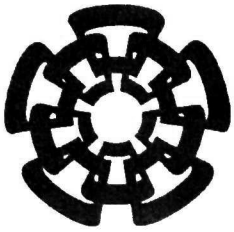


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Transformación del Algoritmo Matricial que Efectúa la Descomposición QR de su Forma Secuencial a su Forma Paralela

Tesis que presenta:

Ricardo Gomez ku

para obtener el grado de:

Maestro en Ciencias

en la especialidad de:

Ingeniería Eléctrica

Directores de Tesis

Dr. Deni Librado Torres Román
Dr. Yuriy Shkvarko Sosnoff

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Diciembre de 2009.

Transformación del Algoritmo Matricial que Efectúa la Descomposición QR de su Forma Secuencial a su Forma Paralela

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Ricardo Gomez ku
Ingeniero en Comunicaciones y Electrónica
Universidad Autónoma de Campeche 2002-2007

Directores de Tesis
Dr. Deni Librado Torres Román
Dr. Yuriy Shkvarko Sosnoff

CINVESTAV del IPN Unidad Guadalajara, Diciembre de 2009.

Agradecimientos

Antes que nada agradezco a Dios con todas mis fuerzas y humildad por darme la vida y abrigarme con seres humanos que me demuestran cariño, los cuales exigen lo mejor de mi día con día; me refiero a mis padres, hermano, camaradas, amigos y conocidos. Es bueno saber que se encuentran ahí para sumar fuerzas, multiplicar las alegrías y dividir las tristezas.

Mi gratitud para con el Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mi estancia en el Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV). Esta institución académica alberga algunos de los mejores investigadores del país de entre los cuales, para mi buena fortuna, puedo contar a algunos de mis profesores a quienes admiro y respeto. En lo particular quiero reconocer a mi asesor el Dr. Deni Torres por la confianza en los momentos decisivos, al Dr. Mariano Aguirre Hernández y al Dr. Ramón Parra por sus sugerencias para mejorar este trabajo, y finalmente al Dr. Manuel Guzmán al cual considero un buen amigo y consejero.

Resumen

Se describe la metodología y las transformaciones aplicadas al algoritmo secuencial que efectúa la descomposición *QR* en base a los grafos de dependencia de datos de cada transformación. Todas las transformaciones aplicadas se reducen a una sola transformación la cual llamamos transformación *espacio-temporal*. Se supone un número no limitado de procesadores, y se presenta una metodología para resolver problemas cuyo número de procesadores virtuales es mayor que el número de procesadores físicos y se describen las transformaciones de *embaldosado* y de *multiproyección* aplicadas al algoritmo secuencial que efectúa la descomposición *QR*. Estas transformaciones se requieren cuando el número de procesadores físicos es limitado. Finalmente se presenta una introducción simple y que forma la base para estudiar e implementar los agendadores más básicos. En particular se presenta pseudocódigo para implementar agendadores con lista de prioridad. La teoría presentada es una plataforma sobre la cual se pueden desarrollar teoría y agendadores para sistemas paralelos más complejos.

Abstract

We describe the methodology and the transformations applied to the sequential algorithm that performs the QR decomposition based on data dependency graphs for each transformation. All the transformations applied are reduced to a single transformation which we call space-time processing. It is an unlimited number of processors, and presents a methodology for solving problems whose number of virtual processors is greater than the number of physical processors and describes the tiling transformations and sequential multicast algorithm applied to making the QR decomposition. These transformations are required when the number of physical processors is limited. Finally, it presents a simple introduction which forms the basis for studying and implementing the most basic scheduled. In particular, it presents pseudocode for implementing shedule with the priority list. The theory presented is a platform on which to develop theory and schedule for more complex parallel systems.

Índice General

Capítulo 1

Introducción

Carezco de un don especial. Sólo soy apasionadamente curioso.
Albert Einstein (1879-1955)

1.1 Introducción

Durante las décadas pasadas, grandes avances tecnológicos han sido alcanzados en el área de los circuitos de gran escala de integración VLSI (*Very Large Scale Integration*). Esta tecnología avanza dando paso a una nueva manera de computación, basada en un alto cómputo paralelo para sistemas de aplicación específica. Una interesante aproximación son los denominados arreglos de procesadores, propuestos originalmente por Kung [16], a finales de la década de los setentas. Un arreglo de procesadores es un dispositivo de cómputo paralelo para una aplicación específica, constituido de un gran número de elementos procesadores simples, interconectados de forma regular y con comunicación local de datos. De manera similar como la sangre circula en el cuerpo humano, los *datos* circulan dentro de los elementos procesadores del arreglo de procesadores, interactuando con otros datos. Los resultados obtenidos en los elementos

procesadores son requeridos por otros elementos procesadores incrementando el cómputo [33]. La regularidad y simplicidad del diseño físico de los elementos de cómputo constituyen características deseables para su implementación directa en silicio, en la forma de circuitos *VLSI*. Excelentes referencias acerca de algoritmos mapeados en un arreglo de procesadores se encuentran en la bibliografía.

Los arreglos de procesadores implementados en un circuito integrado de silicio son típicamente arreglos lineales o bidimensionales de elementos procesadores. Los primeros ejemplos de diseños de arreglos de procesadores fueron concebidos para un propósito específico requiriendo enorme cantidad de creatividad e inspiración de sus inventores. Recientemente un notable interés se incitó en las ciencias computacionales con respecto al desarrollo de métodos formales para sintetizar y generar algoritmos sistólicos. Tales esfuerzos dieron lugar a muchos trabajos en esta área, véase la referencia de Song, Siang, W., en la bibliografía. De cierta forma la mayoría de los métodos propuestos varían poco, difieren ligeramente en el grado de formalismo y la forma de aproximar el problema. En última instancia, todos ellos usan el concepto de *transformación de las dependencias*.

El trabajo realizado en esta tesis presenta las principales ideas involucradas en los métodos de transformación de un algoritmo regular iterativo secuencial a su forma paralela haciendo uso de mapeo lineal y técnicas de proyección. Se describe el método de la ***transformación espacio temporal*** sobre las dependencias de datos mediante representaciones geométricas basados en los grafos de dependencias de datos. Los resultados obtenidos son de vital importancia para explotar el paralelismo en bucles perfectamente anidados en un algoritmo [27].

1.2 Motivación

Las arquitecturas de arreglo de procesadores representan una red de elementos procesadores que procesan datos a través de un sistema. Se requiere que un arreglo de elementos procesadores posea las siguientes características:

i. *sincronismo*. Los datos son procesados por elementos procesadores y viajan a través del arreglo de procesadores accionados por un mismo reloj global.
ii. *modularidad y regularidad.* El arreglo consiste de módulos de procesamiento de datos con interconexión homogénea. Además el arreglo de procesadores puede ser extendido indefinidamente.
iii. *localidad espacial y localidad temporal.* El arreglo de procesadores posee comunicación local espacial la cual se logra con una estructura de interconexión que conectar procesadores cercanos. Hay al menos una unidad de retardo asignada para que la transacción de una señal de un elemento procesador hacia otro pueda ser completada logrando así localidad temporal.

Los esquemas de detección *MIMO* usados en los *transceptores* de los estándares 802.11n y 802.16e se pueden basar en los algoritmos lineales de baja complejidad y eficiencia como el *Zero-Forcing* (***ZF***) o *Minimun Mean Square Error* (***MMSE***). El problema computacional principal para los algoritmos lineales *ZF* y *MMSE* es el cálculo de la *matriz inversa*. Este mismo problema se presenta en otros sistemas más sofisticados de detección. Entre los métodos para el cómputo de una *matriz inversa* se encuentran: los métodos iterativos y los métodos directos. Los métodos iterativos involucran alta complejidad secuencial en los cómputos y no son adecuados para un alto desempeño en implementación [21]. Entre los métodos directos están la *eliminación Gaussiana*, la *descomposición de Cholesky*, la *descomposición LU* [15] y la *descomposición QR*. Estos métodos computan la solución en un número finito de pasos.

En el esquema de *igualación de canal* usado en el *estándar* de *60Ghz, Single Carrier System* (*IEEE* 802.15.3c) se requieren usar algoritmos para *ecualización*. De nueva cuenta el problema computacional principal es encontrar una matriz inversa [28].

Dado la importancia de la factorización matricial *QR* en la implementación de algoritmos de igualación en sistemas de comunicación de muy alta velocidad y en la implementación de algoritmos para detección en sistemas *MIMO*, como manera más efectiva para obtener el hardware del problema de la *matriz inversa* por su estabilidad numérica, seleccionamos el algoritmo de factorización *QR* secuencial para su transformación en un algoritmo paralelo. Dado las aplicaciones que estamos contemplando, es de suma importancia que la factorización *QR* se efectúe en tiempos muy cortos para lo cual se requiere su implementación usando múltiples elementos procesadores.

Un agendador en un sistema paralelo es crucial y es el encargado de indicar en que momento y en que recursos se ejecutan las operaciones o tareas que ejecuta el sistema paralelo. Un agendador está compuesto de un calendarizador y un asignador. El calendarizador se concretiza en un controlador de la circuitería y puede ser implementado de diferentes formas, por ejemplo: distribuido, centralizado, o un híbrido de los dos. La bibliografía sobre la teoría del agendamiento es abundante y llena de heurísticas [32].

1.3 Objetivos

El propósito de la tesis no es proponer un nuevo algoritmo para la descomposición *QR*, sino mostrar como seleccionando un algoritmo secuencial que efectúa la descomposición *QR* podemos transformarlo siguiendo una metodología a un programa de ejecución paralela. Los objetivos generales pueden resumirse de la forma siguiente:

i. modelar un *programa secuencial* como un poliedro el cual tiene embebido un *grafo de dependencia*. El poliedro nos revela el paralelismo existente en el programa secuencial.

ii. exponer la transformación de ***embaldosado***, la cual permite la implementación en hardware de un arreglo de procesadores virtuales con un gran número de éstos con una cantidad menor de procesadores físicos en el arreglo final de procesadores.

iii. explicar la transformación de ***multiproyección*** como una transformación *espacio-temporal*.

iv. presentar aspectos teorícos base para el *estudio* y la *implementación* de ***agendadores*** simples.

1.4 Perspectiva de los Capítulos

Capítulo 1. Presenta la motivación, los objetivos y las contribuciones del trabajo realizado.

Capítulo 2. Describe la metodología y las transformaciones aplicadas al algoritmo secuencial que efectúa la descomposición *QR* en base a los grafos de dependencia de datos de cada transformación. Todas las transformaciones aplicadas se reducen a una sola transformación la cual llamamos transformación *espacio-temporal*. Se supone un número no limitado de procesadores.

Capítulo 3. Se presenta una metodología para resolver problemas cuyo número de procesadores virtuales es mayor que el número de procesadores físicos y se describen las transformaciones de *embaldosado* y de *multiproyección* aplicadas al algoritmo secuencial que efectúa la

descomposición *QR*. Estas transformaciones se requieren cuando el número de procesadores físicos es limitado.

Capítulo 4. Se presenta una introducción simple y que forma la base para estudiar e implementar los agendadores más básicos. En particular se presenta pseudocódigo para implementar agendadores con lista de prioridad. La teoría presentada es una plataforma sobre la cual se pueden desarrollar teoría y agendadores para sistemas paralelos más realistas y complejos.

Capítulo 5. Presenta las conclusiones sobre los resultados obtenidos, hace énfasis en las contribuciones y menciona las áreas de oportunidad identificadas para ampliar el trabajo de investigación iniciado en esta tesis.

1.5 Contribuciones

En el capítulo 2 se presenta una serie de transformaciones del poliedro original al poliedro destino que nos permite obtener una implementación del programa paralelo eficiente. En la literatura no hemos encontrado un desarrollo metódico para implementación del algoritmo *QR* como el que aquí se presenta. Se explican dos desarrollos, en uno de ellos se supone que el número de elementos procesadores físicos es ilimitado y en el capítulo 3 se supone que el número de procesadores físicos es limitado y está determinado por el desempeño que se requiere de la implementación.

Nuestra contribución en el capítulo 4 es de resumir y presentar de una forma clara los resultados más básicos de la teoría del agendamiento, además se presenta la teoría y pseudocódigo para construir agendadores con lista de prioridad bajo las hipótesis antes mencionadas.

En el apéndice B se presenta una breve introducción al tema de grafos que es pertinente para el estudio de dependencias en programas y diseño de agendadores. En este apéndice se pone en perspectiva los agendadores más simples. Se recomienda al lector revisar el apéndice B antes de estudiar el capítulo 4 sobre la teoría del agendamiento.

Capítulo 2

Factorización QR

Matemáticas: el fundamento inconmovible de la ciencia...
Isaac Barrow (1630-1677)

2.1 Introducción

En este capítulo se abordarán las transformaciones implicadas para la implementación en hardware del algoritmo de descomposición QR.

La primera sección presenta una breve introducción a los fundamentos matemáticos, requeridos para el entendimiento de la descomposición QR. Posteriormente se mostrará a detalle las transformaciones, implementadas en lenguaje C, realizadas en el algoritmo original de descomposición QR. Las transformaciones en nuestro código base

comprenden: efectuar normalización, concordancia de índices, eliminación de comunicación global, asignación única concluyendo con la transformación *espacio temporal.*

En la transformación programa destino, se representa el algoritmo inicial, ejecutado *secuencialmente*, a uno modelado de manera *paralela.* Este código está en función de un nuevo espacio de índices, los cuales modelan el programa en función del tiempo y el espacio de procesadores. En base a la información contenida en el programa destino, se obtienen las herramientas necesarias, para la *generación* de las *señales de control*, de la implementación en hardware de la descomposición QR.

2.2 Rotación de Givens

Las matrices de rotación son elementos básicos para construir algoritmos numéricos más complejos que resuelven problemas de matrices. Las matrices de rotación se usan para anular, esto es convertir en cero, a un elemento seleccionado de una matriz. A continuación motivamos la definición de una matriz de rotación [12]. Sean w y w' vectores en el plano, w' se obtiene por la rotación de w en contra las manecillas del reloj un ángulo de β radianes. Ahora encontramos una expresión para la transformación que nos permite encontrar w' a partir de w dado el ángulo de rotación β, la figura 2.1 muestra la situación descrita.

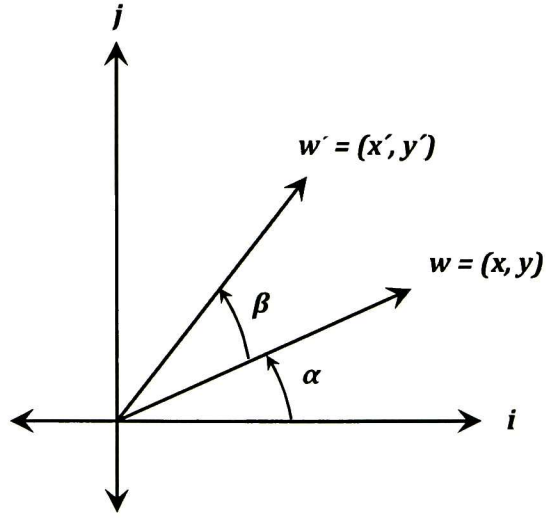


Figura 2.1: Rotación de un vector, en sentido contrario a las manecillas del reloj.

Sea r la longitud del vector, esto es $r=\sqrt{x^2+y^2}$. Por trigonometría se tiene que:

$$
\begin{aligned}
x &= r\cos\alpha, \\
y &= r\ \text{sen}\ \alpha, \\
x' &= r\cos(\alpha+\beta) \qquad \text{y} \\
y' &= r\ \text{sen}\ (\alpha+\beta)
\end{aligned}
$$

Usando identidades trigonométricas tenemos que:

$$
\begin{aligned}
w' &= \begin{bmatrix} r\cos(\alpha+\beta) \\ r\ \text{sen}(\alpha+\beta) \end{bmatrix} \\
w' &= r\begin{bmatrix} \cos\alpha\cos\beta-\text{sen}\alpha\text{sen}\beta \\ \cos\alpha\text{sen}\beta+\text{sen}\alpha\cos\beta \end{bmatrix} \\
w' &= r\begin{bmatrix} \cos\beta & -\text{sen}\beta \\ \text{sen}\beta & \cos\beta \end{bmatrix}\begin{bmatrix} \cos\alpha \\ \text{sen}\alpha \end{bmatrix} \\
w' &= \begin{bmatrix} \cos\beta & -\text{sen}\beta \\ \text{sen}\beta & \cos\beta \end{bmatrix} w
\end{aligned}
$$

La transformación buscada es:

$$
T=\begin{bmatrix} \cos\beta & -\text{sen}\beta \\ \text{sen}\beta & \cos\beta \end{bmatrix} \tag{2.1}
$$

y se llama la matriz de rotación en el plano y es una matriz ortogonal, una matriz A es ortogonal si es cuadrada y tiene columnas ortonormales, $A'A=I$ es decir $A^{-1}=A'$ La figura 2.2 muestra un caso similar a la de la figura 2.1, excepto que ahora el vector w' se ha girado un ángulo de β radianes en el sentido de las manecillas del reloj.

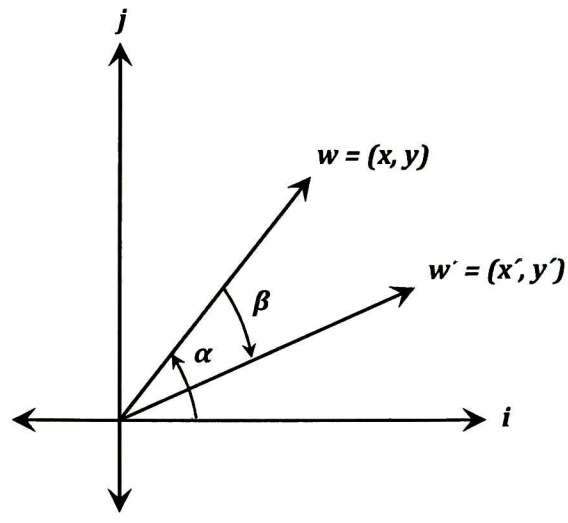


Figura 2.2: Rotación de un vector, en el sentido de las manecillas del reloj.

De un proceso similar al presentado antes se tiene que:

$$w' = \begin{bmatrix} r\cos(\alpha-\beta) \\ r\ \text{sen}(\alpha-\beta) \end{bmatrix}$$

$$w' = r\begin{bmatrix} \cos\beta & \text{sen}\beta \\ -\text{sen}\beta & \cos\beta \end{bmatrix}\begin{bmatrix} \cos\alpha \\ \text{sen}\alpha \end{bmatrix}$$

$$w' = \begin{bmatrix} \cos\beta & \text{sen}\beta \\ -\text{sen}\beta & \cos\beta \end{bmatrix} w$$

w' y w están relacionados por la siguiente matriz de rotación la cual es una matriz ortogonal.

$$T = \begin{bmatrix} \cos\beta & \text{sen}\beta \\ -\text{sen}\beta & \cos\beta \end{bmatrix} \tag{2.2}$$

Para el problema que consideramos más adelante, la selección de la matriz de rotación por la ecuación (2.1) o (2.2) es indistinta, en nuestro desarrollo consideramos la relación (2.2). En la descomposición QR de una matriz, el siguiente problema se resuelve repetidas veces. Sean a y b números reales dados, encontrar $c = \cos\theta$, y $s = \text{sen}\,\theta$ y r tal que:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}=\begin{bmatrix} r \\ 0 \end{bmatrix}=\begin{bmatrix} c\cdot a+s\cdot b \\ -s\cdot a+c\cdot b \end{bmatrix} \tag{2.3}$$

De las relaciones $r = c\cdot a + s\cdot b$, $0 = -s\cdot a + c\cdot b$ y $c^2 + s^2 = 1$ se tiene que:

$$\begin{aligned} r &= \sqrt{a^2+b^2} \\ c &= \frac{a}{r} \\ s &= \frac{b}{r} \end{aligned} \tag{2.4}$$

Donde se supone que $r \neq 0$. La transformación en (2.3), llamada rotación planar, de Givens o de Jacobi, se usa para introducir ceros en los elementos de la matriz real sobre la cual actúa. Los valores de c y s se calculan de tal manera que la relación (2.3) se cumple. Las longitudes de los vectores $[a,\ b]^t$ y $[r,\ 0]$ son iguales. Los cálculos requeridos por la relación (2.4) incluyen división y raíz cuadrada. Los resultados anteriores los podemos resumir en la relación:

$$\begin{bmatrix} \cos\theta & \text{sen}\theta \\ -\text{sen}\theta & \cos\theta \end{bmatrix}\begin{bmatrix} a & x \\ b & y \end{bmatrix}=\frac{1}{\sqrt{a^2+b^2}}\begin{bmatrix} a & b \\ -b & a \end{bmatrix}\begin{bmatrix} a & x \\ b & y \end{bmatrix}=\begin{bmatrix} r & x' \\ 0 & y' \end{bmatrix} \tag{2.5}$$

Donde $\theta = \text{tg}^{-1}\left(\frac{b}{a}\right)$, $r = \sqrt{a^2+b^2}$ y

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x\cdot\cos\theta+y\cdot\text{sen}\theta \\ -x\cdot\text{sen}\theta+y\cdot\cos\theta \end{bmatrix}=\begin{bmatrix} \text{Re}\{(x+jy)e^{-j\theta}\} \\ \text{Im}\{(x+jy)e^{-j\theta}\} \end{bmatrix}=\begin{bmatrix} x' \\ y' \end{bmatrix} \tag{2.6}$$

Dado que $\theta = \text{tg}^{-1}\left(\frac{b}{a}\right)$. Las relaciones (2.5) y (2.6) son útiles para implementar la transformación de Givens en un procesador córdico o en un procesador convencional [20]. Un procesador córdico tiene dos modos de operación: vectorial y rotacional. En modo vectorial se le proporcionan a y b y retorna θ y r, y en modo rotacional si se le proporciona θ y $[a,\ b]$ retorna $[c\cdot a+s\cdot b,\ -s\cdot a+c\cdot b]^t$ donde $c = \cos\theta$ y $s = \text{sen}\theta$. En el córdico en modo vectorial se efectúa una rotación y se calcula un ángulo. En el córdico en modo rotacional sólo se efectúa una rotación. Las relaciones

(2.5) y (2.6) nos ilustran como implementar una rotación de Givens con un procesador córdico y con un procesador convencional. La figura 2.3 muestra los dos modos de operación de un procesador córdico. Tanto en el córdico en modo vectorial como en rotacional se multiplica un vector por una misma matriz de transformación. Observamos que si en el córdico en modo rotacional interpretamos la entrada y la salida como números complejos, en particular $z = x + jy$ y $z' = x' + jy'$, entonces el córdico efectúa la siguiente multiplicación de números complejos: $z' = z \cdot e^{-j\theta}$ El uso de los procesadores córdicos permite implementar la rotación de Givens sin tener que utilizar raíz cuadrada y división. Las operaciones de rotación y cálculo de ángulo de rotación son operaciones primitivas de un procesador córdico y sus tiempos de ejecución son comparables a la de una multiplicación en un procesador convencional.

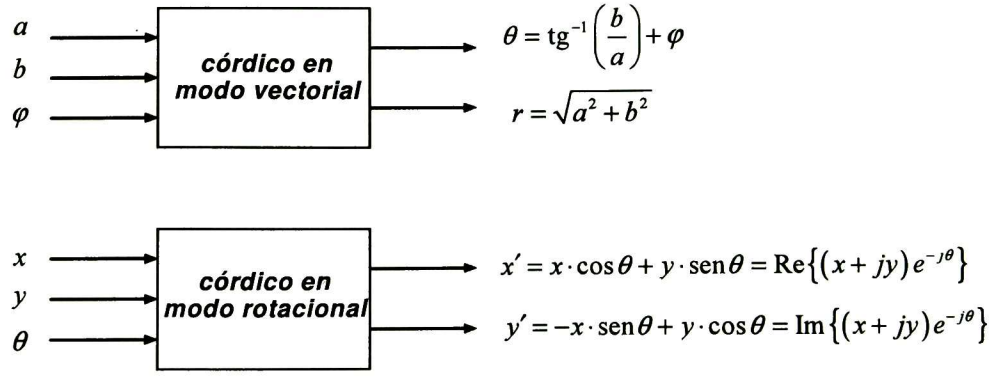


Figura 2.3: Modos de operación de un procesador córdico.

Para simplificar algunas figuras que se presentan más adelante, la figura 2.4 muestra la figura 2.3 para el caso: $f = a + jb$ y $z = x + jy$

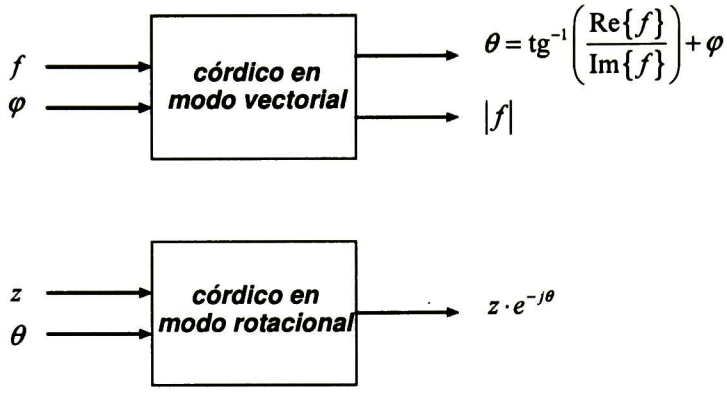


Figura 2.4: Modos de operación de un procesador córdico, cuando la entrada es un número complejo.

Una rotación planar o de Givens, para rotar vectores de n dimensiones, se representa por una matriz de la forma:

$$G(i,k,\theta) = \begin{array}{c} \overbrace{\phantom{\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}}}^{\begin{matrix} i & & k \end{matrix}} \\ \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \end{array} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix}$$

Donde $c = \cos\theta$ y $s = \mathrm{sen}\,\theta$ Note que $g_{ii} = c$, $g_{kk} = c$, $g_{ik} = s$ y $g_{ki} = -s$ donde se supone que $i < k$. La rotación de Givens es una matriz ortogonal. La matriz de rotación de Givens es una matriz de rotación en el plano (i,k) de $\mathbb{R}^n$ El ángulo θ solo tiene significado en dos y tres dimensiones. El producto $G(i,k,\theta)\boldsymbol{x}$, $\boldsymbol{x} \neq 0$ representa una rotación de θ radianes en el sentido de las manecillas del reloj del vector $\boldsymbol{x}$.

Sea $x=(x_1,\ldots,x_i,\ldots,x_k,\ldots,x_n)^t$ entonces:

$$G(i,k,\theta)x=\begin{bmatrix} x_1 \\ \vdots \\ cx_i+sx_k \\ \vdots \\ -sx_i+cx_k \\ \vdots \\ x_n \end{bmatrix} \begin{matrix} \\ \\ \leftarrow \text{fila } i \\ \\ \leftarrow \text{fila } k \\ \\ \\ \end{matrix}$$

Si c y s satisfacen las relaciones (2.4) se tiene que:

$$G(i,k,\theta)x=\begin{bmatrix} x_1 \\ \vdots \\ \sqrt{x_i^2+x_k^2} \\ \vdots \\ 0 \\ \vdots \\ x_n \end{bmatrix} \begin{matrix} \\ \\ \leftarrow \text{fila } i \\ \\ \leftarrow \text{fila } k \\ \\ \\ \end{matrix}$$

Es decir, $G(i,k,\theta)$ anula el elemento k de x y modifica el elemento i de x, de tal forma que el nuevo elemento en la fila i es $\sqrt{x_i^2+x_k^2}$ Cuando la matriz A se multiplica por $G(i,k,\theta)$ sólo se afectan las filas i y k de la matriz A.

Se puede aplicar una secuencia de rotaciones en el plano a todos los componentes abajo de un elemento de un vector. Eliminando el ángulo en la rotación de Givens tenemos que:

$$G(1,2)x=\begin{bmatrix} \sqrt{x_1^2+x_2^2} \\ 0 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix},$$

$$G(1,3)G(1,2)x=\begin{bmatrix}\sqrt{x_1^2+x_2^2+x_3^2}\\0\\0\\x_4\\\vdots\\x_n\end{bmatrix},$$

$$G(1,n)\ldots G(1,3)G(1,2)x=\begin{bmatrix}\sqrt{x_1^2+x_2^2+x_3^2\ldots+x_n^2}\\0\\0\\0\\\vdots\\0\end{bmatrix},$$

El producto de matrices de rotación no produce una matriz de rotación, sin embargo dado que una matriz de rotación es ortogonal se tiene que el producto de matrices de rotación es ortogonal.

