \cup P_2 \cup ... \cup P_k\}$, tal que el costo de parejas de procesos comunicantes q_g que conectan a dos aglomeraciones P_a y P_b sea mínima.

1.4. La Aproximación

El conjunto P de procesos paralelos y el conjunto C de procesos comunicantes puede representarse como una gráfica de software $G_{sw} = \{P, C\}$, donde los vértices son indicados por cada elemento $p_i \in P$ y las aristas son representadas por comunicaciones establecidas por cada par ordenado $q_g \in C$. De la misma forma, la arquitectura de hardware puede representarse como una gráfica $H = \{V, E\}$, donde el conjunto de vértices V representa nodos o unidades de procesamiento, mientras el conjunto de aristas E indica las conexiones de red entre los vértices de una computadora paralela. Ambas gráficas son dirigidas, ponderadas, disjuntas y estáticas.

Una estrategia de aglomeración y mapeo se encarga de agrupar procesos con el objetivo de controlar la granularidad, es decir, mantener un control sobre las cantidades de comunicaciones entre pares de procesos, ya sea que éstos pertenezcan a una misma aglomeración o a distintas. Por su parte, el mapeo se encargar de encontrar una asignación proceso-procesador eficiente para la ejecución del sistema paralelo [2].

La estrategia de aglomeración debe considerar los tiempos de comunicación entre los procesos como un parámetro decisivo en la agrupación de procesos, de tal forma que pueda mitigar los tiempos de comunicación agregados al tiempo de ejecución, con el fin de reducir el impacto en el desempeño global del sistema paralelo.

La estrategia de mapeo es la encargada de encontrar un mapeo eficiente para la gráfica de software sobre la gráfica de hardware, es decir, encontrar una asignación de un proceso a un procesador eficiente para llevar a cabo la ejecución del sistema paralelo.

Para lograr lo anterior, se consideran las propiedades de corte y flujo, y el teorema de Corte mínimo Flujo máximo [3]. El cual aproxima la idea de un biparticionamiento

$G[A, B]$ de una gráfica G, considerando la menor cantidad de flujo que transita de una partición A a una partición B. Dicho teorema se basa en las siguientes propiedades:

1. Corte: Es una partición del conjunto de vértices.
2. Flujo:

   a) Una arista no puede tener un flujo mayor a su capacidad.

   b) La cantidad de flujo que entra a una arista es netamente igual a la cantidad de flujo que sale de ella.

Al trabajar con la gráfica de software G_{sw} se puede aplicar el algoritmo de Ford Fulkerson [3], el cual obtiene un corte mínimo en la gráfica, aglomerando los procesos paralelos en dos particiones A,B [3].

Para obtener k particiones del conjunto P de vértices, se aplica el algoritmo de forma *anidada* sobre las subgráficas resultantes, hasta obtener las particiones deseadas. El objetivo de obtener k aglomeraciones es llevar a cabo un mapeo uno a uno sobre los k nodos o unidades de procesamiento de la gráfica H de hardware, es decir, la cantidad de aglomeraciones que la estrategia consigue es igual a la cantidad de nodos en el sistema distribuido para realizar una asignación directa.

1.5. Contribuciones

La idea principal que provee la estrategia es basarse en los costos de comunicación entre pares de procesos, para realizar una aglomeración eficiente y poder realizar el mapeo sobre la arquitectura de hardware.

La estrategia modela como una gráfica de software al conjunto de procesos de acuerdo con las dependencias y cantidades de comunicación que éstos presentan. En dicha gráfica se aplica un algoritmo para obtener particiones del conjunto general de procesos, minimizando la comunicación. Cada bipartición de la gráfica garantiza ser un corte mńimo del conjunto de vértices, pero no así para los subconjuntos de la aglomeración obtenidos de forma anidada. Aún con esto el objetivo de reducir las cantidades de comunicaciones que van hacia el exterior de una partición se cumple, como se puede observar en el capítulo 5.

Con ésto se puede asegurar que la estrategia de aglomeración y mapeo minimiza (no de forma óptima) la comunicación por paso de mensajes entre procesos disjuntos, reduciendo el tráfico en la red y los retardos que la comunicación provoca.

1.6. Estructura de la Tesis

Para su objetivo la tesis se divide en los siguiente capítulos:

- El capítulo 2, establece los conceptos y terminología necesarios para fundamentar el desarrollo de la solución. Se describen tres metodologías: la primer metodología, llamada PCAM, propuesta por Ian Foster en su libro "Designing Parallel Algorithms"publicado en 1995; la segunda metodología llamada PGM, propuesta por K. Mani Chandy y Stephen Taylor, publicada en el libro "An Introduction to Parallel Programming"; y la última metodología llamada DAOM, propuesta por David Culler, Jaswinder Pal Singh y Anoop Gupta, publicada en el libro "Parallel Computer Arquitecture". Se describe a detalle el problema de mapeo y el problema de aglomeración, incluyendo las aproximaciones existentes para ambos problemas. Se presenta una explicación de teoría de gráficas y se muestre como es que se asocia el problema de aglomeración y mapeo con una representación en gráficas.

- El capítulo 3 presenta dos trabajos relacionados y menciona de qué manera existe una relación entre los trabajos y la propuesta de éste. Se presenta un artículo llamado "On the Mapping Problem"publicado en 1981 por Shahid H. Bokhari, donde muestra la importancia del mapeo para procesos paralelos, modela la estructura de comunicación entre procesos como una gráfica, y presenta un algoritmo heurístico eficiente para resolver problemas de mapeo.

  El segundo trabajo relacionado es el artículo titulado "Una Estrategia de Mapeo Eficiente para Programación Paralela", publicado por el Dr. Jorge Luis Ortega Arjona y por el Dr. Héctor Benítez Pérez, donde presentan una estrategia de mapeo eficiente basada en optimización de comunicaciones entre procesos, así como la distribución balanceada de tareas sobre procesadores en una red.

- El capítulo 4 muestra de forma detallada los pasos de la estrategia propuesta, iniciando con la generación de la gráfica de software en base al diseño del sistema

de software paralelo, la transformación de la gráfica de software a una red de flujo analizando los puntos hacia dónde fluyen los datos, y finalmente se especifica cómo es que el algoritmo del camino aumentado ayuda a realizar la aglomeración del conjunto de procesos, para lograr el mapeo de los subconjuntos aglomerados sobre la arquitectura de hardware.

- El capítulo 5 presenta un par de casos de estudio que se desarrollan en base a las especificaciones de la estrategia detalladas en el capítulo 4. El primer caso de estudio es el problema de ordenamiento, y su solución con el algoritmo de $MergeSort$. El programa paralelo asociado a la solución de este problema se ejecuta sobre una arquitectura de memoria distribuida. La aglomeración de tres conjuntos que se obtienen en base a la estrategia son mapeados sobre tres nodos del sistema distribuidos. Los tiempos de ejecución son comparados con los tiempos obtenidos con un mapeo usando la estrategia de Round Robbin.

  El segundo caso de estudio presenta el problema de la factorización de Cholesky. El programa paralelo asociado a éste caso de estudio, se ejecuta sobre la misma arquitectura de memoria distribuida, solo que ahora haciendo uso de 4 nodos de procesamiento. Primero se analizan los tiempos de ejecución bajo el mapeo con la estrategia de corte y flujo, posteriormente se comparan con los tiempos de ejecución bajo la estrategia de Round Robbin.

- Finalmente el capítulo 6 presenta una serie de ventajas, desventajas, y consideraciones adicionales que deben tomarse en cuenta para la aplicación de la estrategia, así como una sección de conclusiones donde se agregan los comentarios finales del trabajo, y por último se presenta la sección de trabajo futuro.

2

Antecedentes

El capítulo presenta un conjunto de metodologías para el diseño de software paralelo, y detalla cada una de sus etapas. Además se agrega una sección de teoría de gráficas, y finalmente se muestra la forma de modelar un sistema paralelo mediante éstas.

2.1. Metodologías de Diseño

Diversos autores han propuesto varias metodologías de diseño de software paralelo, con el objetivo de establecer pasos que ayuden a obtener fácilmente una traducción de un problema secuencial en un problema paralelo, y considerando también factores de desempeño y eficiencia. A continuación se describen tres metodologías de diseño de software paralelo: PCAM, propuesta por Ian Foster en [1], PGM, propuesta por K. Mani Chandy y Stephen Taylor en [4], y DAOM, propuesta por David Culler, Jaswinder Pal Singh y Anoop Gupta en [5].

2.1.1. Metodología PCAM

Foster sugiere una metodología de diseño estructurada en cuatro etapas nombrada por el acrónimo PCAM, en la que describe que, dado un problema, éste puede particionarse con el objetivo de generar tareas paralelas. Dentro de este conjunto existen pares de tareas que necesitan comunicación para lograr sincronización o sólo para paso de mensajes. Después de ser estructurada la conexión entre éstas sub-tareas, la aglomeración se encarga de agruparlas con el objetivo de encontrar un mapeo eficiente. Por ultimo, el mapeo es la etapa que se encarga de la asignación de los grupos de tareas

paralelas a las unidades de procesamiento [1]. La figura 2.1 muestra las cuatro etapas mencionadas, finalizando con el mapeo sobre los núcleos de ejecución.

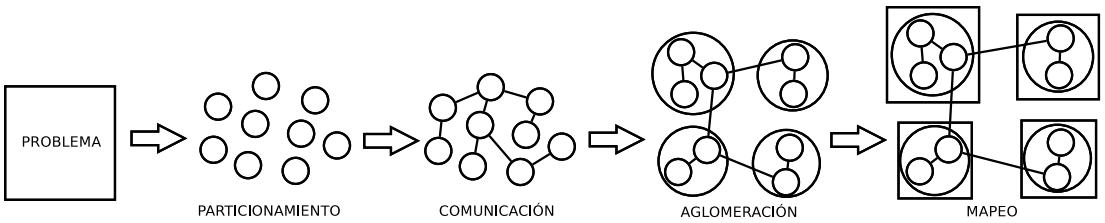

Figura 2.1: Metodología PCAM, propuesta por Ian Foster en 1995 [1].

2.1.2. Metodología PGM

Mani Chandy y Taylor proponen tres actividades que toman en cuenta una arquitectura paralela para mejorar el desempeño de un programa. Particionar un programa en componentes que pueden ser ejecutados en paralelo, ajustando la granularidad de estos módulos, y finalmente mapearlos [4]. La figura 2.2 muestra cómo las tres actividades propuestas en la metodología PGM transforman un problema inicial en un sistema de software paralelo.

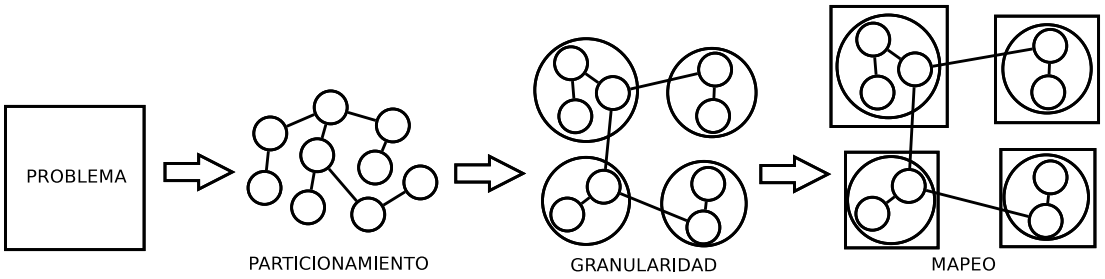

Figura 2.2: Metodología PGM, propuesta por K. Mani Chandy y Stephen Taylor en 1992 [4].

2.1.3. Metodología DAOM

Culler, et al, describen una metodología basada en cuatro pasos. Inicialmente, un programa es descompuesto en tareas (piezas de trabajo hechas por un programa), las cuales, mediante una asignación, son agrupadas en procesos. La orquestación crea la estructura para los accesos de datos necesarios, comunicaciones y sincronización entre

procesos, y finalmente mapea dichos procesos en las unidades físicas de procesamiento [5]. La Figura 2.3 ejemplifica las etapas de la metodología DAOM.

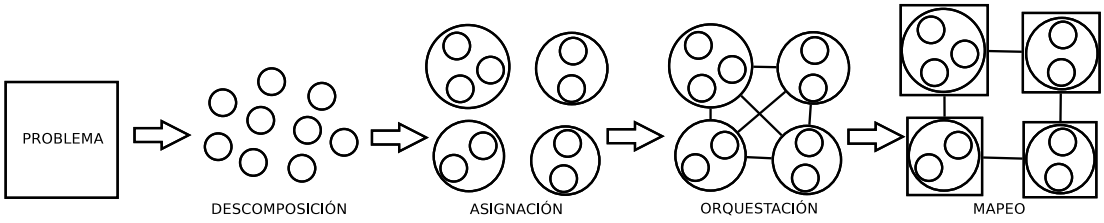

Figura 2.3: Metodología DAOM, propuesta por David Culler, Jaswinder Pal Singh y Anoop Gupta en 1997 [5].

2.1.4. Una Metodología Genérica

Una metodología genérica puede establecerse englobando las tres metodologías PCAM, PGM Y DAOM, considerando cuatro pasos que describen el proceso de diseño se software paralelo. La metodología genérica de diseño inicia estableciendo un problema general (problema secuencial), el cual necesita ser particionado o descompuesto en bloques que puedan ser ejecutados en paralelo. Posteriormente, se estructura el esquema de comunicaciones para los bloques paralelos. Esto depende de las necesidades de datos y/o sincronización que especifique el problema general.

Una vez hecho el esquema de comunicaciones, es necesario agrupar y ajustar la granularidad de cada bloque, con el fin de reducir los costos de comunicación entre grupos de bloques paralelos. El mapeo es una etapa común entre las metodologías, ya que ésta se encarga de la asignación de los grupos a los núcleos físicos de procesamiento de la computadora paralela. A continuación se describe detalladamente cada uno de los pasos de la metodología genérica:

1. **Particionamiento o Descomposición.** El particionamiento consiste en dividir un problema general en módulos que puedan ser ejecutados en paralelo. Esto no implica que la división del problema directamente genere un número de módulos igual al número de procesadores, ya que esta división no se realiza en base al número de procesadores disponibles [1,4]. El número de tareas puede variar dinámicamente en tiempo de ejecución, el número máximo de tareas en un tiempo

determinado delimita la cota superior de procesos (y por lo tanto de procesadores) que pueden ser usados [5].

Si alguna porción de un programa en ejecución no tiene tantos procesos como procesadores, entonces algunos procesadores estarán ociosos y el tiempo de ejecución será subóptimo [5].

Los objetivos principales de la partición son la escalabilidad y minimizar la latencia de comunicación entre procesos [4], así como expresar de manera eficiente el paralelismo, de modo que los procesos ocupen todo su tiempo en procesamiento. Sin embargo, no tanto así que la gestión de tareas llegue a compararse substancialmente a la labor útil [5].

La escalabilidad se refiere a cómo el desempeño de una partición dada cambia cuando se agregan procesadores, mientras que la latencia es el tiempo que tarda un mensaje en viajar a través del canal de comunicación hasta llegar a una computadora o procesador destino [4].

La escala de particionamiento puede indicar cuál es el mínimo número de procesadores que se emplea para resolver el problema. Es preferible particionar en un número de componentes mayor al de procesadores disponibles, para reducir la latencia de comunicación entre procesos [4].

2. **Comunicación.** Las tareas generadas por el paso anterior pueden ejecutarse concurrentemente, mas no de forma independiente. El procesamiento que cada tarea paralela hace típicamente está ligado a datos que puede obtener de otras tareas. Así, los datos deben ser transferidos entre pares de tareas paralelas, con el objetivo de completar el procesamiento global [1].

   La comunicación se requiere para coordinar la ejecución de estas tareas paralelas. La estructura de comunicaciones debe ser apropiada para satisfacer las necesidades de comunicación entre tareas definidas por la etapa anterior [1].

   El envío de un mensaje involucra un costo físico, por lo que se desea evitar agregar tantos canales y operaciones de comunicación innecesarias. Además se busca optimizar el rendimiento de operaciones de comunicación distribuida que permita ejecuciones concurrentes [1].

   Los tipos de comunicaciones puede clasificarse en cuatro grupos [1]:

- Comunicaciones Locales/Globales: Las comunicaciones locales, implican un proceso el cual solo se comunica con un conjunto pequeño de *vecinos*, o procesos geográficamente cercanos. En contraste, las comunicaciones globales, implican un proceso el cual requiere comunicación con la mayoría de los procesos del sistema paralelo.
- Comunicaciones Estructuradas/No Estructuradas: Un proceso puede establecer una *estructura* bien definida de comunicaciones, con sus vecinos, como la de árbol o malla. Por otro lado las comunicaciones *no estructuradas*, presentan un patrón arbitrario de comunicación.
- Comunicaciones Estáticas/Dinámicas: En comunicaciones estáticas, la identidad del par de procesos que se comunican, no cambia a lo largo del tiempo ejecución del sistema. De forma opuesta, las comunicaciones dinámicas, permiten que la identidad del par de procesos comunicantes, cambie y pueda ser altamente variable durante el tiempo de ejecución.
- Comunicaciones Síncronas/Asíncronas: Las comunicaciones síncronas, necesitan que tanto el proceso que envía datos, como el proceso que recibe datos, sean ejecutados de manera coordinada. Por contraparte, en las comunicaciones asíncronas, el proceso receptor puede obtener datos sin la necesidad de un proceso emisor coordinado.

Establecer la estructura de comunicaciones también depende del tipo de descomposición que se haya llevado a cabo, es decir, si la descomposición del problema y/o datos es por dominio, funcional o de actividad [1].

3. **Aglomeración y ajuste de Granularidad.** En las primeras dos etapas se obtienen las tareas paralelas y la estructura de comunicación que satisface los requerimientos de comunicación para dichas tareas. Hasta este punto la arquitectura de hardware no ha sido considerada para la ejecución del sistema. De hecho, la metodología de diseño de software paralelo puede ser altamente ineficiente si se tiene un número mucho mayor de tareas paralelas comparado con el número de unidades de procesamiento [1].

   Esta etapa de aglomeración, se encarga de agrupar tareas en subconjuntos para proveer tareas de mayor tamaño [1]. Consiste en un mecanismo por el cual

las tareas se distribuyen entre diferentes procesos. Dependiendo qué proceso es responsable de ejecutar un cierto tipo de cómputo [5].

La granularidad es la relación entre procesamiento y comunicación de una agrupación de tareas paralelas. Cuando un problema tiene una estructura regular, es posible variar la granularidad para incrementar el procesamiento o para reducir la comunicación. La idea central es emplear la *localidad*, es decir, la agrupación de las tareas para ayudar a reducir la comunicación entre procesadores [4].

Los objetivos principales de la aglomeración de procesos son: el balance en la carga de trabajo, la reducción de los costos de comunicación entre procesos, y la reducción del retardo de gestión de asignación [1,5]. Algunas estrategias se basan en heurísticas bien conocidas para determinar las aglomeraciones. El balance de carga incluye el tiempo de procesamiento, entrada/salida y accesos a datos o comunicación. Si la asignación es determinada completamente en tiempo de ejecución se dice que es una asignación dinámica [5].

4. **Mapeo:** Después del particionamiento y ajuste de granularidad, los componentes de software deben ser asignados o mapeados a los procesadores para su ejecución [4]. El mapeo es la acción de asignar procesos a procesadores. El problema de mapeo se define como el problema de maximizar el número de pares de procesos con comunicación que son asignados a procesadores directamente conectados [2].

   Cada tarea es asignada a un procesador de manera que intente satisfacer los objetivos de maximizar la utilización del procesador y minimizar los costos de comunicación. El mapeo puede ser especificado, estáticamente o determinarse en tiempo de ejecución por algoritmos de balance de carga [3]. La *localidad*, es un aspecto que debe tomarse en cuenta, es decir, para un par de procesos cuyo intercambio de información sea de granularidad fina, entonces éstos deben ser colocados lo mas "cerca"posible uno del otro, con el fin de reducir los tiempos de comunicación [2,4].

Hasta aquí es notable que son estas dos últimas etapas (Aglomeración y Mapeo) en las que se puede minimizar el tiempo de comunicaciones agregado al tiempo de ejecu-

ción global. Ya que, por una parte, la aglomeración considera la agrupación de tareas paralelas con el objetivo de reducir la granularidad de comunicaciones y maximizar el cómputo de cada proceso paralelo, mientras que el mapeo se encarga de asignar dichas aglomeraciones sobre las unidades de procesamiento para llevar a cabo la ejecución del sistema. Por lo que las siguientes secciones, se encargan de describir a detalle la aglomeración y el mapeo para conocer las limitaciones y características de cada uno de ellos.

2.2. Aglomeración, una descripción detallada

Dado que en la etapa de descomposición y particionamiento se establecen tantas tareas paralelas como sea posible, definir una gran cantidad de tareas de grano fino no produce necesariamente un programa paralelo eficiente [1]. La aglomeración considera si es útil agrupar o combinar tareas identificadas en la fase de descomposición, tal que se provee un número menor de tareas de mayor tamaño. También durante ésta etapa se considera si es necesaria la replica de datos y/o cómputo [1]. El replicado de cómputo, se emplea para reducir los requerimientos de comunicación y/o tiempo de ejecución. Es decir, agrupar procesos que hagan cómputo similar, con el fin, de reducir costos de comunicación, equilibrando la relación entre comunicación/cómputo [1].

El número de procesos obtenidos en la fase de aglomeración puede ser aun mayor al número de procesadores disponibles. De esta forma, no se da una solución al problema de mapeo. Alternativamente, se puede elegir obtener una cantidad P de subconjuntos aglomerados para una cantidad P de procesadores disponibles. En este caso se simplifica el problema de mapeo [1].

Un factor crítico que influye en el desempeño del programa paralelo, son los costos de comunicación debido a la cantidad de datos que se envían desde un proceso hasta otro, además de que el procesamiento debe esperar a que los datos sean enviados o recibidos. Se puede mejorar el desempeño reduciendo la cantidad de tiempo que tardan los procesos en comunicarse. Esto puede lograrse incrementando la granularidad de las particiones, aglomerando muchas tareas dentro de una (efecto $superficie - volumen$) [1]. En consecuencia, el efecto $superficie - volumen$ obtiene una descomposición de una dimensión mayor, típicamente mas eficiente. Debido a que se reduce la superficie

(comunicación) requerida por un volumen determinado (cómputo). Por lo tanto, desde el punto de vista de eficiencia, usualmente, es mejor incrementar la granularidad mediante la aglomeración de tareas, que reducir la dimensión de las particiones en la descomposición [1].

2.3. Mapeo, una descripción detallada

El objetivo común de cualquier estrategia de mapeo es minimizar el tiempo total de ejecución, de tal forma que la asignación de procesos a procesadores mejore el desempeño global del sistema [1]. El número de procesos no tiene que ser igual al número de recursos físicos de procesamiento disponibles. Si existe un número mayor de procesos, entonces éstos tienen que ser multiplexados sobre los procesadores disponibles. Si existen un número menor de procesos, algunos procesadores quedarán ociosos [5].

Los procesos pueden ser asignadas a cualquier unidad de procesamiento en diferentes formas, ya que el mapeo no afecta la semántica de los procesos. En particular muchas tareas pueden asignarse a una misma unidad de procesamiento [1] (también se puede considerar que una tarea pueda asignarse a diferentes unidades de procesamiento, lo cual queda fuera del contexto de este trabajo).

El problema de mapeo es de tipo $NP - Completo$[1], lo cual significa que no existe un algoritmo en tiempo polinomial que resuelva dicho problema eficientemente. Aun así, existen muchas estrategias y heurísticas que tratan de resolverlo en un tiempo considerablemente aceptable, ofreciendo una solución efectiva [1].

Formalmente, el problema de mapeo se conoce como Calendarización Multiprocesador. Una instancia del problema define [13]: Sea T un conjunto de tareas, sea m el número de procesadores en la arquitectura, $\forall\ m\ \in \mathbb{Z}^+$, sea longitud $l(t) \in \mathbb{Z}^+$ para cada $t \in T$, y sea $D \in \mathbb{R}$ un plazo límite. ¿Hay una calendarización m-procesador para T que cumpla con el plazo total D?, es decir, una función $\sigma : T \to \mathbb{Z}^+$ tal que, para todo $u \geq 0$, el número de tareas $t \in T$ para las cuales $\sigma(t) \leq u < \sigma(t) + l(t)$, no es más que m tal que, para toda tarea $t \in T$, $\sigma(t) + l(t) \leq D$.

Existen diferentes tipos de algoritmos que proveen un mapeo a partir de algoritmos de balance de carga. A continuación se describen algunos [1]:

[1]La prueba de NP-Completes se basa en reducir el problema de Clendarización Multiprocesador al problema de Partición el cual se muestra en la sección 2.5.

1. **Bisección recursiva:** Particiona el dominio en sub-dominios aproximadamente del mismo tamaño, intentando reducir los costos de comunicación. Puede considerarse como una estrategia análoga a divide y vencerás. Existen algunas modificaciones de esta estrategia, como la bisección recursiva no balanceada y la bisección de gráfica recursiva.