Es usual decir que una secuencia de rotaciones en $\mathbb{R}^{n\times n}$ produce una rotación en $\mathbb{R}^{n\times n}$ como se señaló previamente, pero ésto no es correcto. Sin embargo adoptamos la costumbre. De lo mencionado previamente se desprende el siguiente resultado:

Sea $0\neq x\in\mathbb{R}^n$, x se puede rotar por una secuencia de $n-1$ rotaciones de plano de tal suerte que después de la rotación, x sólo tiene componentes en la dirección de e_i, $0\le i\le n$, donde e_i es un vector unitario en la dirección del eje i. Dicho de otra forma, existe una matriz ortonormal Q formada por el producto de rotaciones planas tal que:

$$Qx=\|x\|e_i=\left[0\quad\ldots\quad 0\quad\underbrace{\sqrt{x_1^2+x_2^2+\ldots+x_i^2+\ldots+x_n^2}}_{\text{columna } i}\quad 0\quad\ldots\quad 0\right]^t\qquad 0\le i\le n$$

En particular:

$$Q=G(i,n)\cdot\ldots\cdot G(i,i+1)\cdot G(i,i-1)\cdot\ldots\cdot G(i,1)$$

Ejemplo 2.1. Sea $x = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5]^t$ y

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \xrightarrow{G(3,1)} \begin{bmatrix} 0 \\ x_2 \\ \sqrt{x_1^2 + x_3^2} \\ x_4 \\ x_5 \end{bmatrix} \xrightarrow{G(3,2)} \begin{bmatrix} 0 \\ 0 \\ \sqrt{x_1^2 + x_2^2 + x_3^2} \\ x_4 \\ x_5 \end{bmatrix} \xrightarrow{G(3,4)}$$

$$\begin{bmatrix} 0 \\ 0 \\ \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} \\ 0 \\ x_5 \end{bmatrix} \xrightarrow{G(3,5)} \begin{bmatrix} 0 \\ 0 \\ \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2} \\ 0 \\ 0 \end{bmatrix}$$

Entonces la matriz ortonormal $\boldsymbol{Q}$ está dada por:

$$\boldsymbol{G}(3,5) \cdot \boldsymbol{G}(3,4) \cdot \boldsymbol{G}(3,2) \cdot \boldsymbol{G}(3,1)$$

■

En la práctica las multiplicaciones de la rotación de Givens no se efectúan de acuerdo al algoritmo de multiplicación de matrices sino se usa un procedimiento de rotación que hace lo equivalente, esto nos libra de efectuar operaciones que no contribuyen al resultado. Más adelante mostramos algoritmos que ilustran la idea. Note que un elemento de $\boldsymbol{x}$ que se ha reducido cero por una rotación nunca toma valores diferentes de cero cuando se aplican más tarde rotaciones. De lo explicado antes se tiene que dada la matriz $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ es posible a través de una secuencia de rotaciones reducir $\boldsymbol{A}$ a $\boldsymbol{R} \in \mathbb{R}^{n \times m}$ donde $\boldsymbol{R}$ es una matriz triangular superior. $\boldsymbol{R}$ se logra anulando todos los elementos de $\boldsymbol{A}$ que están debajo de su diagonal.

La rotación de plano considerada antes aplica solamente a matrices reales. Ahora nos interesa extender las ideas anteriores a matrices complejas. Existen varias formas de lograr la extensión deseada. Sea la siguiente matriz cuyos elementos son números complejos:

$$\begin{bmatrix} x & z \\ y & w \end{bmatrix} \tag{2.7}$$

Ahora encontramos las relaciones que x, y, z y w deben cumplir para que la matriz (2.7) sea una matriz unitaria. De:

$$\begin{bmatrix} x & z \\ y & w \end{bmatrix}\begin{bmatrix} x^* & y^* \\ z^* & w^* \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}=\begin{bmatrix} |x|^2+|z|^2 & x\cdot y^*+z\cdot w^* \\ y\cdot x^*+w\cdot z^* & |y|^2+|w|^2 \end{bmatrix} \tag{2.8}$$

Se tiene que se debe cumplir que $|x|^2+|z|^2=1$, $|y|^2+|w|^2=1$, $y\cdot x^*+w\cdot z^*=0$ y $x\cdot y^*+z\cdot w^*=0$. De aquí se tiene que $|x|=|w|$, $|y|=|z|$ y $\theta_y-\theta_x=\theta_w-\theta_z\pm(2k+1)\pi$, donde k es un entero. Usando estas últimas expresiones en la matriz (2.7) tenemos que se puede reescribir como:

$$\begin{bmatrix} |x|e^{j\theta_x} & |z|e^{j\theta_z} \\ |z|e^{j\theta_y} & |x|e^{j\theta_w} \end{bmatrix} \tag{2.9}$$

Donde $\theta_x=\angle x$, $\theta_z=\angle z$, $\theta_y=\angle y$, $\theta_w=\angle w$, $|x|^2+|z|^2=1$ y $\theta_y-\theta_x=\theta_w-\theta_z\pm(2k+1)\pi$. La matriz compleja (2.9) junto con las restricciones $|x|^2+|z|^2=1$ y $\theta_y-\theta_x=\theta_w-\theta_z\pm(2k+1)\pi$, donde k es un entero, es la forma más general de una transformación matricial unitaria 2×2. Existen seis parámetros en la representación. Si seleccionamos $\theta_w=-\theta_x$ y $\theta_y=-\theta_z\pm(2k+1)\pi$, entonces se satisface la restricción $\theta_y-\theta_x=\theta_w-\theta_z\pm(2k+1)\pi$ y se reduce el número de rotaciones complejas de la transformación en dos. Después de la particularización mencionada y haciendo $|x|=\cos\theta$, $|z|=\operatorname{sen}\theta$. $0\le\theta\le\dfrac{\pi}{2}$, la matriz (2.9) se transforma en:

$$\begin{bmatrix} \cos\theta\cdot e^{j\theta_x} & \operatorname{sen}\theta\cdot e^{j\theta_z} \\ -\operatorname{sen}\theta\cdot e^{-j\theta_z} & \cos\theta\cdot e^{-j\theta_x} \end{bmatrix}$$

Si hacemos $c = \cos\theta \cdot e^{j\theta_x}$ y $s = \operatorname{sen}\theta \cdot e^{-j\theta_z}$, la matriz anterior se puede reescribir como:

$$\begin{bmatrix} c & s^* \\ -s & c^* \end{bmatrix} \tag{2.10}$$

Donde $|c|^2 + |s|^2 = 1$. La matriz (2.10) es la rotación de Givens compleja y es una matriz unitaria. Supondremos que la condición anterior es válida en lo que sigue. Sean a y b números complejos dados, queremos encontrar c y s en general complejos tal que se cumple que:

$$\begin{bmatrix} c & s^* \\ -s & c^* \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} = \begin{bmatrix} c \cdot a + s^* \cdot b \\ -s \cdot a + c^* \cdot b \end{bmatrix}$$

De las relaciones $r = c \cdot a + s^* \cdot b$, $0 = -s \cdot a + c^* \cdot b$ y $|c|^2 + |s|^2 = 1$ se tiene que:

$$\begin{aligned} r &= \frac{c}{a^*}\left(|a|^2 + |b|^2\right) = \frac{s^*}{b^*}\left(|a|^2 + |b|^2\right) \\ &= e^{-j(\theta_c + \theta_a)} \cdot \sqrt{|a|^2 + |b|^2} = e^{-j(\theta_s - \theta_b)} \cdot \sqrt{|a|^2 + |b|^2} \end{aligned}$$

$$\begin{aligned} |c| &= \frac{|a|}{\left(|a|^2 + |b|^2\right)^{\frac{1}{2}}} \\ |s| &= \frac{|b|}{\left(|a|^2 + |b|^2\right)^{\frac{1}{2}}} \end{aligned} \tag{2.11}$$

Donde $\theta_c = \angle c$, $\theta_a = \angle a$, $\theta_s = \angle s$ y $\theta_b = \angle b$. En general, los ángulos de c y s son arbitrarios y se seleccionan para satisfacer otros criterios. Por ejemplo si se requiere que r sea real entonces se debe tener que:

$$\angle c = \angle a^* \quad \text{y} \quad \angle s = \angle b,$$

que es una consecuencia directa de la primera ecuación de (2.11). Si $\theta_a = \angle a$ y $\theta_b = \angle b$ entonces se tiene que:

$$s = |s| e^{j\theta_b} \quad \text{y} \quad c = |c| e^{-j\theta_a} \tag{2.12}$$

Los resultados anteriores los podemos resumir en la relación:

$$\begin{bmatrix} \cos\theta \cdot e^{-j\theta_a} & \text{sen}\theta \cdot e^{-j\theta_b} \\ -\text{sen}\theta \cdot e^{j\theta_b} & \cos\theta \cdot e^{j\theta_a} \end{bmatrix} \begin{bmatrix} a & x \\ b & y \end{bmatrix} = \frac{1}{\sqrt{|a|^2 + |b|^2}} \begin{bmatrix} a^* & b^* \\ -b & a \end{bmatrix} \begin{bmatrix} a & x \\ b & y \end{bmatrix} = \begin{bmatrix} r & x''' \\ 0 & y''' \end{bmatrix} \tag{2.13}$$

Donde $\theta = \text{tg}^{-1}\left(\frac{|b|}{|a|}\right)$, $\theta_a = \angle a$, $\theta_b = \angle b$, $r = \left(|a|^2 + |b|^2\right)^{\frac{1}{2}}$ y

$$\begin{bmatrix} \cos\theta \cdot e^{-j\theta_a} & \text{sen}\theta \cdot e^{-j\theta_b} \\ -\text{sen}\theta \cdot e^{j\theta_b} & \cos\theta \cdot e^{j\theta_a} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x''' \\ y''' \end{bmatrix}$$

Note que la siguiente relación es válida:

$$\begin{aligned} \begin{bmatrix} \cos\theta \cdot e^{-j\theta_a} & \text{sen}\theta \cdot e^{-j\theta_b} \\ -\text{sen}\theta \cdot e^{j\theta_b} & \cos\theta \cdot e^{j\theta_a} \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & e^{j(\theta_a+\theta_b)} \end{bmatrix} \begin{bmatrix} \cos\theta \cdot e^{-j\theta_a} & \text{sen}\theta \cdot e^{-j\theta_b} \\ -\text{sen}\theta \cdot e^{-j\theta_a} & \cos\theta \cdot e^{-j\theta_b} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & e^{j(\theta_a+\theta_b)} \end{bmatrix} \begin{bmatrix} \cos\theta & \text{sen}\theta \\ -\text{sen}\theta & \cos\theta \end{bmatrix} \begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix} \end{aligned} \tag{2.14}$$

Sea la transformación:

$$\begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a' \\ b' \end{bmatrix} \tag{2.15}$$

donde a y b son números complejos, entonces a' y b' son números reales donde $a' = |a|$ y $b' = |b|$. Note que la matriz de transformación en la relación (2.15) es unitaria. La primera matriz de transformación en (2.14) convierte el vector complejo $[a, \ b]^t$ en el vector real $\left[|a|, \ |b|\right]^t$ La segunda matriz de transformación rota el vector real

$[|a|, \ |b|]^t$ y se obtiene el vector real $[r, \ 0]^t$ y la tercera matriz de transformación no tiene efecto en el vector real $[r, \ 0]^t$

La figura 2.5 muestra superprocesadores córdicos que efectúan el procesamiento indicado por las relaciones (2.13) y (2.14). Las rotaciones en (2.13) y (2.14) se pueden efectuar en procesadores convencionales o en superprocesadores córdicos. Con objeto de producir una matriz con sólo números reales en la diagonal, se requiere la siguiente transformación para convertir en real el elemento a la extrema derecha en la diagonal $|z| \cdot e^{j\theta_z}$:

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-j\theta_z} \end{bmatrix}$$

El superprocesador córdico se puede reconfigurar para efectuar esta transformación.

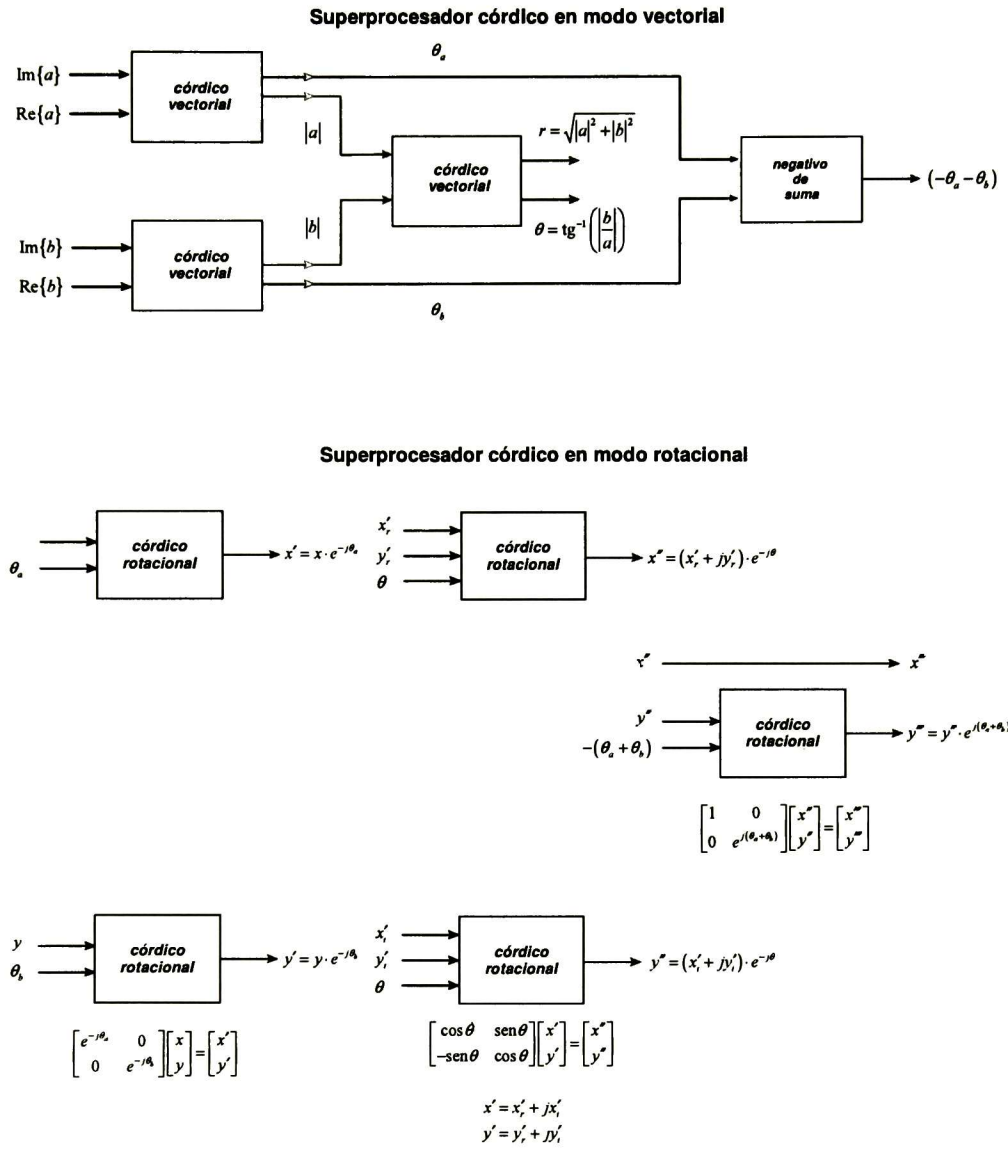


Figura 2.5: Rotación compleja de Givens implementada con superprocesadores córdicos.

A continuación explicamos otra forma de encontrar una transformación que anula un elemento de un vector bidimensional y convierte el otro elemento en un número real. En algunas aplicaciones el resultado descrito antes es un requerimiento. La transformación que presentamos es el producto de otras dos transformaciones donde la primera convierte las dos componentes de un vector bidimensional en reales y la segunda transformación es una transformación de Givens para reales. La transformación buscada

se obtiene de la transformación compuesta (2.14) eliminando la tercera transformación, la cual no tiene efecto en el vector $[r, \ 0]^t$

Consolidando las dos primeras transformaciones de (2.14) en una transformación, se tiene que:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix}\begin{bmatrix} a & x \\ b & y \end{bmatrix}=\begin{bmatrix} \cos\theta\cdot e^{-j\theta_a} & \text{sen}\theta\cdot e^{-j\theta_b} \\ -\text{sen}\theta\cdot e^{j\theta_b} & \cos\theta\cdot e^{j\theta_a} \end{bmatrix}\begin{bmatrix} a & x \\ b & y \end{bmatrix}=\begin{bmatrix} r & x'' \\ 0 & y'' \end{bmatrix} \qquad (2.16)$$

Donde $\theta=\text{tg}^{-1}\left(\left|\frac{b}{a}\right|\right)$, $\theta_a=\angle a$, $\theta_b=\angle b$, $r=\sqrt{|a|^2+|b|^2}$ y

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x'' \\ y'' \end{bmatrix} \qquad (2.17)$$

La transformación resultante (2.16) es unitaria dada que es el producto de dos transformaciones unitarias. La figura 2.6 muestra superprocesadores córdicos en modo vectorial y rotacional que calculan respectivamente la expresión (2.17) y los ángulos θ, θ_a y θ_b. Aquí también se aplica un método similar al explicado antes para convertir todos los elementos de la diagonal en reales.

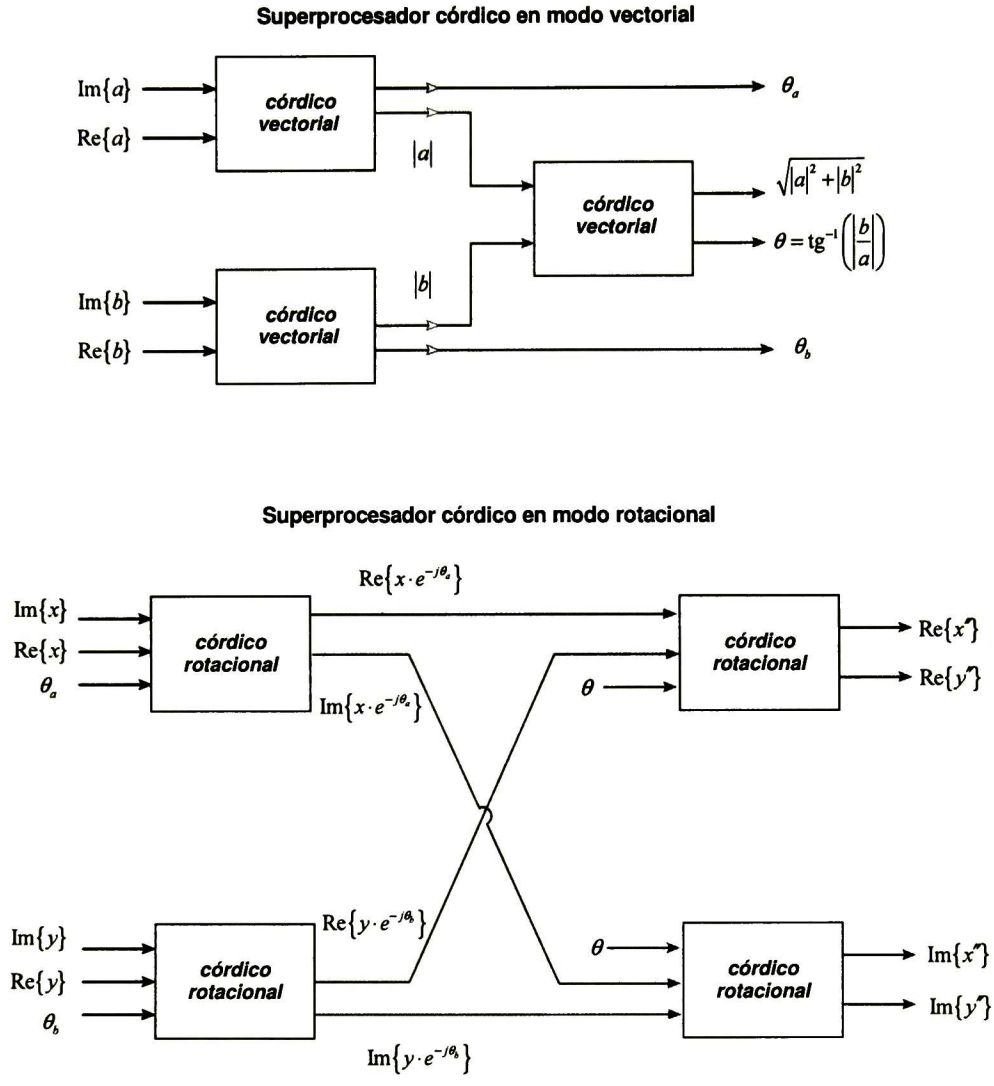


Figura 2.6: Rotación compleja de Givens implementada con superprocesadores córdicos.

A continuación explicamos otra transformación que anula un elemento de un vector bidimensional y convierte el otro elemento en uno real. Nuestro punto de partida es la relación (2.16). Esta relación tiene el inconveniente que requiere cuatro rotaciones complejas. Formamos la siguiente transformación:

$$\begin{bmatrix} e^{j\theta_a} & 0 \\ 0 & e^{j\theta_b} \end{bmatrix}\begin{bmatrix} \cos\theta & \text{sen}\theta \\ -\text{sen}\theta & \cos\theta \end{bmatrix}\begin{bmatrix} e^{-j\theta_a} & 0 \\ 0 & e^{-j\theta_b} \end{bmatrix}\begin{bmatrix} a & x \\ b & y \end{bmatrix} = \begin{bmatrix} \cos\theta & \text{sen}\theta\cdot e^{j(\theta_a-\theta_b)} \\ -\text{sen}\theta\cdot e^{-j(\theta_a-\theta_b)} & \cos\theta \end{bmatrix}\begin{bmatrix} a & x''' \\ b & y''' \end{bmatrix} = \begin{bmatrix} r\cdot e^{j\theta_a} & x''' \\ 0 & y''' \end{bmatrix} \tag{2.18}$$

Donde $\theta = \text{tg}^{-1}\left(\left|\frac{b}{a}\right|\right)$, $\theta_a = \angle a$, $\theta_b = \angle b$, $r = \sqrt{|a|^2 + |b|^2}$ y

$$\begin{bmatrix} \cos\theta & \text{sen}\theta\cdot e^{j(\theta_a-\theta_b)} \\ -\text{sen}\theta\cdot e^{-j(\theta_a-\theta_b)} & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x''' \\ y''' \end{bmatrix} \tag{2.19}$$

La transformación (2.18) es unitaria dado que es el producto de transformaciones unitarias. La figura 2.7 muestra superprocesadores córdicos en modo rotacional y vectorial que calculan respectivamente la relación (2.19) y los ángulos θ y $\theta_a - \theta_b$. Note que en este caso $r\cdot e^{j\theta_a}$ no es en general real en la expresión (2.18). La rotación en la relación (2.18) es útil cuando no se requiere que $r\cdot e^{j\theta_a}$ sea real.

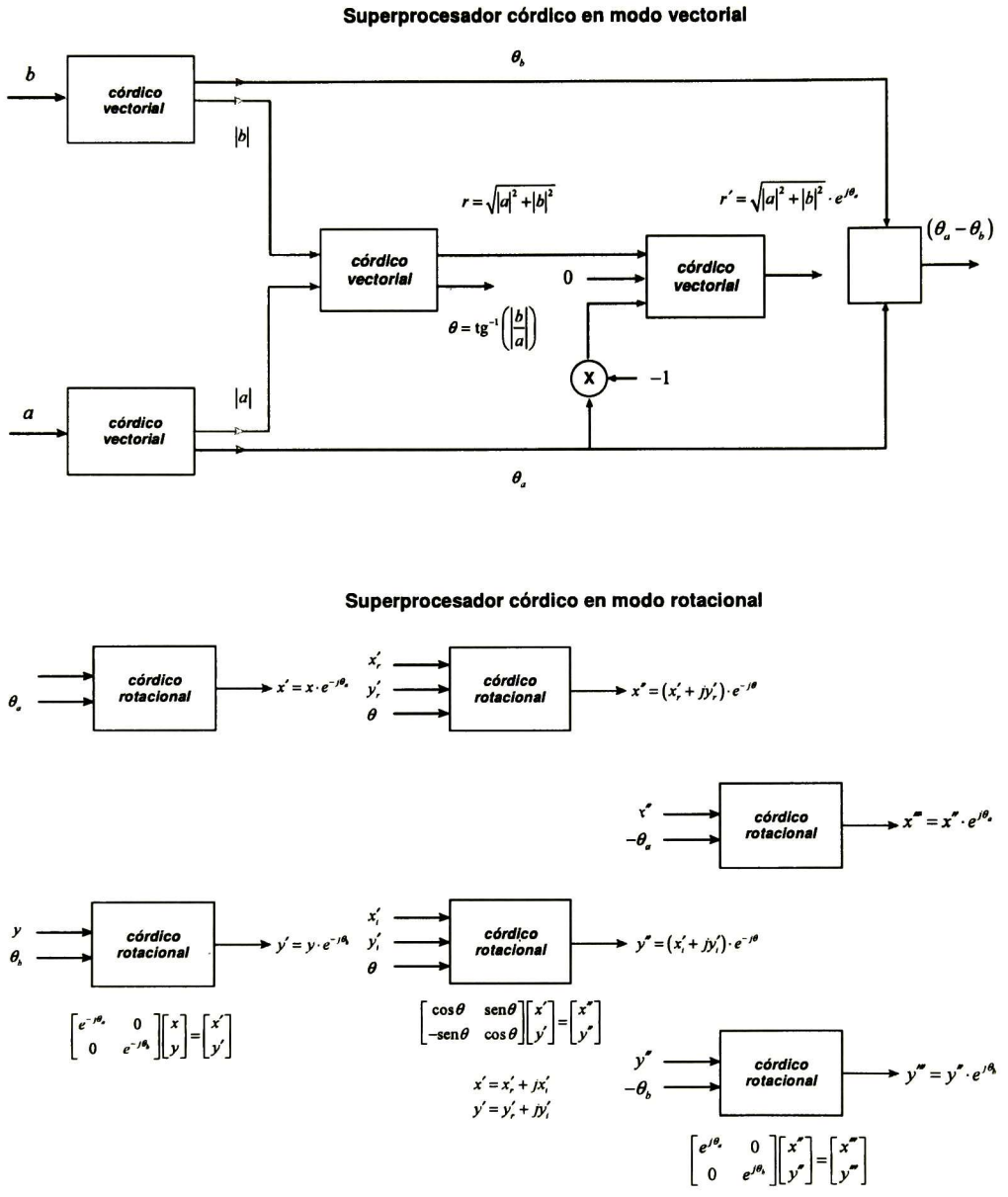


Figura 2.7: Rotación compleja de Givens implementada con superprocesadores córdicos.

2.3 Factorización QR vía Rotación de Givens

La factorización QR de una matriz tiene múltiples aplicaciones entre otras:

i. en la solución de ecuaciones lineales simultáneas,
ii. en mínimos cuadrados,
iii. cálculo de la seudoinversa de Moore Penrose,
iv. cálculo de vectores propios y
v. descomposición en valores singulares.

Existen los siguientes métodos para efectuar la factorización QR.

i. proceso de Gram Schmidt,
ii. proceso de Gram Schmidt modificado,
iii. transformación de Householder y
iv. rotación de Givens.

La factorización QR se puede formular en términos de aplicaciones de transformaciones de Givens. Las transformaciones de Givens se aplican hasta producir una matriz triangular superior. El proceso de convertir $\boldsymbol{A}$ en $\boldsymbol{R}$ se llama triangularización.

La rotación de Givens tiene las siguientes características cuando se usa para la factorización QR.

i. requiere menos operaciones que la transformación de Householder para matrices ralas, una matriz rala es aquella en que la mayor parte de los valores de la matriz son iguales. Usualmente, en una matriz rala el valor que se repite es el cero. Existen variantes de la rotación de Givens que requieren el mismo número de operaciones que la transformación de Householder para matrices densas.
ii. mayor facilidad de ser paralelizada que la transformación de Householder.
iii. tiene menor estabilidad numérica que la transformación de Householder.

Para implementación en circuitos el método de factorización QR por medio de la rotación de Givens se prefiere dado que presenta mayores posibilidades para paralelizar.

Sean $\boldsymbol{A} \in \mathbb{C}^{n \times m}$, $\boldsymbol{Q} \in \mathbb{C}^{n \times n}$ y $\boldsymbol{R} \in \mathbb{C}^{n \times m}$, la descomposición QR de $\boldsymbol{A}$ es de la forma:

$$\boldsymbol{A} = \boldsymbol{QR}$$

Donde Q es una matriz unitaria esto es $QQ^h = I$ y $R = Q^h A$ tiene sus elementos debajo de su diagonal iguales a cero. Es decir, R es una matriz triangular superior. La triangularización se puede aplicar a matrices con diferente cociente $\frac{n}{m}$. Si $n \geq m$, $Q^h A$ se puede escribir como:

$$Q^h A = \begin{bmatrix} R' \\ 0 \end{bmatrix}$$

Donde $R' \in \mathbb{C}^{m \times m}$ y es triangular superior. Si $n < m$, se tiene que:

$$Q^h A = \begin{bmatrix} R' & S \end{bmatrix}$$

Donde $R' \in \mathbb{C}^{n \times n}$ y $S \in \mathbb{C}^{n \times (m-n)}$ y R' es triangular superior. Cuando se tiene este caso en muchas aplicaciones es más conveniente usar la descomposición RQ en lugar de la QR. Otras descomposiciones similares a la QR son la RQ, QL, y LQ. Estas últimas son similares a la QR y RQ respectivamente, la diferencia es que mientras en QR y RQ aparecen matrices triangulares superiores en QL, y LQ son triangulares inferiores. La factorización QR presentada antes se llama factorización QR modificada por algunos autores [12].

Una generalización de la factorización QR es la factorización QRP donde Q y P son como antes y $P \in I^{m \times m}$ y es una matriz de permutación. La inclusión de la matriz P equivale a permitir pivoteo en la factorización QR. La factorización QRP permite detectar dependencia entre las columnas de A. Si A tiene rango k entonces se puede escribir que:

$$Q^h AP = \begin{bmatrix} R' & R'' \\ 0 & 0 \end{bmatrix}$$

Donde $R' \in \mathbb{C}^{k \times k}$ y $R'' \in \mathbb{C}^{k \times (m-k)}$ donde R' es triangular superior y no singular.

2.4 Algoritmo de Solución de la Descomposición *QR*

En el listado 2.1 se presenta la idea fundamental, para la implementación computacional de la descomposición QR cuando la matriz A es real. M es el número de filas, N el número de columnas y $M > N$. El caso $M = N$ no se trata aquí, porque la teoría aquí presentada contiene también a este caso. La primera tarea que el algoritmo efectúa es el cálculo del seno y coseno del ángulo de rotación y después la rotación se aplica a las filas a donde pertenecen el elemento que se aniquila y el elemento de la diagonal que se considera.

$for\ (k = 0; k < N;\ k++)\{$ // barre la diagonal

$for\ (i = M-1;\ i >= k+1;\ i--)\{$ // barre las filas

// efectúa el cálculo

// del coseno y el seno, es decir,

$$c = \frac{a[i-1][k]}{\left(a^2[i][k] + a^2[i-1][k]\right)^{\frac{1}{2}}};$$ // calcula parámetros de rotación

$$s = \frac{a[i][k]}{\left(a^2[i][k] + a^2[i-1][k]\right)^{\frac{1}{2}}};$$

$for\ (j = k; j < N; j++)\{$ // barre las columnas

// efectúa las rotaciones

$$\begin{bmatrix} a[i-1][j] \\ a[i][j] \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a[i-1][j] \\ a[i][j] \end{bmatrix};$$

$\}$

$\}$

$\}$

Listado 2.1: Factorización QR para implementar en procesador secuencial.

En el algoritmo mostrado tenemos tres índices: k, i y j, k barre sobre los elementos de la diagonal, i sobre las filas y j sobre las columnas. Notamos que asociado a cada elemento de la diagonal k, a_{kk} tenemos una matriz A_{kk} definida por:

$$A_{kk} = \begin{bmatrix} a_{kk} & \cdots & a_{k,N-1} \\ \vdots & & \vdots \\ a_{M-1,k} & \cdots & a_{M-1,N-1} \end{bmatrix}$$

Se puede considerar que el índice k produce la secuencia de matrices A_{kk}, $k = 0,\ldots,N-1$ donde se supone que $M > N$ Un recorrido del índice i hace igual a cero todos los elementos de A_{kk}, en la columna k que están abajo del elemento a_{kk}. Un recorrido del índice j efectúa el mismo procesamiento que se le hizo a un elemento de la columna k y fila i, a todos los elementos de la fila i. Podemos considerar que un recorrido de k produce matrices, que un recorrido por i transforma en matrices que tienen ceros en la columna k debajo del elemento a_{kk}. El índice k barre sobre matrices, el índice i sobre filas de estas matrices y el índice j sobre columnas de una fila dada. El índice k maneja objetos agregados mientras el índice j maneja objetos simples. Los objetos simples se usan para construir objetos agregados de diferente complejidad.

El algoritmo del listado 2.1 se puede transformar en el algoritmo mostrado en el listado 2.2, M es el número de filas, N el número de columnas y $M > N$ Aquí en lugar de calcular el seno y el coseno de un ángulo se obtiene el ángulo y en vez de efectuar la rotación por medio de una multiplicación por matriz, la rotación se considera una primitiva. El algoritmo es adecuado para implementarse usando *procesadores cordicos*, en estos procesadores las operaciones de obtención de ángulo y rotación de un número complejo son operaciones primitivas. La primera tarea que el algoritmo efectúa es el cálculo del ángulo de rotación y después la rotación se aplica a las filas a donde pertenecen el elemento que se anula y el elemento de la diagonal que se considera.

```
for (k = 0; k < N; k++){                    // barre la diagonal
    for (i = M − 1; i >= k + 1; i−−){       // barre las filas
                                            // efectúa el cálculo del ángulo, es decir,
                                            // calcula parámetro de rotación
        θ = angle(a[i][k], a[i−1][k]);

        for (j = k; j < N; j++){            // barre las columnas
                                            // efectúa las rotaciones

            [ a[i−1][j] ]                   [ a[i−1][j] ]
            [ a[i][j]   ] = rotate(θ, [ a[i][j]   ]);

        }
    }
}
```

Listado 2.2: Factorización QR para implementación con procesador córdico.

2.5 Transformación del Algoritmo

2.5.1 Algoritmo Base de la Descomposición *QR*

El listado 2.3 muestra la implementación computacional del algoritmo de la descomposición QR, en lenguaje C, en base al listado 2.1. Sea $A = a[i][j]$ la matriz de entrada de datos, de orden $M \times N$. donde: M es el número de filas, N el número de columnas y $M > N$

```
for (k = 0; k < N; k++){                          // barre la diagonal
    for (i = M - 1; i >= k + 1; i--){             // barre las filas

            r = (sqrt( pow(a[i-1][k], 2.0) + pow(a[i][k], 2.0)));
            c = (a[i-1][k]) / r;
            s = (a[i][k]) / r;

        for (j = k; j < N; j++){                  // barre las columnas

                    t1 = a[i-1][j];
            a[i-1][j] = c*t1 + s*a[i][j];
            a[i][j]   = -s*t1 + c*a[i][j];

        }
    }
}
```

Listado 2.3: Implementación en lenguaje C de la factorización QR.

El espacio de iteración o de índices, $x = [i, j, k]^t$, donde los datos son computados está definido por:

$$x = \left\{ (i, j, k)^t \,|\, M - 1 \geq i \geq k + 1, \quad k \leq j \leq N - 1, \quad 0 \leq k \leq N - 1 \right\}$$

Con el código del listado 2.3 se realizarán una serie de transformaciones que comprenden: *normalización, concordancia de índices, eliminación de comunicación global* y *asignación única*. Para explicación de estos términos véase el libro de Wolfe, Michael, en la bibliografía.

Observe que cuando $j = k$ se realiza el cálculo de la hipotenusa, del seno y del coseno dados los catetos de un triángulo, también se calcula la rotación de las filas i e $i-1$. El segundo cálculo es innecesario dado que sabemos que los valores que se obtendrán después de la rotación son la hipotenusa y cero. En particular, tenemos que introducir las siguientes líneas de código:

$$a[i-1][k] = r \qquad \text{y} \qquad a[i][k] = 0$$

En el programa mostrado en el listado 2.4 se evita el cálculo innecesario antes mencionado. Asimismo se añaden al programa las siguientes líneas de código:

```
if (r == 0){
    c = 1;
    s = 0;
}else{
    c = (a[i-1][k]) / r;
    s = (a[i][k]) / r;
}
```

Con el propósito de evitar en el proceso de *Generación de Rotaciones de Givens* (*GG*) un valor indeterminado al evaluar los valores de las variables de seno y coseno s y c respectivamente, dada una posible división por el valor de cero, que se presenta cuando los elementos de la matriz $a[i-1][j]$ y $a[i][j]$ son ambos cero [24].

```
for (k = 0; k < N; k++){                    // barre la diagonal
    for (i = M-1; i >= k+1; i--){           // barre las filas

        r=(sqrt( pow(a[i-1][k],2.0)+pow(a[i][k],2.0)));

        if (r == 0){
            c = 1;
            s = 0;
        }else{
            c = (a[i-1][k])/r;
            s = (a[i][k])/r;
        }

        a[i-1][k]= r;
        a[i][k]  = 0;

        for (j = k+1; j < N; j++){          // barre las columnas

                t1 = a[i-1][j];
            a[i-1][j]= c*t1+s*a[i][j];
            a[i][j]  = -s*t1+c*a[i][j];

        }
    }
}
```

Listado 2.4: Implementación en lenguaje C de la factorización QR.

2.5.2 Hundimiento de Código

En la metodología de transformación de un programa en circuitería que aquí consideramos a cada punto del espacio de iteración definido por los índices (i, j, k) le asignamos una operación que se ejecuta en un procesador. Sin embargo, el programa del listado 2.4 es un bucle imperfecto. Es posible en muchas ocasiones, como aquí es el

caso, transformar un bucle imperfecto a uno perfecto. Para tal efecto se usa la transformación de *hundimiento de código* [40]. El programa del listado 2.5, se obtiene de aplicar la transformación de *hundimiento de código* al programa del listado 2.4. También se realizó la eliminación de la línea de código $a[i][k]=0;$ esto se debe a que la teoría nos indica que este elemento de la matriz es siempre cero. Los elementos que siempre son cero de acuerdo con la teoría nunca los calculamos con el objeto de reducir la circuitería que los implementaría.

```
for (k = 0;k < N; k++){                          // barre la diagonal
    for (i = M-1; i >= k+1; i--){                // barre las filas
        for (j = k; j < N; j++){                 // barre las columnas

            if (j == k){          // Generación de Rotaciones de Givens (GG)

                r=(sqrt( pow(a[i-1][j],2.0)+pow(a[i][j],2.0)));
                if (r == 0){
                  c = 1;
                  s = 0;
                }else{
                  c = (a[i-1][j])/r;
                  s = (a[i][j])/r;
                }
                a[i-1][j]= r;

            }else{            // Ejecución de Rotaciones de Givens (RG)

                    t1 = a[i-1][j];
              a[i-1][j]= c*t1+s*a[i][j];
              a[i][j]  = -s*t1+c*a[i][j];

            }
        }
    }
}
```

Listado 2.5: Código de la factorización QR después de aplicar *hundimiento de código*.

Del programa del Listado 2.5 observamos que para cada valor del índice k se accesan inicialmente dos filas de la matriz A y posteriormente se accesan fila por fila de la matriz A hasta llegar a la fila $k+1$. A continuación presentamos una transformación que nos permite para cada índice de k accesar una a una las filas de la matriz A. Esta transformación, como veremos más adelante, nos simplificará la circuitería considerablemente dado que evita suministrar por el alimentador las filas $M-1$ y $M-2$ de la matriz A durante la primera iteración del índice determinado por i. Se logra así que la alimentación de datos a la circuitería sea uniforme. A continuación explicamos la transformación. Considere la matriz de entrada A:

$$A[M][N]=\begin{bmatrix} a_{00} & \cdots & a_{0,N-1} \\ \vdots & & \vdots \\ a_{M-1,0} & \cdots & a_{M-1,N-1} \end{bmatrix}$$

y las filas $a[M-2][*]$ y $a[M-1][*]$ durante la iteración cuando el índice i es igual a $M-1$. Enseguida se muestra las filas $M-1$ y $M-2$.

$$A[M][N]=\begin{matrix} \\ a[M-2][*]\{ \\ a[M-1][*]\{ \end{matrix}\begin{bmatrix} a_{00} & \cdots & a_{0,N-1} \\ \vdots & & \vdots \\ a_{M-2,0} & \cdots & a_{M-2,N-1} \\ a_{M-1,0} & \cdots & a_{M-1,N-1} \end{bmatrix}$$

La modificación, como se muestra a continuación, al código consiste en aumentar el número de filas de M a $M+1$ en la matriz de entrada de datos A donde todos los elementos de esta nueva fila son cero.

$$A[M+1][N]=\begin{bmatrix} a_{00} & \cdots & a_{0,N-1} \\ \vdots & & \vdots \\ a_{M-1,0} & \cdots & a_{M-1,N-1} \\ 0 & \cdots & 0 \end{bmatrix}$$

La transformación al programa también usa la propiedad del algoritmo de descomposición QR: cuando en una rotación de Givens una de las filas que se transforma es igual a cero la otra no se modifica.

La transformación en el programa se manifiesta cambiando el límite inferior del índice i el cual es ahora M Observe que la transformación expande el espacio de iteración. Esta expansión como se verá adelante incrementa el tiempo de ejecución del algoritmo en un ciclo de reloj sin embargo produce reducciones considerables en la implementación final. La transformación está reflejada en el código del listado 2.6 donde se supone que los elementos de la última fila de la matriz A son iguales a cero.

Sea A la matriz de entrada de datos, de orden $(M+1)\times N$, donde: M es el número de filas, N el número de columnas y $M > N$

```
for (k = 0; k < N; k++){                          // barre la diagonal
    for (i = M; i >= k+1; i--){                   // barre las filas
        for (j = k; j < N; j++){                  // barre las columnas

            if (j == k){                          // Generación de Rotaciones Givens (GG)

                r = (sqrt( pow(a[i-1][j],2.0) + pow(a[i][j],2.0)));
                if (r == 0){
                   c = 1;
                   s = 0;
                }else{
                   c = (a[i-1][j]) / r;
                   s = (a[i][j]) / r;
                }
                a[i-1][j] = r;

            }else{                                // Ejecución de Rotaciones de Givens (RG)

                    t1 = a[i-1][j];
                a[i-1][j] = c*t1 + s*a[i][j];
                a[i][j]   = -s*t1 + c*a[i][j];

            }
        }
    }
}
```

Listado 2.6: Código de la factorización QR modificado para evitar al alimentador suministrar dos valores de la matriz de entrada A en la primera iteración del índice i.

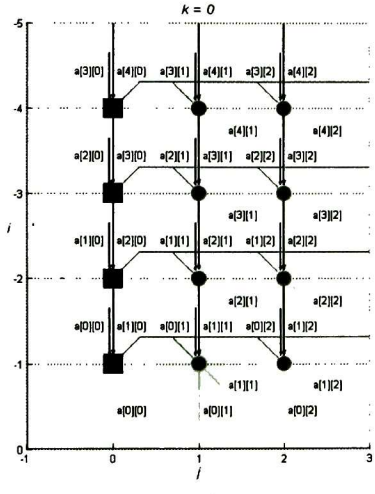
2.5.3 Normalización

La teoría sobre la cual se fundamentan la serie de transformaciones aplicables al código supone que los índices se incrementan positivamente. La transformación que logra esto se llama de normalización. El programa del listado 2.7 se obtiene del programa del listado 2.6 al aplicar una transformación de normalización.

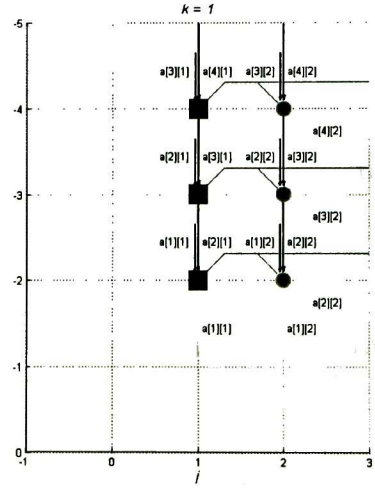
```
for (k = 0;k < N; k++){                          // barre la diagonal
    for (i = -M; i >= -k - 1; i++){              // barre las filas
        for (j = k; j < N; j++){                 // barre las columnas

            if (j == k){                         // Generación de Rotaciones de Givens (GG)

                r=(sqrt( pow(a[-i-1][j],2.0)+ pow(a[-i][j],2.0)));
                if (r == 0){
                    c = 1;
                    s = 0;
                }else{
                    c = (a[-i-1][j])/r;
                    s = (a[-i][j])/r;
                }
                a[-i-1][j]= r;

            }else{                               // Ejecución de Rotaciones de Givens (RG)

                    t1 = a[-i-1][j];
                a[-i-1][j]= c*t1+s*a[-i][j];
                a[-i][j]  = -s*t1+c*a[-i][j];

            }
        }
    }
}
```

Listado 2.7: Código de la factorización QR después de aplicar normalización.

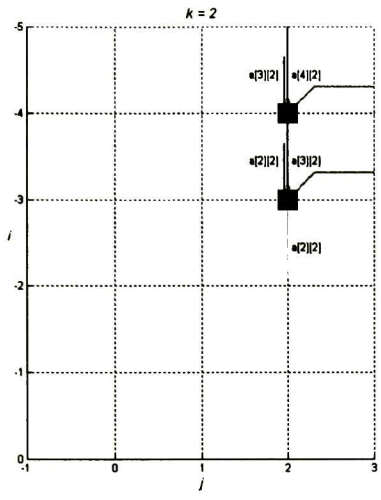
En el grafo de dependencia de la figura 2.8, el cual corresponde al programa del listado 2.7, se observa que los elementos de las matrices parciales, definidas por cada iteración del índice k, $A^{\langle k \rangle}$ resaltados con una flecha de color azul, son los resultados esperados al aplicar la descomposición QR. Las flechas denotadas con color verde son los elementos cuyos valores se han de procesar en la siguiente iteración de k.



a) Iteración $k = 0$.



b) Iteración $k = 1$.



c) Iteración $k=2$.

Figura 2.8: Grafo de dependencia del código de normalización de la descomposición QR, $(M=4, N=3)$.

2.5.4 Concordancia de Indices

La transformación de concordancia de índices aplicada al programa de normalización de la descomposición QR del listado 2.7, consiste en mapear cada una de las asignaciones presentes en el código, al espacio de índices $\boldsymbol{x}=[i,j,k]^t$ determinado por el número de bucles presentes en el programa, como se muestra en el listado 2.8.

```
for (k=0;k<N; k++){                          // barre la diagonal
    for (i=-M; i>=-k-1; i++){                // barre las filas
        for (j=k; j<N; j++){                 // barre las columnas

            if (j==k){                       // Generación de Rotaciones de Givens (GG)

              r[0][0][0]=(sqrt( pow(a[-i-1][j][0],2.0)+pow(a[-i][j][0],2.0)));
              if (r[0][0][0]==0){
                  c[0][0][0]=1;
                  s[0][0][0]=0;
              }else{
                  c[0][0][0]=(a[-i-1][j][0])/r[0][0][0];
                  s[0][0][0]=(a[-i][j][0])/r[0][0][0];
              }
              a[-i-1][j][0]= r[0][0][0];

            }else{                           // Ejecución de Rotaciones de Givens (RG)

              t1[0][0][0]    = a[-i-1][j][0];
              a[-i-1][j][0]= c[0][0][0]*t1[0][0][0]+s[0][0][0]*a[-i][j][0];
              a[-i][j][0]    = -s[0][0][0]*t1[0][0][0]+c[0][0][0]*a[-i][j][0];

            }
        }
    }
}
```

Listado 2.8: Programa después de la transformación de concordancia de índices.

2.5.5 Asignación Unica y Eliminación de Comunicación Global

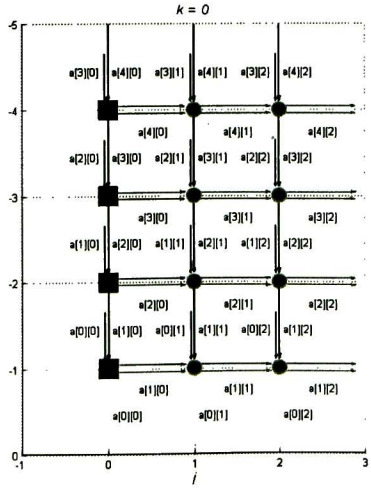
Al efectuar ***asignación única*** en el código a cada variable se le asigna valor una sola vez, durante toda la ejecución del programa. La ***eliminación de comunicación global*** evita la difusión global de datos y la sustituye por múltiples comunicaciones punto a punto. La transformación de asignación única de las variables r, s y c y eliminación de comunicación global de s y c realizado al programa de concordancia de índices, se muestra en el código del listado 2.9.

```
for (k = 0;k < N; k++){                          // barre la diagonal
    for (i = -M; i >= -k-1; i++){                // barre las filas
        for (j = k; j < N; j++){                 // barre las columnas

            if (j == k){                         // Generación de Rotaciones de Givens (GG)

              r[-i][j][0]=(sqrt( pow(a[-i-1][j][0],2.0)+ pow(a[-i][j][0],2.0)));
              if (r[-i][j][0] == 0){
                 c[-i][j+1][0] = 1;
                 s[-i][j+1][0] = 0;
              }else{
                 c[-i][j+1][0] = (a[-i-1][j][0]) / r[-i][j][0];
                 s[-i][j+1][0] = (a[-i][j][0]) / r[-i][j][0];
              }
              a[-i-1][j][0] = r[-i][j][0];
```

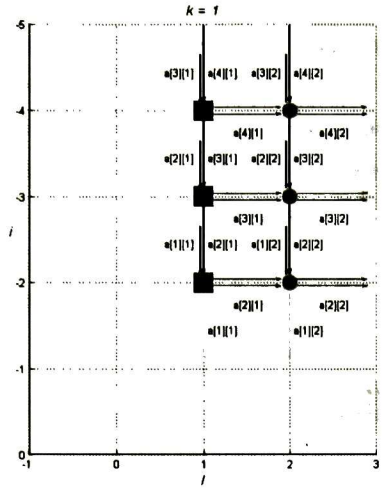
```
        }else{          // Ejecución de Rotaciones de Givens (RG)

            t1[-i][j][0]   = a[-i-1][j][0];
            a[-i-1][j][0]= c[-i][j][0]*t1[-i][j][0]+s[-i][j][0]*a[-i][j][0];
            a[-i][j][0]    = -s[-i][j][0]*t1[-i][j][0]+c[-i][j][0]*a[-i][j][0];
            c[-i][j+1][0]  = c[-i][j][0];
            s[-i][j+1][0]  = s[-i][j][0];

        }
    }
  }
}
```

Listado 2.9: Programa después de efectuar asignación única de las variables r c y s y eliminación de comunicación global de c y s

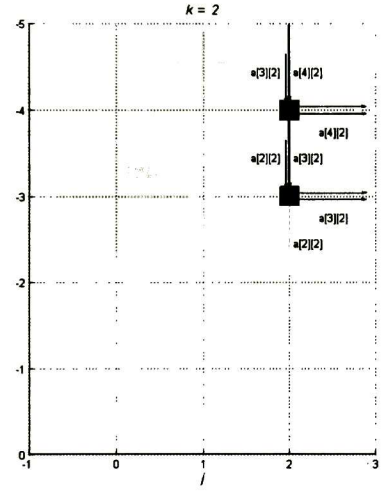
Asignación única de variables r . c y s y eliminación de comunicación global de c y s . En cada recorrido del ciclo i cada fila de a, excepto la última y la primera, es actualizada dos veces, el programa no es de asignación única, sin embargo el programa puede ser ejecutado como un programa secuencial. En el grafo de dependencia de la figura 2.9, correspondiente al programa del listado 2.9, se observa que los elementos de las matrices parciales, definidas por cada iteración del índice k, $A^{\langle k \rangle}$ resaltados con una flecha de color azul, son los resultados esperados al aplicar la descomposición QR. Las flechas denotadas con color verde son los elementos cuyos valores se han de procesar en la siguiente iteración de k.



a) Iteración $k=0$.



b) Iteración $k=1$.



c) Iteración $k=2$.

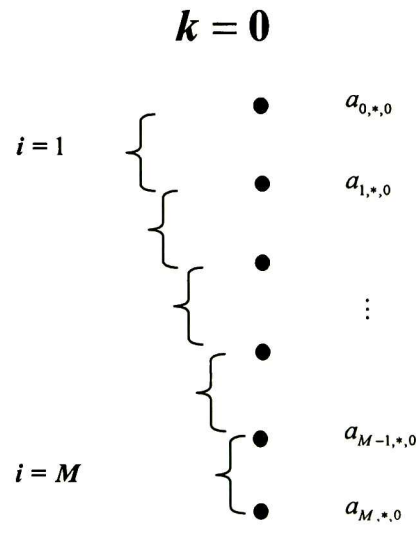
Figura 2.9: Grafo de dependencia de programa con asignación única de las variables r c y s y eliminación de comunicación global de c y s. $(M=4, N=3)$.

2.5.6 Asignación Unica

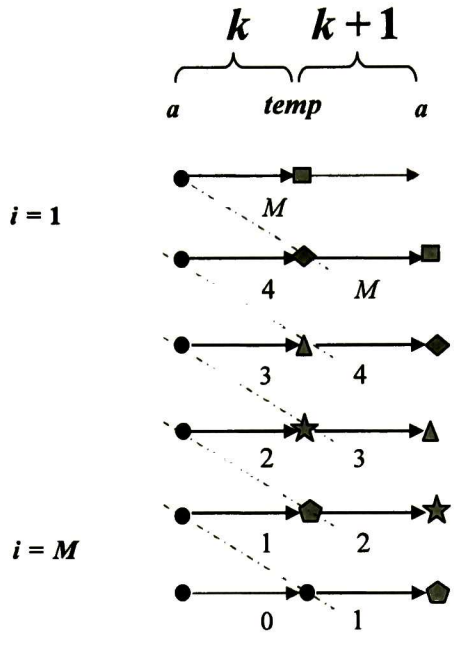
Con el objeto de eliminar la doble actualización al arreglo a introducimos un arreglo temporal denotado como ***temp***. Durante cada iteración de i sucede una actualización de a y una de $temp$, este proceder es regular excepto en la primera y última iteración de i. En la primera iteración se actualizan dos elementos de $temp$ y uno de a, en la última iteración se actualizan dos elementos de a y uno de $temp$.

Como resultado de la iteración $k=0$, dado el arreglo $a[i][j][0]$ se produce el arreglo $a[i][j][1]$ para $0 \le i \le M$ y $0 \le j \le N-1$; para la iteración $k=1$, dado el arreglo $a[i][j][1]$ se produce el arreglo $a[i][j][2]$ para $1 \le i \le M$ y $1 \le j \le N-1$. En general, dado el arreglo $a[i][j][k]$, resultado de una iteración del índice k, se produce el arreglo $a[i][j][k+1]$ para $-M \le i \le -k-1$ y $k \le j \le N-1$.

Se procede de manera similar eliminando la primera fila y la primera columna por cada iteración de k hasta llegar al último elemento de la diagonal de la matriz A. Las figuras 2.10 y 2.11 muestran las dos primeras iteraciones del programa de asignación única que usa el arreglo temporal $temp$.



a)

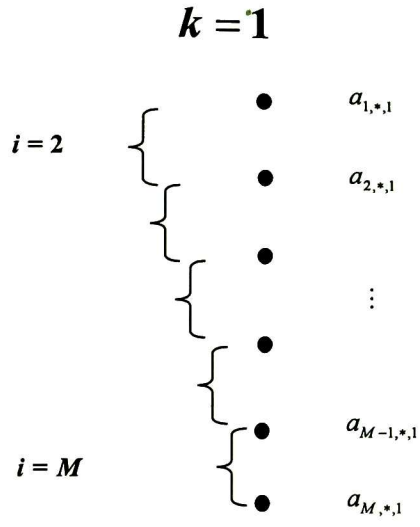


b)

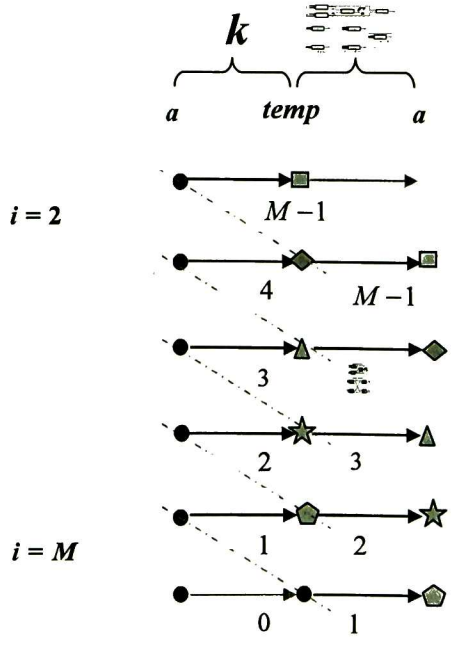
$$
\begin{aligned}
&\text{prólogo} \quad \{ \quad temp_{M,*,0} =^{0} a_{M,*,0} \\
&i = M \quad \begin{cases} a_{M,*,1} =^{1} -s * a_{M-1,*,0} + c * temp_{M,*,0} \\ temp_{M-1,*,1} =^{1} c * a_{M-1,*,0} + s * temp_{M,*,0} \end{cases} \\
&i = M-1 \begin{cases} a_{M-1,*,1} =^{2} -s * a_{M-2,*,0} + c * temp_{M-1,*,0} \\ temp_{M-2,*,1} =^{2} c * a_{M-2,*,0} + s * temp_{M-1,*,0} \end{cases} \\
&i = 1 \quad \begin{cases} a_{1,*,1} =^{M} -s * a_{0,*,0} + c * temp_{1,*,0} \\ temp_{0,*,0} =^{M} c * a_{0,*,0} + s * temp_{1,*,0} \end{cases} \\
&\text{epílogo} \quad \{ \quad a_{0,*,1} = temp_{0,*,0}
\end{aligned}
$$

c)

Figura 2.10: a) Filas de A en la iteración $k = 0$. **b)** Cálculos sobre filas, un cálculo se guarda en un elemento del arreglo *temp* y el otro en un elemento de arreglo a. Excepto en la primera y la última fila. **c)** Transformación de la matriz A en la iteración $k = 0$.



a)



b)

$$
\begin{aligned}
&\text{prólogo} \quad \{ \quad temp_{M,*,1} =^{0} a_{M,*,1} \\
&i = M \quad \begin{cases} a_{M,*,2} =^{1} -s * a_{M-1,*,1} + c * temp_{M,*,1} \\ temp_{M-1,*,1} =^{1} c * a_{M-1,*,1} + s * temp_{M,*,1} \end{cases} \\
&i = M-1 \begin{cases} a_{M-1,*,2} =^{2} -s * a_{M-2,*,1} + c * temp_{M-1,*,1} \\ temp_{M-2,*,1} =^{2} c * a_{M-2,*,1} + s * temp_{M-1,*,1} \end{cases} \\
&i = 1 \quad \begin{cases} a_{2,*,2} =^{M-1} -s * a_{1*,1} + c * temp_{2,*,1} \\ temp_{1,*,1} =^{M-1} c * a_{1,*,1} + s * temp_{2,*,1} \end{cases} \\
&\text{epílogo} \quad \{ \quad a_{1,*,2} = temp_{1,*,1}
\end{aligned}
$$

c)

Figura 2.11: a) Filas de A en la iteración $k = 1$. **b)** Cálculos sobre filas, un cálculo se guarda en un elemento del arreglo *temp* y el otro en un elemento de arreglo a. Excepto en la primera y la última fila. **c)** Transformación de la matriz A en la iteración $k = 1$.