2. **Métodos probabilísticos:** Proveen un balance de carga eligiendo procesadores de forma aleatoria. Si el número de procesos es grande, entonces se espera a que cada procesador tenga aproximadamente el mismo número de procesos. Es recomendable para procesos de granularidad gruesa.

3. **Mapeo cíclico:** Es apropiado cuando la carga computacional varia entre procesos y existe una independencia en la carga de trabajo. Realiza una asignación secuencial de procesos a procesadores, dicha técnica también es conocida como Round Robbin.

4. **Algoritmos locales de balance:** Son útiles para situaciones donde los procesos son creados dinámicamente en tiempo de ejecución. Tiene un costo menor a las técnicas anteriores, ya que no requieren conocimiento del estado global del sistema, solo necesitan conocer un mínimo de información de los vecinos y su estado local para tomar decisiones con respecto al mapeo.

Como puede darse cuenta, el problema de mapeo es de tipo $NP - Completo$ por lo que dar una algoritmo óptimo en tiempo polinomial es imposible. Aún con esto, existen diversas aproximaciones que si bien no son óptimas, al menos tienen un buen desempeño para casos generales. Por otra parte, se puede observar que la aglomeración puede ayudar al mapeo si se considera un número de aglomeraciones igual al de unidades de procesamiento (o múltiplos de éste).

2.4. Teoría de Gráficas

Muchas situaciones del mundo real pueden ser representadas convenientemente como un diagrama, consistente de un conjunto de puntos y líneas que relacionan ciertos pares de ellos. Una abtracción matemática de situaciones de este tipo da pie al concepto de gráfica [12].

Una gráfica G es un par ordenado $(V(G), E(G))$ consistente de un conjunto $V(E)$ de vértices, y un conjunto $E(G)$ de aristas disjunto de $V(G)$, junto con una función ψ_G de incidencia, que asocia con cada arista $e \in E(G)$ un par no ordenado de vértices $u, v \in V(G)$ tal que si $\psi(e) = \{u, v\}$, entonces e se dice que une a u y v [12].

Las gráficas son también así nombradas, porque pueden ser representadas gráficamente mediante puntos que representan vértices y líneas que los interconectan (aristas), lo cual ayuda a entender muchas de sus propiedades [12]. La Figura 2.4 representa una gráfica no dirigida de seis nodos o vértices y seis aristas. La posición de las aristas y/o vértices usualmente no tiene importancia, es decir, no hay una forma correcta de dibujarlas [12].

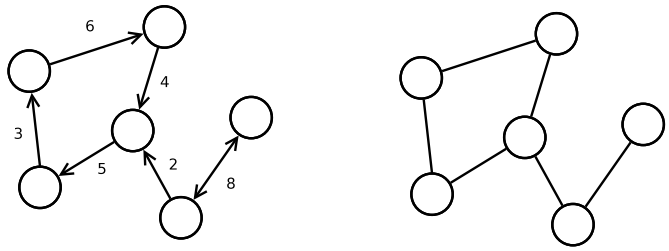

Figura 2.4: Representación de dos gráficas. Del lado izquierdo, una gráfica dirigida con aristas ponderadas. Del lado derecho, una gráfica no dirigida sin aristas ponderadas.

Se dice que una gráfica es simple si no contiene aristas paralelas, es decir, que existen dos o mas aristas cuyo par de vértices son los mismos, ni contiene aristas blucle, la cual es una arista que va de un vértice y finaliza en éste mismo [12].

Una gráfica dirigida o digráfica G_D de la misma forma que una gráfica (normal) es un par ordenado de vértices y aristas. La diferencia principal es la función de incidencia, ya que $\psi_{G_D}(a)$ asigna a cada arista o *arco*[1] un par ordenado de elementos de $V(G_D)$. Si $a \in E(G_D)$ es un arco entonces $\psi_{G_D}(a) = \{u, v\}$, donde a se dice que une a u con v o que u domina a v (y no viceversa para ambos casos). Entonces el vértice u es la cola de a y el vértice v es la cabeza de a [12].

Una digráfica *estricta* es una gráfica dirigida que no contiene arcos paralelos, ni arcos bucle, de la misma forma que una gráfica simple [12].

Un *camino* es una gráfica simple cuyos vértices pueden ser puestos en una secuencia lineal del tal forma que dos vértices son adyacentes si ellos son consecutivos en la

[1]El el caso de gráficas dirigidas puede usarse indistintamente la palabra arista o arco para hacer referencia a los elementos de $E(G_D)$ [12].

secuencia, y son no adyacentes de cualquier otra forma. La longitud de un camino es el número de sus aristas [12].

Una grafica G se dice *conexa* si, para cada partición del conjunto de vértices en dos conjuntos no vacíos X, Y, existe una arista $e \in E(G)$ de la forma $\psi(e) = \{x, y\}$ tal que $x \in X$ y $y \in Y$, de cualquier otra forma, la gráfica es *disconexa* [12].

2.4.1. Red de flujo

Una red de flujo $G_{nf} = (V(G_{nf}), E(G_{nf}))$ es una digráfica estricta, donde cada arco $a = (u, v) \in E(G_{nf})$ tiene una capacidad no negativa $c(a) \geq 0$. Si $E(G_{nf})$ contiene un arco $a = (u, v)$, entonces no puede existir un arco $a_r = (v, u)$ en la dirección contraria [6].

En una red de flujo se consideran dos vértices especiales, un vértice fuente $s \in V(G_{nf})$, y un vértice sumidero o vértice fin $t \in V(G_{nf})$, el vértice fuente es el encargado de generar flujo que es encausado por las aristas de la red [3,6].

Se asume que para cada vértice v de la red de flujo G_{nf}, v se encuentra en al menos un camino de s a t, es decir, $\forall\, v \in V(G_{nf})$. La red de flujo contiene un *camino* $s \to v \to t$, por lo tanto la gráfica es *conexa*, donde cada $v \in V(G_{nf}) - \{s\}$ tiene al menos una arista incidente entonces $|E(G_{nf})| \geq |V(G_{nf})| - 1$ [6]. Normalmente una red de flujo no tiene aristas incidentes en el vértice fuente s o el flujo incidente en s es 0 [6].

Sean A y B dos subconjuntos contenidos en $V(G_{nf})$, entonces:

1. **Corte**

   Un corte de $s - t$ (de un vértice s a un vértice t), donde $s \in A$ y $t \in B$, es una partición de $V(G_{nf})$ en dos grupos $G_{nf} : [A, B]$. La capacidad de un corte se define como $cap(A, B)$ igual a la suma de las capacidades de cada arista $e \in E(G)$ que sale de A [3]. La Figura 2.5 ejemplifica un corte sobre una red de flujo.

   $$cap(A, B) = \sum_{e \text{ sale de } A(G)} c(e)$$

   El problema del corte mínimo se refiere a encontrar un corte $s-t$ tal que $cap(A, B)$ sea la mínima posible [3].

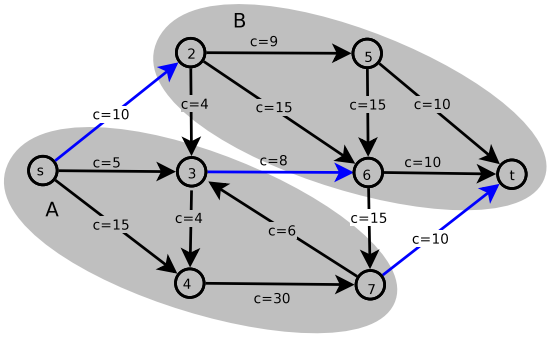

Figura 2.5: Ejemplo de una red de flujo. Las partes sombreadas forman la agrupación A y B respectivamente. La suma de las capacidades de las aristas (s,2),(3,6) y (7,t) iluminadas de color azul, definen la capacidad del corte A, $cap(A, B) = 28$.

2. **Flujo**

   Un flujo de $s - t$ (de un vértice s a un vértice t) se define como una función que satisface [3]:

   a) Capacidad: Para toda arista $e \in E(G)$, el flujo $f(e)$ asignado a ella, debe ser menor o igual a la capacidad $c(e)$ de la misma.

   $$0 \leq f(e) \leq c(e),\ \forall\ e \in E(G_{nf})$$

   b) Conservación: Para cada vértice $v \in V(G) - \{s, t\}$, la suma de flujo asignado $f(e)$ de cada arista e que entra a v, es igual a la suma de flujo asignado $f(e)$ de cada arista que sale de v.

   $$\sum_{e\ entra\ a\ v} f(e) = \sum_{e\ sale\ de\ v} f(e),\ \forall\ v \in V(G) - \{s, t\}$$

   La cantidad de flujo que entra a un vértice debe ser igual a la cantidad de flujo que sale del mismo, sin violar la propiedad de capacidad de cada arista (como puede observarse en la figura 2.6). Dicha propiedad de conservación es equivalente a la ley de Kirchhoff [6].

   $$fval(e) = \sum_{e\ sale\ de\ v} f(e)$$

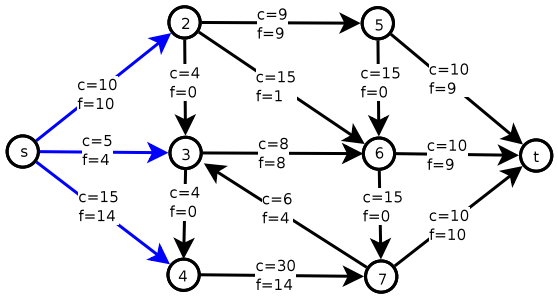

Figura 2.6: Red de flujo. Las aristas (s,2),(s,3) y (s,4) iluminadas de color azul, denotan el flujo saliente de s. Por lo tanto $fval(s) = 28$.

El valor del flujo $fval(v)$ de un vértice $v \in V(G)$ es igual a la suma de flujos asignados $f(e)$ de cada arista $e \in E(G)$ que sale de v [3]. La figura 2.6 ejemplifica el valor de flujo de vértice s.

El problema de flujo máximo consiste en encontrar un flujo $s-t$ de valor máximo [3], enviado desde una fuente s hasta un sumidero t, sin violar la restricción de capacidad en las aristas [6].

3. **Lema del Valor de Flujo**

   Sea f cualquier flujo asociado a alguna arista $e \in E(G_{nf})$ y sea (A, B) cualquier corte $s-t$ de la red de flujo [3]. Entonces, el flujo neto $netf(f_n)$ enviado a través del corte es igual a la suma de flujo f_{out} que sale de todo vértice $v \in A$, cuya arista se dirige a un vértice $u \in B$, menos la suma de flujo f_{in}, de toda arista que entra por algún vértice $x \in A$, proveniente de algún vértice $y \in B$ [3].

$$netf(f_n) = \sum_{e\ sale\ de\ A} f_{out}(e) - \sum_{e\ entra\ a\ A} f_{in}(e)$$

   La figura 2.7 muestra una red de flujo, donde se ejemplifica el lema del valor de flujo.

2.4.2. Gráfica residual

Una red de flujo residual o gráfica residual $G_r = (V_{nf}(G_r), E_{nf}(G_r) + E_r(G_r))$ está formada por los vértices y aristas de la red de flujo y el conjunto $E_r(G_r)$ de aristas de la gráfica residual con capacidades positivas [3]:

$$E_r(G_r) = \{e = (u, v) : f(e) \leq c(e)\} \cup \{e^r = (v, u) : f(e_r) = f(e) \geq 1\}$$

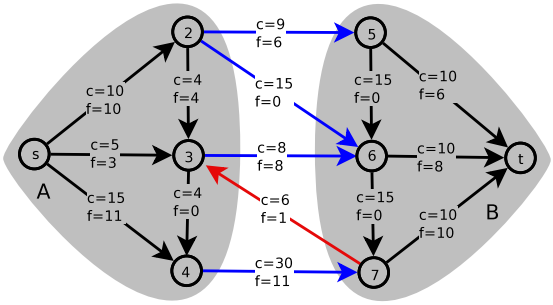

Figura 2.7: Red de flujo. Las aristas (2,5),(2,6),(3,6) y (4,7) iluminadas de color azul, denotan el flujo saliente $f_{out}(e)$ de la partición A, mientras que la arista (7,3) iluminada de color rojo, representa el flujo entrante $f_{in}(e)$ a la partición A. Por lo tanto el valor del flujo $netf(f_n)$ es de 24 unidades.

Dada un arista $e \in E_{nf}(G_r)$, donde $e = (u, v) \mid u, v \in V_{nf}(G_r)$, y $f(e) = x$ y $c(e) = y$, tal que $x, y \in N$ y $x \leq y$. Entonces una arista residual $e_r = (v, u) \in E_r(G_r)$ es capaz de revertir flujo. Para cada arista residual e_r la capacidad residual asignada a la misma se define como [3]:

$$c^r(e) = \begin{cases} c(e) - f(e) \text{ si } e \in E \\ f(e) \text{ si } e^r \in E \end{cases}$$

Una arista en la gráfica residual, puede admitir flujo adicional igual a la capacidad de la arista, menos el flujo asignado en la misma arista [6]. La figura 2.8 muestra una arista del conjunto $E_{nf}(G_r)$ y una arista residual del conjunto $E_r(G_r)$, indica las capacidades y flujos que se asignan a cada una de ellas.

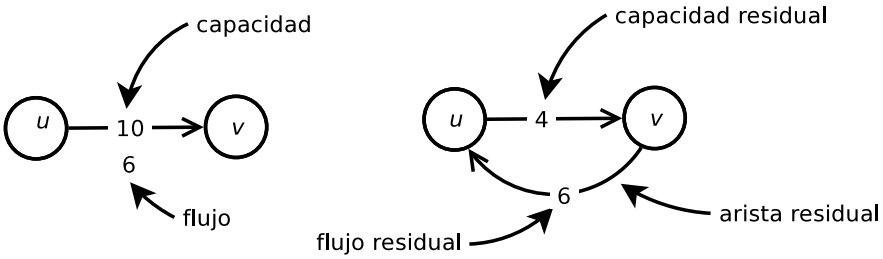

Figura 2.8: A la izquierda, se muestra una arista ponderada con capacidad, y flujo asignado. Del lado derecho, se muestra su arista residual asociada, la cual indica la capacidad residual, y flujo residual asignado.

2.4.3. Algoritmo del camino aumentado o Ford-Fulkerson

El algoritmo de Ford Fulkerson depende de tres conceptos importantes: gráficas residuales, caminos aumentados y cortes. Éstos anteriores hacen que el método sea aplicable para problemas y algoritmos de flujo. Estas ideas son esenciales para el Teorema del Flujo Máximo Corte Mínimo, el cual caracteriza el valor máximo de flujo en términos de un corte mínimo en una red [6].

La idea principal del algoritmo, consiste en encontrar un camino aumentado. Un camino aumentado es un camino simple desde un vértice inicial s, hasta un vértice final t en una gráfica residual, es decir, encontrar una sucesión de aristas con capacidades positivas que dirija flujo hasta un vértice destino t [6]. Cada arista $e \in E_r(G_r)$ que participa en al menos un camino aumentado admite algún flujo positivo adicional sin violar la propiedad de capacidad de una red de flujo [6].

El Algoritmo 1 Ford-Fulkerson(G, s, t) [3], requiere de entrada una red de flujo G_{nf}, un vértice s de donde se genera flujo, y un vértice t hacia donde es dirigido. Inicia con flujo 0 asignado a todas las aristas $e \in E_r(G_r)$. En cada iteración se incrementa el flujo en al menos una unidad sobre cada arista que pertenezca a un nuevo camino aumentado P encontrado. El algoritmo termina cuando no encuentra mas caminos aumentados por donde enviar flujo sobre la red [6].

Algorithm 1 Ford-Fulkerson(G, s, t)

```
for all e ∈ E do
    f(e) ← 0
end for
G_f ← residual graph
while exista un camino aumentado P ∈ G_f do
    f ← Camino_Aumentado(f, P)
    actualiza G_f
end while
return f
```

El Algoritmo 2 Camino_Aumentado(f, P) [3], se encarga de actualizar el flujo $f(e)$ asignado a cada arista $e \in P$ en el camino, determinado en base a la función $cuello\_de\_botella(P)$, la cual encuentra el menor valor de flujo que puede aplicarse a toda arista en el camino

P. La asignación de flujo $f(e)$ a cada arista depende si la arista es normal, o es una arista residual.

Algorithm 2 Camino_Aumentado(f, P)

```
b ← cuello_de_botella(P)
for all e ∈ P do
  if e ∈ E then
    f(e) ← f(e) + b
  else
    f(e_r) ← f(e_r) − b
  end if
end for
return f
```

Aunque en cada iteración del método de Ford Fulkerson se incrementa el valor del flujo en al menos una unidad, particularmente en cada arista $e \in G_{nf}$ el flujo asociado puede crecer o decrecer [6].

La Figura 2.9 ejemplifica una iteración del Algoritmo 3 Ford-Fulkerson(G, s, t), donde tanto en G_{nf}, como en G_r, se muestran las aristas que pertenecen al camino aumentado P, que dirige un flujo de 8 unidades desde el vértice s. En la gráfica residual G_r se observan las arista residuales con flujo residual asignado de 8 unidades, donde cada arista especifica la capacidad residual actual.

Iterativamente, el Algoritmo 3 Ford-Fulkerson(G, s, t) agota los caminos de s a t con aristas de capacidades positivas, obteniendo así el corte sobre la red de flujo G_{nf}, denotando las aristas con ausencia de capacidad disponible entre dos conjuntos de vértices A, B. Los vértices que determinan la partición A del corte son todos aquellos vértices que pueden ser alcanzados mediante una arista con capacidad positiva desde el vértices inicial s sobre la gráfica residual G_r.

La figura 2.10 muestra el resultado de la ejecución del Algoritmo 3 Ford-Fulkerson sobre la misma red de flujo G_{nf} que muestra la Figura 2.9. Puede observarse que el flujo total enviado a través de la red es de 19 unidades. En la parte superior se observa de color azul los flujos asignados a cada arista de la red, mientras que en la parte inferior se ve especificado de color azul las aristas residuales con un flujo residual asignado, las cuales se generaron a través de la ejecución. De color negro se observa a las aristas

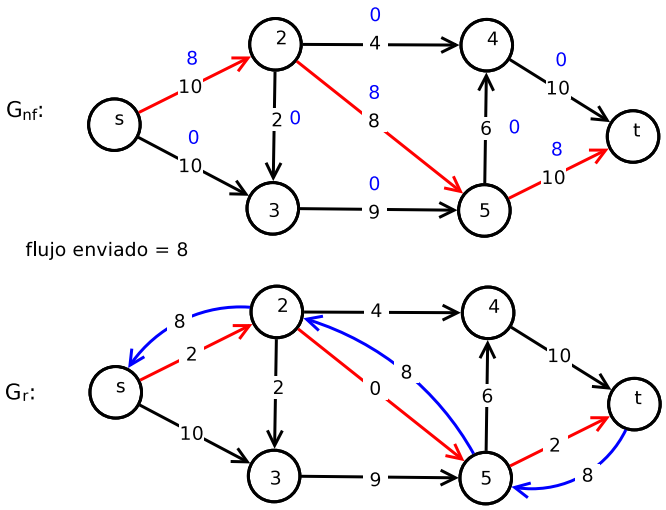

Figura 2.9: En la parte superior, se muestra un ejemplo de la ejecución de la primer iteración del Algoritmo 3 Ford-Fulkerson(G, s, t) sobre la red de flujo G_{nf}. El primer camino encontrado por el algoritmo consiste las de las aristas (s,2),(2,5) y (5,t) iluminadas de color rojo, las cuales son actualizadas con un flujo asignado de 8 unidades. En la parte inferior, se muestra la gráfica residual G_r, donde se indican las aristas residuales (2,s),(5,2) y (t,5) iluminadas de color azul, las cuales cuentan con una flujo residual de 8 unidades.

normales que aun cuentan con capacidad mayor a cero. Finalmente, puede observar de color gris a las arista que saturaron su capacidad a lo largo de la ejecución, y ahora son las que delimitan la partición de los vértices en dos subconjuntos A, B del conjunto general de vértices V_{nf}. Las aristas $(s, 2)$ y $(3, 5)$, delimitan la bipartición de los vértices de la gráfica, formando al subconjunto $A = \{s, 3\}$, y al subconjunto $B = \{2, 4, 5, t\}$.

Al final de la última iteración, cuando no se encuentran más caminos aumentados sobre la gráfica residual, se deriva el siguiente teorema.

- **Teorema del Camino Aumentado**: El flujo f que obtiene el algoritmo de Ford Fulkerson es máximo si y sólo si no existen más caminos aumentados en la red de flujo [6].

2.4.4. Teorema de Flujo Máximo Corte Mínimo

Sea f el flujo obtenido por el algoritmo de Ford Fulkerson. El teorema del camino aumentado asegura que el flujo devuelto por el algoritmo no puede ser incrementado, ya que no existen más caminos aumentados desde el vértice s hasta el vértice t. Lo que

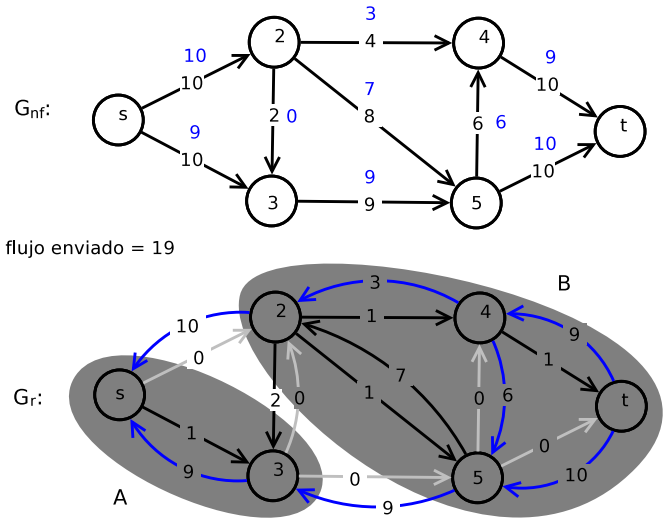

Figura 2.10: Resultado final de la ejecución del Algoritmo 3 Ford-Fulkerson sobre la red de flujo G_{nf}, las regiones sombreadas delimitan la agrupación de vértices en dos subconjuntos A,B. La aristas de color azul son aristas residuales generadas por el algoritmo, las aristas de color negro son aristas que aun cuentan con capacidad positiva y las aristas de color gris son aristas con capacidad saturada.

se busca es garantizar que dicho flujo f es también el máximo que puede pasar por la red [6].

Teorema : Si f es un flujo $s-t$ tal que no existe un camino aumentado en la gráfica residual G_r, entonces existe un corte $s-t$, (A,B) en G para el cual val(f)=cap(A,B). Consecuentemente, f tiene el máximo valor de cualquier flujo en G, y (A,B) tiene la capacidad mínima para cualquier corte $s-t$ en G [6].

2.4.5. Complejidad en tiempo

Asumiendo todas las capacidades como enteras y entre 1 y $C \in N$. Cada valor del flujo f(e) y cada capacidad residual $c^r(e)$ continúan siendo enteros a través de la ejecución del algoritmo [3].

Teorema : El algoritmo termina en a lo más $v(f^*) \leq nC$ iteraciones. Dado que cada flujo que agrega un camino aumentado es a lo menos 1 [3].

Si C=1, Ford-Fulkerson se ejecuta en tiempo O(mn) [3].

Si todas las capacidades son enteras, entonces existe un máximo flujo f para el cual cualquier valor de flujo $f(e)$ es un entero [3].

Introduciendo los conceptos y propiedades de gráficas, note que el algoritmo de FordFulkerson entrega una bipartición que cumple con el Teorema de Flujo Máximo Corte Mínimo. Puede verse que ésto se acerca a la idea de aglomerar tareas paralelas, en base a una división por la cantidad mínima de comunicaciones necesarias.

2.5. Una Representación de la Aglomeración y Mapeo con Gráficas

Esta sección muestra una forma de modelar tanto el problema de aglomeración como el de mapeo usando teoría de gráficas, los cuales están muy cercanamente relacionados con la arquitectura de la máquina paralela (hardware), así como con el propio sistema paralelo (software).

2.5.1. Modelo Tarea-Canal

Un modelo es empleado para representar de forma apropiada y útil el paralelismo, además debe contar con mecanismos que permitan expresar de forma explícita la concurrencia y la localidad, que faciliten el desarrollo de programas escalables y modulares. Para cumplir este propósito considere dos elementos: la tarea y el canal [1].

Un sistema paralelo es una colección de dos o más programas cooperativos (representadas por círculos), que juntos satisfacen una especificación dada [4] y son ejecutadas concurrentemente. Dichos programas están conectados por canales de comunicación (representadas por flechas). Cada tarea encapsula un programa secuencial almacenado en una memoria local y define un conjunto de puertos de entrada y/o salida, los cuales le permiten interactuar con su ambiente. Un canal es una cola de mensajes en la cual un emisor puede colocar paquetes y un receptor puede borrar mensajes. La figura 2.11 ejemplifica el modelo de Tarea-Canal [1].

El número de procesos en un sistema paralelo puede variar o no, dependiendo de la dinámica del sistema. Cada proceso puede ejecutar cuatro funciones básicas:

- Escritura/Lectura de la memoria local.
- Envío de mensajes entre procesos mediante el/los puertos de salida.