La figura 2.11, muestra los cálculos sobre la matriz A en la iteración $k = 1$. La transformación de asignación única, con la inclusión del arreglo temporal *temp*, se muestra en el programa del listado 2.10.

```
for (k=0;k<N; k++){                          // barre la diagonal
    for (i=-M; i>=-k-1; i++){                // barre las filas
        for (j=k; j<N; j++){                 // barre las columnas
            if (j==k){                       // Generación de Rotaciones de Givens (GG)
              if(i==-M){
                r[-i][j][k]= a[-i][j][k];
              }
              r[-i-1][j][k]=(sqrt( pow(a[-i-1][j][k],2.0)+ pow(r[-i][j][k],2.0)));
              if(r[-i-1][j][k]==0){
                c[-i][j+1][k]=1;
                s[-i][j+1][k]=0;
              }else{
                c[-i][j+1][k]=(a[-i-1][j][k])/r[-i-1][j][k];
                s[-i][j+1][k]=(a[-i][j][k])/r[-i-1][j][k];
              }
              if(i ==-k-1){
              a[-i-1][j][k+1]= r[-i-1][j][k];
              }
            }else{                           // Ejecución de Rotaciones de Givens (RG)
              if(i==-M){
                temp[-i][j][k]= a[-i][j][k];
              }
              temp[-i-1][j][k]= c[-i][j][k]*a[-i-1][j][k]+s[-i][j][k]*temp[-i][j][k];
              a[-i][j][k+1]   = -s[-i][j][k]*a[-i-1][j][k]+c[-i][j][k]*temp[-i][j][k];
              if(i ==-k-1){
                a[-i-1][j][k+1]= temp[-i-1][j][k];
              }
              c[-i][j+1][k] = c[-i][j][k];
              s[-i][j+1][k] = s[-i][j][k];
            }
        }
    }
}
```

Listado 2.10: Programa de la descomposición QR después de aplicar la transformación de asignación única.

Las variables del programa de asignación única están representadas por las aristas del grafo de dependencia. La figura 2.12 muestra el grafo de dependencia en tres dimensiones, después de aplicar la transformación de asignación única al código de descomposición QR.

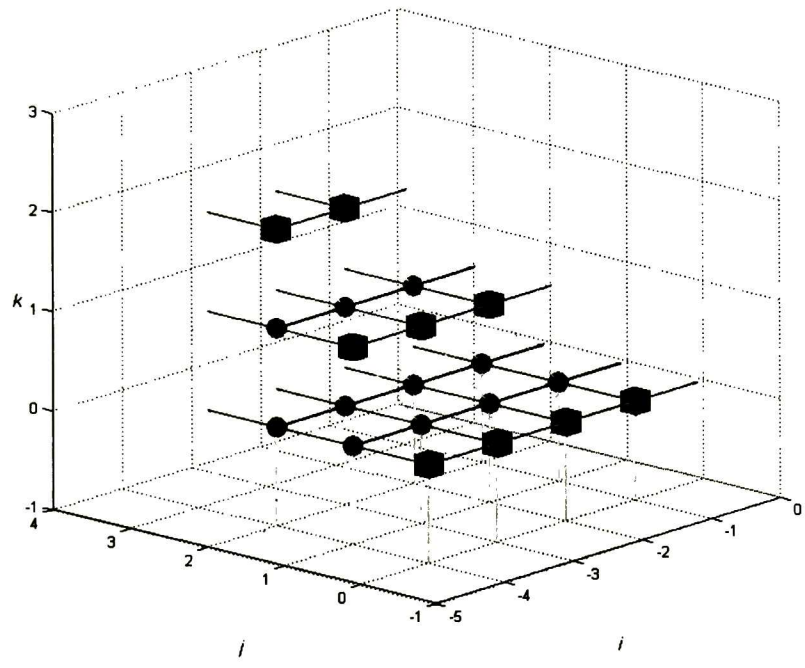


Figura 2.12: Grafo de dependencia del programa de asignación única de la descomposición QR, caso $(M=4, N=3)$.

Las entradas y salidas de los nodos del grafo de dependencia de la figura 2.12 son como sigue:

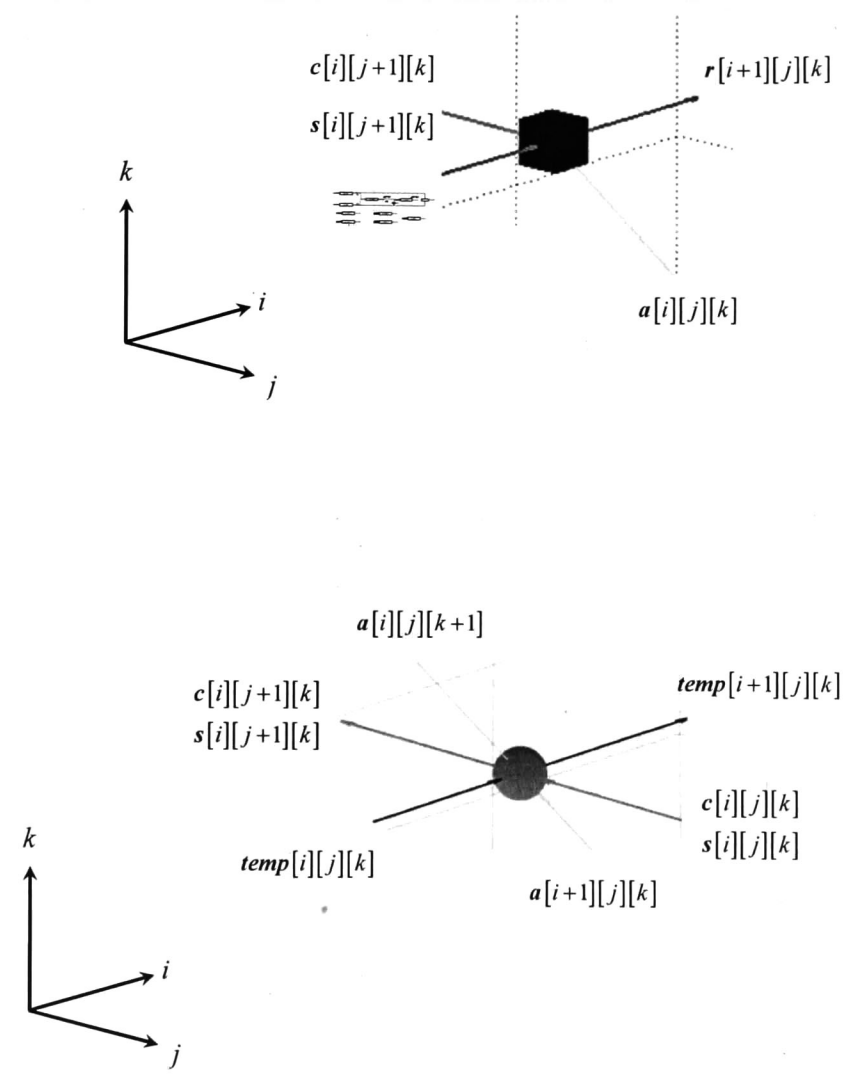


Figura 2.13: Nodos del grafo de dependencia del programa de asignación única.

A continuación mediante un ejemplo se muestra la ejecución del código de asignación única, de la descomposición QR.

Ejemplo 2.2. Sea la matriz de entrada A de orden $(M+1)\times N$ donde $M=4,\ N=3$.

$$A=\begin{bmatrix} 12 & -51 & 4 \\ 6 & 167 & -68 \\ -4 & 24 & -41 \\ 7 & 8 & 9 \\ 0 & 0 & 0 \end{bmatrix}$$

A continuación se muestran los resultados de las matrices parciales $A^{(k)}$. derivados en cada iteración de k.

Iteración $k=0$:

$$A^{(0)}=\begin{bmatrix} 15.6524 & 22.3606 & -8.497 \\ 0 & 106.1311 & -16.3757 \\ 0 & -136.9335 & 71.3609 \\ 0 & -24.8069 & 31.1327 \end{bmatrix}$$

Iteración $k=1$:

$$A^{(1)}=\begin{bmatrix} 175.0142 & -70.1771 \\ 0 & -32.9254 \\ 0 & -17.9133 \end{bmatrix}$$

Iteración $k=2$:

$$A^{(2)}=\begin{bmatrix} 37.4829 \\ 0 \end{bmatrix}$$

El objetivo de ejecutar el código de asignación única de la descomposición QR, es obtener la matriz triangular superior R. La matriz R está dada por:

$$R=\begin{bmatrix} 15.6524 & 22.3606 & -8.497 \\ 0 & 175.0142 & -70.1771 \\ 0 & 0 & 37.4829 \\ 0 & 0 & 0 \end{bmatrix}$$

2.6 Politopo Destino

La matriz de transformación *espacio temporal* $\boldsymbol{T}$ mapea el espacio de índices *espacial* $\boldsymbol{x}=[i,j,k]^t$ a un nuevo espacio de índices *temporal* y *espacial* $y=[t,p_1,p_2]^t$ En lo siguiente se presenta la matriz de transformación *espacio temporal* $\boldsymbol{T}$ la cual define el tiempo de ejecución y efectúa el mapeo hacia un arreglo de procesadores en dos dimensiones $(2D)$.

$$\boldsymbol{T}=\begin{bmatrix}\underline{\boldsymbol{s}}\\ \boldsymbol{P}\end{bmatrix} \tag{2.20}$$

La matriz de transformación lineal $\boldsymbol{T}$ se conforma, en la primera fila por el vector de calendarización denotado:

$$\boldsymbol{s}=[1 \quad 1 \quad 2]^t$$

las filas siguientes representan la matriz de asignación denotada por:

$$\boldsymbol{P}=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

La matriz $\boldsymbol{P}$ está formada por los vectores ortogonales al vector de proyección de asignación de procesador, el cual está dado por:

$$\boldsymbol{d}_i=[1 \quad 0 \quad 0]^t$$

que se lee vector de proyección sobre el eje i. Esta matriz de transformación *espacio temporal* $\boldsymbol{T}$ es una matriz unimodular, es decir es una matriz cuadrada de enteros con determinante +1 ó -1.

$$\boldsymbol{T}=\begin{bmatrix}\underline{\boldsymbol{s}}\\ \boldsymbol{P}\end{bmatrix}=\begin{bmatrix}\underline{1} & \underline{1} & \underline{2}\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$\det(\boldsymbol{T})=1$$

La matriz de transformación *espacio temporal* $\boldsymbol{T}$, cumple la relación:

$$\boldsymbol{Tx}=\boldsymbol{y} \tag{2.21}$$

Donde el vector x representa el espacio de índices $x=[i \quad j \quad k]^t$ del politopo fuente (ver listado 2.10, programa de "asignación única") y el vector y el nuevo espacio de índices definido $y=[t \quad p_1 \quad p_2]^t$ del politopo destino [18].

Aplicando la relación (2.21):

$$\overbrace{\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{T} \overbrace{\begin{bmatrix} i \\ j \\ k \end{bmatrix}}^{x} = \overbrace{\begin{bmatrix} t \\ p_1 \\ p_2 \end{bmatrix}}^{y}$$

Se obtiene lo siguiente:

$$\boldsymbol{x} = \boldsymbol{T}^{-1} y \tag{2.22}$$

$$\overbrace{\begin{bmatrix} i \\ j \\ k \end{bmatrix}}^{x} = \overbrace{\begin{bmatrix} 1 & -1 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{T^{-1}} \overbrace{\begin{bmatrix} t \\ p_1 \\ p_2 \end{bmatrix}}^{y}$$

Luego entonces:

$$\begin{aligned} i &= t - p_1 - 2p_2 \\ j &= p_1 \\ k &= p_2 \end{aligned} \tag{2.23}$$

La relación (2.23) define los valores de x en función del nuevo espacio de iteración determinado por y, en el código del politopo destino.

De los límites inferior y superior de los bucles perfectamente anidados presentes en el listado 2.10, se determina el conjunto de desigualdades que define al politopo fuente, que están dadas por:

$$\begin{aligned} -M \le i \le -k-1 \\ k \le j \le N-1 \\ 0 \le k \le N-1 \end{aligned}$$

Esta información se puede reescribir de la forma:

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{2.24}$$

La relación (2.24) es la representación matricial del sistema de desigualdades, que define al politopo fuente, en función del espacio de iteración (i, j, k).

$$\overbrace{\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} i \\ j \\ k \end{bmatrix}}^{x} \leq \overbrace{\begin{bmatrix} M \\ 0 \\ 0 \\ -1 \\ N-1 \\ N-1 \end{bmatrix}}^{y}$$

limite inferior de i	$i \geq -M$
limite inferior de j	$j \geq k$
limite inferior de k	$k \geq 0$
limite superior de i	$i \leq -k-1$
limite superior de j	$j \leq N-1$
limite superior de k	$k \leq N-1$

El politopo destino está descrito por el conjunto de desigualdades de la relación (2.25), resultado de las relaciones (2.24) y (2.22).

$$\left(\boldsymbol{A}\boldsymbol{T}^{-1}\right)\boldsymbol{y} \leq \boldsymbol{b} \tag{2.25}$$

$$\overbrace{\begin{bmatrix} -1 & 1 & 2 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{(A\cdot T^{-1})} \overbrace{\begin{bmatrix} t \\ p_1 \\ p_2 \end{bmatrix}}^{y} \leq \overbrace{\begin{bmatrix} M \\ 0 \\ 0 \\ -1 \\ N-1 \\ N-1 \end{bmatrix}}^{b}$$

Resolviendo el sistema de desigualdades de la relación (2.25) por el método de eliminación de *Fourier-Motzkin* [1] se obtienen los límites inferior y superior del nuevo espacio de iteración $\boldsymbol{y}$, del programa del politopo destino:

$$-M \le t \le 2*N-4$$

$$\max\left\lceil 0, t-M+4, \frac{1}{2}(t+2), t-N+3 \right\rceil \le p_1 \le \min(t+M, N-1)$$

$$\max(0, t-p_1+2) \le p_2 \le \min\left\lfloor \left(p_1, N-1, \frac{1}{2}(t-p_1+M) \right) \right\rfloor$$

El programa correspondiente al politopo destino está dado en el listado 2.11.

```
for (t = -M; t <= 2*N-4; t++){
  for( p1 = ceil( max(0, t-M+4, (1/2)(t+2), t-N+3) ); p1 <= min(t+M, N-1); p1++ ){
    for( p2 = max(0, t-p1+2); p2 <= floor( min( p1, N-1, (1/2)(t-p1+M) ) ); p2++ ){
      if(p1 == p2){                    // Generaciones de Givens (GG)
        if(t-p1-2*p2 == -M){
          r[-t+p1+2*p2][p1][p2] = a[-t+p1+2*p2][p1][p2];
        }
        r[-t+p1+2*p2-1][p1][p2]=sqrt( pow(a[-t+p1+2*p2-1][p1][p2],2.0)+
                                      pow(r[-t+p1+2*p2][p1][p2],2.0));
        if(r[-t+p1+2*p2-1][p1][p2] == 0){
          c[-t+p1+2*p2][p1+1][p2] = 1;
          s[-t+p1+2*p2][p1+1][p2] = 0;
        }else{
          c[-t+p1+2*p2][p1+1][p2] = (a[-t+p1+2*p2-1][p1][p2])/r[-t+p1+2*p2-1][p1][p2];
          s[-t+p1+2*p2][p1+1][p2] = (a[-t+p1+2*p2][p1][p2])/r[-t+p1+2*p2-1][p1][p2];
        }
        if(t-p1-2*p2 == -p2-1){
          a[-t+p1+2*p2-1][p1][p2+1] = r[-t+p1+2*p2-1][p1][p2];
        }
      }else{                           // Rotaciones de Givens (GR)
        if(t-p1-2*p2 == -M){
          temp[-t+p1+2*p2][p1][p2] = a[-t+p1+2*p2][p1][p2];
        }
        temp[-t+p1+2*p2-1][p1][p2] = c[-t+p1+2*p2][p1][p2]*a[-t+p1+2*p2-1][p1][p2]+
                                     s[-t+p1+2*p2][p1][p2]*temp[-t+p1+2*p2][p1][p2];
        a[-t+p1+2*p2][p1][p2+1]    = -s[-t+p1+2*p2][p1][p2]*a[-t+p1+2*p2-1][p1][p2]+
                                     c[-t+p1+2*p2][p1][p2]*temp[-t+p1+2*p2][p1][p2];
        if(t-p1-2*p2 == -p2-1){
          a[-t+p1+2*p2-1][p1][p2+1] = temp[-t+p1+2*p2-1][p1][p2];
        }
        c[-t+p1+2*p2][p1+1][p2] = c[-t+p1+2*p2][p1][p2];
        s[-t+p1+2*p2][p1+1][p2] = s[-t+p1+2*p2][p1][p2];
      }
    }
  }
}
```

Listado 2.11: Programa de la descomposición *QR* después de aplicar la transformación *espacio temporal*.

El listado 2.12 muestra el programa después de la transformación espacio temporal pero en esta ocasión las funciones afines de acceso a los arreglos no se han modificado con respecto al programa original. En lugar de esto se tienen reasignaciones de los índices [36].

```
for (t = -M; t <= 2*N-4; t++){
  for( p1 = ceil( max(0, t-M+4, 1/2(t+2), t-N+3)); p1 <= min(t+M, N-1); p1++){
    for( p2 = max(0, t-p1+2); p2 <= floor( min( p1, N-1, 1/2(t-p1+M))); p2++){

      // Asignación
        i = t - p1 - 2*p2;
        j = p1;
        k = p2;

      if (j == k){              // Generaciones de Givens (GG)

        if (i == -M){
          r[-i][j][k] = a[-i][j][k];
        }

        r[-i-1][j][k]=(sqrt( pow(a[-i-1][j][k],2.0) + pow(r[-i][j][k],2.0)));

        if (r[-i-1][j][k] == 0){
          c[-i][j+1][k] = 1;
          s[-i][j+1][k] = 0;
        }else{
          c[-i][j+1][k] = (a[-i-1][j][k]) / r[-i-1][j][k];
          s[-i][j+1][k] = (a[-i][j][k]) / r[-i-1][j][k];
        }

        if (i == -k-1){
          a[-i-1][j][k+1] = r[-i-1][j][k];
        }
```

```
}else{              // Rotaciones de Givens (RG)

    if (i == -M){
      temp[-i][j][k] = a[-i][j][k];
    }

    temp[-i-1][j][k] =  c[-i][j][k]*a[-i-1][j][k]+s[-i][j][k]*temp[-i][j][k];
    a[-i][j][k+1]    = -s[-i][j][k]*a[-i-1][j][k]+c[-i][j][k]*temp[-i][j][k];

    if (i == -k-1){
      a[-i-1][j][k+1] = temp[-i-1][j][k];
    }

    c[-i][j+1][k] = c[-i][j][k];
    s[-i][j+1][k] = s[-i][j][k];

  }
 }
}
}
```

Listado 2.12: Programa de la descomposición QR después de aplicar la transformación espacio temporal.

Consideramos realizar los siguientes cambios a los índices (t, p_1, p_2), para una visualización más clara de la información contenida en la transformación *espacio temporal*. En particular aplicamos una transformación de traslación de tal suerte que ahora el tiempo sólo toma valores no negativos.

```
for (t = 0; t <= M + 2*N - 4; t++){
  for( p1 = ceil( max(0, t - 2*M + 4, (1/2)(t - M + 2), t - M - N + 3) ); p1 <= min(t, N - 1); p1++ ){
    for( p2 = max(0, t - p1 - M + 2); p2 <= floor( min( p1, N - 1, (1/2)(t - p1) ) ); p2++ ){
      if (p1 == p2){      // if GG / RG      // Generación de Rotaciones de Givens (GG)
        if (t == p1 + 2*p2){      // if.1.GG
          r[-t + p1 + 2*p2 - M][p1][p2] = a[-t + p1 + 2*p2 - M][p1][p2];
        }
        r[-t + p1 + 2*p2 - 1 - M][p1][p2] = sqrt( pow(a[-t + p1 + 2*p2 - 1 - M][p1][p2], 2.0) +
                                                  pow(r[-t + p1 + 2*p2 - M][p1][p2], 2.0) );
        if (r[-t + p1 + 2*p2 - 1 - M][p1][p2] == 0){
          c[-t + p1 + 2*p2 - M][p1 + 1][p2] = 1;
          s[-t + p1 + 2*p2 - M][p1 + 1][p2] = 0;
        }else{
          c[-t + p1 + 2*p2 - M][p1 + 1][p2] = (a[-t + p1 + 2*p2 - 1 - M][p1][p2]) / r[-t + p1 + 2*p2 - 1 - M][p1][p2];
          s[-t + p1 + 2*p2 - M][p1 + 1][p2] = (a[-t + p1 + 2*p2 - M][p1][p2]) / r[-t + p1 + 2*p2 - 1 - M][p1][p2];
        }
        if (t == p1 + p2 + M - 1){      // if.2.GG
          a[-t + p1 + 2*p2 - 1 - M][p1][p2 + 1] = r[-t + p1 + 2*p2 - 1 - M][p1][p2];
        }
      }else{      // Ejecución de Rotaciones de Givens (GR)
        if (t == p1 + 2*p2){      // if.1.RG
          temp[-t + p1 + 2*p2 - M][p1][p2] = a[-t + p1 + 2*p2 - M][p1][p2];
        }
        temp[-t + p1 + 2*p2 - 1 - M][p1][p2] = c[-t + p1 + 2*p2 - M][p1][p2] * a[-t + p1 + 2*p2 - 1 - M][p1][p2] +
                                               s[-t + p1 + 2*p2 - M][p1][p2] * temp[-t + p1 + 2*p2 - M][p1][p2];
        a[-t + p1 + 2*p2 - M][p1][p2 + 1]     = -s[-t + p1 + 2*p2 - M][p1][p2] * a[-t + p1 + 2*p2 - 1 - M][p1][p2] +
                                                c[-t + p1 + 2*p2 - M][p1][p2] * temp[-t + p1 + 2*p2 - M][p1][p2];
        if (t == p1 + p2 + M - 1){      // if.2.RG
          a[-t + p1 + 2*p2 - 1 - M][p1][p2 + 1] = temp[-t + p1 + 2*p2 - 1 - M][p1][p2];
        }
        c[-t + p1 + 2*p2 - M][p1 + 1][p2] = c[-t + p1 + 2*p2 - M][p1][p2];
        s[-t + p1 + 2*p2 - M][p1 + 1][p2] = s[-t + p1 + 2*p2 - M][p1][p2];
      }
    }
  }
}
```

Listado 2.13: Programa de la descomposición QR después de aplicar la transformación espacio temporal.

Las figuras 2.14, 2.15 y 2.16 muestran diferentes perspectivas del grafo de dependencia del código después de aplicada la transformación *espacio temporal*, el caso mostrado supone $M = 4$ y $N = 3$.

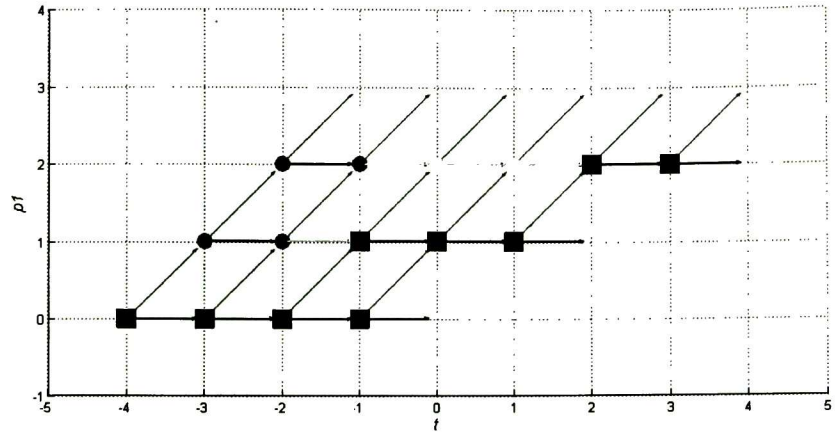


Figura 2.14: Grafo de dependencia del código de la transformación *espacio temporal*, proyectado en el plano t y p_1.

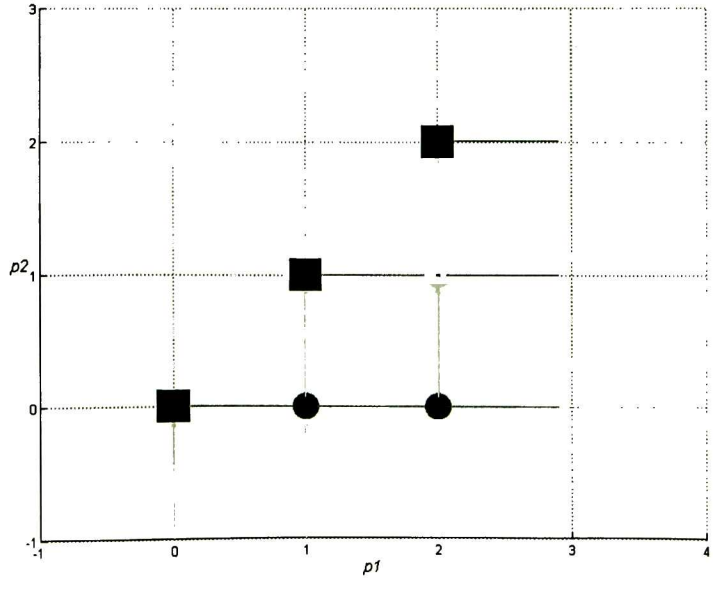


Figura 2.15: Grafo de dependencia del código de la transformación espacio temporal, proyectado en el plano p_1 y p_2.

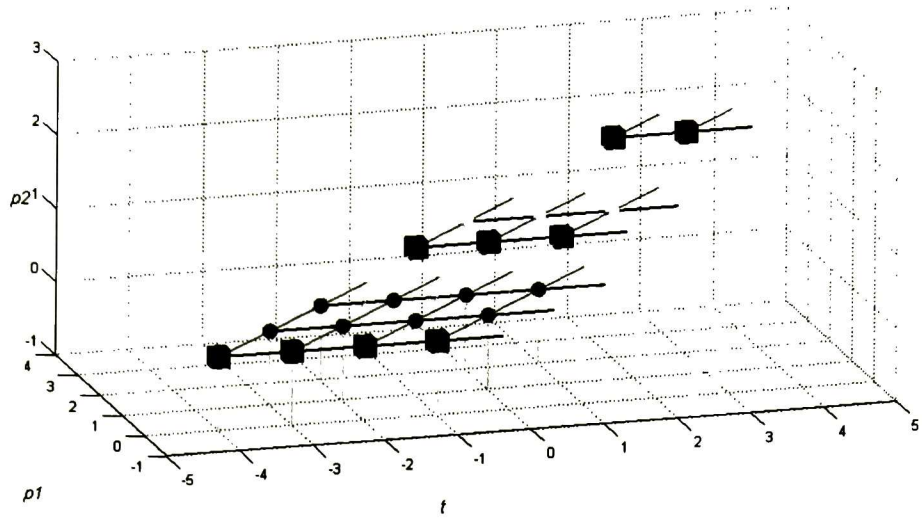


Figura 2.16: Grafo de dependencia del código de la transformación *espacio temporal*, visto en el espacio t, p_1 y p_2.

2.7 Grafo de Flujo de Señales

Los vectores de dependencia de datos, obtenidos de la figura 2.12 son:

$$\begin{aligned} \boldsymbol{s} &= [0 \quad 1 \quad 0]^t \\ \boldsymbol{c} &= [0 \quad 1 \quad 0]^t \\ \boldsymbol{r} &= [1 \quad 0 \quad 0]^t \\ \boldsymbol{temp} &= [1 \quad 0 \quad 0]^t \\ \boldsymbol{a} &= [-1 \quad 0 \quad 1]^t \end{aligned}$$

la matriz de dependencia de datos $\boldsymbol{D}$ es:

$$\boldsymbol{D} = [\boldsymbol{s} \quad \boldsymbol{c} \quad \boldsymbol{r} \quad \boldsymbol{temp} \quad \boldsymbol{a}] = \begin{bmatrix} 0 & 0 & 1 & 1 & -1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

y la matriz de transformación *espacio temporal* $\boldsymbol{T}$ definida en la relación (2.20) :

$$\boldsymbol{T} = \begin{bmatrix} \underline{\boldsymbol{s}} \\ \boldsymbol{P} \end{bmatrix} = \begin{bmatrix} \underline{1} & \underline{1} & \underline{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Los conceptos citados permiten obtener, los nuevos vectores de dependencia de datos mediante:

$$\boldsymbol{\varphi}' = \boldsymbol{T} * \boldsymbol{\varphi} = \begin{bmatrix} \underline{t} \\ \zeta \end{bmatrix} \begin{matrix} \} \\ \} \end{matrix} \begin{matrix} \textit{tiempo de calendarización} \\ \textit{asignación de procesador} \end{matrix}$$

donde φ denota el vector de dependencia en el grafo de dependencia original y φ' el vector de dependencia en el grafo de dependencia destino [23]. Los vectores de dependencia del grafo destino son:

$$s' = T * s = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{1} \\ 1 \\ 0 \end{bmatrix} \begin{matrix} Retardo \\ j \\ k \end{matrix}$$

$$c' = T * c = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{1} \\ 1 \\ 0 \end{bmatrix} \begin{matrix} Retardo \\ j \\ k \end{matrix}$$

$$r' = T * r = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{1} \\ 0 \\ 0 \end{bmatrix} \begin{matrix} Retardo\ Autolazo \\ j \\ k \end{matrix}$$

$$temp' = T * temp = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \underline{1} \\ 0 \\ 0 \end{bmatrix} \begin{matrix} Retardo\ Autolazo \\ j \\ k \end{matrix}$$

$$a' = T * a = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \underline{1} \\ 0 \\ 1 \end{bmatrix} \begin{matrix} Retardo \\ j \\ k \end{matrix}$$

La nueva matriz de dependencia de datos está dada por:

$$D' = [s' \quad c' \quad r' \quad temp' \quad a'] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

La matriz de transformación del politopo original al politopo destino se puede descomponer como a continuación explicamos. La primera fila de la matriz se puede interpretar como el vector de calendarización τ Recordamos que el vector de calendarización cumple con:

$$\tau * (i, j, k) = t = [1 \quad 1 \quad 2] \begin{bmatrix} i \\ j \\ k \end{bmatrix} = i + j + 2k$$

donde (i, j, k) es un punto en el espacio del politopo fuente y t es el tiempo de ejecución de las operaciones de la iteración (i, j, k). Las filas restantes definen un

espacio vectorial y la función π es perpendicular a éste, véase la referencia de Song, Siang, W., en la bibliografía para una derivación.