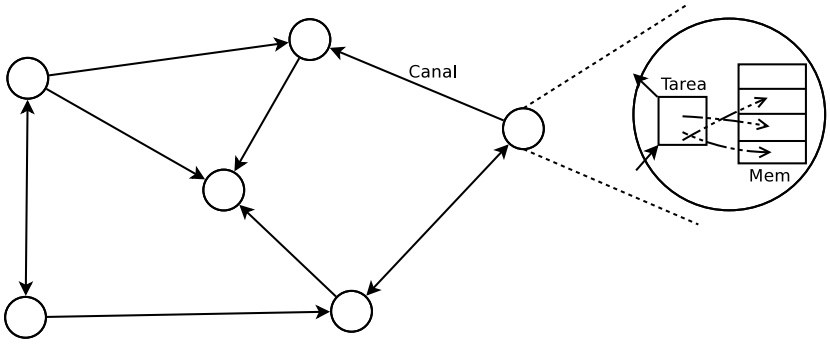

Figura 2.11: Modelo tarea canal, se representan mediante círculos las tareas, las cuales interactuan con memoria local, así mismo cada tarea tiene asignado uno o mas canales de comunicación.

- Recepción de mensajes de otros procesos mediante el/los puertos de entrada.
- Crear tareas nuevas y finalizar su ejecución.

Tanto la operación de envío y recepción de mensajes son asíncronas, es decir, un envío se completa inmediatamente, y una recepción causa que la ejecución de la tarea sea bloqueada hasta recibir el mensaje esperado. Los canales de comunicación asociados a los puertos pueden también cambiar en tiempo de ejecución [1].

El modelo de Tarea-Canal provee la idea de *localidad*, es decir, para cada proceso, éste tiene almacenados sus datos en una memoria local, la cual está "cerca"de él. Por otro lado, cualquier otro dato ajeno es *remoto* al proceso [1].

Note que el modelo de abstracción genera un diagrama muy similar a una gráfica (vease figura 2.11), donde los vértices representan tareas paralelas, y las aristas canales de comunicación entre ellas, tal que una arista es orientada en una dirección, si el proceso donde se genera una arista envía información al proceso donde apunta dicha arista.

Sea $G_{sw} = (V_{sw}, E_{sw})$ una gráfica estática y dirigida, que modela el sistema de software paralelo, sea V_{sw} el conjunto de procesos o tareas paralelas que existenten en el sistema paralelo, y E_{sw} los canales de comunicación que interconectan a dichos procesos.

Una vez hecha la representación del sistema paralelo mediante el modelo de Tarea-Canal, se obtiene una gráfica que indica precisamente la estructura del software del

sistema paralelo. La siguiente sección, muestra como es que el problema de aglomeración puede modelarse para aproximar posteriormente a una solución.

2.5.2. Modelado de la Aglomeración

La aglomeración formalmente puede reducirse a una partición de gráficas. Considere una gráfica $G = (V, E)$ con aristas ponderadas, tal que $|V| = n$, denota la cardinalidad del conjunto de vértices. Para el problema de (k, v) particiones balanceadas, para algún $k \geq 2$, el objetivo es particionar G en componentes a lo mas de tamaño $v(n/k)$, mientras que la capacidad de las aristas que conectan dos componentes separadas es mínima [14], es decir, dada una gráfica G y un $k \in \mathbb{N}^+$, particionar V en k-subconjuntos $V_1, V_2, ..., V_k$, tal que sean disjuntos, de igual tamaño, y el número de aristas con finales en diferentes subconjuntos sea mínima.

Típicamente, los problemas de particionamiento de gráficas caen en la categoría de problemas $NP - Hard$. Las soluciones de estos problemas son generalmente dados usando heurísticas o algoritmos de aproximación [14].

En particular, el problema de k-particiones balanceadas se muestra que es de tipo $NP - Completo$ en [14]. Incluso el problema de (2,1)-particiones balanceadas (que pareciera ser mas sencillo) es también de tipo $NP - Completo$ [13].

Considere un conjunto finito A y un tamaño $s(a) \in \mathbb{Z}$ para cada $a \in A$, ¿existe un subconjunto A' tal que [13]:

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a) \ ?$$

Es relativamente fácil darse cuenta que el problema de partición de gráficas está dentro de los problemas tipo NP, ya que un algoritmo no determinístico necesita solo adivinar un subconjunto A' de A, y verificar en tiempo polinomial que el tamaño de A' es igual al de $A - A'$ [13].

El conjunto de aristas más pequeño para desconectar una gráfica en dos subconjuntos disjuntos, puede ser eficientemente encontrado usando Redes de Flujo y el algoritmo de Ford-Fulkerson. El mínimo conjunto de corte puede separar a uno o mas vértices de la gráfica, por lo que las particiones resultantes pueden ser muy desbalaceadas. Aún cuando un mejor criterio de particionamiento mantiene un mínimo conjunto de corte con particiones aproximadamente del mismo tamaño, ya se ha mencionado que este

problema es de tipo $NP - Completo$ [9], y ya que para los fines de este trabajo el balance de carga en las particiones no es el objetivo principal, éste puede dejarse a un lado como trabajo a futuro.

2.5.3. Modelado del Problema de Mapeo

Una vez que se expresa la estructura de un sistema de software paralelo mediante una gráfica, y dado que la aproximación al problema de aglomeración se aborda con redes de flujo, es necesario establecer una gráfica que represente la arquitectura del sistema de hardware paralela donde la ejecución del sistema de software tome lugar.

La arquitectura de hardware paralela o de una computadora paralela es un conjunto de unidades de procesamiento que pueden comunicarse por un medio, y que son capaces de trabajar de forma cooperativa, ya sea que todos los núcleos de procesamiento compartan memoria, cuenten con una memoria propia e independiente a las demás (memoria distribuida), o una combinación de las dos anteriores. El tipo de medio de comunicación entre procesos puede ser un bus de datos o un medio de red [1,5].

La arquitectura de hardware paralelo se refiere a cómo organizar los componentes de forma que tengan un buen funcionamiento [5]. Una computadora paralela ofrece el potencial de concentrar los recursos computacionales, como memoria y procesadores, sobre un problema computacionalmente importante [1].

Sea $G_{hw} = (V_{hw}, E_{hw})$ una gráfica estática y dirigida, donde V_{hw} es el conjunto de núcleos de procesamiento o procesadores existentes en la arquitectura de hardware paralela, y E_{hw} representa los enlaces de comunicación que interconectan a los procesadores de la arquitectura paralela. Por su parte, el mapeo se encarga de tomar todo proceso paralelo $v \in V_{sw}$ y asignarlo a un núcleo de procesamiento $w \in V_{hw}$ para llevar a cabo su ejecución. Note que todo elemento dentro de una misma aglomeración, es asignado a un mismo núcleo de procesamiento.

2.6. Resumen

A lo largo de este capítulo 2 se plantearon tres metodologías de diseño de software paralelo, las cuales detallan varias etapas para transformar un problema secuencial a un

problema paralelo. Se describe una metodología genérica que engloba las tres metodologías enunciadas describiendo a detalle las etapas de: partcionamiento, comunicación, agolmeareación y ajuste de granularidad, y mapeo. Estas dos últimas etapas de aglomeración y mapeo, se detallan a fondo ya que es aquí es donde la estrategia propuesta toma relevancia.

La sección de gráficas presenta y describe una herramienta matemática que ayuda a proveer un modelo abstracto y presenta un par de teoremas con relevancia para la aproximación de la estrategia expuesta en el presente trabajo. Finalmente la sección de una representanción de la aglomeración y mapeo toma las gráficas y las emplea para formalizar un modelo de representación para los sistemas paralelos, y cómo éstos se relacionan con la aglomeración y asignación a las unidades de procesamiento de la arquitectura paralela.

3

Trabajos Relacionados

En el presente capítulo se abordan dos trabajos relacionados con la estrategia propuesta en esta tesis. El primero es un artículo llamado "Sobre el problema de Mapeo"(On the Mapping Problem) publicado por Sahid Bokhari en 1981, el cual presenta una estrategia de mapeo basada en un algoritmo heurístico sobre una arquitectura de hardware fuertemente conectada. El segundo trabajo se llama "Una Estrategia Eficiente de Mapeo para Programación Paralela"(An Efficient Mapping Strategy for Parallel Programing) publicado por Jorge Luis Ortega Arjona y Héctor Benítez Pérez en 2010, el cual propone una estrategia de mapeo de control no centralizado.

3.1. Sobre el Problema de Mapeo

En una arquitectura de hardware paralela, es importante la asignación de los procesos paralelos sobre los procesadores, tal que los procesos que se comunican entre ellos sean asignados lo mas cerca posible, en procesadores adyacentes [2]. En [2] el problema de mapeo se formula teóricamente en términos de una gráfica, y se muestra su equivalencia de forma más general, con el problema de la gráfica isomorfa.

Debido a la complejidad del problema de mapeo, un algoritmo eficiente para resolver este problema en general no puede obtenerse, así que las investigaciones en esta área se han concentrado en buscar algoritmos heurísticos que arrojen resultados eficientes para algunos casos. En [2] se describe un algoritmo heurístico que emplea secuencias de intercambios de parejas, alternando con saltos probabilísticos. Dicho algoritmo se emplea para resolver problemas prácticos de mapeo sobre una arquitectura de hard-

ware paralela específica, obteniendo resultados aceptables, los cuales se muestran en la sección de rendimiento del algoritmo en [2].

3.1.1. Descripción

Muchas arquitecturas paralelas no son completamente conexas, es decir, no existe una conexión directa para cada par de procesadores en ella. Dos razones por las que comúnmente ésto sucede son: el número de enlaces directos se incrementa cuadráticamente por cada procesador que se aumenta, y el número de entradas y salidas se incrementa linealmente por cada procesador que aumenta [2].

El problema del mapeo es importante cuando se cuenta con un arreglo de procesadores y módulos que se asignan a éstos. El mapeo debe considerar la comunicación que existe entre distintos módulos, y asígnarlos a procesadores lo mas cerca posible, preferentemente en procesadores directamente conectados [2].

Bokhari presenta en [2] una formulación matemática, donde representa inicialmente el problema como una gráfica denotada como $G_p = \{V_p, E_p\}$, donde los nodos representan las tareas o procesos y las aristas la comunicación entre ellos. De la misma forma, representa el arreglo de procesadores de la arquitectura paralela como una gráfica denotado como $G_a = \{V_a, E_a\}$.

La función de mapeo coloca módulos sobre procesadores y es representada como [2]:

$$fm : V_p \longrightarrow_{1-1}^{onto} V_a$$

En [2] se define la calidad del mapeo en base al número de coincidencias de aristas, que se obtiene al sobreponer la gráfica G_p sobre la gráfica G_a, es decir, el número de aristas que caen directamente sobre una arista del gráfica de la arquitectura. A este número se le conoce como la *cardinalidad* del mapeo, y se denota como $|fm|$.

Para buscar el mejor mapeo posible, se debe buscar una función fm tal que tenga una máxima cardinalidad entre las $(|V_p|)!$ funciones posibles [2].

Una vez formulado el problema de mapeo en términos de una gráfica teórica, en [2] se muestra que el problema de mapeo, en su forma más general, es computacionalmente equivalente al problema de la gráfica isomorfa.

Dos gráficas G_1 y G_2 se dicen isomorfos uno al otro si existe una correspondencia uno a uno entre sus vértices y sus aristas, tal que la incidencia de relación se preserva.

Este es un problema clásico de combinatoria aun no resuelto, ya que no se conoce un algoritmo que resuelva este problema para gráficas arbitrarias, aunque existen algunas aproximaciones [2].

Entonces en [2] se propone que, si se encuentra un algoritmo polinomial para resolver el problema general de mapeo, se puede de resolver también el problema de la gráfica isomorfa, ya que se conseguiría mapear la gráfica G_1 sobre la gráfica G_2, y así, para dos gráficas arbitrarias, poder responder la pregunta en tiempo polinomial, ¿G_1 y G_2 son isomorfos?.

La arquitectura de hardware paralela en la que [2] se centra, es una máquina finita de elementos (FEM[1]), la cual consiste de en un arreglo de procesadores interconectados en un patrón de ocho vecinos cercanos. Además de los enlaces de comunicación con cada vecino, que están dedicados a comunicación entre un par específico de procesadores, existe un bus de datos general de tiempo compartido para comunicar cada par de nodos que no sean adyacentes.

La estrategia de mapeo en [2] procura que durante la ejecución del sistema paralelo, exista comunicación entre pares de procesos solo si éstos fueron asignados ya sea en un mismo nodo, o en nodos adyacentes. Por lo tanto, las arista de la gráfica de procesos debe caer sobre una arista de la FEM. De tal manera que Bokhari en [2] afirma que:

> *"La eficiencia de las comunicaciones responde de buena manera debido a los enlaces dedicados de la FEM, por el contrario, cuando las comunicaciones emplean el bus de datos compartido la eficiencia del sistema se ve afectada de forma negativa".*

En [2] Bokhari presenta un algoritmo heurístico de intercambio de parejas, el cual toma como entrada una matriz de adyacencia la cual representa el problema además del tamaño de la matriz de la FEM, y provee como salida una matriz permutada, la cual se asemeja mucho a la matriz de adyacencia de la FEM. El algoritmo implementa una función de búsqueda, la cual intenta mejorar el mapeo considerando todos los posibles intercambios de parejas de los nodos numerados. El intercambio de parejas aumenta la cardinalidad del mapeo, este proceso se repite hasta que no haya forma de aumentar la ganancia. En este momento se deja la búsqueda y si el mapeo que se encuentra por la búsqueda anterior es el mejor encontrado hasta ahora. Entonces, se aplica un salto

[1] Por sus siglas en Inglés *Finite Element Machine*.

probabilístico y el algoritmo regresa a la función de búsqueda. El algoritmo no garantiza encontrar un mapeo óptimo y a veces encuentra mapeos terribles.

Respecto al desempeño que muestra el algoritmo propuesto en [2], se puede observar que en muchos casos el algoritmo es capaz de mejorar el mapeo dinámico. Es difícil determinar que tanto los mapeos obtenidos por el algoritmos se acerca a la solución óptima, ya que para un problema en especifico no hay forma de conocerla. Por otra parte, se puede saber que tanto se acerca la cardinalidad del mapeo final al número de aristas, entonces se puede estar seguro de que está cerca del óptimo.

El algoritmo que se presenta en [2], resulta ser bastante aceptable para el prototipo de una FEM de 6 x 6. Sin embargo, debido a la tasa cuadrática de crecimiento de la matriz de adyacencia de la FEM, el algoritmo tal vez no sea adecuado para arreglos de gran tamaño, para dichos arreglos es posible que sea necesario otro tipo de heurísticas.

3.2. Una Estrategia de Mapeo Eficiente para Programación Paralela

Ortega y Benítez en [15] establecen que para obtener una ejecución efectiva de un sistema paralelo, requiere que el mapeo de los procesos (del software paralelo) en los procesadores (del hardware paralelo) se realice eficientemente. Así entonces, [15] presenta una estrategia eficiente de mapeo basada en optimización de comunicaciones entre procesos, así como la distribución balanceada de tareas sobre procesadores en una red. La estrategia de mapeo propuesta en [15] se desarrolla como un programa paralelo, basada en ejecución de procesos locales e independientes, de tal forma que la estrategia propuesta no requiere de un control centralizado para llevar a cabo el mapeo.

3.2.1. Descripción

En [15] Ortega y Benítez, definen el problema de mapeo en términos topológicos, refiriéndose a encontrar una gráfica que represente el sistema de procesos comunicantes y asignarlo sobre una gráfica que represente los procesadores en red, tal que los procesadores vecinos alojen procesos vecinos. Ya que la complejidad del problema de mapeo no permite dar un solución óptima, en [15] se propone una estrategia de mapeo que obtiene resultados aceptables.

Al igual que en [2], en [15] se propone la representación de un programa paralelo en forma de una gráfica, en donde cada proceso es un vértice y cada canal de comunicación entre dos procesos es una arista. De la misma forma, la arquitectura de hardware paralelo puede representarse como una gráfica, donde ahora cada procesador es un vértice y cada interconexión entre ellos es una arista. Entonces, el problema de mapeo reduce la gráfica de software sobre la gráfica de hardware, resultando en una distribución de procesos a procesadores tal que la ejecución optimice [15]:

- Balance de carga
  Los procesos son asignados a los procesadores tal que la carga de procesamiento debe estar bastante distribuida sobre todos los procesadores de la arquitectura. En tal situación el sistema paralelo se considera estar balanceado.

- Optimización de las comunicaciones
  La comunicación entre dos procesos debe estar distribuida tan uniformemente como sea posible sobre todas las conexiones entre procesadores. Cuando esto sucede se dice que las comunicaciones son óptimas.

De acuerdo al último punto, la optimización de comunicaciones significa que procesos vecinos deben ser mapeados sobre procesadores lo más cercanos posible (de preferencia vecinos). De otra forma para cada comunicación entre dos procesos, mas de una conexión tiene que ser usada. Por lo tanto Ortega y Benítez en [15] afirman que:

> *"Dichas comunicaciones representan una carga para los procesos intermedios, los cuales tienen que encaminar los datos hasta llegar al procesos destino. Esto produce una sobre carga de comunicaciones, debido a la retransmisión de los datos, provocando un incremento del tiempo global de ejecución del sistema paralelo".*

Sin embargo, comúnmente no es posible realizar tal mapeo en procesadores vecinos. Por lo tanto, el objetivo de la estrategia que se presenta en [15], persigue que:

> *"Los procesos deber se asignados de tal menera que, el costo comunicaciones globales se mantengan en lo mínimo posible".*

Ortega y Benítez proveen en [15], una descripción formal para establecer la plataforma de hardware paralela, el sistema de software, las restricciones que se presentan,

la carga de trabajo y el mapeo óptimo. A continuación se mencionan cada una de ellas [15]:

1. Plataforma de hardware paralelo
   La plataforma se define con memoria distribuida de la forma:

   $$HW = (< P, A_{hw} >, C_{hw})$$

   donde:

   $< P, A_{hw} >$ es un grafo no dirigido que representa la red, donde $P = P_1, P_2, P_3...P_n$ es el conjunto de N de procesadores de la plataforma de hardware. A_{hw} es la matriz de adyacencia que representa al grafo. C_{hw} es la matriz de costos de comunicación asociados a la plataforma de hardware.

2. Sistema de Software paralelo
   Se define como una red estática de procesos comunicantes de la forma:

   $$SW = (< Q, A_{sw} >, C_{sw}, t)$$

   donde:

   $< Q, A_{sw} >$ es un grafo dirigido donde $Q = Q_1, Q_2, Q_3...Q_n$ es el conjunto de procesos paralelos en el sistema. A_{sw} es la matriz de adyacencia que representa el grafo. C_{sw} es la matriz de costos de comunicación asociada al sistema de software. $t : Q \rightarrow Z$ es una función que el tiempo de procesamiento de un proceso Q_i, Z son los pasos atómicos computacionales.

3. Restricciones
   El proceso de mapeo implica que algunos procesos deben colocarse en procesadores en particular, debido a que cada procesador ofrece capacidades únicas necesarias para el sistema de software paralelo. Este hecho debe considerarse para la descripción formal como se muestra a continuación.

   $$R \subseteq Q \times P$$

Lo cual significa que el proceso Q_i debe ser forzosamente mapeado sobre el procesador P_m.

4. Carga

   La carga de procesos se refiere a la cantidad de tiempo que un procesador emplea en ejecutar un proceso o un grupo de procesos. En términos de tiempo por proceso sobre un procesador, la carga l puede ser definida como:

$$l(P_m) = \sum_{\pi(Q_i)=P_m} t(Q_i)$$

   La función $t(Q_i)$ regresa el tiempo producido por el proceso, lo cual tiende a ser difícil de estimar, ya que no hay forma de determinar cual es el tiempo que un proceso requiera de un procesador. En general, lo que se hace es medir el tiempo durante la ejecución real. Otra forma de calcular este tiempo es determinando la complejidad de un proceso. Aun así, no es posible estimar el tiempo de forma real, ya que la complejidad no se puede obtener automáticamente, o los procesos puedan tener variaciones dinámicas en tiempo de ejecución.

5. Mapeo Óptimo

   Un mapeo óptimo puede definirse en términos de la función:

$$\pi : Q \to P$$

   donde

   $\pi(Q_i) = P_m$, función la cual toma en cuenta que

   $l(P_m) \approx \frac{\sum_{n=1}^{N} l(P_n)}{N}$

   lo cual significa que los procesadores en su mayoría están balanceados.

   $min(c_{sw}[i,j] \cdot c_{hw}[t(P_m), t(P_n)])$

   Lo cual significa que los costos de comunicación son minimizados.

La estrategia de mapeo propuesta por Ortega y Benitez en [15], se caracteriza por, ser distribuida, hacer uso del vecino óptimo mas cercano para obtener una optimización global, se basa en el intercambio de información de los costos del balance de carga y

comunicación entre procesadores vecinos, permite migración de procesos de acuerdo a la demanda de optimización, para acelerar el proceso de mapeo.

Los pasos generales de la estrategia de mapeo que se presenta en [15], cuando se ejecuta en paralelo en cada procesador, son los siguientes:

1. Intercambio de información con procesos vecinos y permitir la migración de procesos a procesadores vecinos.

2. La optimización de la comunicación obtiene procesos con altos costos de comunicación y considera, si es necesario, alojarlos en procesadores vecinos.

3. Balance de carga local.

Inicialmente, todos los procesos son mapeados en cualquier procesador. Para cada iteración, los procesos se migran de un procesador a otro, con el fin de cumplir con la demanda de optimización. Los costos de comunicación local se reducen debido a que los procesos que causan altos costos se migran a otros procesadores.

La estrategia que se presenta en [15] toma como criterio más importante el balance de carga sobre la minimización de comunicaciones. Por lo tanto, dicha estrategia consigue un mapeo subóptimo, el cual cumple con la característica del balance de carga. Si por el contrario, se toma la decisión de reducir los costos de comunicación, entonces la estrategia selecciona un proceso que se migra hacia algún procesador, sólo si tal proceso genera una alto costo de comunicación.

La estrategia de mapeo distribuida, lleva a cabo un mapeo $\pi : Q \rightarrow P$ considerando lo siguiente:

- El costo de comunicación $C_{hw}[m, n]$ entre cuales quiera dos procesos Pm y Pn es:

$$C_{hw}[m,n] = \begin{cases} 1 & \text{si } A_{hw}[m,n] = 1 \\ 2 \times d_{mn} & \text{de otra forma} \end{cases}$$

  donde d_{mn} denota la distancia mas corta entre P_m y P_n.

- El costo de comunicación para procesos conectados directamente es 1.

En [15] se define el costo de comunicación entre dos procesos cualesquiera como el conjunto $C_{sw}[i,j] = 1$, aunque durante la ejecución de la estrategia de mapeo existen tres tipos diferentes de comunicaciones que deben considerarse:

- La comunicaciones internas, que se emplean entre dos procesos alojados en un mismo procesador, tal que el costo se denotara como $C_{sw}[i,j] = 1$.
- Comunicaciones cortas, las cuales se llevan a cabo entre dos procesos mapeados sobre procesadores directamente conectados (vecinos). Se denota un costo de $C_{sw}[i,j] = 1$.
- Comunicaciones lejanas, las cuales comunican procesos que se encuentran mapeados distantes (no vecinos), donde el valor del costo se denota como $C_{sw}[i,j] \cdot C_{hw}[m,n]$.

Considerando los tres tipos de comunicación que define [15], la decisión de seleccionar un proceso candidato para ser migrado se toma por una función de atracción, la cual se define en base a las siguientes suposiciones:

1. Cada procesador tiene la información acerca de las distancias más cortas para cada procesador en la red.
2. Cada procesador P_m tiene que informar acerca de todos los canales de comunicación $I_{hw}[m,n]$ a cada procesador vecino P_n, através del cual la distancia más corta hacia los otros procesadores puede obtenerse.
3. Cada proceso tiene la ubicación *probable* de todos los demás procesos.

En éste el último punto, se dice que es *probable* ya que cuando un proceso se migra, solo los vecinos conocen la ubicación actual. Por lo tanto, la información puede no estar actualizada.

En cada iteración de la estrategia, se selecciona un proceso mapeado con el máximo valor de atracción. Dicha consideración cumple por completo con los requerimientos del balance de carga. Cualquier proceso con el máximo valor de atracción es enviado a un procesador vecino, decrementando su valor de probabilidad.

> *"En este caso se descuidan los requerimientos del balance de carga para asegurar una minimización en los costos de comunicación"*

[15].

3.3. Resumen

A lo largo del capítulo 3 se describen dos trabajos relacionados a la estrategia de mapeo propuesta. Similar a los trabajos relacionados, es necesario obtener a partir de un problema paralelo una gráfica que representa la estructura del sistema paralelo y la estructura de comunicación de datos. Bokhari en [2] toma la gráfica de software y realiza una equivalencia con el problema de la gráfica isomorfa, de tal forma afronta el problema de mapeo específico para una máquina de elementos finitos. Por otra parte Ortega y Benítez en [15] proponen una estrategia eficiente de mapeo distribuida, en la cual se basa en intercambio de información entre procesos vecinos, para garantizar uno de dos objetivos, ya sea el balance de carga o la minimización de comunicaciones en el mapeo de un sistema paralelo.

4

La estrategia de aglomeración

El capítulo presenta la estrategia que se propone en este trabajo de tesis de forma detallada, tomando como referencia la metodología de diseño genérica expuesta en la sección 2.1.4, y mostrando los requerimientos y pasos necesarios para llevar a cabo la aglomeración de procesos paralelos, con el fin de encontrar un mapeo eficiente sobre una arquitectura distribuida. Además, presenta algunas consideraciones adicionales sobre el modelo y un caso especial de redes de flujo.

4.1. Descripción de la estrategia

Dado un sistema paralelo, éste puede modelarse de manera abstracta. Dicha representación se conoce como la gráfica de software del sistema paralelo.

La gráfica de software indica precisamente los procesos que componen al sistema, así como los enlaces de comunicación de datos que transmite cada par de éstos. La transmisión de los datos es necesaria para la ejecución del sistema. De esta forma, poder caracterizar la cantidad de flujo de datos que son transferidos sobre las aristas de comunicación de la gráfica de software.

Con el fin de realizar la aglomeración de los procesos, es necesario formular la gráfica de software en una red de flujo la cual debe cumplir con las propiedades descritas en la sección 2.4.1.

Para poder formular la gráfica de software en una red de flujo es necesario:

- Agregar dos vértices especiales, un vértice fuente s y un vértice fin t, tal que todo flujo saliente de s debe ser igual al flujo entrante a t.

- Asignar a cada arista de la gráfica de software una capacidad que delimita la cantidad de datos que puede pasar sobre ella.
- Determinar la cantidad de flujo de datos que transmite cada proceso sobre cada aristas de la gráfica de software. Dicha cantidad de flujo puede ser menor o igual a la capacidad de la arista correspondiente.

Formulada la gráfica de software en una red de flujo, se ejecuta sobre ésta un algoritmo basado en el algoritmo de Ford-Fulkerson, para obtener una partición del conjunto de vértices. La partición resultante cumple que: "La cantidad máxima de flujo que va de la partición A a la partición B más la cantidad máxima de flujo en sentido contrario (de la partición B a la particón A) es mínima para cuales quiera dos particiones". Se itera sobre las subgráficas resultantes para obtener las particiones (aglomeraciones) deseadas. Cabe mencionar que el certificado de optimalidad es garantizado para la primer iteración, aunque para particiones posteriores esta garantía se pierde.

La figura 4.1 muestra los pasos de la estrategia de mapeo, ilustrando un ejemplo simple de cuáles son las entradas de cada etapa y el resultado de cada una de ellas.

4.2. Consideraciones

Existen algunos detalles que pasan desapercibidos en el capítulo 2, los cuales son importantes considerar, ya que la estrategia de aglomeración debe cubrir la mayoría de los posibles escenarios que puedan presentarse. Por lo tanto, la presente sección se dedica a tomar en cuenta dichos detalles para plantear una solución ante estos problemas.

4.2.1. Adaptación del Modelo de Red de Flujo para Procesos Paralelos

Representar la gráfica de software que caracteriza la estructura y comunicación del sistema paralelo en una red de flujo, ayuda a obtener un particionamiento (A, B) del conjunto de vértices V_{nf}, e iterativamente así aproximar las particiones a una aglomeración de procesos para proseguir con un mapeo directo sobre las unidades de procesamiento del sistema distribuido.

De manera general, para cualquier red de flujo arbitraria (como se muestra en la figura 4.2) se puede encontrar una partición (A, B) tal que la o las aristas que unen

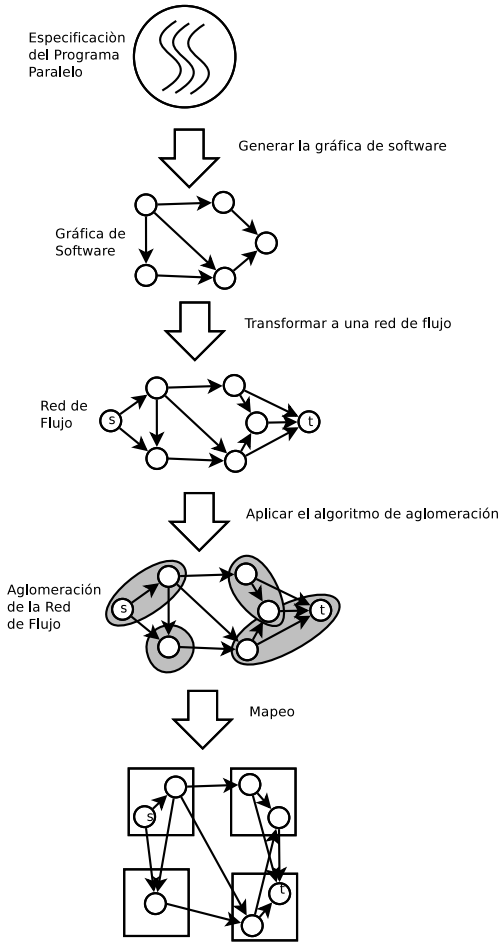

Figura 4.1: Secuencia de pasos de la estrategia de aglomeración basada en propiedades de corte y flujo en gráficas.

a estas particiones sumen una capacidad mínima, es decir, encontrar un corte mínimo el cual indica la máxima cantidad de flujo que puede pasar de una partición A a una partición B (por el *Teorema de Corte Mínimo Flujo Máximo*).

Considere el sistema paralelo que se presenta en la figura 4.3. Se puede observar que el corte mínimo está conformado por las aristas $(s, 2)$ y $(3, 5)$, con lo cual se obtiene un corte mínimo de valor 19. Esto se percibe al observar el flujo asignado a cada arista en la red, y notar que las aristas que pertenecen al corte mínimo se encuentran con capacidades saturadas.

Note que la arista $(2, 3)$ es incidente a la partición A. Según la definición de corte dice que:

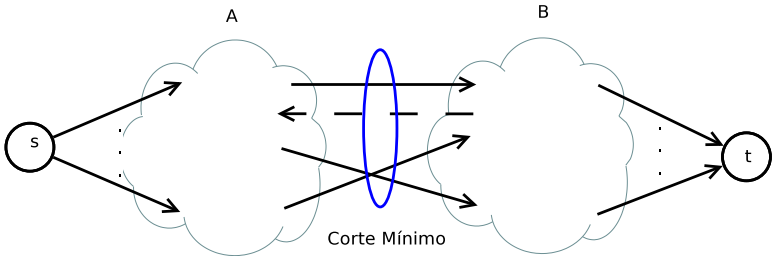

Figura 4.2: Generalización de una red de flujo, particionamiento por corte mínimo.

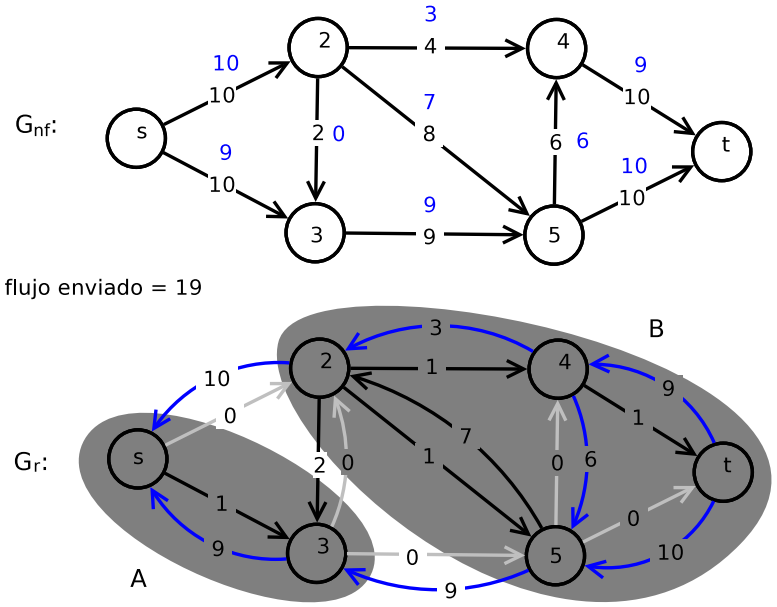

Figura 4.3: Red de Flujo, las areas en gris indican el particionamiento de vértices obtenido por el algoritmo de Ford Fulkerson.

> *"La capacidad de un corte es igual a la suma de la capacidad de sus aristas excedentes de la partición".*

Con lo cual se puede ver que para toda arista incidente a la partición A, ésta es despreciada para la capacidad del corte. Ya que el objetivo del particionamiento es minimizar la cantidad de flujo que se transmite de la partición A a la partición B y viceversa, entonces se puede decir que:

> *"Un corte mínimo, para el problema de aglomeración de procesos paralelos, debe considerar a toda arista tanto incidente como excedente de toda partición, ya que no solo importa si el flujo va de la partición A a la partición B, si no también en sentido contrario".*

Si se hace esta observación en el corte mínimo mostrado en la figura 4.3 se tiene que, el valor del corte no es 19, si no 19 + 2 = 21 (por el flujo asignado de la arista $(2, 3)$ incidente a la partición A).

Buscando intuitivamente un nuevo corte mínimo, ahora considerando la restricción anterior, se encuentra que puede conformarse un corte mínimo con las aristas $(5, t)$, $(5, 4)$ y $(2, 4)$, con las que se obtiene un corte de valor 20 (como se observa en la figura 4.4) el cual es menor al corte obtenido por el *algoritmo de Ford-Fulkerson* mostrado en la figura 4.3.

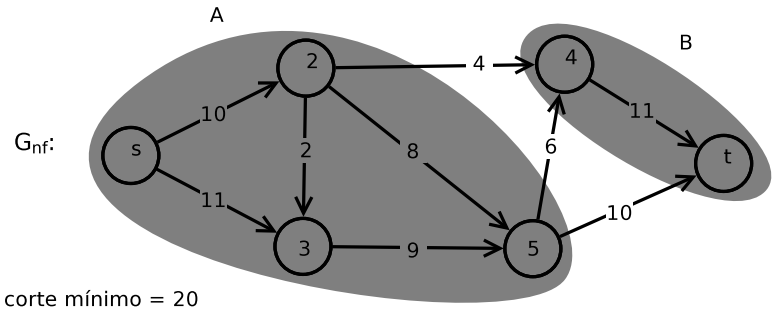

Figura 4.4: Corte mínimo considerando aristas incidentes y exedentes.

Por lo que un algoritmo que tome esta consideración debe ser capaz de encontrar un corte mínimo, donde un corte mínimo (de cuerdo a la adaptación para el modelo de procesos paralelos) es:

$$mincut(G_{nf}) = \sum cap(e)$$

tal que $e \in E_{nf}$ es una arista, cuyos vértices finales se encuentran en particiones disjuntas, independientemente del sentido en que conecten a las particiones.

Un algoritmo propuesto que encuentra este corte mínimo se muestra a continuación:

Considere el algoritmo de Ford-Fulkerson presentado en la sección 2.4.3. Puede notarse que éste algoritmo se encarga de encontrar caminos desde un vértice s hasta un vértice t, tal que en cada iteración actualiza el flujo asignado a las aristas que participan en cada camino hasta agotar la capacidad de ellas. Viendo el funcionamiento del algoritmo por etapas, puede decirse que la primer etapa es donde se encuentran caminos simples que solo consideran aristas $e \in E_{nf}$, es decir, solo caminos con aristas

normales. Una vez que los caminos usando éstas son agotados, la segunda etapa encuentra caminos simples considerando ahora aristas $e \in \{E_{nf} \cup E_r\}$, es decir, tanto aristas *normales* como aristas *residuales*. Hasta este punto el algoritmo de Ford-Fulkerson terminaría sin la consideración plantada anteriomente, por lo que la tercer etapa (propuesta) es, si existe al menos una arista en el corte mínimo que va de un vértice de la particion B a un vértice de la particion A, es decir, existe al menos una arista de la forma $e = (y, x)$ *tal que* $y \in B$ y $x \in A$, entonces para toda arista de esta forma invierte su orientación a $e = (x, y)$, continua con el algoritmo de FordFulkerson hasta encontrar un nuevo corte mínimo, tal que el corte no contenga arista incidentes en A.

Algorithm 3 Ford-Fulkerson-Extendido(G, s, t)

```
mincut(G) ← Ford-Fulkerson(G, s, t)
while Exista una e ∈ mincut(G) incidente en A do
  for all e incidente en A do
    G' ← G − e
    G' ← G' ∪ e_inversa
  end for
  continua con Ford-Fulkerson(G', s, t)
end while
```

La pregunta que surge observando este algoritmo anterior es:

- ¿qué pasa si el conjunto de aristas e que son incidentes en la partición A tiene flujo asignado $f(e) > 0$?.

  Para responder esta pregunta, primero se muestra que este caso nunca sucede, es decir, para toda arista que el algoritmo orienta en sentido contrario ésta tiene un flujo asignado $f(e) = 0$. Por un momento supongase que si esto no sucede, entonces el flujo sobre la red sería alterado, ya que al invertir la orientación de una arista, el flujo asignado a ésta seria revertido al vértice de donde el flujo proviene, y con esto se violaría la propiedad de conservación de flujo sobre la red, particularmente sobre dicho vértice.

  Sea $G = \{V, E\}$ una red de flujo, y sea $mincut(G) = \{e_1, e_2, ..., e_k\}$ las aristas que conforman el corte mínimo de la gráfica G. Note que $mincut(G)$ también entrega

una bipartición G:[A,B]. Entonces:

$\forall\ e \in mincut(G)$ tal que $e = (y, x)$ donde $y \in B$ y $x \in A$ entonces $f(e) = 0$.

Demostración:
Por contradicción supóngase que $f(e) > 0$ *tal que* $e \in mincut(G)$*, es decir, existe una arista incidente a la partición A, tal que e tiene flujo asignado mayor a cero.*

Dado que $mincut(G)$ *es el corte mínimo de G, entonces*
$\forall\ e' \in mincut(G)$,
$f(e') = cap(e')$,
donde $e' = (a, b) \mid a \in A\ y\ b \in B$, *y*
no existe un camino P_{s-t} *con capacidades positivas.*

Ya que $cap(e) > 0$*, por lo tanto,* $\exists\ e_r$ *una arista residual asociada a e,*
donde $f(e_r) > 0$*, es decir,* e_r *tiene capacidad de transmitir flujo de* $A \rightarrow B$*, con lo que existen dos casos:*

1. $\exists\ P'_{s-t}$*, al menos un camino que usa a* e_r *para transmitir flujo de s a t. Por lo tanto, contradice que* $mincut(G)$ *sea el corte mínimo.*
2. *No existe un camino* P'_{s-t} *que use a* e_r *para transmitir flujo de s a t, lo cual indica que existe un corte mínimo de menor tamaño que* $mincut(G)$ *que se ha saturado antes, lo que también contradice la suposición inicial.*

Por lo tanto se demuestra que, $\forall\ e \in mincut(G)$ *tal que* $e = (y, x)$ *donde* $y \in B$ *y* $x \in A$ *entonces* $f(e) = 0$.

Una pregunta obligada que se genera al presentar el Algoritmo 3 es, ¿cómo demostrar que el algoritmo extendido de Ford-Fulkerson continúa siendo correcto?, es decir, demostrar que la garantía de obtener un corte mínimo sigue vigente.

- Puede notarse que el algoritmo extendido de Ford-Fulkerson primero obtiene un corte mínimo inicial sobre una gráfica G. Considere los dos casos posibles de un corte mínimo arbitrario $mincut(G) = \{e_1, e_2, e_3, ..., e_k\}$:

1. $\forall\ e_i \in mincut(G)$ es de la forma $e_i = (x, y) \mid x \in A$ y $y \in B$, es decir, para toda arista del corte mínimo, ésta va de la partición A a la B y no viceversa.

   Sobre este caso, el algoritmo extendido de Ford-Fulkerson no varía en lo absoluto con la aproximación original para obtener el corte mínimo de flujo máximo. Por lo tanto, para mostrar la correctez del algoritmo extendido, solo tomaría remitir a la prueba del algoritmo original.

2. $\exists\ e_i = (y, x) \in mincut(G) \mid y \in B$ y $x \in A$, es decir, existe al menos una arista que incide en la partición A.

   Este caso, a diferencia del anterior, es donde la extensión del algoritmo de Ford-Fulkerson toma sentido. En inicio se puede mostrar (por la prueba anterior) que el flujo sobre G', una vez reorientadas las aristas incidentes a la partición A, no es alterado debido a que las aristas reorientadas tienen flujo asignado $f(e) = 0$.

   Dado que el conjunto de aristas reorientadas pertenecen al conjunto de aristas de $mincut(G')$, lo que hace el algoritmo es habilitar una nueva capacidad en el corte ya mínimo. Si existen aún más caminos por los cuales seguir enviando flujo de s a t, entonces debe haber otro corte más pequeño $mincut'(G')$, el cual se satura antes que $mincut(G')$. De lo contrario, $mincut(G')$ sigue siendo el corte mínimo sobre G'.

Considerando los dos casos anteriores, el algoritmo extendido de Ford-Fulkerson obtiene el corte mínimo considerando tanto aristas incidentes como excedentes a cada partición $G : [A, B]$.

4.2.2. Caso especial de una red de flujo

De acuerdo al teorema de Flujo Máximo Corte Mínimo (Sección 2.4.4), para toda gráfica arbitraria, existe un conjunto de aristas las cuales forman corte de capacidad mínima o un *cuello de botella* para todo flujo que es enviado desde el vértice inicial s hasta un vértice final t.

Considere el siguiente caso: Sea $G = \{V, E\}$ una gráfica dirigida, donde toda arista $e \in E$ tiene una capacidad $cap(e) \in \mathbb{Z}^+$, tal que para todo vértice $v \in \{V - s, t\}$, la suma de las capacidades incidentes es igual a la suma de las capacidades excedentes (similar a la propiedad de conservación de flujo, sólo que en las capacidades). Es decir:

$$\sum_{e\ entra\ a\ v} cap(e) \ = \sum_{e'\ sale\ de\ v} cap(e')$$

Si la igualdad entre las suma se cumple para todo vértice en la gráfica (menos s y t), entonces la red de flujo no cuenta con un corte mínimo exclusivo. Ya que las capacidades se conservan a lo largo de la red de flujo, se dice que la red es de capacidad homogénea. Por lo que cualquier corte en la red es del mismo valor (podría decirse que todo corte en la red es mínimo). Por otro lado, el flujo máximo que puede ser enviado a través de la red satura por completo a todas las aristas (debido a la conservación de capacidades). Por lo que cualquier corte en la red es de flujo máximo.

Si se conoce que la red de flujo cumple esta condición, buscar el corte mínimo para determinar el flujo máximo sobre la red es trivial, por lo que puede considerar un parámetro más para llevar acabo la aglomeración. Un algoritmo que bajo esta condición obtiene algomeraciones basadas en las propiedades de corte y flujo se muestra a continuación:

Sea max_flow el valor de máximo flujo que se encuentra sobre una red G de capacidades homgéneas, con un vértice inicial s y un final t, sea k el número de aglomeraciones que se desean obtener de la red de flujo, y sea n el número de caminos aumentados que se encontraron durante el Algoritmo de Ford-Fulkerson. Entonces el Algoritmo 4 Division_de_Flujo(G, s, t, k) hace un balance de los n caminos aumentados que existen en G sobre las k particiones que se desean obtener, y a cada unos de los vértices que conforman los caminos aumentados se le asigna una aglomeración especifica.

Note que para cumplir con la condición que s y t no pertenecen al mismo conjunto, entonces para $k \geq 2$, el vértice inicial s es agregado al primer conjunto de aglomeración encontrado y el vértice final t es agregado al k-ésimo conjunto de aglomeración encontrado. Un caso de estudio donde se presenta la conservación de capacidades sobre una red de flujo se presenta en la sección 5.1.

Algorithm 4 Division_de_Flujo(G, s, t, k)

```
max_flow ← Ford-Fulkerson(G, s, t)
n ← CaminosAumentados(G, s, t)
agl ← n/k
i ← 1
for l ← 1, hasta l ≤ n do
  for all arista e = (x, y) ∈ {cam_l, cam_{l+1}, ..., cam_{l+agl}} do
    agrega al vértice y a la i-esima aglomeración
  end for
  l ← l + agl
  i ← i + 1
end for
```

4.3. La Estrategia de Aglomeración

Tomando en cuenta las consideraciones de la sección anterior para la estrategia de aglomeración de procesos paralelos, se procede aquí a detallar cada uno de los pasos que conforman dicha estrategia.

4.3.1. Generación de la gráfica de Software

Una gráfica de software es una representación abstracta que permite modelar la estructura de conexión entre procesos de un sistema paralelo.

La gráfica de software se genera en base a la especificación de un sistema paralelo, representando a los procesos en forma de vértices, y a comunicaciones entre pares de ellos como aristas de la gráfica. Cada arista en la gráfica de software tiene asignada una ponderación entera independiente mayor a cero.

La gráfica de software además de proveer una especificación de la estructura de comunicación entre procesos, también es idealmente empleada, por su sencillez y fidelidad, con que puede representar la estructura de un sistema paralelo. Por otra parte, es fácil, dada una gráfica, poder expresarla en términos de una red de flujo.

De acuerdo a la metodología genérica de diseño de software paralelo que se especifica en el capítulo 2, en las etapas de descomposición y comunicación es donde se obtiene

dicha gráfica de software.

Sea $G_{sw} = (V_{sw}, E_{sw})$ una digráfica, conexa y sin aristas paralelas. Entonces:

1. La etapa de descomposición es la encargada de expresar eficientemente el paralelismo. Es aquí, donde se identifican los procesos paralelos que pueden conformaran el sistema. En esta etapa se construye el conjunto de vértices V_{sw}, identificando cada proceso del sistema. Cada proceso p_n del sistema paralelo, se agrega al conjunto de vértices V_{sw} de la gráfica de software.

$$V_{sw} = V_{sw} \cup \{p_n\}$$

2. La etapa de comunicación organiza las relaciones de comunicaciones entre pares de procesos del sistema. Es aquí donde mediante el conjunto de vértices V_{sw} se genera el conjunto de aristas E_{sw} por cada par de procesos que requieran comunicación para su correcta ejecución.

   Sean p_i , $p_j \in V_{sw}$ dos procesos. Si p_i establece un canal de comunicación directo a p_j, entonces existe una arista e_{ij} que debe ser incluida en E_{sw}.

$$E_{sw} = E_{sw} \cup \{e_{ij}\} \; tal \; que, \; e_{ij} = (p_i, p_j)$$

   e_{ij} es de la forma (p_i, p_j), ya que la comunicación va del proceso p_i al proceso p_j. Además, e_{ij} tiene asociada una capacidad $c(e_{ij})$ donde:

$$c(e_{ij}) = n, \; tal \; que, \; n \in \mathbb{N}^+$$

   La capacidad de cada arista se establece como la suma de la cantidad de datos que es transmitido sobre ella durante tiempo de ejecución. La obtención de las cantidades de comunicaciones (capacidades de las aristas) son totalmente dependientes del sistema paralelo y el tipo de comunicación que éste emplee. Existen tres tipos de comunicación que se clasifican en relación al determinismo que éstas pueden tener:

   - Comunicación determinista: Si las comunicaciones pueden ser determinadas en su totalidad y representadas mediante un número entero positivo.

- Comunicación dinámica: Si las comunicaciones se incrementan o decrementan con relación al sistema paralelo. Entonces, se emplea una cantidad de comunicación estimada en el tamaño del sistema.
- Comunicación no determinista: Si las comunicaciones no pueden ser determinadas por completo, es decir, no pueden representarse con un valor entero positivo, ya que depende condicionalmente de la naturaleza que el sistema paralelo resuelve.

Para comunicaciones deterministas y dinámicas, es posible determinar o estimar una valor entero positivo preciso para caracterizar la cantidad de flujo de datos que transita sobre una arista. En caso contrario, cuando las comunicaciones son no deterministas, se pierde precisión sobre las aristas con comunicaciones. El problema se da ya que la cantidad de comunicaciones es totalmente dependiente de la naturaleza del sistema.

3. Para cualquier vértice en un gráfica de software, éste puede clasificarse de acuerdo a:

   - Vértices iniciales o generadores de flujo: Pueden realizar algún tipo de procesamiento. Principalmente, son procesos que generan y envían los datos que otros procesos necesitan para su ejecución. En ocasiones, puede que un proceso de este tipo también reciba datos del algún otro vértice generador.
   - Vértices de procesamiento-redistribución: Reciben datos de los vértices generadores. Pueden realizar algún tipo de procesamiento y lo redistribuyen a otros vértices.
   - Vértices de finales: Reciben datos de vértices generadores o de redistribución. Hacen procesamiento, y pueden regresar una respuesta al o los vértices de donde fueron recibidos los datos, en ocasiones, los vértices finales pueden tener comunicación entre ellos.

De tal forma que al identificar, los procesos del sistema paralelo, la estructura de comunicaciones entre cada par de procesos, y la clasificación que recibe cada uno de ellos, entonces se puede genera la gráfica de software G_{sw}. Una vez construida, es necesario modelar dicha gráfica obtenida en una red de flujo, proceso que se muestra

en el siguiente paso de la estrategia. Si el sistema paralelo está basado en algún patrón de diseño de software paralelo, entonces la gráfica de software es muy parecida a éste.