Las filas restantes definen una matriz P tal que:

$$P * x = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} j \\ k \end{bmatrix}$$

donde (i, j, k) es un punto en el espacio del politopo fuente y (p_1, p_2) son las coordenadas donde se encuentra el procesador que ejecuta la operación de la iteración (i, j, k). Otra manera de efectuar la proyección es usar el vector de proyección de asignación de procesador π, recordamos que π es un vector perpendicular al espacio definido por las filas de la matriz P. La función de asignación π cumple con:

$$Proy_{\pi}(i, j, k) = (p_1, p_2)$$

donde $Proy_{\pi}$ es el operador de proyección en la dirección de π y de igual manera que antes (i, j, k) es un punto en el espacio del politopo fuente y (p_1, p_2) son las coordenadas donde se encuentra el procesador que ejecuta la operación de la iteración (i, j, k) [26]. En la figura 2.17 se observa la topología de interconexión de los elementos procesadores y la figura 2.18 muestra el grafo de flujo de señales.

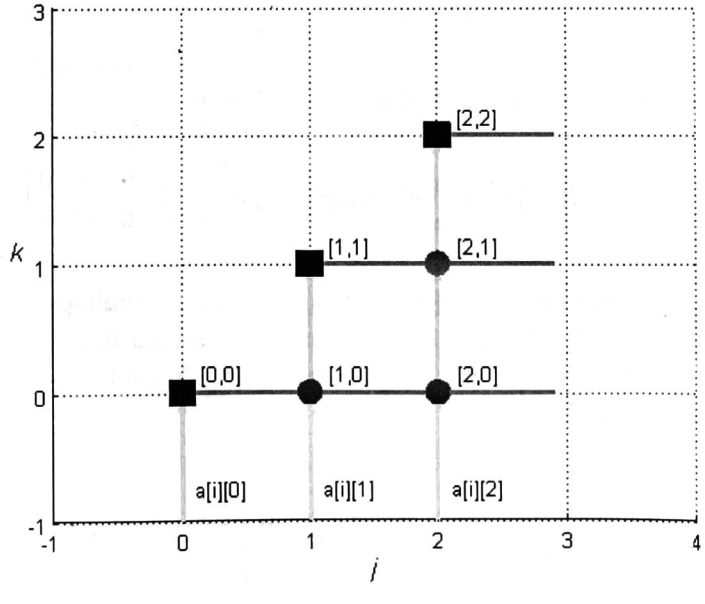
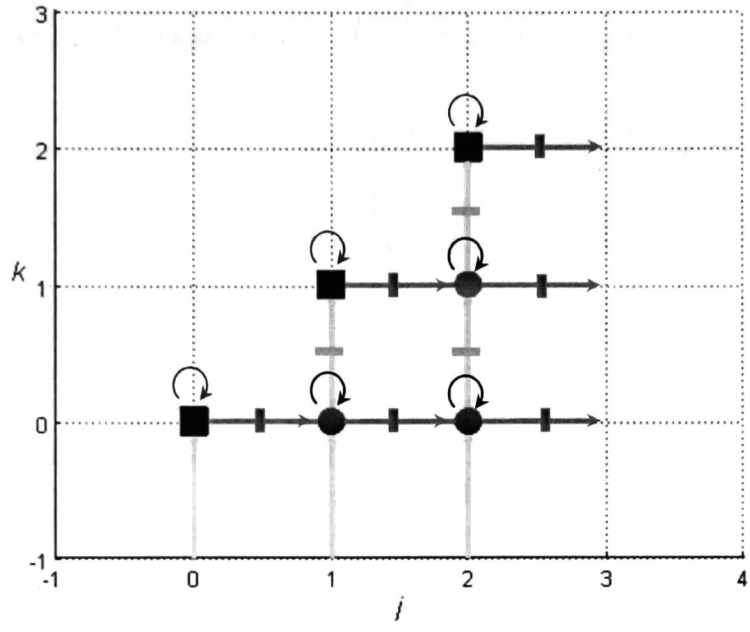


Figura 2.17: Grafo de la topología de interconexión del arreglo de procesadores.



$a[3][0]$	0	0
$a[2][0]$	$a[3][1]$	0
$a[1][0]$	$a[2][1]$	$a[3][2]$
$a[0][0]$	$a[1][1]$	$a[2][2]$
0	$a[0][1]$	$a[1][2]$
0	0	$a[0][2]$

Figura 2.18: Grafo de flujo de señales de la descomposición QR, dado el vector de proyección de asignación de procesador π, $(M=4, N=3)$.

La tabla 2.1 muestra el tiempo de activación de cada uno de los elementos procesadores, donde el número asignado a cada procesador corresponde a su ubicación sobre los ejes coordenados, ver figura 2.17. De la tabla se observa que se requieren siete ciclos de reloj para la ejecución del algoritmo por el arreglo de procesadores, con una utilización de los elementos procesadores de $\eta_{\pi} = \frac{10}{21}$. Como el lector puede comprobar, la proyección en la dirección de π proporciona una utilización mayor de los elementos procesadores en comparación con proyecciones en otras direcciones.

t	*Procesador físico*						η
	00	*10*	*20*	*11*	*21*	*22*	
0	*1*						1/6
1	*1*	*1*					1/3
2	*1*	*1*	*1*				1/2
3	*1*	*1*	*1*	*1*			2/3
4		*1*	*1*	*1*	*1*		2/3
5			*1*	*1*	*1*		1/2
6					*1*	*1*	1/3
7						*1*	1/6
						η_π	10/21

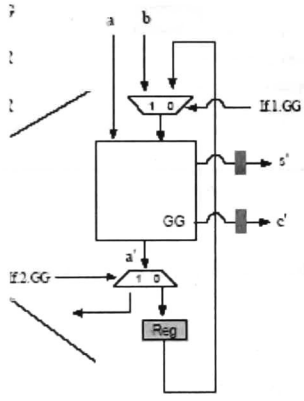
Tabla 2.1: Tiempo de activación de cada elemento procesador, dado el vector de proyección de asignación de procesador π

2.8 Extracción de Información para la Implementación

En esta sección se describen la trayectoria de datos y la generación de señales de control necesarios para la implementación del arreglo bidimensional de procesadores de la descomposición QR.

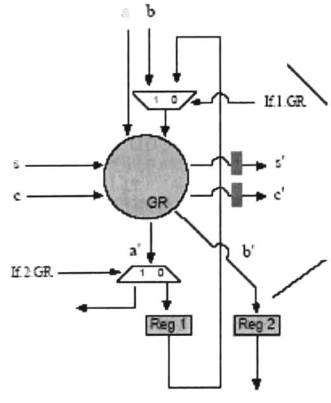
2.8.1 Trayectoria de Datos

La trayectoria de datos está dada por el grafo de flujo de señales de la figura 2.18, en el cual observamos dos tipos de elementos procesadores. La figura 2.19 muestra las funciones internas de cada tipo de elemento procesador.



$$r = \sqrt{a^2 + b^2}$$
$$c' = \frac{a}{r}$$
$$s' = \frac{a}{r}$$
$$a' = r$$

a) Elemento procesador que efectúa la *generación de las rotaciones de Givens* (*GG*).



$$a' = c * a + s * b$$
$$b' = -s * a + c * b$$

b) Elemento procesador que efectúa la *ejecución de las rotaciones de Givens* (*RG*).

Figura 2.19: Elementos procesadores presentes en la descomposición QR.

2.8.2 Generación de Señales de Control

Las señales de control para su estudio se dividen en: *señales de control de habilitación* y *señales de control funcional.*

2.8.2.1 Control de Habilitación

El control de habilitación calendariza la activación de cada elemento procesador del arreglo de procesadores físico. La ventaja de su aplicación es un ahorro considerable en el consumo de potencia del arreglo. La figura 2.20 muestra el diagrama de Gantt de la activación de los elementos procesadores del arreglo bidimensional que efectúa la descomposición QR. La tabla 2.2 es la tabla de activación de los procesadores. Si observamos con cuidado vemos que la tabla 2.2 y la figura 2.20 contienen la misma información.

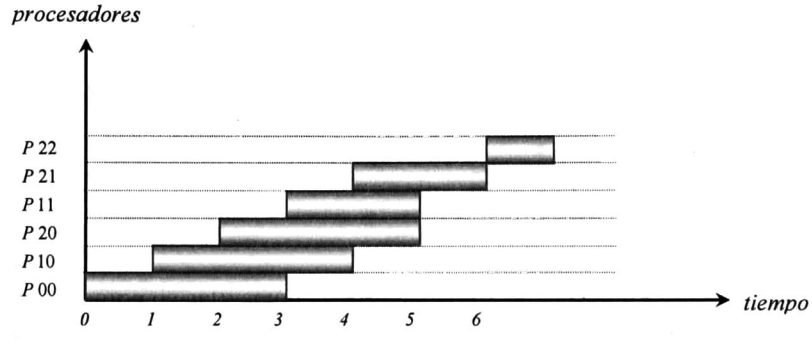


Figura 2.20: Diagrama de Gantt del arreglo paralelo de procesadores físicos que efectúan la descomposición QR.

t	*Procesador físico*					
	00	*10*	*20*	*11*	*21*	*22*
0	*1*					
1	*1*	*1*				
2	*1*	*1*	*1*			
3	*1*	*1*	*1*	*1*		
4		*1*	*1*	*1*	*1*	
5			*1*	*1*	*1*	
6					*1*	*1*
7						*1*

Tabla 2.2: Tiempo de activación de los elementos procesadores del *arreglo bidimensional* que efectúa la descomposición QR.

2.8.2.2 Control Funcional

El control funcional especifica la función que el elemento procesador efectúa al momento de estar habilitado. Las siguientes tablas contienen la información correspondiente a la activación de cada condicional, véase el listado 2.13 para observar a que condicional se refiere cada tabla.

t	*Procesador físico*					
	00	*10*	*20*	*11*	*21*	*22*
0	*1*					
1						
2						
3				*1*		
4						
5						
6						*1*
7						

Tabla 2.3: Tiempo de activación de la condicional *if.GG / RG* condición $(p_1 = p_2)$.

t	*Procesador físico*					
	00	*10*	*20*	*11*	*21*	*22*
0	*1*					
1						
2						
3				*1*		
4						
5						
6						*1*
7						

Tabla 2.4: Tiempo de activación de la condicional *if.1.GG* condición $(t = p_1 + 2 * p_2)$.

t	*Procesador físico*					
	00	*10*	*20*	*11*	*21*	*22*
0						
1						
2						
3	*1*					
4						
5				*1*		
6						
7						*1*

Tabla 2.5: Tiempo de activación de la condicional *if.2.GG* condición $(t = p_1 + p_2 + M - 1)$.

t	*Procesador físico*					
	00	*10*	*20*	*11*	*21*	*22*
0						
1		*1*				
2			*1*			
3						
4					*1*	
5						
6						
7						

Tabla 2.6: Tiempo de activación de la condicional *if.1.RG* condición $(t = p_1 + 2 * p_2)$.

t	Procesador físico					
	00	10	20	11	21	22
0						
1						
2						
3						
4		1				
5			1			
6					1	
7						

Tabla 2.7: Tiempo de activación de la condicional *if.2.RG* condición $(t = p_1 + p_2 + M - 1)$.

La tabla 2.8 condensa la información de las tablas anteriores, muestra el tiempo de activación de cada una de las señales de *control funcional*.

t	Procesador físico					
	00	10	20	11	21	22
0	if.GG/GR if.1.GG					
1		if.1.RG				
2			if.1.RG			
3	if.2.GG			if.GG/GR if.1.GG		
4		if.2.RG			if.1.RG	
5			if.2.RG	if.2.GG		
6					if.2.RG	if.GG/GR if.1.GG
7						if.2.GG

Tabla 2.8: Tiempo de activación de las señales de *control funcional*.

Capítulo 3

Embaldosado

El conocimiento que busca la geometría es el conocimiento de lo eterno.
Platón (Aprox. 429-347 a. C.), La República, VII, 52.

3.1 Introducción

En este capítulo, se expone el desarrollo matemático, para efectuar la transformación de *embaldosado*, en programas modelados en lenguaje C. Realizar embaldosado al espacio de iteración del procesador requiere la aplicación de la transformación de *minería de tiras* y la transformación de *permutación de bucles*. El embaldosado permite la realización, con una menor cantidad de hardware, de una implementación física en la cual se requiera un número considerable de elementos procesadores y módulos de hardware. La aplicación de embaldosado, reduce considerablemente el área física en una implementación de hardware, proporcionando además una utilización mayor en el número de elementos procesadores del arreglo físico (*eficiencia*). Lo anterior, obliga a efectuar tiempos mayores en la ejecución de un algoritmo computacional, con la implicación de considerar de acuerdo a la aplicación específica, un compromiso

respecto a *tiempo*, *área* y *eficiencia*; como se expondrá en este capítulo, con la descomposición QR.

En la sección final de este capítulo se aborda la transformación *espacio temporal* de *multiproyección*, de la cual se obtienen la trayectoria de datos y el controlador de ésta, para la implementación del arreglo de procesadores que efectúa la descomposición QR. En la transformación de multiproyección se consideran los casos de una implementación física: con un número *ilimitado* de elementos procesadores, y con un número *limitado* de elementos procesadores en el arreglo de elementos procesadores.

3.2 Embaldosado

El embaldosado es una transformación sobre los bucles usada para crear automáticamente baldosas de cómputo del algoritmo y con ellas cubrir todo el espacio de iteración. La técnica de embaldosado permite la implementación en hardware de un arreglo de procesadores virtuales con un número menor de procesadores físicos en el arreglo final, permitiendo así la ejecución de problemas que requieren un gran número de procesadores virtuales en un hardware que tiene menor cantidad de procesadores físicos. Las técnicas convencionales de embaldosado combinan dos transformaciones: *minería de tiras* y *permutación de bucles* [37].

La minería de tiras es usada para dividir una dimensión del espacio de iteración en tiras. Descompone un bucle en dos bucles anidados, el bucle externo (*el apuntador de tira*) barre las tiras consecutivas de iteración y el bucle interno (*ejecutor de tira*) ejecuta las iteraciones dentro de una tira. La transformación de minería de tiras siempre es *legal*.

A continuación presentamos un ejemplo de transformación de minería de tiras. Sea el siguiente bucle:

$$
\begin{aligned}
&for\,(I = L; I <= U; I++)\{\\
&\qquad Loop \quad body;\\
&\}
\end{aligned}
$$

El intervalo cerrado de valores de índices enteros que va de L a U se divide en tiras de tamaño B_{II} que empieza en offset definido entre $0 \le offset \le B_{II}$. El siguiente código se obtiene después de aplicar minería de tiras al bucle anterior. Nótese que el índice interior I (*el apuntador de tira*) barre una tira y el índice exterior II (*el ejecutor de tira*) barre las tiras.

$$
\begin{aligned}
&for\left(II = \left\lfloor \frac{(L - offset)}{B_{II}} \right\rfloor * B_{II} + offset; II <= U; II += B_{II}\right)\{\\
&\quad for\left(I = \max(II, L); I <= \min\left((II + B_{II} - 1), U\right); I++\right)\{\\
&\qquad Loop \quad body;\\
&\quad \}\\
&\}
\end{aligned}
$$

La *transformación de permutación de bucles* es usada para establecer el orden en el cual los ciclos de un bucle anidado son barridos. La permutación de bucles es una transformación unimodular que generaliza la transformación de *intercambio de bucles* [34]. La transformación de permutación de bucles no siempre es legal, lo que implica que la transformación de embaldosado no es legal si las permutaciones que la conforman no son legales [14].

La transformación de intercambio de bucles es la transformación que invierte el orden de dos bucles adyacentes perfectamente anidados. La transformación de permutación de bucles, en cambio, permite que más de dos bucles permuten a la vez y no requiere que sean bucles adyacentes. Una permutación de bucles λ sobre bucles perfectos transforma las iteraciones $(I_1, I_2, \ldots, I_n)$ a $(I_{\lambda 1}, I_{\lambda 2}, \ldots, I_{\lambda n})$. Esta transformación puede ser expresada en una matriz identidad I_λ, de orden $n \times n$, con las filas permutadas por λ. Los límites de los bucles después de aplicar una transformación de permutación de bucles se obtienen por el método de eliminación de *Fourier-Motzkin*. La matriz de permutación de bucles P_λ es una transformación ***legal*** si y sólo si:

$$\varphi \in D \quad : \quad P_\lambda * \varphi \geq 0$$

A continuación explicamos dos formas de hacer embaldosado. En la primera partimos de un bucle anidado cuyos índices son (t, p_1, p_2) y llegamos a un bucle anidado cuyos índices son $(p_1 p_1, p_2 p_2, t, p_1, p_2)$. En la segunda partimos del mismo bucle anidado que en el caso anterior y llegamos a un bucle anidado cuyos índices son $(t, p_1 p_1, p_2 p_2, p_1, p_2)$. Las transformaciones mencionadas las logramos por secuencias de transformaciones de permutación de bucles y de minería de tiras [14]. Si M denota una transformación minería de tiras y P_λ denota una transformación de permutación de bucles entonces la primera transformación se logra mediante:

$$\begin{bmatrix} t \\ p_1 \\ p_2 \end{bmatrix} \rightarrow P_0 \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ t \end{bmatrix} \rightarrow M_0 \rightarrow \begin{bmatrix} p_1 p_1 \\ p_1 \\ p_2 \\ t \end{bmatrix} \rightarrow P_1 \rightarrow \begin{bmatrix} p_1 p_1 \\ p_2 \\ p_1 \\ t \end{bmatrix} \rightarrow M_1 \rightarrow \begin{bmatrix} p_1 p_1 \\ p_2 p_2 \\ p_2 \\ p_1 \\ t \end{bmatrix} \rightarrow P_2 \rightarrow \begin{bmatrix} p_1 p_1 \\ p_2 p_2 \\ t \\ p_1 \\ p_2 \end{bmatrix}$$

y la segunda transformación se efectúa mediante:

$$\begin{bmatrix} t \\ p_1 \\ p_2 \end{bmatrix} \rightarrow \boldsymbol{M}_0 \rightarrow \begin{bmatrix} t \\ p_1 p_1 \\ p_1 \\ p_2 \end{bmatrix} \rightarrow \boldsymbol{P}_0 \rightarrow \begin{bmatrix} t \\ p_1 p_1 \\ p_2 \\ p_1 \end{bmatrix} \rightarrow \boldsymbol{M}_1 \rightarrow \begin{bmatrix} t \\ p_1 p_1 \\ p_2 p_2 \\ p_2 \\ p_1 \end{bmatrix} \rightarrow \boldsymbol{P}_1 \rightarrow \begin{bmatrix} t \\ p_1 p_1 \\ p_2 p_2 \\ p_1 \\ p_2 \end{bmatrix}$$

A continuación se efectúa la primera transformación de embaldosado a la transformación *espacio temporal*.

```
for (t = Lt; t <= Ut; t++){
   for (p1 = Lp1; p1 <= Up1; p1++){
      for (p2 = Lp2; p2 <= Up2; p2++){
            loop  body;
      }
   }
}
```

la matriz de permutación inicial $\boldsymbol{P}_0$ sobre los índices $(t, p1, p2)$ es:

$$\boldsymbol{P}_0 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

al aplicar la transformación se obtiene:

$$
\begin{array}{l}
for\,(p1 = L'p1;\ p1 <= U'p1;\ p1++)\{ \\
\quad for\,(p2 = L'p2;\ p2 <= U'p2;\ p2++)\{ \\
\quad\quad for\,(t = L't;\ t <= U't;\ t++)\{ \\
\quad\quad\quad\quad loop\quad body; \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}
$$

la transformación de minería de tiras $\boldsymbol{M}_0$ aplicada al bucle $p1$ es:

$$
\begin{array}{l}
for\,(p1p1 = L'p1;\ p1p1 <= U'p1;\quad p1p1+ = Bp1)\{ \\
\\
\quad for\,(p1 = \max(p1p1, L'p1);\ p1 <= \min(p1p1 + Bp1 - 1, U'p1);\quad p1++)\{ \\
\quad\quad for\,(p2 = L'p2;\ p2 <= U'p2;\quad p2++)\{ \\
\quad\quad\quad for\,(t = L't;\ t <= U't;\quad t++)\{ \\
\quad\quad\quad\quad\quad loop\quad body; \\
\quad\quad\quad \} \\
\quad\quad \} \\
\quad \} \\
\}
\end{array}
$$

la matriz de permutación $\boldsymbol{P}_1$ aplicada sobre los índices $(p1, p2, t)$ es:

$$
\boldsymbol{P}_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

al aplicar la transformación se obtiene:

$$
\begin{array}{l}
for\,(p1p1 = L'p1;\ p1p1 <= U'p1;\quad p1p1 += Bp1)\{ \\
\quad for\,(p2 = L''p2;\ p2 <= U''p2;\quad p2++)\{ \\
\quad\quad for\,(p1 = \max(p1p1, L''p1);\ p1 <= \min(p1p1 + Bp1 - 1, U''p1);\quad p1++)\{ \\
\quad\quad\quad for\,(t = L't;\ t <= U't;\quad t++)\{ \\
\quad\quad\quad\quad loop\quad body; \\
\quad\quad\quad\} \\
\quad\quad\} \\
\quad\} \\
\}
\end{array}
$$

la transformación de minería de tiras M_1 aplicada al bucle $p2$ es:

$$
\begin{array}{l}
for\,(p1p1 = L'p1;\ p1p1 <= U'p1;\quad p1p1 += Bp1)\{ \\
\quad for\,(p2p2 = L''p2;\ p2p2 <= U''p2;\quad p2p2 += Bp2)\{ \\
\quad\quad for\,(p2 = \max(p2p2, L''p2);\ p2 <= \min(p2p2 + Bp2 - 1, U''p2);\quad p2++)\{ \\
\quad\quad\quad for\,(p1 = \max(p1p1, L''p1);\ p1 <= \min(p1p1 + Bp1 - 1, U''p1);\quad p1++)\{ \\
\quad\quad\quad\quad for\,(t = L't;\ t <= U't;\quad t++)\{ \\
\quad\quad\quad\quad\quad loop\quad body; \\
\quad\quad\quad\quad\} \\
\quad\quad\quad\} \\
\quad\quad\} \\
\quad\} \\
\}
\end{array}
$$

la matriz de permutación final $\boldsymbol{P}_2$ aplicada sobre los índices $(p2, p1, t)$ es:

$$\boldsymbol{P}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

al aplicar la transformación se obtiene:

```
for (p1p1 = L'p1; p1p1 <= U'p1;   p1p1+ = Bp1){
  for (p2p2 = L''p2; p2p2 <= U''p2;   p2p2+ = Bp2){

    for (t = L'''t; t <= U'''t;   t++){
      for (p1 = max(p1p1, L'''p1); p1 <= min(p1p1 + Bp1 - 1, U'''p1);   p1++){
        for (p2 = max(p2p2, L'''p2); p2 <= min(p2p2 + Bp2 - 1, U'''p2);   p2++){
            loop   body;
        }
      }
    }
  }
}
```

El listado 3.1 muestra la primera transformación de embaldosado a los índices $(p1, p2)$ de la transformación *espacio temporal.*

```
for (p1p1 = 0; p1p1 <= N - 1; p1p1+ = Bp1){
  for (p2p2 = 0; p1 <= min(N - 1, p1p1 + Bp1 - 1); p2p2+ = Bp2){

    for (t = max(0, p1p1, p1p1 + 2 * p2p2, p2p2, 2 * p2p2 - M + 2, 6 - 3 * M + 6 * p2p2);
    t <= min(M + 2 * N - 4, 2 * M + p1p1 + Bp1 - 5, M + N + p1p1 + Bp1 - 4)); t++){
      for( p1 = ceil( max( p1p1, 0, 1/2 (t - M + 2), t - M - N + 3, p2p2, 2 * p2p2 - M + 2 ) );
      p1 <= min(p1p1 + Bp1 - 1, t, N - 1, t - 2 * p2p2); p1++){
        for (p2 = max(p2p2, 0, t - p1 - M + 2);
        p2 <= floor( min( p2p2 + Bp2 - 1, p1, N - 1, 1/2 (t - p1), p1p1 + Bp1 - 1 ) ); p2++ ){

          if (p1 == p2){              // Generación de Rotaciones de Givens (GG)

            if (t == p1 + 2 * p2){
              r[-t + p1 + 2 * p2 - M][p1][p2] = a[-t + p1 + 2 * p2 - M][p1][p2];
            }

            r[-t + p1 + 2 * p2 - 1 - M][p1][p2]=sqrt(pow(a[-t + p1 + 2 * p2 - 1 - M][p1][p2],2.0) +
                                                  pow(r[-t + p1 + 2 * p2 - M][p1][p2],2.0));

            if (r[-t + p1 + 2 * p2 - 1 - M][p1][p2] == 0){
              c[-t + p1 + 2 * p2 - M][p1 + 1][p2] = 1;
              s[-t + p1 + 2 * p2 - M][p1 + 1][p2] = 0;
            }else{
              c[-t + p1 + 2 * p2 - M][p1 + 1][p2] = (a[-t + p1 + 2 * p2 - 1 - M][p1][p2]) / r[-t + p1 + 2 * p2 - 1 - M][p1][p2];
              s[-t + p1 + 2 * p2 - M][p1 + 1][p2] = (a[-t + p1 + 2 * p2 - M][p1][p2]) / r[-t + p1 + 2 * p2 - 1 - M][p1][p2];
            }

            if (t == p1 + p2 + M - 1){
              a[-t + p1 + 2 * p2 - 1 - M][p1][p2 + 1] = r[-t + p1 + 2 * p2 - 1 - M][p1][p2];
            }
```

```
        }else{          // Ejecución de Rotaciones de Givens (RG)

          if (t == p1+2*p2){
            temp[-t+p1+2*p2-M][p1][p2] = a[-t+p1+2*p2-M][p1][p2];
          }

          temp[-t+p1+2*p2-1-M][p1][p2] =  c[-t+p1+2*p2-M][p1][p2]*a[-t+p1+2*p2-1-M][p1][p2]+
                                          s[-t+p1+2*p2-M][p1][p2]*temp[-t+p1+2*p2-M][p1][p2];
          a[-t+p1+2*p2-M][p1][p2+1]    = -s[-t+p1+2*p2-M][p1][p2]*a[-t+p1+2*p2-1-M][p1][p2]+
                                          c[-t+p1+2*p2-M][p1][p2]*temp[-t+p1+2*p2-M][p1][p2];

          if (t == p1+p2+M-1){
            a[-t+p1+2*p2-1-M][p1][p2+1] = temp[-t+p1+2*p2-1-M][p1][p2];
          }

          c[-t+p1+2*p2-M][p1+1][p2] = c[-t+p1+2*p2-M][p1][p2];
          s[-t+p1+2*p2-M][p1+1][p2] = s[-t+p1+2*p2-M][p1][p2];

        }
      }
    }
  }
}
```

Listado 3.1: Programa de la descomposición QR después de aplicar embaldosado a los índices $(p1, p2)$ de la transformación *espacio temporal.*

En la figura 3.1 se observa como las baldosas cubren el espacio de iteración $(t, p1, p2)$ del listado 3.1.

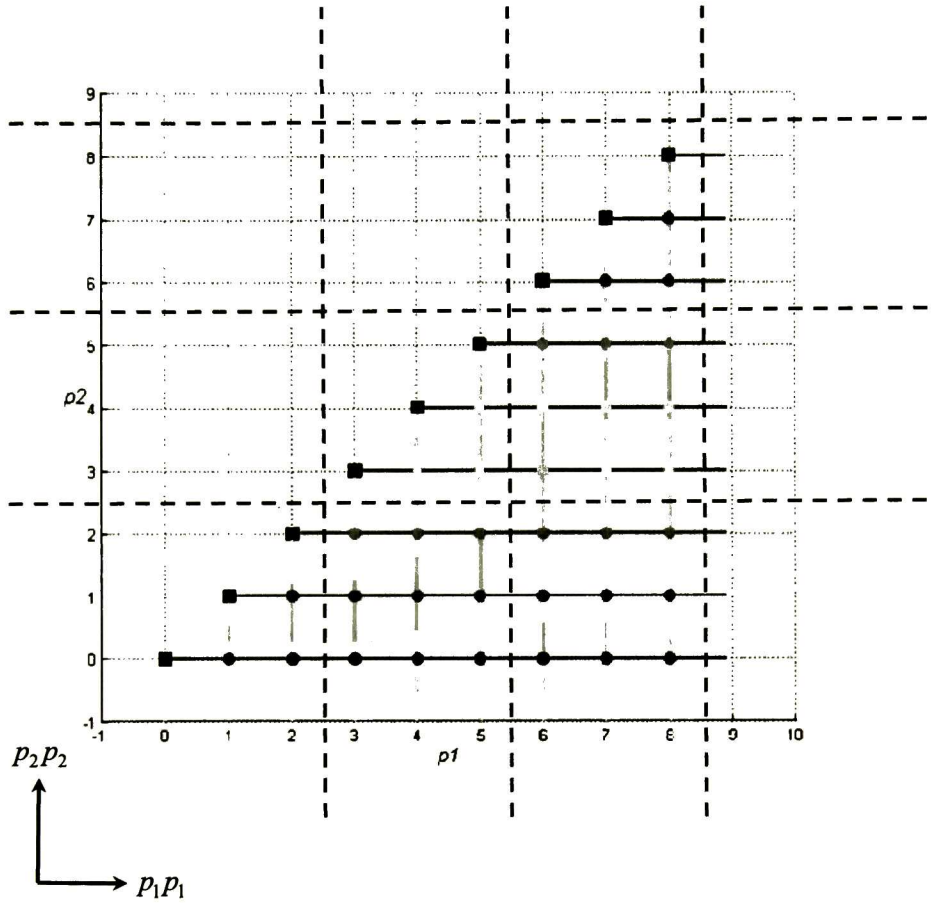


Figura 3.1: Embaldosado $\left(B_{p_1}=3, B_{p_2}=3\right)$ del espacio de iteración de la transformación *espacio temporal*, $\left(M=10, N=9\right)$.