4.3.2. Transformación a una red de flujo

Una red de flujo es una digráfica ponderada de acuerdo a su capacidad, con un flujo asignado y cumple con las características descritas en la sección 2.4.1. Una red de flujo se emplea para modelar la cantidad de datos que se envían desde un vértice inicial a través de la red y hasta un vértice final. Dada una red de flujo, existe un algoritmo que permite hacer una partición del grupo de vértices, el cual toma en cuanta la cantidad de flujo transmitido por cada vértice como un factor para realizar el particionamiento o corte de la red de flujo.

Sea $G_{sw} = (V_{sw}, E_{sw})$ la gráfica de software obtenida del paso anterior, y sea $G_{nf} = (V_{nf}, E_{nf})$ una red de flujo, tal que G_{nf} es dirigida y no contiene aristas paralelas. Entonces:

1. Sean s, t dos vértices, s un vértice fuente y t un vértice sumidero, se agregan al conjunto de vértices de la red de flujo. Entonces:

   $$V_{nf} = V_{sw} \cup \{s, t\}$$

   donde s y t son dos vértices especiales que no cumplen con la propiedad de conservación.

2. Para cada arista $e_k \in E_{sw}$ se asigna un valor entero $f(e_k) = x$ que determina su cantidad de flujo asignado, tal que x es menor o igual a la capacidad $c(e_k)$ de la misma [3].

   $$\forall\ e_k \in E_{sw},\ f(e_k) = x,\ tal\ que,\ x \in \mathbb{N}\ y\ 0 \leq f(e_k) \leq c(e_k)$$

   entonces se agregan al conjunto de aristas de la red de flujo:

   $$E_{nf} = E_{nf} \cup \{e_k\}$$

   Para un estado de inicialización $f(e_k)$ se asigna a 0 (flujo nulo).

3. El vértice s debe tener una arista e_g dirigida a cada vértice generador de flujo o vértice inicial v_{gen} de la gráfica de software, tal que:

$$E_{nf} = E_{nf} \cup \{e_g\} \; tal \; que, \; e_g = (s, v_{gen}) \; \forall \, v_{gen} \; \in \; V_{sw}$$

Normalmente los vértices generadores de flujo representan a procesos main o procesos que delegan trabajo a procesos esclavos. En algunas ocasiones, los vértices generadores de flujo pueden comunicarse entre ellos, aunque generalmente existe uno por gráfica.

La capacidad $c(e_g)$ de cada arista $e_g = (s, v_{gen})$ debe ser igual o mayor a la suma de las capacidades de las aristas que excedan de v_{gen} y vayan a cualquier vértice $u \in V_{nf}$.

$$c(e_g) \geq \sum c(e_{out}) \; tal \; que, \; e_{out} = (v_{gen}, u), \;\; \forall \, v_{gen} \in V_{nf}$$

Si esta propiedad no se cumple, es posible que las aristas e_g alteren el resultado del corte mínimo flujo máximo. Por motivos de la implementación del algoritmo de Ford-Fulkerson, es recomendable que la capacidad $c(e_g)$ sea al menos mayor en una unidad que la suma de las capacidades $c(e_{out})$.

4. Para cada vértice final v_f, debe existir una arista e_f dirigida al vértice t:

$$E_{nf} = E_{nf} \cup \{e_f\} \; tal \; que, \; e_f = (v_{fin}, t), \;\; \forall \, v_{fin} \; \in \; V_{sw}$$

Los vértices finales representan a aquellos procesos esclavos, cuyo propósito general es realizar tareas de procesamiento con datos recibidos, y no tanto comunicarse con los demás, aunque en algunas ocasiones pueden comunicarse entre procesos esclavos o regresar resultados a procesos superiores.

La capacidad $c(e_f)$ de cada arista $e_f = (v_{fin}, t)$ debe ser igual o mayor a la suma de las capacidades de aristas que finalizan en v_{fin} y vienen de cualquier vértice $u \in V_{nf}$.

$$c(e_f) \geq \sum c(e_{in}) \; tal \; que, \; e_{in} = (u, v_{gen})$$

Es importante asegurar que para cada vértice, excepto s y t, se cumpla con las propiedades de una red de flujo.

Ahora el vértice fuente puede encausar flujo de datos por medio de la gráfica de software hasta el vértice sumidero. En la figura 4.5 se muestra la agregación de ambos vértices. Las aristas punteadas se agregan para dirigir el flujo generado a partir del vértice s y hasta el vértice t.

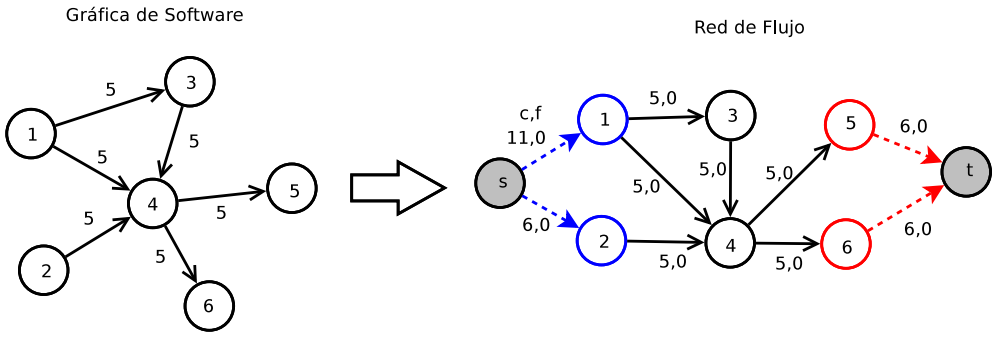

Figura 4.5: Transformación de una gráfica de software a una red de flujo. Los vértices sombreados s y t son los vértices especiales que se agregan. Las flechas punteadas de color azul, indican las aristas que se agregan para conectar al vértice s con los vértices generadores de flujo (de color azul), y las flechas punteadas de color rojo, indican las aristas que se agregan para conectar a todo vértice final (de color rojo) con el vértice t. Cada arista tiene asignado un numero entero que delimita su capacidad y uno que indica su flujo asignado (el cual se inicia en cero).

Una vez que la red de flujo es construida, es posible ahora aplicar sobre ella un algoritmo para aglomerar (particionar) el conjunto de vértices V_{nf}, tomando como parámetro principal la cantidad de comunicaciones que se transmiten entre pares de procesos del sistema paralelo.

4.3.3. Aplicación del algoritmo

Ya que la red de flujo se ha modelado, es necesario aplicar un algoritmo para obtener las aglomeraciones de procesos paralelos necesarias. El Algoritmo de Ford Fulkerson extendido (sección 4.2.1) es un algoritmo iterativo empleado para calcular el máximo flujo que se puede transitar por una red, y por consiguiente, puede determinar el corte mínimo sobre la misma. Es decir, el algoritmo obtiene una partición de vértices en dos conjuntos, la cual cumple que la cantidad de datos transmitidos y datos recibidos entre dos aglomeraciones sea mínima. El algoritmo de Ford-Fulkerson (y por

lo tanto Ford-Fulkerson extendido) está basado en tres importantes elementos, gráficas residuales (sección 2.4.2), caminos aumentados (sección 2.4.3) y cortes (secció 2.4.1) [6].

Sea $G_{nf} = \{V_{nf}, E_{nf}\}$ una red de flujo. De acuerdo con las consideraciones expuestas al inicio de este capítulo, la red de flujo G_{nf} puede ser de dos tipos:

1. Sea G_{nf} una red de flujo con capacidades homogéneas:

   Considere el caso especial que se expone en la sección 4.2.2, y el Algoritmo 4 Division_de_Flujo(G, s, t, k) que se propone para la solución de éste.

   El algoritmo de Division_de_Flujo(G, s, t, k) toma como parámetros de entrada una red de flujo G, un vértice inicial s, un vértice final t, y un número k de particiones que se desean obtener sobre la red de flujo G. El algoritmo entrega como salida k subconjuntos de vértices.

2. Sea G_{nf} una red de flujo convencional o de capacidades heterogeneas:

   Considere el Algoritmo 3 Ford-Fulkerson-Extendido(G, s, t) expuesto en la sección 4.2.1, el cual obtiene una bipartición considerando la bidireccionalidad de comunicaciones entre éstas.

   El algoritmo de Ford-Fulkerson-Extendido(G, s, t) toma como parámetros de entrada una red de flujo G, un vértice inicial s y un vértice final t. El algoritmo entrega como salida un par de conjuntos de vértices A y B, los cuales intercambian una mínima cantidad de comunicaciones (con respecto a la red G). Dado que el objetivo es conseguir k subconjuntos de procesos paralelos, el algoritmo 3 se aplica recursivamente sobre el subconjunto de cardinalidad mayor de los subconjuntos ya computados, hasta obtener los k subconjuntos deseados.

Puede notar que en cualquier de los casos anteriores, la salida de ambos algoritmos entrega k subconjuntos de procesos paralelos $A_1, A_2, ..., A_k$ tal que $A_1 \cup A_2 \cup .... \cup A_k = V_{nf}$.

Asumiendo que el número de unidades de procesamiento en el sistema distribuido es igual a k, entonces un algoritmo de mapeo directo puede sólo asignar cada una de las aglomeraciones a cada una de las k unidades de procesamiento para su ejecución.

4.4. Resumen

En el capítulo 4 se detalla la estrategia de aglomeración propuesta, descrita en tres etapas que van desde los requerimientos iniciales, la adaptación a un ambiente de redes de flujo, y hasta la aglomeración de procesos en k subconjuntos de acuerdo a las propiedades que la red presenta. También se muestra algunas consideraciones adicionales que ayudan a que el modelo con que se representan los procesos paralelos del sistema sea más apegado a la realidad, y tomando en cuenta los posibles escenarios que puedan presentarse.

La generación del grafo de software se realiza analizando el código para detectar el número de subprocesos (o threads) que se generan desde un proceso principal, y la relación que hay entre procesos paralelos mediante primitivas de comunicación.

Una vez conformado el grafo de software, la transformación a una red de flujo es necesaria para modelar la cantidad de datos transmitidos desde un punto inicial hasta un punto final. Sobre la red de flujo obtenida, se ejecuta un algoritmo para obtener los k subconjuntos del sistema paralelo, los cuales minimizan la comunicación hacia afuera de los subconjuntos. Dado que se obtienen k subconjuntos, y suponiendo que se tiene el mismo número de unidades de procesamiento, el mapeo sólo se encarga de realizar una asignación uno a uno de las aglomeraciones sobre las unidades de procesamiento.

El siguiente capítulo muestra dos casos de estudio en los que la estrategia se aplica para conseguir minimizar los tiempo de comunicación.

5

Casos de Estudio

A lo largo de este capítulo se presentan dos casos de estudio, los cuales sirven de ejemplo para mostrar cómo la aplicación de la estrategia propuesta en el capítulo 4, se lleva a cabo sobre un sistema paralelo en particular. La sección 5.1 aborda un problema clásico de computación, donde dado un conjuntos de números, se desea obtener un orden de dichos elementos. El segundo caso de estudio es la descomposición de Cholesky, la cual es una factorización de una matriz de características particulares.

El diseño del sistema paralelo de ambos casos de estudio se llevan a cabo siguiendo la metodología genérica de diseño de software paralelo, expuesta en el capítulo 2.

5.1. Ordenamiento MergeSort

5.1.1. El Problema

El ordenamiento de elementos es un problema común e importante en computación. Dado un conjunto de n elementos, se desea generar una secuencia ordenada que contenga los mismo elementos [1].

El *MergeSort* es un algoritmo clásico divide y vencerás basado en comparaciones, propuesto por John Von Neumann en 1945 [9,10]. El ordenamiento consiste en dividir los elementos en dos grupos, recursivamente ordenar los grupos más pequeños, y entonces unir las dos listas previamente ordenadas para finalmente obtener un ordenamiento por completo de los elementos [9]. Unir se refiere a combinar dos o más conjuntos ordenados en uno solo ordenadamente [10], la siguiente función ejemplifica la partición de los n datos en dos grupos, que son ordenados recursivamente.

$$MergeSort(A[1,n])$$

$$MergeSort(MergeSort(A[1,n/2]),\ MergeSort(A[n/2+1,n]))$$

El caso base de la recursión ocurre cuando el sub-arreglo a ser ordenado consiste de un solo elemento. En este punto ya no es posible una partición para aplicar un reordenamiento recursivo. El Algoritmo 5 MergeSort($array[], low, high$) toma como entrada un $array[]$ de elementos a ser ordenados, un valor entero low y $high$ los cuales respectivamente delimitan la cota inferior y superior del arreglo a ordenar.

Algorithm 5 MergeSort($array[], low, high$)

```
if low < high then
  middle=(low+high)/2
  MergeSort(array[], low, middle)
  MergeSort(array[], middle, high)
  Merge(array[], low, middle, high)
end if
```

5.1.2. Descomposición y Comunicación

De acuerdo a la metodología de diseño genérica, se sugiere descomponer el problema lo más fino posible, tal que pueda ser expresado eficientemente el paralelismo.

Puede observarse que el Algoritmo 5 MergeSort($array[], low, high$) recursivamente divide un arreglo inicial de n elementos en mitades hasta el caso base. Una forma de llevar a cabo la descomposición de este problema, es formar un árbol binario de procesos donde la raíz delega la mitad de elementos a dos procesos hijos, de tal manera que el caso base de la recursión finaliza en las hojas de dicho árbol. La figura 5.1 muestra una gráfica de árbol binario que representa la descomposición y estructura de comunicación del correspondiente sistema paralelo, el cual, ordena un conjunto de n elementos implementando el Algoritmo 5 MergeSort($array[], low, high$).

La figura 5.1 indica al proceso 1 como la raíz del árbol, el cual delega un arreglo de tamaño $n/2$ a ambos hijos (procesos 2 y 3). Ambos reciben y dividen a la mitad los elementos para delegar de nuevo a procesos de capas inferiores a lo largo de la profundidad d del árbol, hasta llegar a los procesos hoja, los cuales se encargan de

Algorithm 6 Merge($array[], low, middle, high$)

```
for i = low to i = middle do
    HL[i] = array[i]
end for
for j = middle to j = high do
    HR[j] = array[j]
end for
while not(j = HR.length or i = HL.length do
    if HR[j] <= HL[i] then
        s[k++]=HRji++]
    else
        s[k++]=HL[i++]
    end if
end while
while i¡HR.length do
    S[k++]=HR[j]
end while
while j¡HL.length do
    S[k++]=HL[i]
end while
```

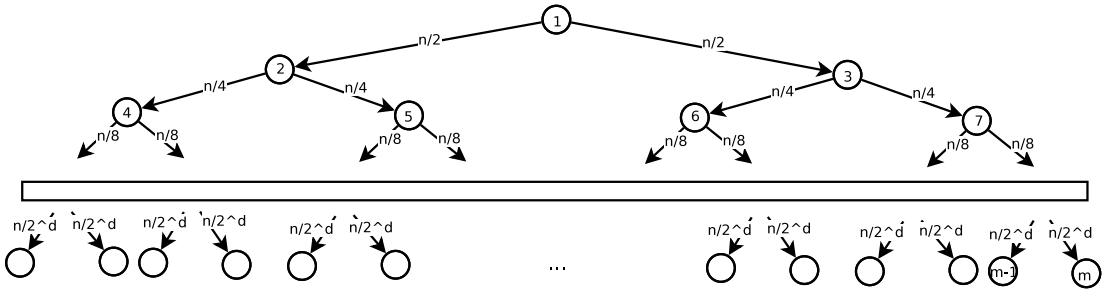

Figura 5.1: Árbol binario que representa la estructura de comunicaciones de procesos paralelos basado en el algoritmo secuencial de ordenamiento Merge. En cada capa del árbol el arreglo es dividido entre 2^d, donde d es la profundidad del mismo.

ordenar un arreglo de tamaño 2 para regresarlo a su proceso padre respectivo, el cual se encarga de unir ambos arreglos delegados de forma ordenada, hasta concluir con el arreglo ordenado de tamaño n por el proceso raíz.

Por simplicidad para el análisis y desarrollo del presente caso de estudio, sólo se consideran conjuntos de elementos de tamaño de potencias de 2. También puede notarse que para ordenar un conjunto de n elementos, el árbol binario correspondiente está conformado por $n - 1$ procesos. Esto indica que el sistema paralelo cambia de tamaño (número de procesos) proporcionalmente al tamaño de los datos de entrada, lo cual haría un sistema paralelo diferente en cada caso (y por lo tanto mapeado diferente) al momento de incrementar el tamaño de los datos de entrada. Por lo que en el diseño del sistema paralelo se considera el desarrollo de un sistema que no varíe con respecto al tamaño de su entrada.

Se analiza un caso base del problema de ordenamiento, donde se desea ordenar un conjunto de 32 elementos. El sistema paralelo correspondiente se ejecuta sobre un sistema distribuido de 3 nodos, por lo que se desean obtener 3 aglomeraciones de dicho sistema paralelo.

Tomando como referencia la estructura expuesta en la figura 5.1, se tienen 31 procesos, donde el proceso raíz es encargado de dividir el arreglo en dos mitades de 16 elementos. Sucesivamente se lleva a cabo la descomposición del arreglo a lo largo de la altura del árbol hasta llegar a los procesos hoja, los cuales reciben un arreglo de 2 elementos cada uno (el cual ya no es dividido). La figura 5.2 muestra la gráfica de comunicación para este caso concreto.

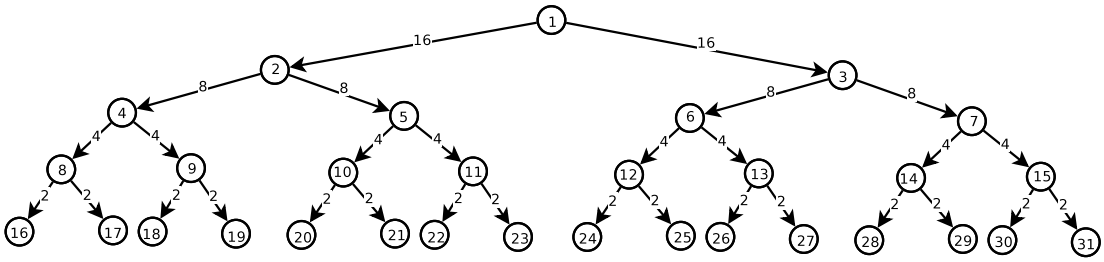

Figura 5.2: Árbol binario que representa la estructura de comunicaciones de procesos paralelos basado en el algoritmo secuencial de ordenamiento Merge, para un ordenamiento de 32 elementos.

La suma de unidades de comunicación entre procesos paralelos de acuerdo a la estructura que se presenta en la figura 5.2, se conforma de 4 capas de procesos, donde los datos son divididos y enviados a procesos de capas inferiores. De acuerdo a la naturaleza de este problema en particular, la cantidad de datos se conserva a lo largo del árbol binario (conservación de capacidad). Por lo tanto los 32 elementos iniciales son retransmitidos por las 4 capas. De ahí se obtiene que la cantidad de comunicaciones totales (CT) son:

$$CT = 128 \; unidades$$

Note que el ascenso de los datos a partir de los procesos hoja hacia la raíz generan la misma cantidad de flujo que el descenso de los mismos. Por razones de simplicidad, la red de flujo que se presenta en la figura 5.3 sólo muestra el flujo descendente de los datos, ya que ésto no altera en lo absoluto los resultados obtenidos.

Si se observa la figura 5.2 o 5.3, es notorio que la gráfica cumple con la propiedad de conservación de capacidad, lo cual indica que para todo vértice en la gráfica (excepto s y t) la suma de las capacidades incidentes al vértice es igual a la suma de las capacidades excedentes al mismo. Por lo tanto, se considera como un caso especial de una red de flujo.

5.1.3. Transformación a una red de flujo

Retomando la gráfica que se muestra en la figura 5.2, la transformación de dicha gráfica a una red de flujo consiste en agregar dos vértices especiales s y t, y un conjunto

de aristas que conectan a s con los vértices generadores y a t con los vértices finales de la gráfica.

Puede notarse que de acuerdo al problema de ordenamiento y al comportamiento de la descomposición del Algoritmo 5 MergeSort($array[], low, high$), el único proceso generador es el proceso 1, ya que éste se encarga de generar y distribuir los datos con los que todos los demás procesos de capas inferiores computan el resultado. Por otra parte, identificar los procesos finales es sencillo, dado que las hojas del árbol binario donde la recursión logra el caso base, corresponden a los procesos hasta donde los datos son redistribuidos. Por lo tanto se consideran a éstos como procesos finales.

Ahora que los procesos generadores y finales son identificados, es necesario agregar una arista por cada uno de éstos que conecten a los vértices s y t con la gráfica. Por lo que, se agrega una arista (s,1) para conectar al vértice s con el vértice generador 1, y para cada vértice final v_{fin} se agrega una arista (v_{fin},t) para conectar los vértices finales al vértice t.

Las capacidades de las aristas que se agregan, para el caso de una red de flujo de capacidades homogéneas, al menos deben preservar la propiedad de conservación de capacidad en la red, o ser mas grande que ésta. Es decir, para la arista (s,1) basta con que su capacidad sea al menos tan grande como la suma de las capacidades de aristas excedentes al vértice 1. Por lo tanto, $cap((s,1))$=32. Para las capacidades de aristas que van de un vértice final v_{fin} al vértice t, la condición es similar: para las aristas (v_{fin},t), la capacidad debe ser al menos igual a la suma de las capacidades de aristas incidentes a cada v_{fin}. Por lo que, $cap((v_{fin},t))$=2.

La figura 5.3 muestra el resultado de la transformación de la gráfica presentada en la figura 5.2, a una red de flujo.

5.1.4. Aglomeración de procesos

Como se plantea a lo largo del capítulo 4, para este tipo de casos especiales de red de flujo se aplica el Algoritmo 4 Division_de_Flujo(G, s, t, k). Tomando en cuenta que se desean obtener 3 aglomeraciones para ser mapeadas sobre un sistema distribuido de 3 nodos. Entonces una instancia para el Algoritmo 4 que resuelve el problema de obtener una 3-aglomeración de la red de flujo presentada en la figura 5.3 es:

$$Division\_de\_Flujo(G, s, t, 3)$$

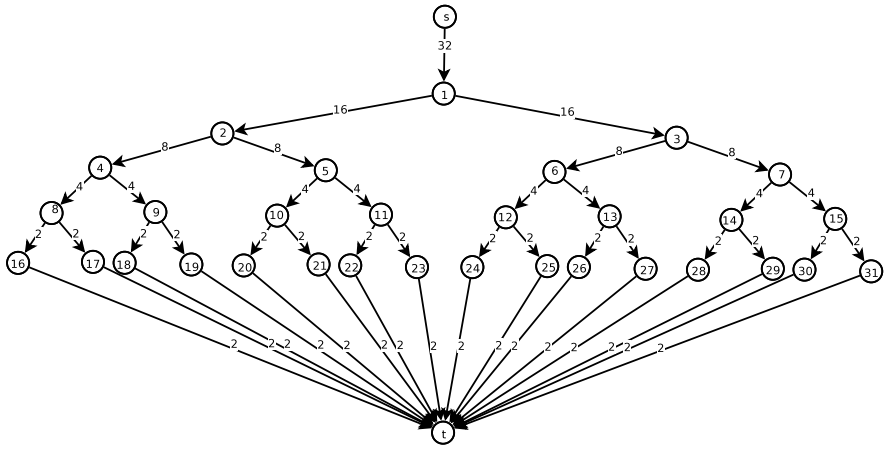

Figura 5.3: Red de flujo obtenida a partir de la gráfica que muestra la figura 5.2

donde G es la red de flujo que muestra la figura 5.3, s indica el vértice donde el flujo será generado y t el vértice a donde será dirigido el flujo, por último, el 3 indica el número de particiones que se desean obtener.

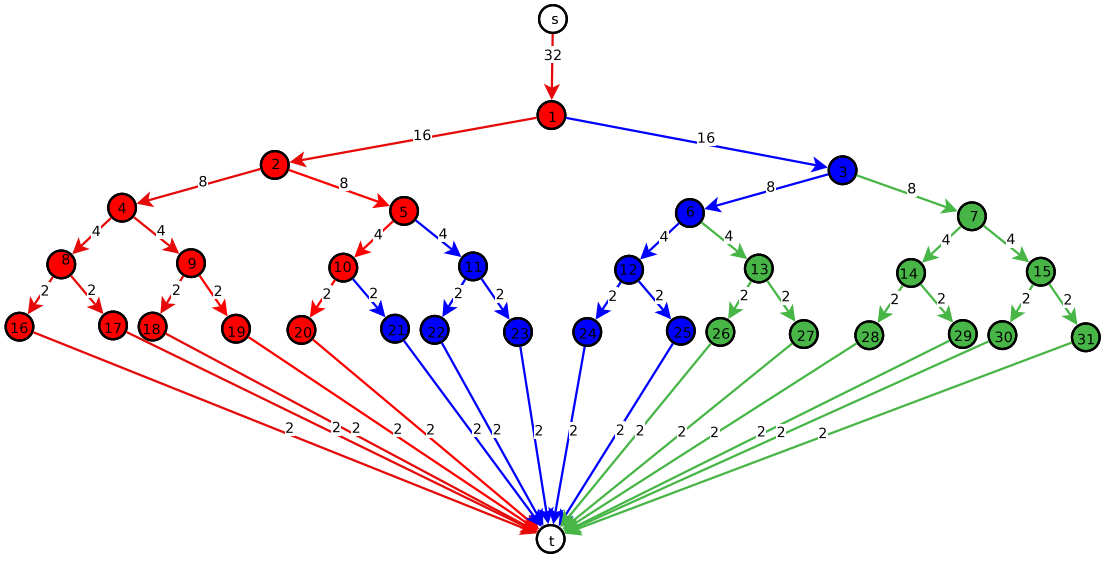

Figura 5.4: Partición en 3 aglomeraciones mediante el algoritmo de division de flujo de la red que expone la figura 5.3.

La figura 5.4 muestra el resultado de aplicar el Algoritmo 4 Division_de_Flujo sobre la red de flujo que se presenta en la figura 5.3. Por lo tanto, las particiones obtenidas se conforman de la siguiente manera:

- A (roja) = {1,2,4,5,8,9,10,16,17,18,19,20}
- B (azul) = {3,6,11,12,21,22,23,24,25}
- C (verde) = {7,13,14,15,26,27,28,29,30,31}

La figura 5.5 muestra la aglomeración de los procesos de acuerdo a los subconjunto listados anteriormente, tal que cada conjunto de procesos se mapea directamente sobre una unidad de procesamiento del sistema distribuido.

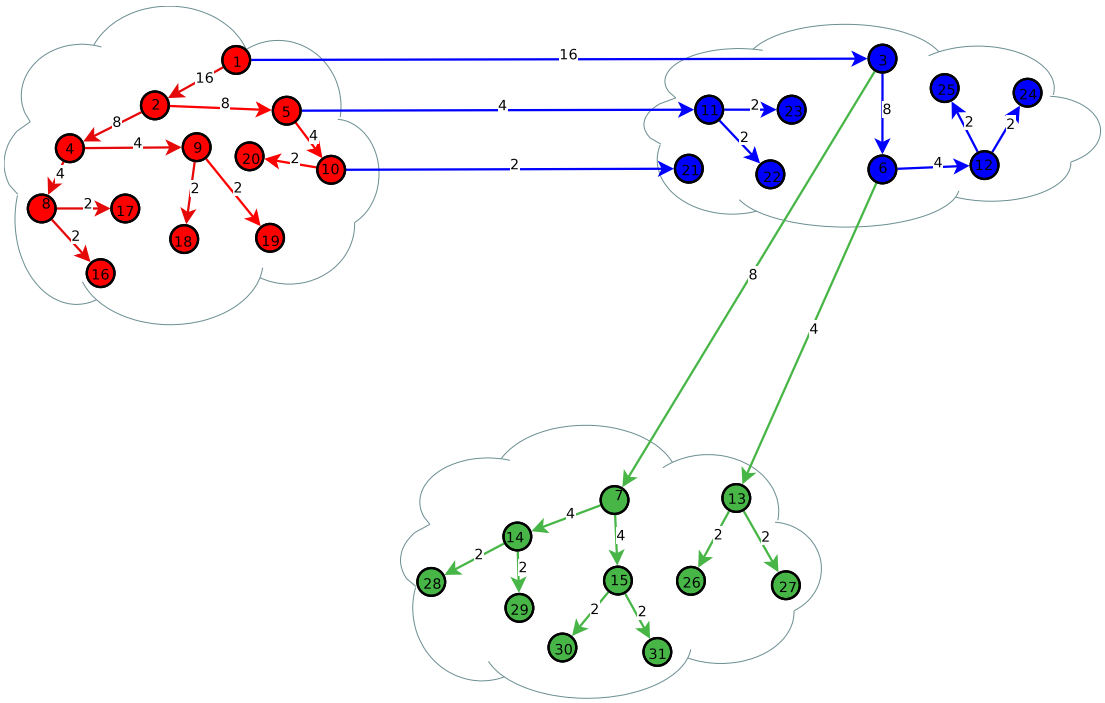

Figura 5.5: Aglomeración de los procesos paralelos que muestra la figura 5.2 de acuerdo a la división por flujo parcial.

Como se menciona en el planteamiento del problema en el capítulo 1, el tiempo de comunicación por memoria compartida toma tiempo constante, denotado como t_{cons}, mientras que el tiempo de comunicación sobre un medio de red puede ser variable, y se denota como t_{var}, tal que $t_{cons} << t_{var}$. Si se calcula el tiempo de comunicación del programa paralelo para el ordenamiento MergeSort, se tiene que:

Sea At_{out} el tiempo total de comunicaciónes excedentes de la aglomeración A, es decir, el tiempo que tardan las comunicaciones que van de un proceso $p_i \in A$ a un proceso

$p_j \notin A$ (comunicaciones exteriores). Por lo tanto, el tiempo total de comunicaciones excedentes de la aglomeración A, se obtiene al realizar la suma de las capacidades de las aristas (1,3), (5,11) y (10,21). Sea At_{in} el tiempo total de comunicaciones internas de la aglomeración A, es decir, el tiempo que tardan las comunicaciones que van de un proceso $p_i \in A$ a un proceso $p_j \in A$ (comunicaciones interiores). Por lo tanto, el tiempo total de comunicaciones internas de la aglomeración A, se obtiene al realizar la suma de las capacidades de las aristas (1,2), (2,4), (2,5), (4,8), (4,9), (5,10), (8,16), (8,17), (9,18), (9,19) y (10,20). De tal forma se tiene que:

$$At_{out} = 22\ t_{var},\ At_{in} = 54\ t_{cons}$$

De la misma forma se calculan los costos de comunicaciones exteriores e interiores para el resto de las aglomeraciones, con lo que se obtiene:

$$Bt_{out} = 12\ t_{var},\ Bt_{in} = 20\ t_{cons}$$

$$Ct_{out} = 0\ t_{var},\ Ct_{in} = 20\ t_{cons}$$

La suma de los tiempos de comunicación tanto externas como internas, de todas las aglomeraciones forman el tiempo total de comunicaciones del sistema paralelo. Cuando el sistema paralelo se aglomera bajo la estrategia basada en redes de flujo, éste tiempo se denotado como TT_{nf}.

$$TT_{nf} = At_{out} + At_{in} + Bt_{out} + Bt_{in} + Ct_{out} + Ct_{in}$$

Por lo tanto, el tiempo total de comunicaciones del sistema paralelo es:

$$TT_{nf} = 34\ t_{var} + 94\ t_{cons}$$

5.1.5. Mapeo por Round Robbin

Considere la gráfica que modela procesos paralelos y comunicaciones para el problema de ordenamiento por MergeSort (figura 5.2), para establecer un punto de comparación con la aglomeración basada en propiedades de corte y flujo, durante esta sección se compara dicha estrategia contra Round Robbin, que es ampliamente usada por la mayoría de los sistemas operativos para una calendarización de procesos.

De acuerdo con el mapeo de Round Robbin, el proceso 1 es asignado a la aglomeración A (rojo), el proceso 2 es asignado a la aglomeración B (azul), el proceso 3 es asignado a la aglomeración C (verde), y de forma circular el siguiente proceso es asignado de nuevo a la aglomeración A. Consecutivamente se realiza dicha asignación hasta que los procesos sean repartidos por completo. La figura 5.6 ejemplifica la descripción anterior para la asignación de los procesos paralelos en una 3-partición mediante Round Robbin.

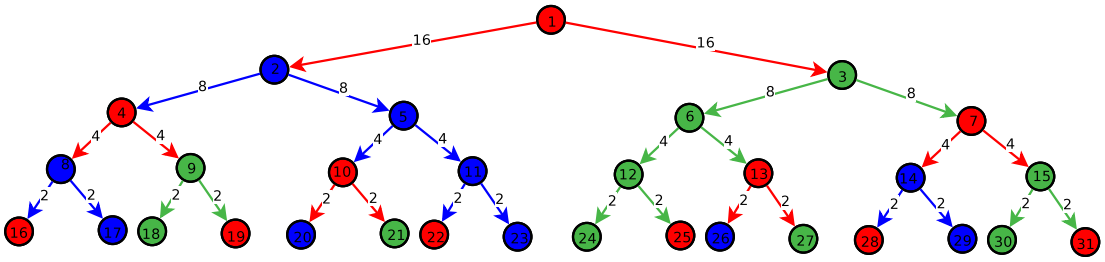

Figura 5.6: Asignación de procesos usando Round Robbin.

Por lo que las aglomeraciones están compuestas como se muestra a continuación:

- A (roja) = {1,4,7,10,13,16,19,22,25,28,31}
- B (azul) = {2,5,8,11,14,17,20,23,26,29}
- C (verde) = {3,6,9,12,15,18,21,24,27,30}

Gráficamente las aglomeraciones pueden verse en la figura 5.7, la cual muestra 3 conjuntos de procesos contruidos por medio de Round Robbin. Cada conjunto de procesos se mapea sobre un nodo del sistema distribuido.

Sea At_{out} el tiempo total de comunicaciónes excedentes de la aglomeración A. Dicho tiempo, se obtiene al realizar la suma de las capacidades de las aristas (4,8), (1,2), (7,14), (10,20), (13,26), (1,3), (7,15), (4,9), (10,21) y (13,27). Sea At_{in} el tiempo total de comunicaciones internas de la aglomeración A. Notese que no existen aristas internas en la aglomeración A. De tal forma se tiene que:

$$At_{out} = 56\ t_{var},\ \ At_{in} = 0\ t_{cons}$$

Realizando el cálculo de los tiempos de comunicación para el resto de las aglomeraciones, se tiene que:

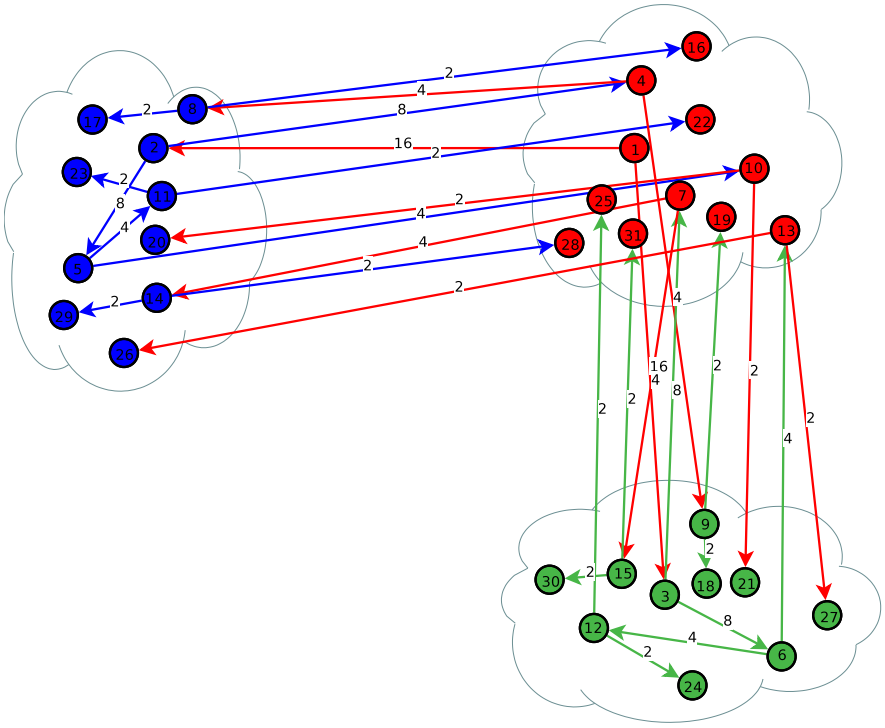

Figura 5.7: Aglomeración de procesos e interconexión entre algomeraciones usando Round Robbin como metodo de asginación.

$$Bt_{out} = 18\ t_{var},\ Bt_{in} = 18\ t_{cons}$$

$$Ct_{out} = 18\ t_{var},\ Ct_{in} = 18\ t_{cons}$$

La suma de los tiempos de comunicación tanto externas como internas, de todas las aglomeraciones, forman el tiempo total de comunicaciones del sistema paralelo. Para un mapeo del sistema con Round Robbin éste tiempo se denota como TT_{rr}. Si se realizan los cálculos para obtener dicho tiempo, se tiene que:

$$TT_{rr} = At_{out} + At_{in} + Bt_{out} + Bt_{in} + Ct_{out} + Ct_{in}$$

Por lo tanto, el tiempo total de comunicaciones del sistema paralelo es:

$$TT_{rr} = 92\ t_{var} + 36\ t_{cons}$$

5.1.6. Comparación

Note que para ambos tiempos de comunicaciones TT_{rr} y TT_{nf}, la suma de sus tiempos variables t_{var} y tiempos constantes t_{cons} respectivamente, suma la misma cantidad de comunicaciones totales CT igual a 128 unidades, como se obtiene en la etapa de descomposición y comunicación.

Considere ambos tiempos de comunicación del sistema paralelo TT_{nf} y TT_{rr}. Note que las comunicaciones interiores bajo el mapeo por redes de flujo es de 94 unidades, mientras que por el mapeo de Round Robbin es de 36 unidades. En general es preferible maximizar la cantidad de comunicaciones interiores o por memoria compartida, lo cual se consigue por medio de la aglomeración basada en redes de flujo. De forma similar, al comparar las cantidades de comunicación exteriores o por paso de mensajes, se nota que la aglomeración basada en redes de flujo obtiene un total de 34 unidades. Por el lado del mapeo de Round Robbin, se obtienen 92 unidades. De forma general, es deseable minimizar la cantidad de comunicaciones exteriores, lo cual también se consigue con la estrategia propuesta.

Si se enfoca el análisis sobre las comunicaciones exteriores, las cuales se sabe que consumen un tiempo mayor, se puede notar que dichas comunicaciones en el sistema paralelo se consiguen minimizar de 92 unidades por Round Robbin a 34 unidades por la estrategia propuesta. Con lo que se consigue reducir a un 63 % las comunicaciones exteriores.

Teóricamente, para este caso de estudio se puede ver que la estrategia de aglomeración basada en redes de flujo tiene un mejor comportamiento con respecto a Round Robbin, en la maximización de comunicaciones locales y minimización de las comunicaciones remotas.

A continuación, se presenta un conjunto de resultados que muestran datos experimentales de 6 ejecuciones del sistema paralelo de ordenamiento basado en el Algoritmo 5 MergeSort($array[], low, high$) sobre un Cluster[1]. El Cluster cuenta con 4 nodos de 8 núcleos cada uno, 16 GB de memoria RAM y sistema operativo Ubuntu 12.04 LTS (GNU/Linux 3.2.0-27-generic x86_64), conectados por una switch Gigabit Ethernet.

Las ejecuciones de cada experimentación se realizan para un mismo conjunto de datos de entrada, la cantidad de procesos paralelos del sistema no varía en relación

[1]Del Departamento de Ingeniería en Sistemas, Control y Automatización del Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas de la UNAM.

No Ejec	T total (RR)	T total (NF)
1	411 ms	430 ms
2	460 ms	460 ms
3	441 ms	442 ms
4	437 ms	449 ms
5	430 ms	390 ms
6	451 ms	411 ms
7	441 ms	440 ms
8	421 ms	401ms
9	442 ms	460 ms
10	428 ms	431 ms
T Prom	436.2 ms	431.4 ms

Cuadro 5.1: 10 ejecuciones del sistema paralelo para un ordenamiento de 32 elementos.

con el tamaño de los datos de entrada, la única variación entre ambas ejecuciones del sistema es la forma en como los procesos son asignados a los nodos del Cluster.

La tabla 5.1 muestra los resultados obtenidos en la primera de las experimentaciones, la cual consiste en los tiempos de 10 ejecuciones del sistema paralelo sobre el Cluster mencionado sólo haciendo uso de 3 nodos de procesamiento. De lo datos de la experimentación que muestra la tabla 5.1, un primer dato a considerar es el tiempo que reporta la ejecución número 5 mapeada con la estrategia basada en redes de flujo de 390 ms, el cual puede ver que es el menor tiempo reportado, considerando ambos esquemas de mapeo. El segundo dato a considerar es el tiempo reportado por las ejecuciones número 2 bajo ambos esquemas de 460 ms, el cual es el mayor tiempo reportado en general. Por último en promedio para la estrategia de Round Robbin se obtiene un tiempo de 436.2 ms, mientras que el tiempo promedio para la estrategia basada en redes de flujo es de 431.4 ms. Por lo que la diferencia entre los tiempos muestra que la estrategia basada en redes de flujo mejora en 4.8 ms, lo cual representa el 1.1 % del tiempo de ejecución total.

Puede verse que los resultados anteriores muestran un mejor tiempo de ejecución para la estrategia basada en redes de flujo de forma experimental, aunque de forma general esto no quiere decir que así sea para problemas de ordenamiento mayores.

No Ejec	T total (RR)	T total (NF)
1	760 ms	804 ms
2	702 ms	706 ms
3	740 ms	777 ms
4	773 ms	784 ms
5	790 ms	763 ms
6	731 ms	787 ms
7	719 ms	802 ms
8	720 ms	817 ms
9	748 ms	809 ms
10	798 ms	804 ms
T Prom	748.1 ms	785.3 ms

Cuadro 5.2: 10 ejecuciones del sistema paralelo para un ordenamiento de 8192 elementos.

Para ampliar la experimentación con el fin de arrojar datos mas concisos. Ahora considerese la misma gráfica de software y por lo tanto la misma red de flujo para el problema de ordenamiento, solo que ahora se realiza un ordenamiento de 8,192 números (2^{13}) por lo que cada hoja ordena 512 elementos. La tabla 5.2 muestra los resultados de los tiempos promedio de las 10 ejecuciones bajo el mismo Cluster usando los mismos 3 nodos.

Para la segunda experimentación el promedio de tiempo de ejecución para la estrategia de Round Robbin es de 748.1 ms, mientras que para la estrategia basada en redes de flujo es de 785.3 ms. Puede notarse que para esta cantidad de datos la estrategia de Round Robbin presenta un mejor desempeño con una diferencia de 37.2 ms mejorando en un 5 % del tiempo de ejecución.

De forma similar en la tabla 5.3 se muestran 10 ejecuciones del sistema paralelo ordenando 1,048,576 elementos (2^{20}). En cada proceso hoja se delegan arreglos de tamaño 65,536 para su ordenamiento. El promedio de las ejecuciones para Round Robbin es de 78,749.2 ms. Por su parte, con la estrategia propuesta se obtiene un promedio de 81,507.8 ms. Una vez mas puede observa que la estrategia de Round Robbin mejora en promedio 2,758.6 ms, que representan el 3.5 % del tiempo de ejecución total.

Note que el porcentaje de mejora que obtiene Round Robbin en la tercera ex-

No Ejec	T total (RR)	T total (NF)
1	69338 ms	79592 ms
2	75309 ms	78940 ms
3	79004 ms	83489 ms
4	82008 ms	82270 ms
5	84982 ms	81591 ms
6	81075 ms	84949 ms
7	80743 ms	84928 ms
8	77848 ms	77755 ms
9	78585 ms	81721 ms
10	78600 ms	79843 ms
T Prom	78,749.2 ms	81,507.8 ms

Cuadro 5.3: 10 ejecuciones del sistema paralelo para un ordenamiento de 1,048,576 elementos.

perimentación (tabla 5.3) es menor que el porcentaje que se obtiene de la segunda experimentación (tabla 5.2).

La tabla 5.4 muestra los datos de una cuarta experimentación para un ordenamiento de 4,194,204 elementos (2^{22}), tal que cada hijo ordena 262,144 elementos. Puede verse que de nuevo la estrategia basada en redes de flujo vuelve a obtener tiempos de ejecución menores, y en promedio muestra una ventaja con un tiempo de 424,100.7 ms, frente a los 431,576.4 ms que se obtienen mediante Round Robbin. La diferencia entre ambos tiempos es de 7,475.7 ms lo cual representa el 1.7 % del tiempo de ejecución total.

La tabla 5.5 que muestra los resultados del quinto experimento el cual consta de 10 ejecuciones del sistema paralelo para un ordenamiento de 8,388,608 elementos (2^{23}), cada hijo ordena de manera independiente arreglos de tamaño 524,288.

Para los datos que muestra la tabla 5.5 el promedio de ejecución para la estrategia de Round Robbin es de 863,844.9 ms, mientras que para la estrategia basada en redes de flujo se obtiene un promedio de 843,078 ms. La diferencia entre ambos tiempos de ejecución son 20,766 ms, lo cual representa un 2.4 % del tiempo de ejecución total.

Finalmente la tabla 5.6 muestra los últimos datos experimentales, donde el sistema paralelo se encarga de ordenar 16,777,216 elementos (2^{24}) delegando a cada una de las hojas arreglos de tamaño 1,048,576 elementos.

No Ejec	T total (RR)	T total (NF)
1	414108 ms	407206 ms
2	421504 ms	428735 ms
3	472324 ms	436719 ms
4	412169 ms	403574 ms
5	418649 ms	432429 ms
6	416026 ms	424889 ms
7	411771 ms	413634 ms
8	470415 ms	422560 ms
9	426422 ms	424452 ms
10	452376 ms	446809 ms
T Prom	431,576.4 ms	424,100.7 ms

Cuadro 5.4: 10 ejecuciones del sistema paralelo para un ordenamiento de 4,194,204 elementos.

No Ejec	T total (RR)	T total (NF)
1	974459 ms	770388 ms
2	929662 ms	806303 ms
3	887337 ms	883273 ms
4	975441 ms	792662 ms
5	770622 ms	888132 ms
6	850999 ms	808648 ms
7	831308 ms	987233 ms
8	808658 ms	787397 ms
9	766199 ms	858757 ms
10	843764 ms	847990 ms
T Prom	863,844.9 ms	843,078 ms

Cuadro 5.5: 10 ejecuciones del sistema paralelo para un ordenamiento de 8,388,608 elementos.

No Ejec	T total (RR)	T total (NF)
1	3644854 ms	3574897 ms
2	3710090 ms	3403640 ms
3	3669390 ms	3361170 ms
4	3729738 ms	3375164 ms
5	3822645 ms	3457327 ms
T Prom	3,715,343.4 ms	3,434,439.6 ms

Cuadro 5.6: 5 ejecuciones del sistema paralelo para un ordenamiento de 16,777,216 elementos.

Para los datos de la tabla 5.6 los promedios obtenidos son de 3,715,343.4 ms para la estrategia de Round Robbin, mientras que para la estrategia basada en redes de flujo se obtienen 3,434,439.6 ms, dando una diferencia de 280,903.8 ms lo cual representa un 7.6 % del tiempo de ejecución total.

Los datos experimentales que se obtienen a partir de la tabla 5.4 en adelante, muestran una reducción cada vez mas perceptible en el tiempo total de ejecución del sistema paralelo, lo cual sugiere de manera práctica que a partir de este punto la estrategia de algomeración basada en redes de flujo mejora de manera incremental, como se muestra en las gráficas de las figuras 5.8 y 5.9.

5.2. Descomposición de Cholesky

5.2.1. El Problema

Si una matriz es simétrica y definida positiva, entonces tiene una descomposición triangular especial. Ser simétrica significa que $a_{i,j} = a_{j,i}$ para $i, j = 1, \ldots, N$, mientras que definida positiva significa que [11] :

$$v \cdot A \cdot v > 0, \quad \textit{para todo vector } v$$

Las matrices simétricas, definidas positivas, son bastante especiales, y aparecen frecuentemente en algunas aplicaciones. Su factorizacíon especial, llamada descomposición de Cholesky, tiene un factor de dos veces mas rápida a sus alternativas para resolver sistemas de ecuaciones lineales [11].

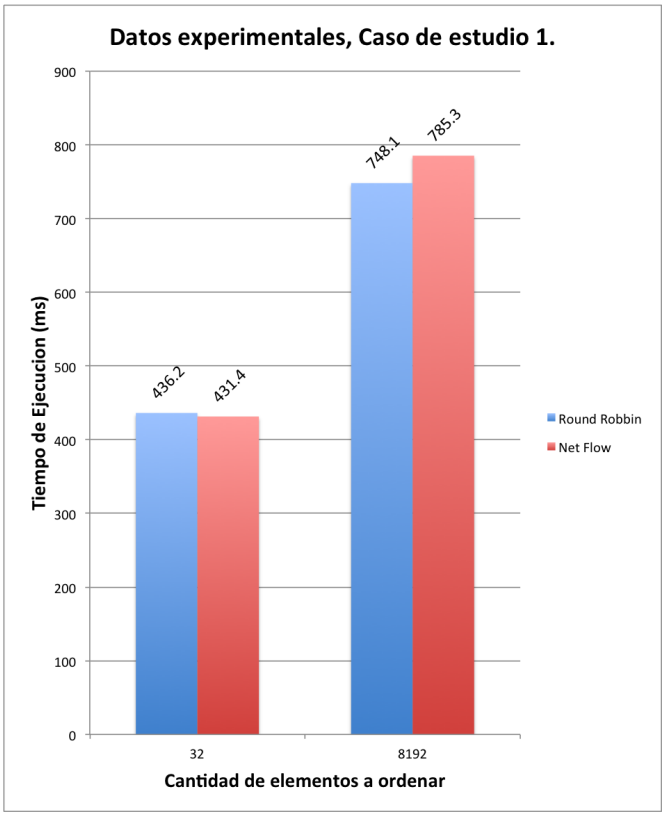

Figura 5.8: Gráfica comparativa entre los tiempo de ejecución obtenidos con Round Robbin (color azul) y la estrategia basada en redes de flujo (color rojo). Gráfica que muestra los tiempos de ejecución, obtenidos por las primeras dos experimentaciones.