El listado 3.2 muestra la segunda transformación de embaldosado a los índices $(p1, p2)$ de la transformación *espacio temporal*, observe que se considera al tiempo como el bucle más externo.

```
for (t = 0; t <= M + 2*N - 4; t++){

  for (p1p1 = 0; p1p1 <= N - 1; p1p1+ = Bp1){
    for (p2p2 = 0; p1p1 <= min(N - 1, p1p1 + Bp1 - 1); p2p2+ = Bp2)

      for (p1 = ceil(max(p1p1, 0, t - 2*M + 4p1, t - M - p1p1 - Bp1 + 3, (1/2)(t - M + 2), t - M - N + 3, p2p2)));
      p1 <= min(p1p1 + Bp1 - 1, t, N - 1, t - 2*p2p2); p1++){
        for (p2 = max(p2p2, 0, t - p1 - M + 2);
        p2 <= floor(min(p2p2 + Bp2 - 1, p1, N - 1, (1/2)(t - p1), (1/2)(p1 + M - 2), p1p1 + Bp1 - 1)); p2++){


          if (p1 == p2){              // Generación de Rotaciones de Givens (GG)

            if (t == p1 + 2*p2){
              r[-t + p1 + 2*p2 - M][p1][p2] = a[-t + p1 + 2*p2 - M][p1][p2];
            }

            r[-t + p1 + 2*p2 - 1 - M][p1][p2] = sqrt(pow(a[-t + p1 + 2*p2 - 1 - M][p1][p2], 2.0) +
                                                pow(r[-t + p1 + 2*p2 - M][p1][p2], 2.0));

            if (r[-t + p1 + 2*p2 - 1 - M][p1][p2] == 0){
              c[-t + p1 + 2*p2 - M][p1 + 1][p2] = 1;
              s[-t + p1 + 2*p2 - M][p1 + 1][p2] = 0;
            }else{
              c[-t + p1 + 2*p2 - M][p1 + 1][p2] = (a[-t + p1 + 2*p2 - 1 - M][p1][p2]) /
                                                  r[-t + p1 + 2*p2 - 1 - M][p1][p2];
              s[-t + p1 + 2*p2 - M][p1 + 1][p2] = (a[-t + p1 + 2*p2 - M][p1][p2]) /
                                                  r[-t + p1 + 2*p2 - 1 - M][p1][p2];
            }

            if (t == p1 + p2 + M - 1){
              a[-t + p1 + 2*p2 - 1 - M][p1][p2 + 1] = r[-t + p1 + 2*p2 - 1 - M][p1][p2];
            }
```

```
}else{          // Ejecución de Rotaciones de Givens (GR)

  if(t == p1+2*p2){
    temp[-t+p1+2*p2-M][p1][p2]=a[-t+p1+2*p2-M][p1][p2];
  }

  temp[-t+p1+2*p2-1-M][p1][p2]= c[-t+p1+2*p2-M][p1][p2]*a[-t+p1+2*p2-1-M][p1][p2]+
                                s[-t+p1+2*p2-M][p1][p2]*temp[-t+p1+2*p2-M][p1][p2];
  a[-t+p1+2*p2-M][p1][p2+1]   =-s[-t+p1+2*p2-M][p1][p2]*a[-t+p1+2*p2-1-M][p1][p2]+
                                c[-t+p1+2*p2-M][p1][p2]*temp[-t+p1+2*p2-M][p1][p2];

  if(t == p1+p2+M-1){
    a[-t+p1+2*p2-1-M][p1][p2+1]=temp[-t+p1+2*p2-1-M][p1][p2];
  }

  c[-t+p1+2*p2-M][p1+1][p2]=c[-t+p1+2*p2-M][p1][p2];
  s[-t+p1+2*p2-M][p1+1][p2]=s[-t+p1+2*p2-M][p1][p2];

 }
 }
 }
 }
 }
}
```

Listado 3.2: Programa de la descomposición QR después de aplicar embaldosado a los índices $(p1, p2)$ de la transformación *espacio temporal*.

3.3 Extracción de Información para la Implementación

En esta sección se describen la trayectoria de datos y la generación de señales de control necesarios para la implementación de la *baldosa* bidimensional de procesadores que efectúa la descomposición QR.

3.3.1 Trayectoria de Datos

La figura 3.2 muestra la baldosa que implementa la transformación de embaldosado $(p1p1, p2p2, t, p1, p2)$, se considera para fines ilustrativos una baldosa de dimensiones $(Bp1 = 3,\ Bp2 = 3)$.

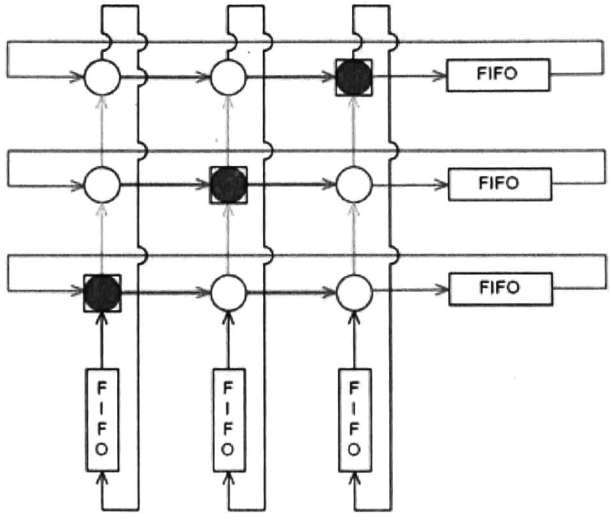


Figura 3.2: Baldosa de dimensiones $(Bp1 = 3,\ Bp2 = 3)$ que efectúa la transformación de embaldosado $(p1p1, p2p2, t, p1, p2)$ de la descomposición QR.

3.3.2 Generación de Señales de Control

Las señales de control para su estudio se dividen en: *señales de control de habilitación* y *señales de control funcional.*

3.3.2.1 Control de Habilitación

El control de habilitación calendariza la activación de cada elemento procesador de la baldosa física. La ventaja de su aplicación es un ahorro considerable en el consumo de potencia. La tabla 3.1 muestra el diagrama de Gantt de los elementos procesadores presentes en la baldosa bidimensional que efectúa la descomposición QR.

tiempo	*Procesador físico*								
	00	*10*	*20*	*01*	*11*	*21*	*02*	*12*	*22*
0	*1*								
1	*1*	*1*							
2	*1*	*1*	*1*						
3	*1*	*1*	*1*		*1*				
4	*1*	*1*	*1*		*1*	*1*			
5	*1*	*1*	*1*		*1*	*1*			
6	*1*	*1*	*1*		*1*	*1*			*1*
7		*1*	*1*		*1*	*1*			*1*
8			*1*		*1*	*1*			*1*
9						*1*			*1*
10									*1*
11	*1*								
12	*1*	*1*							
13	*1*	*1*	*1*	*1*					
14	*1*	*1*	*1*	*1*	*1*				
15	*1*	*1*	*1*	*1*	*1*	*1*	*1*		
16	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	
17	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
18		*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
19			*1*		*1*	*1*	*1*	*1*	*1*
20						*1*		*1*	*1*
21									*1*
22	*1*								
23	*1*	*1*							
24	*1*	*1*	*1*						
25	*1*	*1*	*1*		*1*				
26		*1*	*1*		*1*	*1*			
27			*1*		*1*	*1*			
28						*1*			*1*
29									*1*

Tabla 3.1: Diagrama de Gantt de la baldosa bidimensional $(Bp1=3,\ Bp2=3)$ que efectúan la descomposición QR, $(M=7, N=6)$.

3.3.2.2 Control Funcional

El control funcional especifica la función que el elemento procesador efectúa al momento de estar habilitado. La tabla 3.2 muestra el tiempo de activación de las condicionales presentes en la transformación de embaldosado $(p1p1, p2p2, t, p1, p2)$.

tiempo	*Procesador físico*								
	00	*10*	*20*	*01*	*11*	*21*	*02*	*12*	*22*
0	*if.GG GR if.1 GG*								
1		*if.1.RG*							
2			*if.1.RG*						
3					*if.GG GR if.1.GG*				
4						*if.1.RG*			
5									
6	*if.2 GG*								*if.GG GR if 1 GG*
7		*if.2.RG*							
8			*if.2.RG*		*if.2 GG*				
9						*if.2.RG*			
10									*if.2 GG*
11	*if.1.RG*								
12		*if.1 RG*							
13			*if.1.RG*						
14									
15									
16									
17	*if.2.RG*								
18		*if.2.RG*		*if.2.RG*					
19			*if.2.RG*		*if.2.RG*		*if.2.RG*		
20						*if 2.RG*		*if 2.RG*	
21									*if.2.RG*
22	*if.GG GR if.1.GG*								
23		*if.1.RG*							
24			*if.1.RG*						
25	*if.2.GG*				*if.GG GR if.1.GG*				
26		*if.2.RG*				*if.1.RG*			
27			*if.2.RG*		*if.2.GG*				
28						*if.2.RG*			*if GG GR if.1.GG*
29									*if.2.GG*

Tabla 3.2: Tiempo de activación de las señales de *control funcional*, presentes en la transformación de embaldosado $(p1p1, p2p2, t, p1, p2)$, $(M = 7, N = 6)$.

3.4 Multiproyección

3.4.1 Transformación de Politopo fuente a Politopo Destino

Sea T la matriz de transformación *espacio temporal*, la cual define el tiempo de ejecución y efectúa el mapeo hacia un arreglo de procesadores en una dimensión $(1D)$.

$$T=\begin{bmatrix}\underline{S}\\ p\end{bmatrix}=\begin{bmatrix}s_k\\ \underline{s_{ij}}\\ p_j\end{bmatrix}=\begin{bmatrix}0 & 0 & 1\\ \underline{1} & \underline{1} & \underline{0}\\ 0 & 1 & 0\end{bmatrix} \tag{3.1}$$

La matriz de transformación *espacio temporal* T, cumple la relación (3.2).

$$Tx = y \tag{3.2}$$

Donde el vector x representa el espacio de índices $x=[i \quad j \quad k]^t$ del politopo fuente y el vector y el nuevo espacio de índices definido $y=[t_1 \quad t_2 \quad p]^t$ del politopo destino al efectuar multiproyección. Note que en este caso el tiempo es un vector.

Aplicando la relación (3.2):

$$\overbrace{\begin{bmatrix}0 & 0 & 1\\ 1 & 1 & 0\\ 0 & 1 & 0\end{bmatrix}}^{T}\overbrace{\begin{bmatrix}i\\ j\\ k\end{bmatrix}}^{x}=\overbrace{\begin{bmatrix}t_1\\ t_2\\ p\end{bmatrix}}^{y}$$

Se obtiene la relación:

$$x = T^{-1}y \tag{3.3}$$

$$\overbrace{\begin{bmatrix}i\\ j\\ k\end{bmatrix}}^{x}=\overbrace{\begin{bmatrix}0 & 1 & -1\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}}^{T^{-1}}\overbrace{\begin{bmatrix}t_1\\ t_2\\ p\end{bmatrix}}^{y}$$

Luego entonces:

$$\begin{aligned} i &= t_2 - p \\ j &= p \\ k &= t_1 \end{aligned} \tag{3.4}$$

La relación (3.4) define los valores de $\boldsymbol{x}$ en función del nuevo espacio de iteración determinado por y, en el código del politopo destino al efectuar multiproyección.

La relación $\boldsymbol{Ax} \leq \boldsymbol{b}$ es la representación matricial del sistema de desigualdades, que define al politopo fuente, en función del espacio de iteración (i, j, k).

$$\overbrace{\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} i \\ j \\ k \end{bmatrix}}^{x} \leq \overbrace{\begin{bmatrix} M \\ 0 \\ 0 \\ -1 \\ N-1 \\ N-1 \end{bmatrix}}^{y}$$

limite inferior de i	$i \geq -M$
limite inferior de j	$j \geq k$
limite inferior de k	$k \geq 0$
limite superior de i	$i \leq -k-1$
limite superior de j	$j \leq N-1$
limite superior de k	$k \leq N-1$

El politopo destino al efectuar multiproyección está descrito por el conjunto de desigualdades de la relación (3.5).

$$\left(\boldsymbol{AT}^{-1}\right) y \leq \boldsymbol{b}$$

$$\overbrace{\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 0 & 0 \\ 1 & 1 & -1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}}^{(AT^{-1})} \overbrace{\begin{bmatrix} t_1 \\ t_2 \\ p \end{bmatrix}}^{y} \leq \overbrace{\begin{bmatrix} M \\ 0 \\ 0 \\ -1 \\ N-1 \\ N-1 \end{bmatrix}}^{b} \tag{3.5}$$

Resolviendo el sistema de desigualdades de la relación (3.5) por el método de eliminación de *Fourier-Motzkin* se obtienen los límites inferior y superior del nuevo espacio de iteración y, del programa del politopo destino al efectuar multiproyección:

$$\begin{aligned} 0 \le t_1 &\le \min(M-2, N-1) \\ t_1 \le t_2 &\le \min(M+N-3-t_1) \\ \max(t_1, t_1+t_2-M+2) \le p &\le \min(t_2, N-1) \end{aligned}$$

El listado 3.3 muestra el programa correspondiente al politopo destino al efectuar *multiproyección*.

```
for (t1 = 0; t1 <= N-1;  t1++){
  for (t2 = t1; t2 <= min(M+N-3-t1);  t2++){
    for (p = max(t1, t1+t2-M+2); p <= min(t2, N-1);  p++){

      if(t1 == p){              // Generación de Rotaciones de Givens (GG)
        if (t2 == p){
          r[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
        }
        r[p-t2+M-2][p][t1]=sqrt( pow(a[p-t2+M-2][p][t1],2.0)+
                                 pow(r[p-t2+M-1][p][t1],2.0));
        if (r[p-t2+M-2][p][t1] == 0){
          c[p-t2+M-1][p+1][t1] = 1;
          s[p-t2+M-1][p+1][t1] = 0;
        }else{
          c[p-t2+M-1][p+1][t1] = (a[p-t2+M-2][p][t1]) / r[p-t2+M-2][p][t1];
          s[p-t2+M-1][p+1][t1] = (a[p-t2+M-1][p][t1]) / r[p-t2+M-1][p][t1];
        }
        if (t2+t1 == p+M-2){
          a[p-t2+M-2][p][t1+1] = r[p-t2+M-2][p][t1];
        }

      }else{              // Ejecución de Rotaciones de Givens (RG)
        if (t2 == p){
          temp[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
        }
        temp[p-t2+M-2][p][t1] =  c[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                 s[p-t2+M-1][p][t1]*temp[p-t2+M-1][p][t1];
        a[p-t2+M-1][p][t1+1]  = -s[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                 c[p-t2+M-2][p][t1]*temp[p-t2+M-1][p][t1];
        if (t2+t1 == p+M-2){
          a[p-t2+M-2][p][t1+1] = temp[p-t2+M-2][p][t1];
        }
        c[p-t2+M-1][p+1][t1] = c[p-t2+M-1][p][t1];
        s[p-t2+M-1][p+1][t1] = s[p-t2+M-1][p][t1];
      }
    }
  }
}
```

Listado 3.3: Programa de la descomposición *QR* después de aplicar la transformación *espacio temporal de multiproyección.*

3.4.2 Embaldosado

Las transformaciones convencionales de embaldosado: *minería de tiras* y *permutación de bucles*, presentadas en la sección 3.2, son aplicables a la multiproyección.

A continuación realizamos dos formas de hacer embaldosado. En la primera partimos de un bucle anidado cuyos índices son (t_1,t_2,p) y llegamos a un bucle anidado cuyos índices son (pp,t_1,t_2,p). En la segunda partimos del mismo bucle anidado que en el caso anterior y llegamos a un bucle anidado cuyos índices son (t_1,t_2,pp,p). Las transformaciones mencionadas las logramos por secuencias de transformaciones de permutación de bucles y de minería de tiras. Si $\boldsymbol{M}$ denota una transformación minería de tiras y $\boldsymbol{P}_\lambda$ denota una transformación de permutación de bucles entonces la primera transformación se logra mediante:

$$\begin{bmatrix} t_1 \\ t_2 \\ p \end{bmatrix} \rightarrow \boldsymbol{P}_0 \rightarrow \begin{bmatrix} p \\ t_1 \\ t_2 \end{bmatrix} \rightarrow \boldsymbol{M}_0 \rightarrow \begin{bmatrix} pp \\ p \\ t_1 \\ t_2 \end{bmatrix} \rightarrow \boldsymbol{P}_1 \rightarrow \begin{bmatrix} pp \\ t_1 \\ t_2 \\ p \end{bmatrix}$$

y la segunda transformación se efectúa mediante:

$$\begin{bmatrix} t_1 \\ t_2 \\ p \end{bmatrix} \rightarrow \boldsymbol{M}_0 \rightarrow \begin{bmatrix} t_1 \\ t_2 \\ pp \\ p \end{bmatrix}$$

Los listados 3.4 y 3.5 corresponden respectivamente a la primera y segunda transformación de embaldosado del espacio de iteración del listado 3.3.

```
for (pp = 0; pp <= N-1;  pp+= Bp)
  for (t1 = 0; t1 <= floor(min(pp+Bp-1,M-2,N-1,M+N-pp-3,M+N-3,pp+Bp+M-3,
  0.5*(M+N-3),0.5*(M+pp+Bp-3),0.5*(2*M+N-pp-5),0.25*(3*M+N-7))); t1++){
    for (t2 = max(t1-M+1,pp-M+1,1-M,2*t1-2*M+3);
    t2 <= min(N-2-t1,pp+Bp-2-t1,M+N-2*t1-4); t2++){
      for (p = max(pp,0,t1,2*t1-M+2,t1+t2+1);
      p <= min(pp+Bp-1,N-1,t2+M-1,M+N-t1-3); p++){
        if (t1 == p){              // Generación de Rotaciones de Givens (GG)
          if (t2 == p){
            r[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
          }
          r[p-t2+M-2][p][t1]=sqrt(pow(a[p-t2+M-2][p][t1],2.0)+
                                  pow(r[p-t2+M-1][p][t1],2.0));
          if (r[p-t2+M-2][p][t1] == 0){
            c[p-t2+M-1][p+1][t1] = 1;
            s[p-t2+M-1][p+1][t1] = 0;
          }else{
            c[p-t2+M-1][p+1][t1] = (a[p-t2+M-2][p][t1]) / r[p-t2+M-2][p][t1];
            s[p-t2+M-1][p+1][t1] = (a[p-t2+M-1][p][t1]) / r[p-t2+M-1][p][t1];
          }
          if (t2+t1 == p+M-2){
            a[p-t2+M-2][p][t1+1] = r[p-t2+M-2][p][t1];
          }
        }else{              // Ejecución de Rotaciones de Givens (RG)
          if (t2 == p){
            temp[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
          }
          temp[p-t2+M-2][p][t1] =  c[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                   s[p-t2+M-1][p][t1]*temp[p-t2+M-1][p][t1];
          a[p-t2+M-1][p][t1+1]  = -s[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                   c[p-t2+M-2][p][t1]*temp[p-t2+M-1][p][t1];
          if (t2+t1 == p+M-2){
            a[p-t2+M-2][p][t1+1] = temp[p-t2+M-2][p][t1];
          }
          c[p-t2+M-1][p+1][t1] = c[p-t2+M-1][p][t1];
          s[p-t2+M-1][p+1][t1] = s[p-t2+M-1][p][t1];
        }
      }
    }
  }
}
```

Listado 3.4: Programa de la descomposición QR después de aplicar embaldosado al índice p de la transformación *espacio temporal de multiproyección*.

```
for (t1 = 0; t1 <= N-1;  t1++){
  for (t2 = t1; t2 <= min(M+N-3-t1);  t2++){
    for (pp = max(t1, t1+t2-M+2); pp <= min(t2, N-1);  pp+= Bp){
      for (p = max(pp, t1, t1+t2-M+2); p <= min(pp+Bp-1, t2, N-1);  p++){
        if(t1 == p){            // Generación de Rotaciones de Givens (GG)
          if(t2 == p){
            r[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
          }
          r[p-t2+M-2][p][t1]=sqrt(pow(a[p-t2+M-2][p][t1],2.0)+pow(r[p-t2+M-1][p][t1],2.0));
          if(r[p-t2+M-2][p][t1] == 0){
            c[p-t2+M-1][p+1][t1] = 1;
            s[p-t2+M-1][p+1][t1] = 0;
          }else{
            c[p-t2+M-1][p+1][t1] = (a[p-t2+M-2][p][t1])/r[p-t2+M-2][p][t1];
            s[p-t2+M-1][p+1][t1] = (a[p-t2+M-1][p][t1])/r[p-t2+M-1][p][t1];
          }
          if(t2+t1 == p+M-2){
            a[p-t2+M-2][p][t1+1] = r[p-t2+M-2][p][t1];
          }
        }else{          // Ejecución de Rotaciones de Givens (RG)
          if(t2 == p){
            temp[p-t2+M-1][p][t1] = a[p-t2+M-1][p][t1];
          }
          temp[p-t2+M-2][p][t1] =  c[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                   s[p-t2+M-1][p][t1]*temp[p-t2+M-1][p][t1];
          a[p-t2+M-1][p][t1+1]  = -s[p-t2+M-1][p][t1]*a[p-t2+M-2][p][t1]+
                                   c[p-t2+M-2][p][t1]*temp[p-t2+M-1][p][t1];
          if(t2+t1 == p+M-2){
            a[p-t2+M-2][p][t1+1] = temp[p-t2+M-2][p][t1];
          }
          c[p-t2+M-1][p+1][t1] = c[p-t2+M-1][p][t1];
          s[p-t2+M-1][p+1][t1] = s[p-t2+M-1][p][t1];
        }
      }
    }
  }
}
```

Listado 3.5: Programa de la descomposición QR después de aplicar embaldosado al índice p de la transformación *espacio temporal de multiproyección.*

Capítulo 4

Introducción a la Teoría del Agendamiento

El álgebra es generosa; con frecuencia da más de lo que se le pide.
Jean Le Rond D'Alembert (1717-1783)

4.1 Introducción

Un enfoque para disminuir el tiempo de ejecución de un programa es mejorar la velocidad de operación del procesador donde se ejecuta. Otro enfoque es particionar el programa y usar múltiples procesadores simultáneamente para la ejecución del programa. En general el tiempo de ejecución de un programa se mejora utilizando los dos enfoques. En la actualidad se empieza a acercarse a los límites de desempeño que los componentes electrónicos tienen, esto hace más atractivo el segundo enfoque. En este capítulo estamos interesados solamente en el segundo enfoque, al cual se le llama **procesamiento en paralelo**. El procesamiento paralelo es el enfoque preferido en el diseño de máquinas de alto desempeño. Este tipo de procesamiento requiere identificar en el programa original las operaciones que se pueden ejecutar en paralelo; aunque no es parte de este capítulo la explicación de cómo un programa se puede particionar en

operaciones. La teoría del agendamiento es crucial para el diseño eficiente de sistemas de procesamiento paralelo.

En un problema de agendamiento se tienen tareas u operaciones y máquinas o procesadores y se requiere decidir en qué tiempos y en cuales procesadores se ejecutan las operaciones. El problema de decidir que operaciones ejecuta cada procesador se llama el **problema de la asignación** y el problema de decidir en que tiempos se ejecuta una operación se llama **problema de la calendarización**. La resolución del problema de agendamiento se efectúa bajo restricciones y se optimiza una función objetivo. Algunos autores se refieren con el nombre de problema de calendarización al problema de agendamiento.

Los *conceptos* y *teoremas* presentados en este capítulo constituyen a mi juicio los rudimentos de la teoría del agendamiento. Estos *teoremas* y *lemas* fueron probados por Darte, A., and Y., Robert [5], Bruno, J., Coffman, E., and Sethi, R., [4], Sinnen, O., Sousa, Leonel A., Sandnes, Frode E., *Toward a Realistic Task Scheduling Model.* IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 3, March 2006., Ravindran, K., *Task Allocation and Scheduling of Concurrent Applications to Multiprocessor Systems.* Ph.D. Thesis, Electrical Engineering and Computer Sciences University at Berkeley, 2007. [29] y Sih, Gilbert C., *Multiprocessor Scheduling to Account for Interprocessor Communication.* Ph.D. Thesis, Electronics Research Laboratory. College of Engineering University of California, Berkeley, April, 1991. [31]. La contribución de esta tesis en este aspecto radica en resumir y presentar en una forma clara los resultados más básicos de la teoría del agendamiento, además se presenta la teoría y pseudocódigo para construir agendadores con *lista de prioridad.* Existen literalmente miles de artículos sobre agendamiento, aquí sólo tocamos la superficie. El problema general de agendamiento es un problema computacional difícil de optimización combinatoria discreta multiobjetivo sujeto a muchas restricciones. Se encuentran cotas para el tiempo de ejecución óptimo y número de procesadores óptimo. Un programa es modelado como una relación de orden parcial, en el conjunto de sus operaciones. Estos parámetros óptimos serán útiles para evaluar la calidad de los que se obtienen de implementaciones de calendarizadores y arquitecturas reales.

4.2 Complejidad de Algoritmos

Introducimos aquí breve e informalmente algunos conceptos de complejidad en tiempo de algoritmos. Muchos de los calendarizadores son problemas computacionalmente complejos, también llamados no polinomiales difíciles o NP difíciles.

La clase de los problemas P sólo contiene problemas que pueden ser resueltos en tiempo polinomial, mientras que la clase de los problemas NP sólo contiene problemas cuya solución puede ser verificada en tiempo polinomial. Hasta ahora no ha sido posible demostrar que $P = NP$ note que $P \subseteq NP$ esto es todo problema que es resoluble en tiempo exponencial es verificable en tiempo exponencial. Para los problemas NP sólo se conocen algoritmos de complejidad exponencial y, como ya se mencionó, no ha sido posible hasta ahora encontrar un algoritmo de complejidad polinomial que resuelva un problema NP. Todos los algoritmos en la clase NP son polinomialmente equivalentes en complejidad en tiempo en el sentido que un algoritmo de la clase se puede convertir en otro algoritmo de la clase usando una transformación de complejidad polinomial en tiempo. La clase de problemas NP es tal que si un problema se puede resolver con un algoritmo de complejidad polinomial, todos se pueden resolver con un algoritmo de complejidad polinomial. Sean q y r problemas donde q es un problema en NP. q se puede reducir en tiempo polinomial a r entonces se dice que r **NP difícil**. r es **NP completo** si r es un problema NP. Dado que no se conocen algoritmos polinomiales en la clase de los NP difíciles y se cree que esto nunca sucederá, en la práctica se considera que para resolver un problema NP difícil sólo existen algoritmos de complejidad exponencial.

Dado un problema, la demostración de que es de complejidad en tiempo NP difícil o al menos de esta complejidad, es útil en el sentido de que no es conveniente tratar de encontrar una solución eficiente dado que es altamente improbable encontrarla. Para demostrar que un problema X es al menos de complejidad en tiempo NP difícil basta con encontrar un problema Y en la clase de algoritmos NP difíciles tal que Y se puede transformar en X con complejidad en tiempo polinomial. La transformación anterior, llamada transformación polinomial, si existe, por lo general es compleja y difícil de encontrar. El interesado en la teoría de la complejidad se le recomienda el libro de Garey y Johnson en la bibliografía [9].

Las siguientes son tres características deseables de un algoritmo que resuelve un problema.

i. resuelve el problema óptimamente,
ii. es de complejidad polinomial y
iii. resuelve todas las instancias del problema.

Cuando nos topamos con un problema NP difícil en la práctica se requiere abandonar al menos uno de los requerimientos anteriores. Los algoritmos $\rho-$aproximativos son heurísticos y encuentran soluciones subóptimas. Un algoritmo $\rho-$aproximativo resuelve un problema subóptimamente con la garantía de que su solución está dentro de un factor multiplicativo pequeño ρ del óptimo verdadero, esto se efectúa sin saber cual es el óptimo. Los algoritmos $\rho-$aproximativos son de complejidad polinomial y resuelven todas las instancias del problema. Desafortunadamente muchos algoritmos heurísticos no son $\rho-$aproximativos. Los algoritmos heurísticos más comunes son algoritmos incrementales y voraces que producen resultados subóptimos de complejidad polinomial en tiempo. Por lo general, se usan estudios experimentales para demostrar su eficiencia computacional y la calidad de sus soluciones, estas dos cifras de mérito son comúnmente usadas para comparar algoritmos heurísticos. En general no existe un algoritmo heurístico que es mejor que otro en todas las circunstancias. Un algoritmo heurístico es mejor que otro algoritmo heurístico si dada una población de problemas encuentra una solución más cerca a la óptima más frecuentemente que el otro. Un algoritmo heurístico se dice cuasi óptimo si las soluciones que arroja están dentro de cierto rango del óptimo la mayor parte del tiempo. Una inconveniencia común con los algoritmos heurísticos es su inflexibilidad ante cambios del problema que se resuelve. Por lo general un algoritmo heurístico requiere rediseño si se añaden nuevas hipótesis y restricciones, es decir, sacrifica flexibilidad para ser computacionalmente eficiente.

La gran mayoría de las variantes del problema de agendamiento son problemas NP difíciles. Para una demostración o referencia a ésta véase la bibliografía. Dado que problemas de agendamiento simples son NP difíciles no sería aventurado decir que casi todos los problemas de agendamiento son NP difíciles.

4.3 Agendamiento de Grafos Dirigidos con Comunicación sin Retardos

Se proporciona una breve introducción al agendamiento, inicialmente se considera un número ilimitado y homogéneo de procesadores, posteriormente se considera un número limitado de procesadores. Se considera que los canales de comunicación no tienen retardos. Este último tema se trata con más profundidad más adelante en este capítulo. Un programa se modela por medio de un grafo dirigido acíclico (dag) donde los nodos representan operaciones y los arcos representan precedencia. Dos operaciones están conectadas por un arco si y sólo si la operación en la punta del arco requiere la información producida por la operación en la cola del arco. Una operación es una computación indivisible y puede ser una asignación, un procedimiento o un programa completo.

Dado un dag, $G=(S,R)$, le asociamos a cada operación $o \in S$ un **costo** w **de computación** definido por la función $w(o) \geq 0$. Suponemos inicialmente que tenemos un número ilimitado de procesadores. La relación R define la precedencia entre operaciones. La figura 4.1 muestra un grafo como el descrito. La función w nos proporciona una medida del cómputo que cada uno de las operaciones requiere que por ahora suponemos que es el tiempo de computación y que todos los procesadores son iguales. En el caso que el tiempo de ejecución de una operación es dependiente de los datos suponemos que el valor dado por la función w es el peor caso de costo de computación. Se supone que se conoce $w(o)$ antes de ejecutar la operación o. En el caso de procesadores heterogéneos una vez que se sabe en que procesadores se ejecutan las operaciones la función w nos sirve para calcular los tiempos de ejecución de las operaciones. En el caso de procesadores homogéneos, antes de asignar procesadores a las operaciones se puede calcular el tiempo de ejecución a partir de $w(o)$. Suponemos aquí que el costo de computación es igual al tiempo de computación. La función w tiene como dominio las operaciones y contradominio los naturales. La selección de los naturales es vez de los racionales no es una restricción dado que la cardinalidad de S es finita y los costos racionales se podrían escalar para obtener los costos naturales. Se supone que la red que interconecta los procesadores es totalmente conectada y que la comunicación entre dos procesadores es sin retardo. Los procesadores no intervienen en la comunicación y múltiples procesadores se pueden comunicar a la vez. El modelo considerado es adecuado para circuitos digitales sincrónicos, sistemas multiprocesador con memoria compartida y para arquitecturas basadas en microprograma, además es requerido en algunas situaciones como subproblema para cuando se resuelve el mismo problema pero con retardos de comunicación considerados.

El modelo de programa por medio de un grafo de dependencia no permite tener ejecución condicional de operaciones. Sin embargo la definición de una operación

puede tener condicionales lo cual en general se manifiesta como un tiempo de ejecución variable de la operación. El tiempo depende de las alternativas que se hayan tomado en los condicionales de la operación. En este caso se toma como costo de computación el más costoso. En el modelo del grafo dirigido se supone que siempre se tienen disponibles los datos de entrada a cualquier tiempo requerido. Con respecto a los datos de salida se supone que cuando se producen siempre hay lugar en donde guardarlos. En una implementación real se pueden requerir de alivio tanto para las entradas como para las salidas.

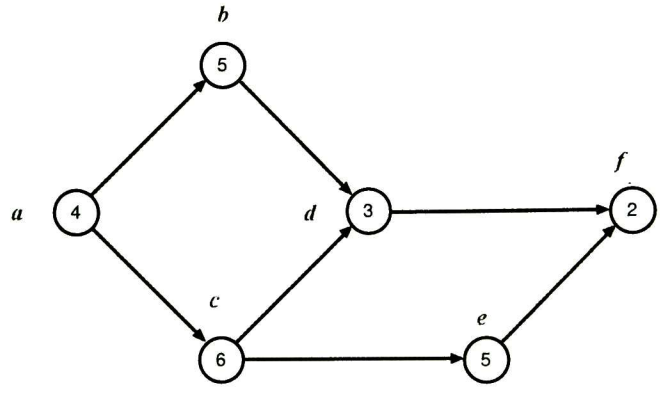


Figura 4.1: Ejemplo de grafo con costos de computación asociados a los nodos.

Un **calendarizador** nos indica en que tiempo se despacha el inicio de la ejecución de cada operación. Suponemos que la función calendarizadora tiene como dominio el conjunto de las operaciones S y como contradominio el conjunto de los naturales. Esta última consideración no es una restricción dado que si tomamos como contradominio los racionales dado que hay un número finito de operaciones siempre podemos escalar los racionales para obtener naturales. La selección es natural para las aplicaciones que tienen un reloj global que es compartido por todos los procesadores. El calendarizador asigna un tiempo de iniciación, $\tau(v)$ a cada operación $v \in S$. Un **asignador** nos indica en que procesador se ejecuta una operación. Notamos que el orden en que se ejecutan las operaciones asignadas a un procesador están determinadas por la función de calendarización. En general el diseño de un calendarizador y asignador son problemas dependientes. Un **agendador** se define como el par (τ, π). Un agendador produce una partición de los nodos del grafo dirigido, todos los nodos que se asignan aun mismo procesador pertenecen a la misma clase de equivalencia que en este contexto se le acostumbra a llamar **cúmulo**. Un agendador normalmente se implementa como un algoritmo que tiene como entradas un modelo del programa que se quiere ejecutar y un modelo del sistema del multiprocesador donde se planea ejecutar el programa. La salida del algoritmo son las funciones de calendarización y de asignación. Por abuso de lenguaje y dado que no conduce a confusiones al algoritmo se acostumbra llamarlo también agendador.

Un **diagrama de Gantt** es una ayuda visual que muestra la evolución de todos los procesadores en el tiempo. Un diagrama de Gantt es una manera de describir un agendador, es decir, toda la información para construir las funciones de calendarización y asignación se encuentran en el diagrama de Gantt. Se distinguen dos estados de un procesador: ocupado y desocupado. Un **procesador ocupado** ejecuta una operación. La figura 4.2 muestra un diagrama de Gantt donde se tienen cuatro elementos procesadores. El agendador tiene como propósito embaldosar, sujeto a un criterio de optimización, un pedazo del espacio tiempo-procesador respetando las precedencias entre operaciones.. Por ejemplo si el criterio de optimización es disminuir el tiempo de ejecución, el embaldosado resultante tendrá pocos lugares con procesadores desocupados.

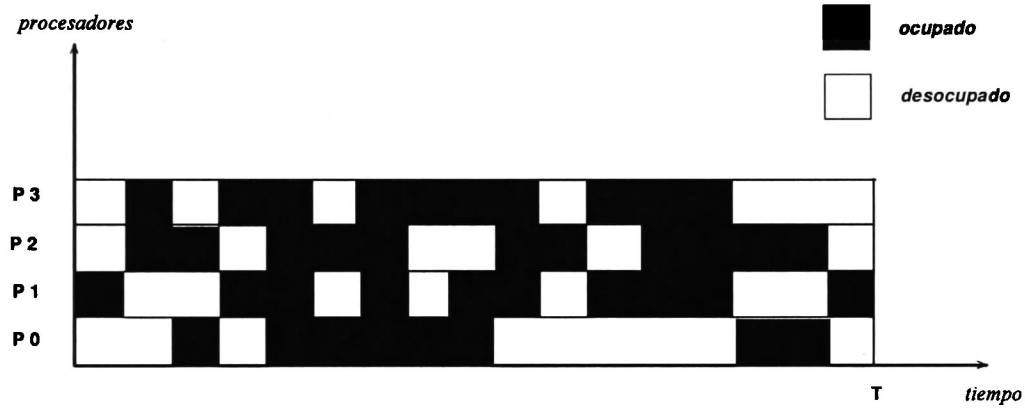


Figura 4.2: Diagrama de Gantt de un sistema paralelo.

Por construcción del diagrama de Gantt se tiene:

Lema 4.1. Para todo diagrama de Gantt son válidas las relaciones:

$$mT = \sum_{v \in S} w(v) + d \tag{4.1}$$

$$mT_{opt} \geq \sum_{v \in S} w(v) \tag{4.2}$$

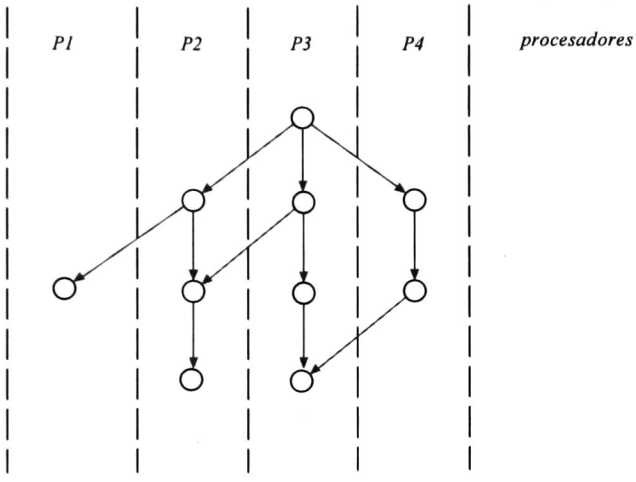
$$T_{opt} \geq \max_{v} w(v) \tag{4.3}$$

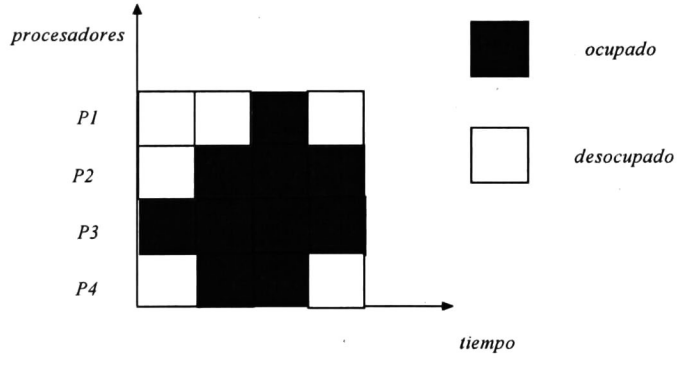
Donde m es el número de procesadores, S es el conjunto de operaciones, $w(v)$ es el costo de computación de la operación v, $d \geq 0$ es el tiempo acumulado que los procesadores se la pasan desocupados, T es el tiempo de ejecución y T_{opt} es el tiempo de ejecución más corto que puede producir un calendarizador arbitrario.

■

La suma de la relación (4.2) se llama el **tiempo de ejecución secuencial** y es el tiempo que le tomaría a un sólo procesador ejecutar todas las operaciones del grafo de dependencias. La relación (4.1) nos dice que el área del diagrama de Gantt puede ser calculado por dos formas al menos: la suma del acumulado de los tiempos de cuando los procesadores están desocupados y de la suma acumulada de los tiempos de cuando los procesadores están ocupados, la otra forma de calcular la área es el producto del número de procesadores por el tiempo de ejecución. La relación (4.2) nos indica que el tiempo de ejecución óptimo es al menos igual al promedio de tiempo de ejecución de cada procesador. La relación (4.3) indica que el tiempo de ejecución óptimo es al menos igual al tiempo de ejecución de cualquier operación.

Ejemplo 4.1. Considere los dags y sus correspondientes diagramas de Gantt en la figura 4.3. Los dags son equivalentes, sólo están dibujados de manera diferente. Los diagramas de Gantt muestran dos calendarizadores diferentes, los dos tienen el mismo tiempo de ejecución, sin embargo el segundo usa menos procesadores. El segundo calendarizador usa los procesadores más eficientemente.





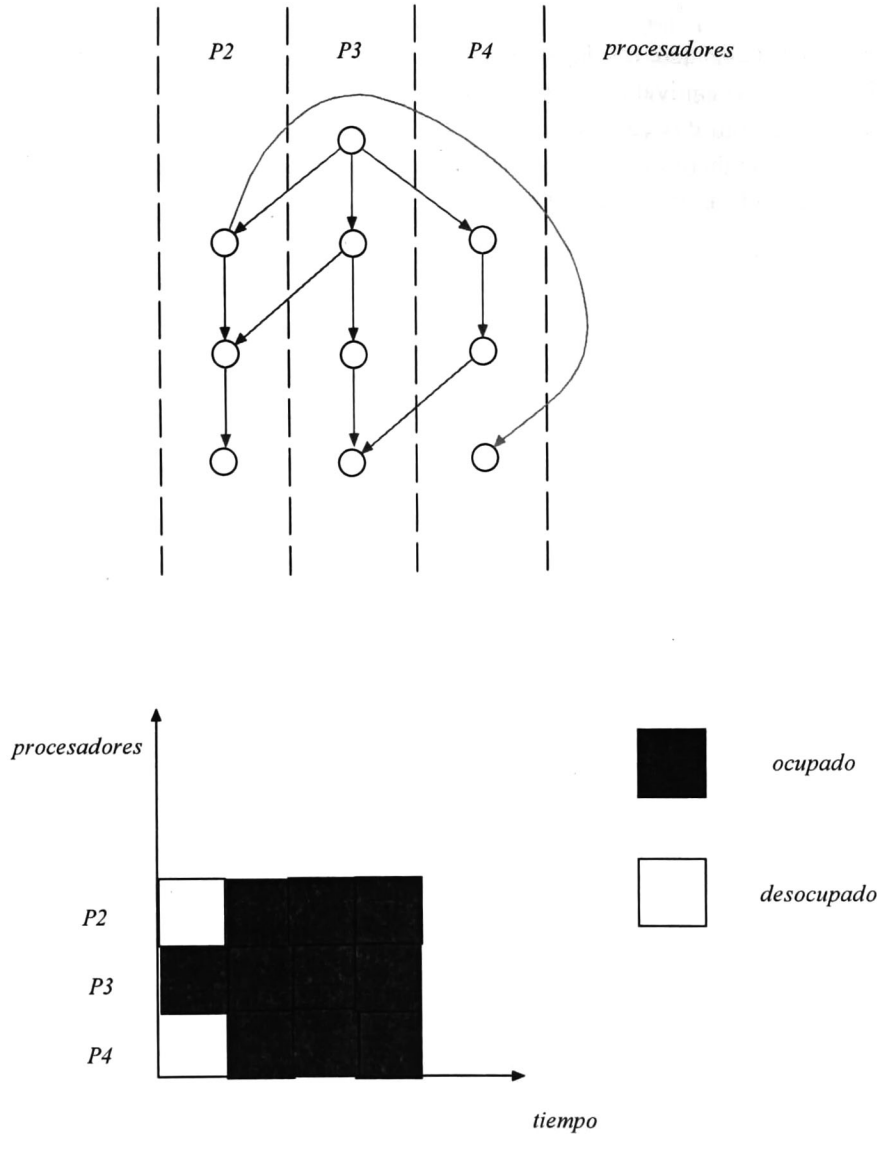


Figura 4.3: Grafos de dependencia y correspondientes diagramas de Gantt para dos calendarizadores diferentes.

■

Un **asignador** y un **calendarizador** son **válidos** o **factibles** si se satisface que:

i. una operación se ejecuta sólo si sus operaciones predecesoras ya se ejecutaron,
ii. no se asignan dos o más operaciones al mismo tiempo al mismo elemento procesador y
iii. la ejecución de una operación la realiza sólo un elemento procesador.

La primera propiedad, también llamada **restricción de precedencia**, nos indica que un calendarizador válido no viola las precedencias establecidas por el dag que modela el programa y la segunda que un procesador no puede ejecutar más de una operación a la vez. Esta última propiedad, también llamada **restricción de procesador**, establece una relación entre el calendarizador y el asignador. Todos los calendarizadores que aquí tratamos son válidos. Un asignador recobra particular importancia cuando el número de operaciones es mayor que el número de procesadores. Siempre que nos referimos a un calendarizador y un asignador suponemos que son válidos.

Si $G=(S,R)$ es el dag que representa el programa y τ y π son respectivamente las **funciones calendarizador** y **asignador** entonces tenemos que las propiedades que definen un calendarizador válido son:

i. $u,v\in S, u\neq v, uRv \quad y \quad \pi(u)\neq\pi(v)\Rightarrow\tau_f(u)\leq\tau(v)$

Donde $\tau_f(u)$ es el **tiempo de terminación** de la operación u y se define como:

$$\tau_f(u)\triangleq\tau(u)+w(u)$$

$\tau(u)$ es el **tiempo de despacho** o **de inicialización** de la operación u, se tiene de manera similar para $\tau(v)$.

ii. $u,v\in S, u\neq v$ y $\pi(u)=\pi(v)\Rightarrow\tau_f(u)\leq\tau(v)$ o $\tau_f(v)\leq\tau(u)$ y
iii. τ es una función no una relación.

La propiedad ii fuerza un orden en las operaciones que se asignan al mismo procesador y la operación se ejecuta sin desalojo hasta su terminación una vez que ha iniciado su ejecución. Se dice que una operación está liberada cuando no ha sido ejecutada y todos sus procesadores inmediatos han sido ejecutados o no tiene predecesores. El **tiempo de activación de una operación** v se define como:

$$\tau_r(v) = \max_{u \in P_r(v)} \tau_f(u)$$

y es el tiempo al cual se activa la operación v y es el máximo tiempo de terminación de los predecesores de v. Cuando una operación no tiene predecesores su tiempo de activación es cero por definición. Una operación liberada está lista para ser calendarizada y asignada. Usando la definición anterior, la restricción de precedencia se puede reformular como:

$$v \in S, \quad u \in P_r(v), \quad u \neq v \quad \text{y} \quad \pi(v) \neq \pi(u) \Rightarrow \tau_r(v) \leq \tau(v)$$

Del resultado anterior se tiene que si seleccionamos $\tau(v)$ tal que $\tau(u) = \tau_r(v)$ entonces se tiene un calendario válido. Sea $\alpha = (\tau, \pi)$ un agendador del grafo de dependencia $G = (S, R)$ extendido con costo de computación $w(v) \geq 0, \quad v \in S$ El costo de computación es una medida de las instrucciones que se requieren para ejecutar la operación. El **tiempo de ejecución del agendador** cuando se usan m procesadores se define como:

$$T_\alpha(m) = \max_{v \in S} \tau_f(v) - \min_{v \in S} \tau(v)$$

si $\min_{v \in S} \tau(v) = 0$ se tiene que:

$$T_\alpha(m) = \max_{v \in S} \tau_f(v)$$

Sin pérdida de generalidad suponemos que todos los calendarizadores inician al tiempo cero, es decir, que la expresión anterior da el tiempo de ejecución. τ_r no depende de a que procesadores se asignan las operaciones. El **tiempo de terminación de un procesador** es el tiempo en que termina la última operación que procesa, más formalmente, si p_i es un procesador:

$$\tau_f(p_i) = \max_{\substack{o \in S \\ \pi(o) = p_i}} \tau_f(o)$$

Los procesadores usados por un agendador son aquellos a los que se les asigna al menos una operación. Si U es el conjunto de **procesadores usados** entonces:

$$U = \{x | x = \pi(o) \quad \text{y} \quad o \in S\}$$

donde S es el conjunto de operaciones. Si se tienen m procesadores disponibles sucede que $|U| \leq m$.

A continuación mostramos que una función de calendarización se puede definir sobre los nodos de un grafo dirigido sin ciclos pero no sobre un grafo dirigido que contiene ciclos.

Teorema 4.1. Sea $G=(S,R)$ un grafo dirigido finito con una función w de costo de computación definida en S y evaluada en N Existe un calendarizador para G si y sólo si G no tiene ciclos.

Demostración. Si hay un ciclo, sea $s_0 \rightarrow s_1 \rightarrow ... \rightarrow s_k \rightarrow s_0$ de aquí se tiene que $s_0 R s_0$ y la función calendarizadora trataría de producir un función que cumple con $\tau(s_1)+w(s_1) \leq \tau(s_1)$. Dado que $w(s_1)>0$ se tiene una contradicción, es decir, la función calendarizadora no existe. Si no hay ciclo por el teorema 2.1 tenemos que hay al menos una clasificación topológica de (S,R) y de ésta se obtiene un calendarizador secuencial.

■

La **longitud computacional de una trayectoria** μ del grafo dirigido acíclico $G=(S,R)$ aumentado con costo de computación $w(v) \geq 0$, $v \in S$. Se define como la suma de los costos de computación asociados a los nodos en la trayectoria, más formalmente:

$$l_c(\mu)=\sum_{o \in \mu} w(o)$$

Note la diferencia entre esta definición y la definición donde la longitud de una trayectoria es el número de aristas de ésta. $l_c(\mu)$ se puede interpretar como el tiempo que toma ejecutar todas las operaciones que forman la trayectoria μ. La ejecución de las operaciones de una trayectoria es secuencial dado la dependencia entre las operaciones que forman la trayectoria. La **trayectoria crítica** μ_c de un grafo extendido con costos de computación es la trayectoria de peso más grande del grafo. La trayectoria crítica en general no es única. Más formalmente, la trayectoria crítica μ_c es la trayectoria μ que satisface:

$$l_c(\mu_c)=\max_{\mu} l_c(\mu)$$

donde μ es una trayectoria del grafo bajo consideración. Como el siguiente lema lo muestra, la longitud de una trayectoria crítica es una cota por abajo al tiempo de ejecución del grafo de dependencias.

Lema 4.2. Sean $G=(S,R)$ un dag finito extendido con costo de computación $w(o)\geq 0$ *para* $o\in S$, τ un calendarizador para G con m elementos procesadores, $T_\tau(m)$ el tiempo de ejecución para el calendarizador τ entonces:

$$T_\tau(m)\geq \sum_{j=1}^{i} w(v_j)\geq 0$$

para una trayectoria arbitraria $\mu=(v_0v_1\ldots v_k)$ del grafo G.

Demostración. Sea la trayectoria $\mu=v_0v_1\ldots v_k$, $k\geq 0$ de G y un calendarizador τ que satisface:

$$\tau(v_{n+1})\geq \tau(v_n)+w(v_n)\geq 0$$

Aplicando repetidamente la relación anterior se tiene que:

$$\tau(v_{k+1})\geq \tau(v_0)+\sum_{j=1}^{i} w(v_j) \qquad (4.4)$$

Dado que $\tau(v_0)\geq 0$ y $T_\tau(m)\geq \tau(v_{k+1})$ de la relación (4.4) se tiene que:

$$T_\tau(m)\geq \sum_{j=1}^{i} w(v_j)\geq 0$$

■

El significado de la trayectoria crítica es que sin importar el número de procesadores que se usen no es posible reducir el tiempo de ejecución abajo del costo de la trayectoria crítica. Es decir, no existe calendarizador que pueda ejecutar las operaciones del dag en menos tiempo que el peso de la trayectoria crítica. Un calendarizador puede tener diferentes metas, por ejemplo, optimizar tiempo de ejecución, optimizar número de procesadores bajo ciertas restricciones.

T_{opt} se llama el tiempo óptimo de ejecución y es el tiempo más corto en que se puede ejecutar las operaciones de un dag dado, con un número ilimitado de procesadores. Más formalmente sea s un calendarizador arbitrario entonces $T_{opt} = \min_s T_s$ donde T_s es el tiempo de ejecución cuando se usa el calendarizador s. Un calendarizador con tiempo de ejecución T_{opt} se llama **calendarizador óptimo en tiempo**. Note que si se tiene que $T_\tau(m) = l_c(\mu_c)$ entonces τ es un calendarizador óptimo. Sea T el tiempo de ejecución de un calendarizador y dag dados T es mejor mientras más cerca esté de T_{opt}. Note que por definición $T_{opt} \leq T$ Observamos que el calendarizador secuencial es óptimo en el número de procesadores y siempre requiere un sólo procesador para efectuar su cómputo. Un problema de mayor interés es encontrar el mínimo número de procesadores dado un tiempo de ejecución. N_{opt} es el número mínimo de procesadores que se requieren para efectuar las operaciones de un dag dado bajo la restricción que el tiempo de ejecución es T_{opt}. Un calendarizador con tiempo de ejecución T_{opt} y número de procesadores N_{opt} se llama **calendarizador óptimo en tiempo y procesadores**. Sea m el número de procesadores que un calendarizador y dag dados requieren. m es mejor mientras más cerca este de N_{opt}. Note que por definición $N_{opt} \leq m$.

Sea N_t el número de operaciones en ejecución al tiempo t cuando se usa un calendarizador óptimo en tiempo con tiempo de ejecución T_{opt}. D_t es el número de operaciones en ejecución cuando se tiene un calendarizador, no necesariamente óptimo, al tiempo t. D_t es el **perfil de ejecución del calendarizador** y N_t es el **perfil de paralelismo del programa**. El máximo número de procesadores requerido, por el segundo calendarizador $D_{\max}$, está dado por:

$$D_{\max} = \max_{0 \leq t \leq T-1} D_t \tag{4.5}$$

Definámos $N_{\max}$ por la siguiente expresión:

$$N_{\max} = \max_{0 \leq t \leq T_{opt}-1} N_t$$

Notamos que si el número de procesadores m es tal que $m > N_{\max}$ entonces, el paralelismo expuesto por el grafo no es suficiente para mantener todos los procesadores ocupados. En este caso el paralelismo de un sistema real está restringido por el programa. Si $m < N_{\max}$ también hay restricción al paralelismo pero en este caso está ocasionado por el número de elementos procesadores de la máquina.

Dos medidas de desempeño de sistemas paralelos son la eficiencia del uso de los procesadores y el factor de aceleración. La **eficiencia del uso de los elementos procesadores** se define como la utilización promedio de cada uno de los m procesadores y está dada por:

$$\eta(m) = \frac{\sum_{t=0}^{T(m)-1} D_t}{T(m) * D_{\max}}$$

Donde se supone que $D_{\max} = m$. El **factor de aceleración** es:

$$\sigma(m) = \frac{\sum_{t=0}^{T(m)-1} D_t}{T(m)} = \eta(m) * D_{\max}$$

Note que:

$$D \triangleq \sum_{t=0}^{T(m)-1} D_t = \sum_{t=0}^{T_{opt}(\infty)-1} N_t = T(1) \triangleq N$$

N se llama el **tiempo total de computación** y es el tiempo que un calendarizador secuencial con un procesador se tardaría en ejecutar el dag. Note que el factor de aceleración expresa la ganancia en tiempo de ejecución del sistema paralelo con respecto al tiempo de ejecución de un sistema secuencial.

El grafo de dependencia de un programa contiene la información completa acerca del paralelismo de éste. Esta información puede ser resumida por el perfil de paralelismo del grafo. Indicaciones del paralelismo de un programa aún más resumidas son el **paralelismo máximo** y el **paralelismo promedio** el cual se define como el número promedio de procesadores ocupados durante el tiempo de ejecución con un número ilimitado de procesadores. Esta definición es equivalente a las siguientes dos definiciones.

i. el factor de aceleración para un número ilimitado de procesadores y
ii. el cociente del tiempo total de computación sobre el tiempo de ejecución óptimo.

Según se puede ver de la siguiente relación:

$$\frac{\sum_{t=0}^{T_{opt}(\infty)-1} N_t}{T_{opt}(\infty)} = \frac{N}{T_{opt}(\infty)} = \frac{T_{opt}(1)}{T_{opt}(\infty)}$$

Lema 4.3. Sea $G=(S,R)$ un dag con $|S|$ finito, m el número de procesadores requeridos para implementar un calendarizador de tiempo óptimo, T_{opt}, el tiempo de ejecución óptimo y $n=|S|$. Entonces $\left\lceil \frac{n}{T_{opt}} \right\rceil \leq m$.

■

El resultado del lema 4.3 es claro si consideramos que los procesadores no tienen tiempo de ocio y que hacen su ejecución en T_{opt} en este caso se tienen $\frac{n}{T_{opt}}$ procesadores, si algunos procesadores tienen tiempos de ocio se obtiene la cota. Debido a las precedencias del dag puede suceder que se necesiten más procesadores que la cota por abajo en el lema 4.3.

Teorema 4.2. Sea $G=(S,R)$ un dag finito, $s(m)$ su calendarizador con m elementos procesadores, entonces la eficiencia η y el factor de aceleración σ satisface que:

$$0 \leq \eta \leq 1 \quad \text{y} \quad \sigma \leq D_{\max} \tag{4.6}$$

Es decir, que el máximo factor de aceleración que se puede obtener con m procesadores, está acotado por $D_{\max}$.

Demostración. Dada la relación (4.5) se tiene que $D_t \leq D_{\max}$, para $t=0, \ldots, T-1$, sumando los términos D_t, $t=0, \ldots, T-1$ se tiene que:

$$0 \leq \sum_{t=0}^{T-1} D_t \leq T * D_{\max}$$

Al dividir por $T * D_{\max}$ se obtiene la primera expresión de (4.6) y al dividir por T se obtiene la segunda.

■

Se dice que existe **factor de aceleración lineal** si la eficiencia permanece constante conforme se incrementa el número de procesadores. Es una situación idealizada donde se obtiene incremento del factor de aceleración sin afectar la eficiencia.

El siguiente teorema nos dice que dado un dag y m elementos procesadores mientras más procesadores se usan mayor es la posibilidad de reducir su tiempo de ejecución óptimo. Es decir, que al aumentar el número de procesadores los tiempos de ejecución óptimos para el número de procesadores no se ampliarán. Más adelante presentamos casos más complejos donde este resultado no es necesariamente cierto.