En vez de buscar factores triangulares máximos y mínimos arbitrariamente, la descomposición de Cholesky construye una matriz triangular inferior L cuya transpuesta L^T puede funcionar como la parte triangular superior [11]:

$$L \cdot L^T = A$$

Rescribiendo la ecuación anterior de forma más detallada se tiene que[11]:

$$L_{i,i} = \left(a_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2 \right)^{1/2}$$

y

$$L_{j,i} = \frac{1}{L_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} L_{i,k} L_{j,k} \right) \quad j = i+1, i+2, ..., N$$

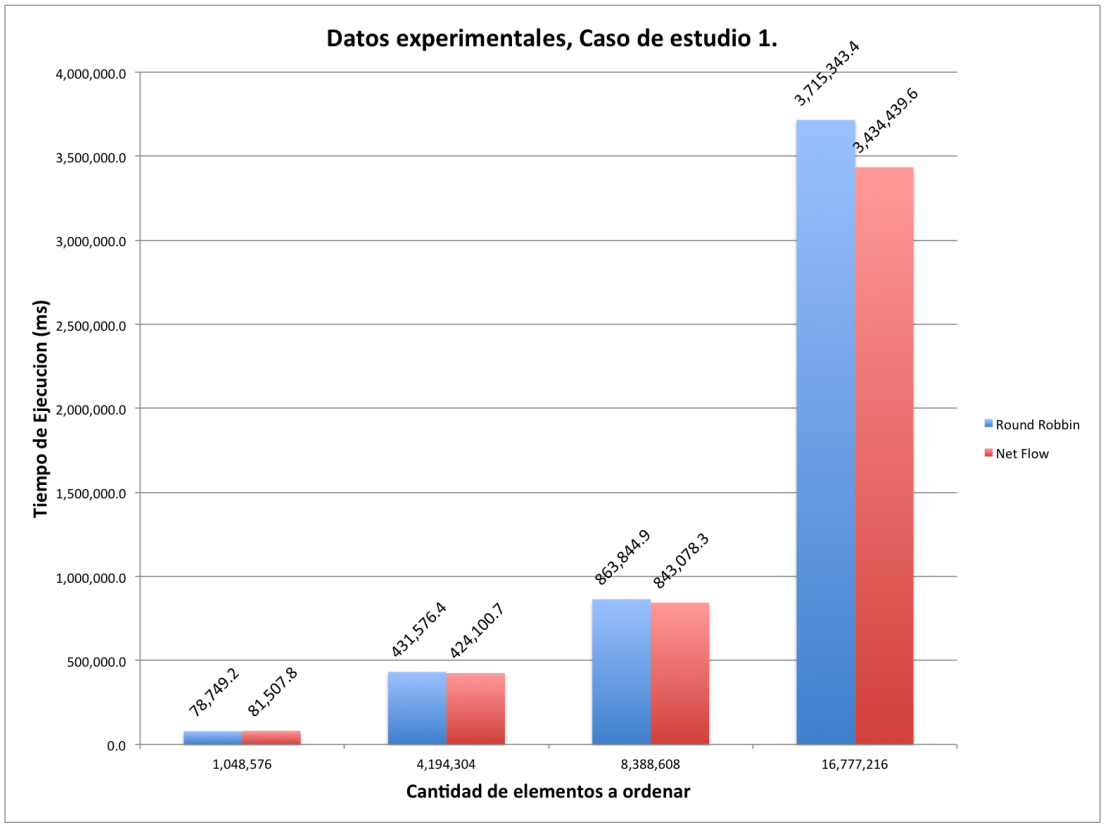

Figura 5.9: Gráfica comparativa entre los tiempo de ejecución obtenidos con Round Robbin (color azul) y la estrategia basada en redes de flujo (color rojo). Gráfica que muestra los tiempos de ejecución, obtenidos por el resto de las experimentaciones.

La figura 5.10 muestra la dependencia de datos para calcular la descomposición de Cholesky en una matriz de tamaño 4x4. Recuerdese que la matriz necesita forzosamente ser simétrica y definida positiva. Debido a esto el triángulo superior es despreciado (casillas en negro) y solo se trabaja con el trángulo inferior y la diagonal. Las casillas en gris denotan que el elemento correspondiente a esa posición se ha calculado en su totalidad, y está listo para ser utilizado por otras casillas que necesitan de él.

5.2.2. Descomposición y Comunicación

La metodología genérica de diseño de software paralelo expuesta en el Capítulo 2 sugiere descomponer el problema lo mas fino posible tal que pueda ser expresado eficientemente el paralelismo. Si se observa la figura 5.10, puede darse cuenta que existen

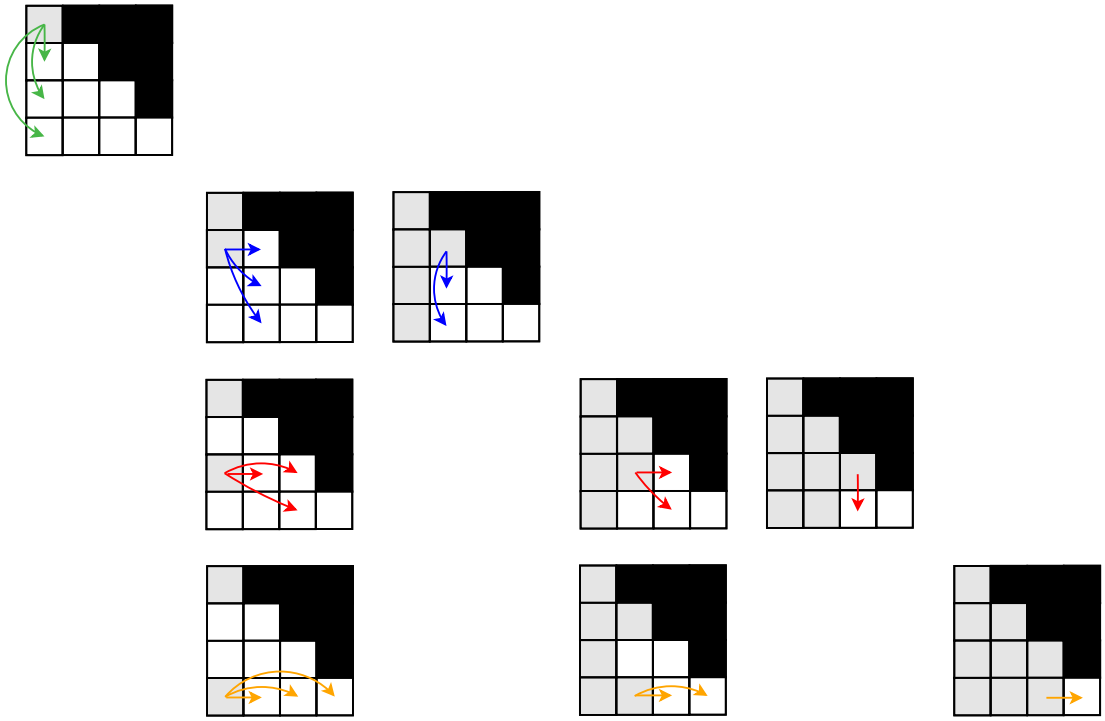

Figura 5.10: Dependencia de datos para la descomposición de Cholesky para una matriz de tamaño 4x4. Las casillas en color gris denotan que el elemento de esa localidad ha sido calculado por completo. Una vez obtenido el valor final es enviado a otras localidades que necesitan de éste.

varias formas de hacer la descomposición del problema. Particularmente, para este caso de estudio se elige una descomposición por celdas.

Cada elemento de la matriz (figura 5.10) puede representarse como proceso paralelo, que necesita datos de otros procesos para llevar a cabo su cómputo local (a excepción de la celda [1,1]). Las flechas que salen de cada casilla representan el envío de mensajes hacia otros procesos. Las flechas que entran a cada casilla representan la recepción de mensajes provenientes de otros procesos. Considerado lo anterior, la figura 5.11 muestra la gráfica de software que modela la estructura de comunicación del programa paralelo. Note que el intercambio de mensajes es unitario.

Debido a que el sistema paralelo se ejecuta en un ambiente de memoria distribuida, es necesario considerar la distribución de los datos iniciales para cada proceso. Cada proceso (celda de la matriz) necesita un elemento (o un conjunto) correspondiente a su dominio de datos. El proceso 1 se encarga de distribuir los datos iniciales correspon-

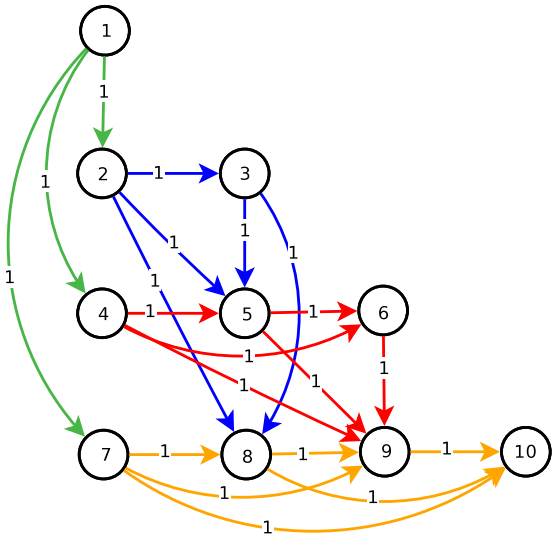

Figura 5.11: Gráfica de software para una descomposición por celda de la matriz de 4x4.

dientes a cada proceso. Por lo tanto, se agrega comunicación adicional a la gráfica, la cual se muestra en la figura 5.12.

La descomposición por celda entrega una división fina del problema, en la cual se pueden observar varios detalles. Uno de ellos es que hay procesos que esperen mucho tiempo antes de obtener los elementos necesarios para llevar a cabo su cómputo local. Tal es el caso del proceso 10, el cual debe esperar a que toda la dependencia de datos se lleve a cabo antes de poder entregar el valor final correspondiente a su casilla. Dicho proceso solo desperdicia tiempo de procesamiento, esperando hasta que sus datos sean entregados. La observación anterior sucede en muchos otros procesos que esperan por la entrega de datos necesarios, lo cual conduce a gastar recursos de cómputo.

5.2.3. Transformación a una red de flujo

Según la clasificación de vértices que se presenta en la sección 4.3.1, es necesario identificar aquellos procesos iniciales y finales, los cuales son adyacentes a los nuevos vértices especiales que se agregan de acuerdo a lo que indica la sección 4.3.2. Puede observarse que el vértice 1 y el vértice 10 son los únicos procesos generador y final respectivamente, por lo que agregar dos vértices especiales s y t en este caso no es necesario, ya que puede considerarse directamente a los vértices 1 y 10 como los vértices especiales fuente y sumidero.

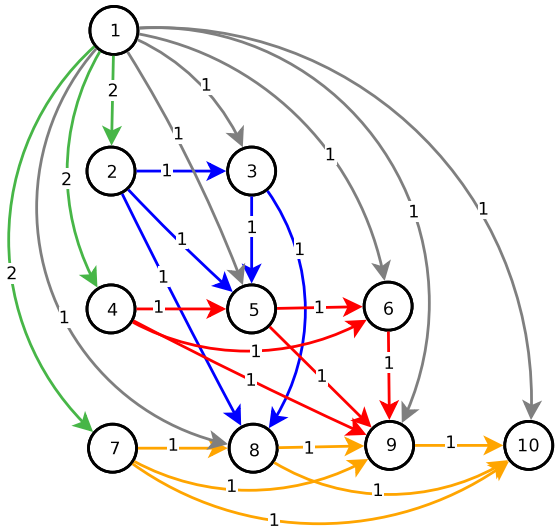

Figura 5.12: Gráfica de software para una descomposición por celda de la matriz de 4x4, considerando distribución inicial de datos requeridos.

La ponderación de las aristas de la red de flujo se obtiene de la cantidad de datos necesarios para el procesos receptor correspondiente. Para el caso de la matriz de tamaño 4 x 4, cada proceso se encarga de calcular un valor, por lo que el envío de datos siempre es de una unidad.

Una vez que se ha descompuesto el problema en módulos que puedan ser ejecutados de forma paralela, y que la estructura de comunicación para el intercambio de datos se ha modelado como una gráfica de software (como lo muestra la figura 5.12), ahora es necesario aplicar el algoritmo de aglomeración para obtener las particiones que se mapean sobre el sistema distribuido.

Si se suman las unidades de comunicación entre pares de procesos del sistema paralelo que muestra la figura 5.12, se obtiene que la cantidad de comunicaciones totales CT es:

$$CT = 29 \; unidades.$$

considerando tanto la distribución de datos iniciales como la redistribución de datos para el cómputo del resultado final.

5.2.4. Aglomeración de procesos

Para la red de flujo que expresa la estructura del sistema paralelo de este caso de estudio (figura 5.12), se aplica el Algoritmo 3 Ford-Fulkerson-Extendido$(G, 1, 10)$ para determinar las agrupaciones de procesos que se mapean sobre un sistema distribuido de 4 nodos. Por lo tanto, se desean obtener 4 aglomeraciones de procesos.

El primer corte que obtiene el Algoritmo 3 Ford-Fulkerson-Extendido$(G, 1, 10)$ sobre la red de flujo de la figura 5.12 se conforma por las aristas $mincut(G) = \{(1,10), (7,10), (8,10), (9,10)\}$. Por lo tanto, la primer partición resultante (llámese A) está constituida sólo por el proceso $A = \{10\}$. La figura 5.13 ilustra la primer partición obtenida.

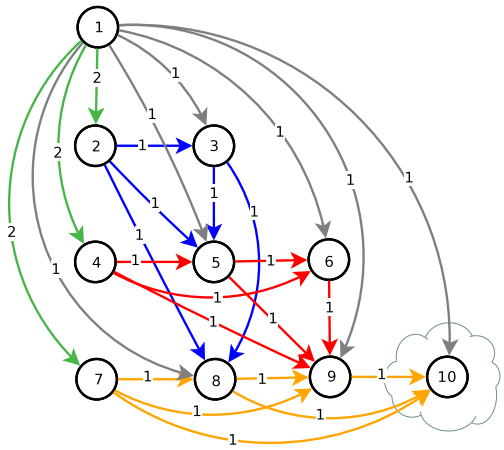

Figura 5.13: Gráfica de software para una descomposición por celda de la matriz de 4x4. La nube delimita la primera partición que se obtiene al ejecutar el Algoritmo 3 Ford-Fulkerson-Extendido(G, s, t).

Dado que el proceso 10 pertenece ya a una aglomeración A, es necesario dejar de considerarlo en la gráfica, y por lo tanto, debe buscarse otro vértice que funcione como un vértice sumidero. El proceso 9 cumple ahora con las condiciones para ser el nuevo vértice especial t. La segunda iteración del algoritmo 3 sobre la red de flujo de la figura 5.12 obtiene un corte mínimo de valor 6, el cual está conformado por las aristas $mincut = \{(1,9), (4,9), (5,9), (6,9), (7,9), (8,9)\}$. Por lo tanto, la segunda aglomeración (llámese B) se conforma por el proceso $B = \{9\}$. La figura 5.14 muestra la segunda iteración del Algoritmo 3 sobre la gráfica del sistema paralelo. La nube delimita la segunda aglomeración obtenida, el corte mínimo es ahora de valor 6.

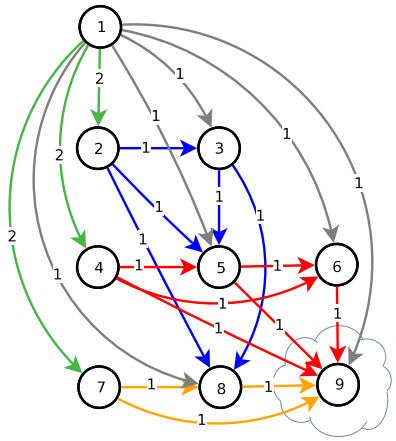

Figura 5.14: Segunda iteración del algoritmo 3 sobre la gráfica de software que muestra la figura 5.14. La nube delimita la segunda partición que se obtiene.

El vértice sumidero de nuevo es particionado en una aglomeración B, y por lo tanto, deja de ser considerado para futuras iteraciones. Ahora se puede observar en la figura 5.15 que tanto el vértice 8 como el vértice 6 pueden funcionar como nuevos vértices sumideros. Dado que no pueden existir dos vértices sumideros en una red de flujo es necesario crear un vértice sumidero t, el cual recibe una arista por cada vértice final. En este caso, los vértices 8 y 6 generan una arista (por cada uno). Por lo tanto la capacidad de la arista que va del vértice 6 al vertice t es igual o mayor a la suma de las capacidades de las aristas que inciden en 6. De la misma forma se calcula la capacidad de la arista que va de 8 a t. La figura 5.15 muestra las aristas mencionadas.

La figura 5.15 muestra la tercer iteración sobre la gráfica del sistema paralelo. El Algoritmo 3 Ford-Filkerson encuentra el corte mínimo de valor 7, el cual esta conformado por las aristas $mincut = \{(1,6), (3,8), (5,6), (4,6), (2,8), (1,8), (7,8)\}$. Dicho corte genera la tercer aglomeración (llámese C) la cual contiene a los vértices $C = \{6, 8, t\}$. Note que el vértice agregado t no representa a ningún proceso ni comunicación, y solo es empleado para encontrar el corte mínimo en la gráfica.

Esta ultima iteración también genera la cuarta aglomeración (llámese D) deseada, la cual está compuesta por los procesos $D = \{1, 2, 3, 4, 5, 7\}$ (figura 5.16).

La aglomeración de procesos se ha completado obteniendo el número de particiones deseadas, y están compuestas de acuerdo a la siguiente lista:

- $A = \{10\}$

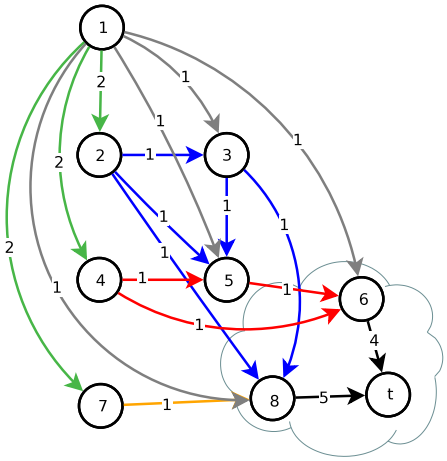

Figura 5.15: Tercera iteración del algoritmo 3 sobre la gráfica de software que muestra la figura 5.6. La nube delimita la tercer partición que se obtiene.

- $B = \{9\}$
- $C = \{6, 8\}$
- $D = \{1, 2, 3, 4, 5, 7\}$

Considerando que la ejecución de este ejemplo se hace sobre un sistema distribuido de 4 nodos[1], el mapeo sobre éste es trivial (uno a uno), lo cual hace que las comunicaciones entre pares de procesos en distintas aglomeraciones sean comunicaciones que forzosamente pasen sobre el medio de red que interconecta al sistema distribuido. La figura 5.16 muestra una visión general de las particiones obtenidas mediante la estrategia de aglomeración basada en propiedades de corte y flujo, que se mapean uno a uno sobre el sistema distribuido.

Como se menciona en el planteamiento del problema (Capitulo 1) el tiempo de comunicación entre dos procesos por memoria compartida toma tiempo constante, denotado como t_{cons}, mientras que el tiempo de comunicación sobre un medio de red puede ser variable, y se denota como t_{var}, tal que $t_{cons} << t_{var}$.

Si se calcula el tiempo de comunicación del sistema paralelo que lleva a cabo la factorización de Cholesky, asumiendo la aglomeración de procesos que muestra la figura 5.16 se tiene que:

[1]Cluster que fue empleado para el caso de estudio 1, solo que incorporando un nodo más.

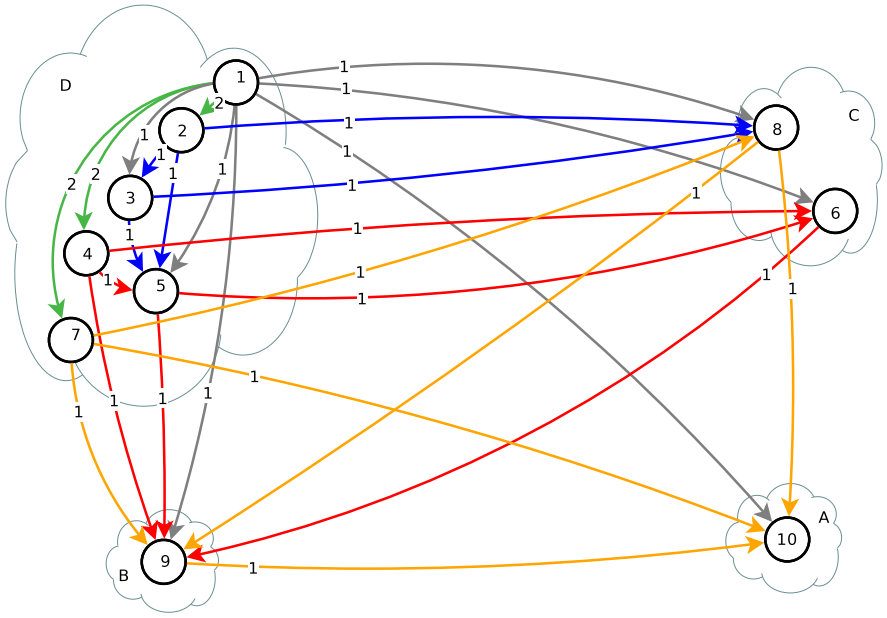

Figura 5.16: Agrupación de procesos e interconexión de aglomeraciones usando la estrategia basada en corte y flujo.

Sea Dt_{out} el tiempo total de comunicaciónes exteriores de la aglomeración D, es decir, que van de un proceso $p_i \in D$ a un proceso $p_j \notin D$. Entonces D_{out}, se obtiene al sumar las capacidades de las aristas (1,8), (1,6), (1,9), (1,10), (2,8), (3,8), (4,6), (4,9), (5,6), (5,9), (7,8), (7,9) y (7,10). Sea Dt_{in} el tiempo total de comunicaciones internas de la aglomeración D, es decir, que van de un proceso $p_i \in D$ a un proceso $p_j \in D$. Entonces D_{in}, se obtiene de la suma de las capacidades de las aristas (1,2), (1,3), (1,4), (1,5), (1,7), (2,3), (2,5), (3,5) y (4,5).

Por lo tanto se tiene que para la aglomeración D:

$$Dt_{out} = 13\ t_{var},\ Dt_{in} = 12\ t_{cons}$$

De la misma forma se calculan el total de comunicaciones exteriores e interiores para el resto de las aglomeraciones.

$$Ct_{out} = 3\ t_{var},\ Ct_{in} = 0\ t_{cons}$$

$$Bt_{out} = 1\ t_{var},\ Bt_{in} = 0\ t_{cons}$$

$$At_{out} = 0\ t_{var},\ At_{in} = 0\ t_{cons}$$

La suma de los tiempos de comunicación de todas las aglomeraciones forman el tiempo total de comunicaciones del sistema paralelo, aglomerado bajo la estrategia basada en redes de flujo, denotado como TT_{nf}.

$$TT_{nf} = 17\ t_{var} + 12\ t_{cons}$$

5.2.5. Mapeo por Round Robbin

De la misma forma como se llevó a cabo en el caso de estudio 1, y por motivos de establecer un punto de referencia, durante la presente sección se procede a realizar un mapeo del sistema paralelo que presenta el caso de estudio 2, por medio de la estrategia Round Robbin sobre el mismo sistema distribuido de 4 nodos.

De acuerdo con el mapeo por Round Robbin, el proceso 1 se asigna a la aglomeración A, el proceso 2 se asigna a la aglomeración B, el proceso 3 se asigna a la aglomeración C, el proceso 4 se asigna a la aglomeración D, y circularmente el siguiente procesos se asigna de nuevo a la aglomeración A, consecutivamente se procede de la misma forma hasta que los procesos sean repartidos por completo. La figura 5.17 muestra una gráfica particionada en cuatro aglomeraciones que representan cada uno de los nodos del sistema distribuido en el cual se desean mapear.

Realizando de la misma forma el cálculo de los tiempo de comunicación como se hizo con anterioridad, se tiene que:

$$At_{out} = 12\ t_{var},\ At_{in} = 3\ t_{cons}$$

$$Bt_{out} = 4\ t_{var},\ Bt_{in} = 0\ t_{cons}$$

$$Ct_{out} = 5\ t_{var},\ Ct_{in} = 0\ t_{cons}$$

$$Dt_{out} = 5\ t_{var},\ Dt_{in} = 0\ t_{cons}$$

De lo anterior se puede obtener la suma total de comunicaciones del sistema paralelo que se mapea bajo el método de Round Robbin, se obtiene:

$$TT_{rr} = 26\ t_{var} + 3\ t_{cons}$$

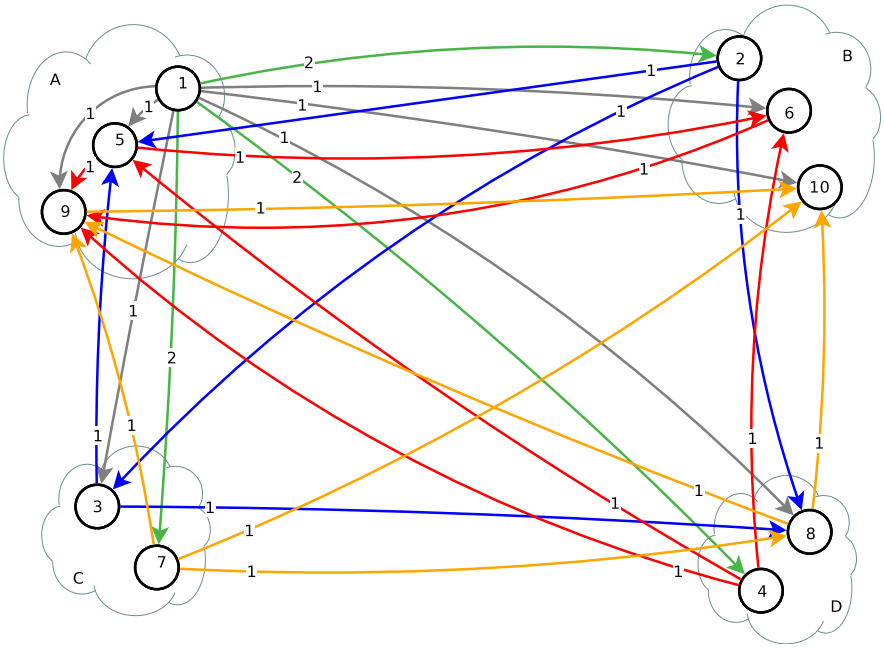

Figura 5.17: Interconexión de aglomeraciones obtenidas mediante Round Robbin.

5.2.6. Comparación

Note que para ambos tiempos de comunicación TT_{rr} y TT_{nf}, sus tiempos variables t_{var} y tiempos constantes t_{cons} respectivamente, suman la misma cantidad de comunicaciones totales CT, igual a 29 unidades, como se obtiene en la etapa de descomposición y comunicación. Si se consideran ambos tiempos de comunicación del sistema paralelo mapeado por ambas técnicas se tiene que, bajo la estrategia basada en propiedades de corte y flujo, el tiempo total de comunicaciones TT_{nf} es de $17_{t_{var}} + 12_{t_{cons}}$, mientras que para el mapeo por Round Robbin se obtiene un TT_{rr} de $26_{t_{var}} + 3_{t_{cons}}$.

Puede ver que de nuevo la estrategia de aglomeración propuesta logra maximizar las comunicaciones locales con 12 unidades, contra las 3 unidades de comunicación obtenidas por medio de Round Robbin. Así mismo, la estrategia de aglomeración logra una minimización de comunicaciones exteriores de 17 unidades, contra las 26 unidades que entrega el mapeo por Round Robbin, logrando reducir dichas comunicaciones a un 65 %.

Teóricamente, puede observarse un mejor desempeño (con respecto a Round Robbin) en comunicaciones bajo la estrategia de aglomeración propuesta, ya que se consigue tanto la minimización de comunicaciones exteriores como la maximización de comuni-

No Ejec	T total (RR)	T total (NF)
1	2159 ms	2364 ms
2	2276 ms	2331 ms
3	2327 ms	2405 ms
4	2198 ms	2409 ms
5	2230 ms	2370 ms
6	2314 ms	2417 ms
7	2086 ms	2323 ms
8	2310 ms	2403 ms
9	2119 ms	2408 ms
10	2249 ms	2377 ms
T Prom	2,226.8 ms	2,380.7 ms

Cuadro 5.7: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 4 x 4.

caciones interiores, lo cual es preferible considerando la ejecución del sistema paralelo bajo un ambiente de memoria distribuida. Contrario al caso de estudio 1, puede observar que las aglomeraciones que obtiene la estrategia basada en corte y flujo presentan un notorio desbalance de carga.

A continuación se presenta una serie de resultados, que muestran los datos experimentales obtenidos por una conjunto de ejecuciones del sistema paralelo que calcula la descomposición de Cholesky. El sistema paralelo cumple que no varía con respecto al tamaño de su entrada, y se mantiene el mismo conjunto de datos para cada ejecución bajo ambos esquemas de mapeo.

La tabla 5.7 muestra la primera de las experimentaciones del sistema paralelo. En promedio, el tiempo de ejecución del sistema para computar la factorización de Cholesky para una matriz de 4 x 4, que se mapea con Round Robbin es de 2,226.8 ms, mientras que bajo la estrategia de aglomeración basada en corte y flujo se obtiene un tiempo promedio de 2,380.7 ms. La diferencia entre ambos tiempo es de 153.9 ms, donde se puede observar que Round Robbin reporta un mejor tiempo promedio de ejecución, mejorando en un 6.9 % del tiempo de cómputo con respecto a la estrategia propuesta.

La segunda experimentación considera una entrada al sistema paralelo de una matriz de 16 x 16. La tabla 5.8 muestra los tiempos de 10 ejecuciones, obteniendo un tiempo

No Ejec	T total (RR)	T total (NF)
1	2269 ms	2373 ms
2	2308 ms	2363 ms
3	2310 ms	2382 ms
4	2231 ms	2415 ms
5	2275 ms	2373 ms
6	2276 ms	2378 ms
7	2244 ms	2400 ms
8	2321 ms	2368 ms
9	2242 ms	2413 ms
10	2243 ms	2407 ms
T Prom	2,271.9 ms	2,387.2 ms

Cuadro 5.8: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 16 x 16.

promedio para el mapeo por Round Robbin de 2,271.9 ms. Por otro lado, el tiempo promedio que se reporta con el mapeo por redes de flujo es de 2,387.2 ms. La diferencia entre ambos tiempos promedios es de 115.3 ms, lo cual representa un 5.08 % de ventaja para el mapeo por Round Robbin.

Los datos de la tercer experimentación se muestran en la tabla 5.9, los cuales se obtienen de 10 ejecuciones del sistema paralelo para una matriz de 64 x 64. Bajo el mapeo por Round Robbin se reporta un tiempo promedio de 2,258.2 ms, contra el tiempo promedio que se obtiene mediante la estrategia de aglomeración propuesta de 2,399.2 ms. De nuevo puede notarse una diferencia de 141 ms a favor de Round Robbin mejorando en un 6.24 % del tiempo de cómputo.

Los datos correspondientes a la experimentación número cuatro se muestran en la tabla 5.10, con una matriz de entrada de 512 x 512 elementos. El tiempo promedio obtenido bajo el mapeo por Round Robbin es de 2,592.8 ms, mientras que para la estrategia de aglomeración por redes de flujo es de 2,677.5 ms, dando una diferencia favorable para Round Robbin de 84.7 ms, lo cual representa el 3.2 % de tiempo de cómputo.

La tabla 5.11 muestra los datos obtenidos por la quinta experimentación, la cual conciste en calcular la descomposición de Cholesky de una matriz de 2048 x 2048, en la

No Ejec	T total (RR)	T total (NF)
1	2306 ms	2418 ms
2	2329 ms	2424 ms
3	2181 ms	2349 ms
4	2287 ms	2344 ms
5	2294 ms	2411 ms
6	2056 ms	2429 ms
7	2214 ms	2394 ms
8	2330 ms	2422 ms
9	2223 ms	2368 ms
10	2362 ms	2433 ms
T Prom	2,258.2 ms	2,399.2 ms

Cuadro 5.9: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 64 x 64.

No Ejec	T total (RR)	T total (NF)
1	2606 ms	2577 ms
2	2640 ms	2606 ms
3	2555 ms	2687 ms
4	2635 ms	2691 ms
5	2569 ms	2696 ms
6	2602 ms	2694 ms
7	2555 ms	2709 ms
8	2633 ms	2692 ms
9	2582 ms	2714 ms
10	2551 ms	2709 ms
T Prom	2,592.8 ms	2,677.5 ms

Cuadro 5.10: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 512 x 512.

No Ejec	T total (RR)	T total (NF)
1	5151 ms	5558 ms
2	4882 ms	5504 ms
3	4880 ms	5439 ms
4	4896 ms	5559 ms
5	4977 ms	5520 ms
6	4939 ms	5560 ms
7	5139 ms	5524 ms
8	5209 ms	5291 ms
9	5097 ms	5409 ms
10	5217 ms	5563 ms
T Prom	5,038.7 ms	5,492.7 ms

Cuadro 5.11: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 2048 x 2048.

cual se obtiene un tiempo promedio de 5,038.7 ms bajo el mapeo por Round Robbin, mientras que el tiempo promedio por redes de flujo es de 5,492.7 ms, con lo que se tiene una diferencia de 454 ms a favor del mapeo por Round Robbin, mejorando en un 9 % del tiempo de cómputo total.

Los datos de la penúltima experimentación, la cual considera una entrada al sistema paralelo de una matriz de 8,192 x 8,192 elementos, se muestran en la tabla 5.12. Si se observan los tiempos promedios de ejecución de ambas estrategias de mapeo, puede notarse que para Round Robbin se obtiene un tiempo promedio de 104,705 ms. Por otra parte, el tiempo promedio que se obtiene para las ejecuciones con la estrategia propuesta es de 103,833.4 ms, dando una diferencia de 871.6 ms, los cuales representan el 0.83 % del tiempo de cómputo total, favorable para la estrategia basada en redes de flujo.

Finalmente la tabla 5.13 muestra los datos de 5 ejecuciones del sistema paralelo con una entrada de una matriz de tamaño 16,384 x 16,384 elementos. El tiempo promedio que se obtiene de las ejecuciones del sistema que se mapea por Round Robbin es de 779,318 ms, mientras que el tiempo promedio para las ejecuciones del sistema que se mapea por la estrategia propuesta es de 761,796.4 ms, con una diferencia de 17,521.6

No Ejec	T total (RR)	T total (NF)
1	104591 ms	103690 ms
2	104220 ms	102791 ms
3	104072 ms	105571 ms
4	105336 ms	104590 ms
5	107664 ms	105441 ms
6	104336 ms	103415 ms
7	104316 ms	102 649 ms
8	104728 ms	102210 ms
9	104461 ms	102844 ms
10	103326 ms	105133 ms
T Prom	104,705 ms	103,833.4 ms

Cuadro 5.12: 10 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 8192 x 8192.

ms favorable para la estrategia propuesta, los cuales corresponden a 2.25 % del cómputo total.

Considerando los datos obtenidos por la serie de tablas anteriores, puede notarse que la ventaja que muestra la técnica de Round Robbin se mantiene durante las experimentaciones iniciales, hasta el punto donde las comunicaciones empiezan a ser de un tamaño considerable, como lo muestra la quinta y sexta experimentación. Puede darse cuenta que aún cuando se observa una ventaja de hasta el 2.2 % para la estrategia basada en redes de flujo, ésta no resulta ser tan significativa como sucede en el caso de estudio 1.

No Ejec	T total (RR)	T total (NF)
1	783203 ms	758255 ms
2	790703 ms	755287 ms
3	786728 ms	775306 ms
4	757062 ms	761095 ms
5	778894 ms	759039 ms
T Prom	779,318 ms	761796.4 ms

Cuadro 5.13: 5 ejecuciones del sistema paralelo para la descomposición de Cholesky para una matriz de 16384 x 16384.

6

Análisis y Conclusiones

A lo largo de éste último capítulo se describen varias ventajas, desventajas y limitaciones, que surgen a raíz de la implementación de la estrategia de aglomeración basada en propiedades de corte y flujo, las cuales deben ser consideradas para la aplicación de la misma. Por último se anexa una sección de conclusiones finales y una sección donde se plantea el trabajo a futuro.

6.1. Resumen del Trabajo

En el presente trabajo de tesis, se aborda el problema de aglomeración y mapeo, se propone una estrategia de aglomeración para el mapeo de un sistema paralelo en un ambiente de memoria distribuida.

En dicha estrategia, se representa al sistema paralelo como una gráfica, la cual ayuda a expresar la estructura y organización del mismo sistema. Esta gráfica puede transformase de manera sencilla, solo identificando una serie de vértices a los cuales se conectan dos vértices especiales, en una red de flujo. La cual, adicionalmente, permite modelar la cantidad de comunicaciones entre cada par de procesos del sistema.

En una red de flujo, puede aplicarse el algoritmo de Ford-Fulkerson Extendido. El cual, consigue un k-particiónamiento de la gráfica, donde k se considera igual al númuero de núcleos de procesamiento en el sistema distribuido.

Finalmente, las k aglomeraciones de procesos, son asignadas de manera directa, es decir, de uno a uno, sobre cada unidad de procesamiento de la máquina paralela para llevar a cabo la ejecución del sistema paralelo.

6.2. Reenunciado de la Hipótesis

Dado un conjunto de procesos, donde hay comunicación entre algunos pares de ellos. Entonces, existe una forma de aglomerar dicho conjunto, considerando como principal criterio de aglomeración, la reducción de comunicaciones remotas, y el incremento de comunicaciones locales. Tal que se mitiga el tiempo agregado por comunicaciones, al tiempo global de ejecución del sistema paralelo.

6.3. Ventajas

La representación de procesos y comunicaciones por medio de una gráfica, ayuda a modelar la estructura de un sistema paralelo, en [2] y [15] hacen uso de dicha representación, para luego abordar el problema de mapeo baje diferentes perspectivas.

En [2], Bokhari muestra que existe una equivalencia entre el problema de mapeo y el de la gráfica isomorfa. Tal que resolver el problema de mapeo, se traduce en, encontrar un isomorfismo de dos gráficas, las cuales respectivamente representan la estructura del sistema paralelo y la arquitectura de la maquina paralela.

Bokhari propone en [2] un algoritmo heurístico, que consiste en intercambios de parejas, con saltos probabilísticos. Para llevar a cabo el mapeo del sistema paralelo, se describe una máquina paralela conectada con un patrón de 8 vecinos. Por lo que, el problema de mapeo se reduce a una arquitectura paralela en particular.

En [15], Ortega y Benítez, proponen un estrategia para el mapeo de sistemas paralelos. La estrategia propuesta puede asegurar uno de dos objetivos, ya sea que, la prioridad de los requerimientos sea, minimizar la comunicación o mantener un balance de carga estable para la ejecución del sistema.

La estrategia sugerida por Ortega y Benítez en [15], se caracteriza principalmente por ser una estrategia distribuida, la cual toma decisiones en base al entorno local del procesador, construye una topología por procesador lo cual permite conocer la probable ubicación de procesos alojados en cualquier procesador de la red, y migrar procesos por la máquina paralela, de acuerdo a los requerimientos de balance de carga o minimzación de comunicaciones.

La migración de procesos, puede conllevar consecuencias desagradables en un sistema paralelo, ya que al mover un proceso a otro procesos, se debe también mover todos los datos del entorno del proceso.

La estrategia que se propone en este trabajo, no considera la migración de los procesos, ni se fija para una arquitectura paralela en particular. Además tiene como criterio principal la reducción de las comunicaciones externas, es decir, las comunicaciones que hacen uso de un medio de red para intercambiar información de un procesador a otro, dentro de un sistema distribuido. Este tipo de comunicaciones en general es más costosa, e impacta directamente en el desempeño de un sistema paralelo.

La estrategia de aglomeración basada en propiedades de corte y flujo, logra una maximización de comunicaciones por memoria compartida, y una minimización de comunicaciones por paso de mensajes. Con lo cual, la estrategia de aglomeración, consigue un impacto positivo en el desempeño de un sistema paralelo.

6.4. Desventajas y Limitaciones

6.4.1. No determinismo de comunicaciones

Una parte importante del Teorema de Corte Mínimo Flujo Máximo, es la capacidad de las aristas, ya que ésta representa una cota superior de flujo permitido en cada una de ellas. La estrategia de aglomeración se basa en esta propiedad, por lo que, la precisión con que las comunicaciones del sistema paralelo sean representadas en la red de flujo influye directamente en la eficiencia de la estrategia de aglomeración. Por lo tanto si la naturaleza del sistema paralelo impide conocer de manera precisa la capacidad de al menos una arista, entonces ésto impactaría de forma negativa en el desempeño de la estrategia.

Es posible usar un valor promedio o estimado sobre las capacidades de las aristas que no pueden ser determinadas, pero debido a que la gráfica no expresa de manera leal las cantidades reales de comunicación del sistema paralelo, ésto afectaría la precisión con que el algoritmo encuentra las particiones.

6.4.2. No garantía sobre balance de carga

La estrategia de aglomeración principalmente considera la cantidad de comunicaciones como el único parámetro de particionamiento, es decir, no garantiza que las particiones obtenidas se encuentren balanceadas.

Puede considerar los dos casos de estudio que se presentan en el capítulo 5. Se observa que, la aglomeración del primer sistema paralelo (ordenamiento $MergeSort$),

entrega un balance de carga cercano al de Round Robbin (el cual además es óptimo). Mientras que para el caso de estudio 2 (descomposición de Cholesky), el balance de carga que entrega la estrategia propuesta esta muy alejado al óptimo. Esto se debe a que el balance de carga no se considera como un objetivo de la estrategia propuesta, mas aún éste puede ser incluido en trabajo a futuro.

6.4.3. No garantía de optimalidad en las particiones

Aún cuando el Teorema de Corte Mínimo Flujo Máximo demuestra que para toda red de flujo existe un corte mínimo y por lo tanto una bipartición óptima, para una k-partición de un red de flujo no se garantiza que ésta sea óptima. Por lo tanto, el par de algoritmos que se proponen en el capítulo 4 para obtener una k-partción de la gráfica del sistema paralelo, sólo ofrecen una aproximación a una k-partición óptima.

Al no conseguir una k-partición óptima, no se puede garantizar que las comunicaciones exteriores de las aglomeraciones que obtiene la estrategia propuesta, sean mínimas.

6.5. Interpretación y Análisis de Resultados

Considere los dos casos de estudio que se presentan en el capítulo 5. Teóricamente, puede darse cuenta que al determinar las aglomeraciones de un sistema paralelo por medio de la estrategia propuesta, se observa una reducción en la cantidad de comunicaciones entre procesos de aglomeraciones diferentes (comparado con Round Robbin), dichas comunicaciones se consideran mas costosas, por lo que impactan de manera negativa en el desempeño del sistema paralelo.

En el caso de estudio del ordenamiento por $MergeSort$, se logra una mejora considerable en el tiempo de ejecución de hasta un 7.6 %. Lo cual sugiere de manera práctica, que la mejora en tiempo de ejecución (comparada con Round Robbin) incrementa de manera proporcional cuando crece el tamaño de los datos de entrada.

El segundo caso de estudio, muestra una mejora en el tiempo de ejecución de menor impacto. Cabe mencionar que para este caso de estudio, Round Robbin mantiene la ventaja en tiempo de ejecución durante las primeras cinco experimentaciones, la cual llega a ser de hasta un 9 %. Posteriormente, en las últimas dos experimentaciones, se logra apreciar una ventaja menor de hasta un 2.2 % para la estrategia propuesta.

Para ambos casos de estudio, se logra observar que a partir de un punto la ventaja que logra el mapeo con la estrategia propuesta, se incrementa.

6.6. Conclusiones

La comunicación entre dos procesos paralelos por memoria compartida, en general, es mas rápida o menos costosa que una comunicación por paso de mensajes. Por lo tanto, es preferible maximizar las comunicaciones locales y minimizar las comunicaciones remotas en un sistema paralelo, para poder mitigar los retados que las comunicaciones remotas inherentemente agregan al tiempo de ejecución del sistema paralelo.

A mayor cantidad de comunicaciones remotas, se logra un impacto proporcional negativo en el desempeño global de sistema paralelo. Por lo que, al conseguir el objetivo de minimización/maximización, también se logra afectar de manera positiva el desempeño global del sistema, el cual se ve reflejado en un menor tiempo de ejecución. Puede notarse que dicho objetivo se consigue teóricamente para ambos casos de estudio, más aún, puede apreciarse en los tiempos de ejecución de las experimentaciones, que el primer caso presenta un mayor impacto positivo, a diferencia del segundo caso de estudio.

Nótese que, el sistema paralelo del primer caso de estudio que se mapea con la estrategia propuesta, logra un balance de carga cercano al óptimo (el cual se obtiene por medio de Round Robbin). Mientras que para el sistema paralelo del segundo caso de estudio que se mapea por medio de la misma estrategia, logra un balance de carga ineficiente lejano al que se consigue por medio de Round Robbin (el cual es óptimo). Por lo tanto puede considerarse que, el mapeo que se logra para el segundo caso de estudio impacta de forma negativa en la eficiencia del mismo, tal que el impacto positivo de la reducción de comunicaciones remotas no logra percibirse de manera considerable.

Por lo que, se podría aventurar a suponer que:

> *"El tiempo de ejecución óptimo de un programa paralelo en una arquitectura de memoria distribuida, se obtiene al garantizar tanto el balance de carga óptimo, como la reducción óptima de comunicaciones remotas".*

Si bien la estrategia de aglomeración, no logra la reducción óptima de las comunicaciones remotas, y por lo tanto, no consigue agregar el tiempo de comunicaciones mínimo

al tiempo de ejecución del sistema, la estrategia si logra tener un impacto positivo en el desempeño de sistema (comparándolo con Round Robbin), sólo variando el mapeo de procesos.

6.7. Contribuciones

La principal aportación es proveer una estrategia de aglomeración para el mapeo de procesos paralelos, en una arquitectura de hardware de memoria distribuida, que basa sus decisiones de particionamiento en la reducción de comunicaciones remotas. Dicha estrategia se describe detalladamente a lo largo del capítulo 4, mencionando sus características, requerimientos de entrada y detalles del resultado que entrega.

Las experimentaciones de dicha estrategia se presentan en los casos de estudio del capítulo 5. Se muestra que si un sistema paralelo se mapea bajo diferentes criterios, en una arquitectura de memoria distribuida, entonces, éste puede variar su desempeño, y por lo tanto, su tiempo de ejecución.

6.8. Trabajo Futuro

Una extensión del presente trabajo que resulta interesante, es un estudio experimental mas detallado, que ayude a apreciar de mejor forma el comportamiento general de la estrategia de aglomeración.

Debido a que Round Robbin obtiene un balance de carga óptimo, se podría proponer un algoritmo que garantice obtener k-particiones óptimas con comunicación mínima, y evaluar experimentalmente mediante el mapeo de un sistema paralelo, que impacto tiene, un mapeo con comunicación óptima mńima, contra un mapeo con balance de carga óptimo.

Ya que una de las desventajas de la estrategia propuesta es que no garantiza un balance de carga óptimo, resultaría muy interesante extender la estrategia para que considere ahora, tanto un balance de carga, como la reducción de comunicaciones. Y evaluar que tanto se logra mejorar, en los tiempos de ejecución de un sistema paralelo.

7

Referencias

[1] *Design and Building Parallel Programs* v1.3, Ian Foster, Chapter 2 Designing Parallel Algorithms, 1995 An Online Publishing Project.

[2] *On the Mapping Problem*, Shahid H. Bokhari, IEEE Transactions on Computers, Vol. c-30, No. 3, March 1981.

[3] *Algorithm Design*, Jon Kleinberg, Éva Tardos, Chapter 7: Network Flow, 2005 Pearson-Addison Wesley

[4] *An Introduction to Parallel Programming*, K. Mani Chandy, Stephen Taylor, Part II Parallel Program Design Chapter 7, 1992 Jones and Bartlett.

[5] *Parallel Computer Arquitecture, A Hardware/Software Approach*, David Culler, Jaswinder Pal Singh, Anoop Gupta, Chapter 2 Parallel Programs, 1997 Morgan Kaufmann.

[6] *Introduction to Algorithms*, Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Second Ed. (2001), Section 26.2: The Ford-Fulkerson method, MIT Press and McGraw-Hill.

[7] *Highly Parallel Computing*, George S. Almasi, Allan Gottlieb, Chapter 1 Overview, The bejamin/Cummings Publishing, 1989.

[8] *Introduction to Parallel Computing*, Blaise Barney, Lawrence Livermore National Laboratory.

[9] *The Algorithm Design Manual*, Steven S. Skiena, Chapter 4 Sorting and Searching, Second Edition Springer 2008.

[10] *The Art of Computer Programming Vol.* 3, Donald E. Knuths, Sorting and Searching, Chapter 5.2 Internal Sorting, Eddison-Wesley.

[11] *Numerical Recipes in C, The art of Scientific Computing* William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Cambridge University Press, Second Edition, 1992.

[12] *Graph Theory*, J. A. Bondy, U.S.R. Murty, Springer, 2008.

[13] *Computers andIntractability, A Guide of the Theory of NP−Completeness*, Michael R. Garey, David S. Johnson, Bell Telephone Laboratories, 1979.

[14] *Balanced Graph Partitioning*, Konstantin Andreev, Harald Racke.

[15] *An Efficient Mapping Strategy for Parallel Programming*, Jorge Luis Ortega Arjona, Héctor Benítez Pérez. 2011.