Teorema 4.3. Sean $G=(S,R)$ un dag finito y $T_{opt}(m)$ el tiempo óptimo con un calendarizador con m procesadores entonces:

$$T_{opt}(1) \geq T_{opt}(2) \geq \ldots T_{opt}(m) \geq T_{opt}(m+1) \geq \ldots T_{opt}(\infty) \qquad (4.7)$$

Demostración. Una manera de obtener un calendarizador con $m+1$ elementos procesadores es usar un calendarizador óptimo en tiempo con m elementos y nunca usar un elemento procesador. Lo anterior implica que:

$$T(m+1)=T_{opt}(m)$$

donde $T(m+1)$ es el tiempo de ejecución con el calendarizador con $m+1$ procesadores, por definición:

$$T_{opt}(m+1)\leq T(m+1)$$

De aquí se tiene que:

$$T_{opt}(m)\geq T_{opt}(m+1)$$

De donde se tiene la relación (4.7).

■

En el teorema anterior los procesadores mencionados son los disponibles, el que ahora se presenta se refiere a los procesadores usados.

Teorema 4.4. Sea un dag finito $G=(S,R)$ extendido con costos de computación $w(o)\geq 0$ para $o\in S$. m es el número de procesadores. C^q y C^{q+1} son clases de calendarizadores que usan respectivamente q y $q+1$ procesadores donde

$q < m$ y $q+1 \leq m$. Para calendarizadores óptimos en cada una de las clases C^q y C^{q+1} se cumple que:

$$T_{opt}^{(q+1)} \leq T_{opt}^{(q)}$$

Demostración. Sean $T_{opt}^{(q)}$ y $T_{opt}^{(q+1)}$ calendarizadores óptimos en las clases de calendarizadores C^q y C^{q+1} respectivamente. Q es el conjunto de procesadores usados por el calendarizador $T_{opt}^{(q)}$ Sea la operación $o \in S$ tal que $\pi(o) = p' \in Q$ y el tiempo de iniciación de ejecución de o es $\tau_{opt}^{(q)}(o)$. La operación o se recalendariza en procesador $p \in P - Q$ donde P es el conjunto de procesadores disponibles, el tiempo de activación de la operación o para esta nueva circunstancia es $\tau_r(o)$, el cual es menor o igual al que se tenia con q procesadores, esto se debe a que es la única operación en p. Recordamos que en el caso que nos concierne τ_r no depende de en que procesadores se asignan las operaciones. Los tiempos de activación de los sucesores de la operación no son más tarde de los que se tenían con q procesadores usados. Es decir, el nuevo calendario es válido si se selecciona $\tau(u) = \tau_r(u)$. El nuevo calendario usa $q+1$ procesadores y así tenemos que $T_{opt}^{(q+1)} \leq T^{(q+1)} \leq T_{opt}^{(q)}$ que es el resultado buscado.

■

El teorema anterior no es en general válido para cuando el grafo de dependencia también incluye costos de comunicación.

Existen situaciones donde se pueden usar menos de $N_{\max}$ procesadores y se puede lograr el tiempo mínimo de ejecución de un algoritmo. Es de interés calcular el óptimo número de procesadores, N_{opt}, con el cual se tiene el mínimo tiempo de ejecución de un algoritmo. T_{opt} y N_{opt} son características del grafo de dependencia y del mejor calendarizador. Dado un calendarizador arbitrario y un grafo de dependencia se puede obtener T y N para este par. T_{opt} y N_{opt} son los mejores valores contra los que se compara T y N La eficiencia óptima del uso de los elementos procesadores está dada por la siguiente expresión:

$$\eta_{opt} = \frac{\sum_{t=0}^{T_{opt}-1} N_t}{T_{opt} * N_{opt}} \tag{4.8}$$

El factor óptimo de aceleración del algoritmo, se expresa mediante:

$$\sigma_{opt} = \frac{\sum_{t=0}^{T_{opt}-1} N_t}{T_{opt}} = \eta_{opt} * N_{opt}$$

A continuación se muestra un calendarizador libre donde se supone que el costo de computación en general no es unitario. El calendarizador libre que explicamos es un calendarizador óptimo en tiempo. En este calendarizador cada operación se asigna a un procesador distinto. Se supone número ilimitado de procesadores. Cada operación inicia su ejecución tan pronto como es posible, es decir, en cuanto una operación se activa se inicia su ejecución, esto es se cumple que $\tau_r(u) = \tau(u)$ para toda operación o. $\tau_r(v)$ es la suma de los costos de computación de una trayectoria crítica que empieza en un elemento minimal del grafo de dependencias y termina en la operación v, el costo de v no se incluye en $\tau_r(v)$. El tiempo de iniciación de una operación está determinado por los tiempos de finalización de sus predecesores inmediatos.

Entrada: $G=(S,R)$ es un dag que define las relaciones de dependencia entre operaciones y costos de computación asociados a cada operación.
Salida: funciones de calendarización $\tau(v)$ y de asignación $\pi(v)$ para todas las operaciones $v \in S$.

```
p = 0;                                  /* variable para llevar registro de procesador asignado
Q = {x | x es elemento minimal de (S,R)};   /* Q es una cola con operaciones activadas

forEach(o ∈ Q)                          /* elementos de Q se accesan en orden
    {
      τ_r = 0;                          /* variable para calcular tiempo de activación
      forEach(v ∈ P_r'(o))              /* calcula tiempo de activación de o
              {
                τ_r = max(τ_r, τ_f(v));
              }
      τ(o)   = τ_r;                     /* tiempo de inicialización de o
      τ_f(o) = τ_r + w(o);              /* tiempo de terminación de o
      π(o)   = p++;                     /* asigna procesador a o
        W    = activeSuc(o);            /* encuentra sucesores inmediatos
                                        /* activados de operación o
        Q    = Q ∪ W;
    }
```

En el programa mostrado no se maneja un reloj global explícito. El siguiente programa muestra un calendarizador libre donde si se maneja el reloj global.

Entrada: $G=(S,R)$ es un dag que define las relaciones de dependencia entre operaciones y costos de computación asociados a cada operación.
Salida: funciones de calendarización $\tau(v)$ y de asignación $\pi(v)$ para todas las operaciones $v \in S$

```
timeStep = 0;                    /* variable para llevar registro de tiempo
F = ∅;                           /* operaciones que terminaron ejecución
P = ∅;                           /* todos los procesadores están desocupados

while(S − F ≠ ∅)
    {
      F = F ∪ releaseProcessor(timeStep, P);
      Q = updateReadyOp(F, S, R);                  /* calcula tiempo de iniciación
                                                      de operaciones activadas
      assignProcessorAndStartTime(Q, timeStep, P); /* asigna procesadores
                                                      a operaciones
      timeStep++;
    }
```

A continuación se explica brevemente los procedimientos que aparecen en el programa anterior.

releaseProcessor$(timeStep, P)$: retorna el conjunto de operaciones que terminan su ejecución al tiempo *timeStep*. Se actualizan los procesadores para marcar como desocupados los que se liberan de operaciones al tiempo *timeStep*.

updateReadyOp(F, S, R): retorna un conjunto con las operaciones que se pueden ejecutar dado que ya se han ejecutado las que están en el conjunto F, es decir, retorna las operaciones activas y proporciona el tiempo de inicio de las operaciones activas las cuales inician en el tiempo *timeStep*.

assignProcessorAndStartTime$(Q, timeStep, P)$: asigna las operaciones activadas en procesadores libres de P actualiza el estado de los procesadores para marcar como ocupados los recién asignados

Dado que a cada operación se le asigna un procesador la asignación producida por el agendador libre cumple con la restricción de precedencia. Que se cumple con la restricción de precedencia se ve del hecho de que el tiempo de inicialización de una

operación es igual al tiempo de activación de la operación. Así tenemos que el agendador libre es válido. Notamos que cada operación inicia su ejecución al tiempo más temprano posible, es decir, en cuanto la operación se activa y los tiempos de activación de las operaciones no se pueden reducir. Como consecuencia tenemos que el tiempo de ejecución es igual a la longitud de la trayectoria crítica del grafo de dependencia. Así hemos demostrado el siguiente teorema.

Teorema 4.5. Sea el dag $G=(S,R)$ extendido con el costo de computación $w(v)\geq 0, \quad v\in S$. Se supone que se tiene un número no limitado de procesadores. Se tiene que el agendador libre produce un calendarizador óptimo en tiempo.

Dos calendarizadores útiles en el estudio de calendarizadores son el calendarizador libre y el calendarizador libre inverso. En ambos casos se supone un número ilimitado de procesadores. El calendarizador libre despacha a ejecución todas las operaciones activadas, es decir, se tiene que $N_t = D_t$ para toda t como consecuencia el perfil de ejecución del calendarizador libre se puede usar para determinar el perfil de paralelismo de un programa representado por un dag. El calendarizador libre inverso puede introducir dilación en la ejecución de algunas operaciones activadas, sólo despacha a ejecución los que son inaplazables en el sentido que una dilación haría que el tiempo de ejecución del dag aumentara. Como se ve a continuación ambos calendarizadores tienen tiempo de ejecución óptimos. Los dos calendarizadores son válidos dado que no despachan a ejecución operaciones, que no están activadas. No se requieren más que $N_{\max}$ procesadores para lograr el tiempo mínimo de ejecución de un programa cuando se usa el calendarizador libre.

Ejemplo 4.2. En la figura 4.4 se muestra el recorrido del calendarizador libre a través del grafo de dependencia de un programa, a partir del grafo, se obtiene la información contenida en la tabla 4.1. En la figura 4.4, 0,1,...,6 denotan los tiempos a los cuales inicia la ejecución de las operaciones del programa. Suponemos que todas las operaciones tiene el mismo costo de computación.

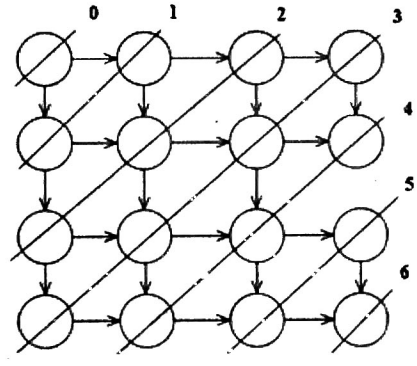


Figura 4.4: Grafo de dependencia.

Tiempo	Número de procesadores ocupados
0	1
1	2
2	3
3	4
4	3
5	2
6	1

Tabla 4.1: Número de elementos procesadores activados al aplicar el calendarizador libre.

La tabla 4.1 contiene el número de elementos procesadores que realizan un cómputo, en un tiempo determinado por el calendarizador libre. A partir de esta información se grafica el perfil de paralelismo de los elementos procesadores contra el tiempo en la figura 4.5.

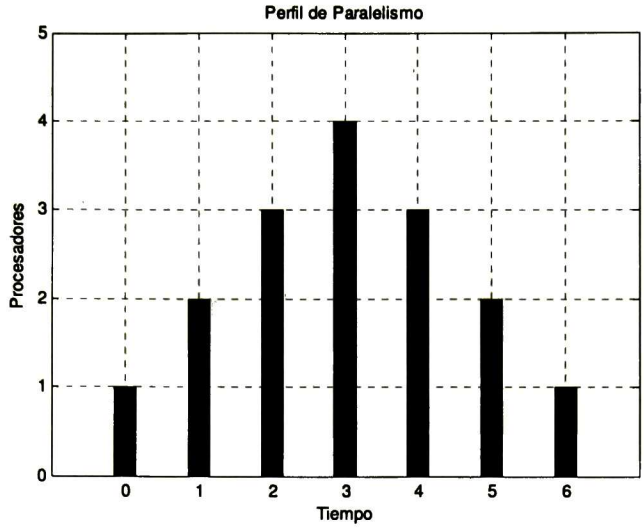


Figura 4.5: Perfil de paralelismo.

Calculamos la eficiencia del uso de los elementos procesadores con la ecuación (2.1):

$$\eta = \frac{\sum_{t=0}^{T_{opt}-1} N_t}{T_{opt} * N_{\max}} = \frac{16}{7*4} = 0.571$$

■

En el caso anterior no hay límite al número máximo de elementos procesadores que requiere el calendarizador libre. Este caso no es realista para cuando se requiere aumentar la eficiencia y se puede tolerar la consecuencia de tener tiempos de ejecución más largos que el tiempo óptimo de ejecución.

Ejemplo 4.3. En el ejemplo 4.2 el calendarizador libre requiere cuatro elementos procesadores para ejecutar el programa. Si analizamos a detalle, observamos que podemos ejecutar el algoritmo con el tiempo mínimo y con sólo tres procesadores, lo anterior se muestra en la figura 4.6, donde $0,1,\ldots,6$ denotan los tiempos a los cuales inicia la ejecución de las operaciones del programa. El lector puede fácilmente comprobar que no se puede tener un tiempo de ejecución menor para un número de procesadores menor a tres. En base al grafo de dependencia se obtiene la información contenida en la tabla 4.2.

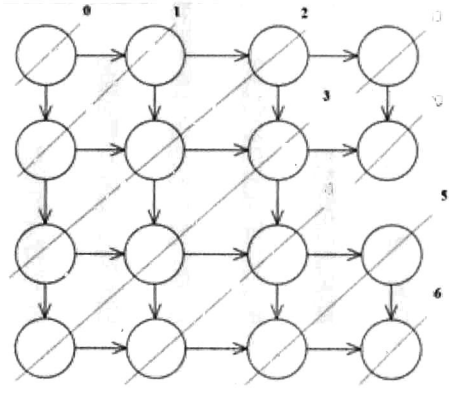


Figura 4.6: Grafo de dependencia.

Tiempo	Número de procesadores ocupados
0	1
1	2
2	3
3	3
4	3
5	3
6	1

Tabla 4.2: Número de elementos procesadores activados al aplicar el calendarizador libre.

La tabla 4.2 contiene el número de elementos procesadores que realizan un cómputo, en un tiempo determinado por el calendarizador libre. De aquí se puede ver que no se requieren más de tres procesadores. A partir de esta información se grafica el perfil de paralelismo de los elementos procesadores contra el tiempo en la figura 4.7.

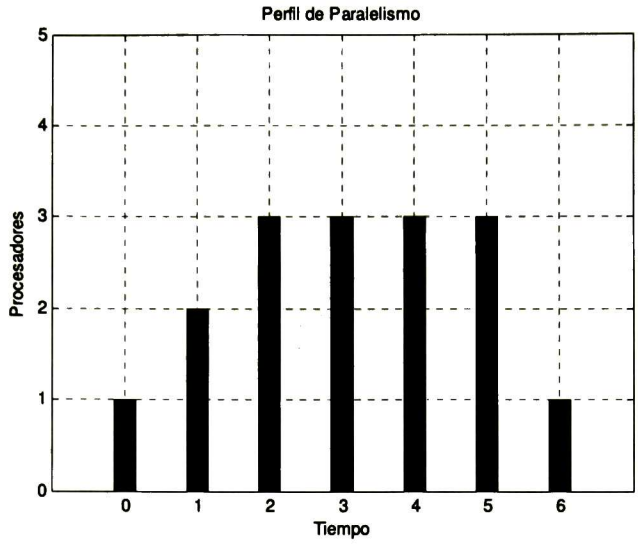


Figura 4.7: Perfil de paralelismo.

Calculamos la eficiencia del uso de los elementos procesadores, cuando se aplica el calendarizador libre con la ecuación (2.1):

$$\eta_{opt} = \frac{\sum_{t=0}^{T_{opt}-1} N_t}{T_{opt} * N_{opt}} = \frac{16}{7*3} = 0.76$$

La cual es una eficiencia mayor que la del ejemplo anterior, de hecho esta es la mayor eficiencia que se puede alcanzar para el perfil de paralelismo dado.

■

Note que un algoritmo cualquiera no se puede correr en un tiempo menor al tiempo óptimo y que cualquier algoritmo que se ejecuta en el tiempo óptimo no puede usar los procesadores con una mejor eficiencia que la eficiencia óptima. Es decir, el tiempo mínimo de ejecución usando un calendarizador arbitrario es mayor o igual al tiempo óptimo y la máxima eficiencia en el uso de los elementos procesadores es menor o igual que la eficiencia óptima.

Teorema 4.6 (de Graham y Brent). Sea $G=(S,R)$ un dag extendido con costo de computación $w(v)$ unitario para $v \in S$ entonces se tiene que:

$$T_{opt}(1) \leq T(m) \leq \frac{T_{opt}(1)-T_{opt}(\infty)}{m}+T_{opt}(\infty) \tag{4.9}$$

Donde m es el número de procesadores y $T(m)$ es el tiempo de ejecución de un calendarizador voraz. También se tiene que:

$$\sigma(m) \geq \frac{m\dfrac{T_{opt}(1)}{T_{opt}(\infty)}}{\left(m+\dfrac{T_{opt}(1)}{T_{opt}(\infty)}-1\right)} \tag{4.10}$$

y

$$\eta(m) \geq \frac{\dfrac{T_{opt}(1)}{T_{opt}(\infty)}}{\left(m+\dfrac{T_{opt}(1)}{T_{opt}(\infty)}-1\right)} \tag{4.11}$$

■

Las cotas del teorema anterior son válidas para cualquier calendarizador voraz: no hay procesador ocioso si hay operación activada, lista a ser ejecutada. La forma de la cota no depende del grafo de dependencia.

El **tiempo más temprano de ejecución** de una operación está dado por la longitud de la trayectoria más larga que termina en la operación y que empieza en una operación que no tiene predecesor y es igual al tiempo de activación de la operación. Este tiempo es independiente del calendarizador usado.

El **tiempo más tarde de ejecución** de una operación está dado por la diferencia entre el tiempo óptimo de ejecución del grafo de dependencia menos la longitud más larga de una trayectoria que termina en la operación y se inicia en una operación que no tiene sucesores en el grafo de dependencia. La trayectoria es sobre el grafo de dependencia inverso el cual se obtiene del grafo de dependencia al invertir la dirección de todos los arcos del grafo original.

Notamos que los tiempos de iniciación de ejecución de las operaciones en el caso del calendarizador libre ocurren a los tiempos más tempranos de ejecución de las operaciones. Así tenemos que el calendarizador libre puede ser usado para producir los tiempos más tempranos de ejecución.

Sea $G=(S,R)$ un dag finito con $n=|S|$ finito. A continuación definimos las funciones $tl(v)$ y $bl(v)$ para $v\in S$, las cuales se llaman respectivamente **nivel de la operación v desde el inicio** y **nivel de la operación v desde el fin**. $tl(v)$ es el costo de la trayectoria de mayor costo de una operación sin predecesor a la operación v, se excluye el peso de v. $bl(v)$ es el costo de la trayectoria de mayor costo que empieza en v y termina en un nodo sin sucesores, se incluye el peso de v. En los dos casos puede haber más de una trayectoria de mayor costo. $bl(v)$ también se llama el costo de la **trayectoria crítica** de v. Se tiene que:

$$tl(v)=\begin{cases} 0, & \text{si } v \text{ no tiene predecesor} \\ \max\limits_{u}\{tl(u)+w(u)\}, & \text{si } u \text{ es predecesor inmediato de } v \end{cases}$$

$$bl(v)=\begin{cases} w(v), & \text{si } v \text{ no tiene sucesor} \\ \max\limits_{u}\{bl(u)+w(v)\}, & \text{si } u \text{ es sucesor inmediato de } v \end{cases}$$

$tl(v)$ es el tiempo más temprano al cual se puede ejecutar la operación v. Al tiempo de iniciar la ejecución de la operación v, el tiempo que resta para terminar la ejecución del grafo de dependencia esta dado por $bl(v)$. $T_{opt}-bl(v)$ es el tiempo más tarde al cual se puede ejecutar la operación v. $tl(v)$ y $bl(v)$ son también llamados respectivamente el **conivel** y **nivel del nodo** v.

Sea el grafo dirigido $G=(S,R)$, el grafo inverso de G, denotado por G^{-1}. De acuerdo con el método de la **trayectoria crítica**, $tl(v)$ se calcula navegando el grafo (S,R) en forma progresiva y $bl(s)$ navegándolo en forma regresiva. Note que la navegación en forma regresiva es equivalente a navegación en forma progresiva del grafo inverso.

A cada nodo se le asocia su **holgura** o **movilidad** la cual se define como la diferencia entre su tiempo más tarde de ejecución y su tiempo más temprano de ejecución, en particular:

$$s(v)=T_{opt}-bl(v)-tl(v)$$

La holgura de una operación es el tiempo que la operación se puede retardar después de su tiempo más temprano de ejecución sin afectar el tiempo de ejecución. Una operación que tiene holgura cero se dice que está en la trayectoria crítica. Cualquier extensión en el tiempo de una operación crítica tiene como efecto una extensión del tiempo de ejecución. Las operaciones que no son críticas pueden mover sus tiempos de inicialización o alargar su duración hasta agotar su holgura. Las operaciones en la ruta crítica no se pueden alargar sin afectar el tiempo de ejecución.

Sea S el conjunto de operaciones de un dag. Ahora generamos dos particiones de S. La partición E_i, $0 \le i \le T_{opt}$ está inducida por los tiempos más tempranos de despacho y la partición L_i, $0 \le i \le T_{opt}$ está inducida por los tiempos más tardes de despacho. En particular:

i. $p, o \in E_i$ si y sólo si o y p tienen i como el tiempo más temprano de despacho y
ii. $p, o \in L_i$ y sólo si o y p tienen i como el tiempo más tarde de despacho.

E_0 contiene las operaciones que no tienen predecesor y L_0 las que no tienen sucesor. E_i tiene los predecesores de los elementos en E_{i+1}. El calendarizador libre requiere ejecutar las operaciones en E_i al tiempo i. El **calendarizador libre inverso** requiere ejecutar las operaciones en L_i al tiempo i.

$E(o)$ y $L(o)$ son respectivamente el tiempo **más temprano de despacho y el tiempo más tardío de despacho de la operación** o. Sea $C_i = E_i \cap L_i$, $0 \le i \le T_{opt} - 1$, entonces C_i es el conjunto de operaciones que se ejecutan al tiempo i. Si la operación o está en la trayectoria crítica se tiene que $E(o) = L(o)$ y como existe al menos una trayectoria crítica se tiene que $C_i \neq \varnothing$ para $0 \le i \le T_{opt} - 1$. $[E(o), L(o)]$ es el rango u holgura permisible de tiempo de despacho para la operación o, que no afecta el tiempo de ejecución óptimo.

Teorema 4.7. Sea $G(S,R)$ un dag con $|S| = n$ finito y N_{opt} el número óptimo de procesadores requeridos por el calendarizador óptimo, entonces se tienen las siguientes cotas inferior y superior para N_{opt}:

$$\max\left\{\max_i C_i, \frac{N}{T_{opt}}\right\} \le N_{opt} \le \min\left\{\max_i |E_i|, \max_i |L_i|\right\}$$

■

4.3.1 Agendador con Lista de Prioridad

Dos clases de algoritmos heurísticos para crear agendadores son: basados en lista de prioridad y basados en aglutinamiento. La gran mayoría de agendadores pertenecen a alguna de estas clases. En esta sección estudiamos agendadores con lista de prioridad.

Los agendadores con listas de prioridad que inicialmente presentamos son algoritmos heurísticos 2-aproximativos. El objetivo de estos agendadores es minimizar el tiempo de ejecución y para esto usan alguna de las siguientes estrategias: orientada al procesador y orientada a la operación. En la estrategia orientada al procesador siempre se asignan operaciones activadas a los elementos procesadores libres. Es decir, que los agendadores con listas de prioridad no dejan desocupado de manera deliberada un elemento procesador cuando existen operaciones activadas. En la estrategia orientada a la operación una operación activada se asigna al procesador que permite su terminación más temprana o lo que es lo mismo su iniciación más temprana. Una operación es **liberada** cuando todas sus predecesoras han sido ejecutadas y es **activada** si es liberada y todos los resultados que requiere para su ejecución están disponibles. Note que si no hay retardo de comunicación los conceptos de operación liberada y activada son equivalentes, sin embargo no sucede así si hay retardos de comunicación.

El modelo de computación más simple de computación paralela que se nos presenta es cuando no hay relación de precedencia entre las operaciones y éstas no se comunican entre sí. Se requiere un agendador que minimice el tiempo de ejecución. Si el número de procesadores es ilimitado, la solución directa es asignar una operación a cada procesador y empezar la ejecución de todas las operaciones al tiempo cero, como consecuencia se tiene el mismo número de procesadores que de operaciones. En seguida estudiamos algoritmos de agendamiento que suponen un número limitado de procesadores donde las operaciones son independientes.

Sea S el conjunto de operaciones y P el de procesadores. Inicialmente consideramos el caso donde no existen restricciones de precedencia entre las operaciones, esto es se pueden ejecutar en cualquier orden, y suponemos que el número de éstas es n. Los procesadores no requieren comunicarse entre ellos y hay m de ellos. La función w, que toma valores en los enteros no negativos, asocia a cada operación su costo de computación. Sea π la función que asigna cada operación a un procesador. El tiempo total de ejecución viene dado por:

$$T = \max_{p \in P} \sum_{p=\pi(v)} w(v)$$

Es decir, que T es el tiempo de ejecución del procesador, que tarda más ejecutando operaciones. Dada la descripción del problema anterior y una $k \in \mathbb{R}^+$ El diseño del asignador π tal que $T < k$ es un problema NP completo (Bruno et al [1974]) en el caso

de dos o más procesadores homogéneos. Dado este resultado recurrimos a los algoritmos aproximativos y a los heurísticos.

Uno de los algoritmos heurísticos más comunes para agendar las operaciones de un grafo dirigido extendido con costos de computación es el algoritmo con listas de prioridad. Los **agendadores con listas de prioridad** son una clase de agendadores incrementales y voraces en la cual asociado a cada operación se tiene una prioridad y todas las operaciones activadas están en una lista ordenada por prioridad, de allí el nombre del agendador. La estructura básica del algoritmo es muy simple. Inicialmente se le asigna una prioridad a cada operación y se construye una lista de prioridad donde se ordenan las operaciones de acuerdo a su prioridad. La prioridad de las operaciones sólo se calcula una vez al inicio del algoritmo. El algoritmo es incremental y voraz en el sentido que va construyendo un agendador conforme va considerando cada una de las operaciones y que no se tienen procesadores desocupados si se tienen tareas listas a ser ejecutadas. Cada vez que se desocupa un procesador se selecciona la operación de mayor prioridad de la lista de prioridad y se asigna al procesador desocupado, este procedimiento se repite hasta agotar las operaciones en la lista. En el primer algoritmo con listas de prioridad que tratamos todas las operaciones tienen la misma prioridad y no hay precedencia entre ellas. Es en la última fase del algoritmo donde se efectúa la asignación de procesadores y calendarización de operaciones. El agendador con lista de prioridad asigna un procesador a una operación si el procesador permite la ejecución más temprana de la operación.

Observamos que el agendador con lista de prioridad construye agendadores parciales válidos antes de tener el agendador final válido. Dado que el algoritmo es incremental un agendador parcial siempre se construye sobre los agendadores parciales que le preceden en el proceso de construir un agendador final. No se desechan o cambian decisiones tomadas con anterioridad. La asignación de una operación a un procesador no depende de que operaciones futuras se van a asignar. El algoritmo es voraz en el sentido que cada operación se asigna al procesador menos cargado. El criterio de asignación de operación a procesador trata de mantener la carga de todos los procesadores balanceada. El problema descrito corresponde al caso de ejecución de programas independientes por un multiprocesador por lotes.

Sólo consideramos agendadores con lista de prioridad sin desalojos, esto es una vez que una operación inicia su ejecución no se interrumpe para que su elemento procesador sea asignado a otra operación. Cuando hay desalojo las operaciones que se ejecutan siempre son las que tienen mayor prioridad y una operación puede ser interrumpida. En el caso sin desalojo puede suceder que una operación de menor prioridad se ejecuta mientras una de mayor prioridad y activada espera por un elemento procesador.

A continuación se presenta un algoritmo que efectúa el agendamiento con listas de prioridad. Una operación puede estar en sólo una de los siguientes estados: en ejecución, activada, no activada y ejecutada. Todas las operaciones activadas están en la

cola de operaciones activadas Q. Las operaciones en ejecución están en la cola P y las no activadas en el conjunto S. En general diferente asignación de prioridades resulta en diferentes agendadores.

Entrada: conjunto de operaciones S y vector de costo de computación $w(v)$ asociado a los elementos de $v \in S$.
Salida: funciones de calendarización $\tau(v)$ y de asignación $\pi(v)$ para todas las operaciones $v \in S$.

```
enqueuePriorityQ(Q,S);          /* inserta de acuerdo a su prioridad en Q

forEach(o ∈ Q)                  /* se seleccionan por orden de prioridad
        {
          p = assignProc(P,o);  /* asigna procesador de P a la operación o
          schedule(o,p);        /* calendariza operación o en procesador p
        }
```

Listado 4.1: Agendador con lista de prioridad.

El objeto del calendarizador es minimizar el tiempo de ejecución del agendador. Si el tiempo de iniciación de la operación $o \in S$ es $\tau(o)$ entonces su tiempo de terminación está dado por:

$$\tau_f(o) = \tau(o) + w(o)$$

el tiempo de ejecución del agendador es:

$$T = \max_{o \in S} \tau_f(o)$$

Es decir, que minimizando los tiempos de terminación de las operaciones y como consecuencia los tiempos de iniciación de éstas se minimiza la longitud de ejecución del agendador.

el tiempo de iniciación de la operación o cuando se asigna al procesador p es:

$$\tau(o) = \max_{\substack{u \in S' \\ \pi(u)=p}} \tau_f(u) \tag{4.12}$$

donde S' es el conjunto de operaciones a las cuales ya se les ha asignado procesador y tiempo de iniciación. Note que $\tau(o)$ es en este caso el tiempo al cual el procesador p termina la última operación que le fue asignada. La asignación de procesador a la operación o es tal que $\pi(o) = p_{\min}$ donde $p_{\min}$ es el procesador que minimiza la expresión:

$$\max_{\substack{u \in S' \\ \pi(u)=p}} \tau_f(u) \tag{4.13}$$

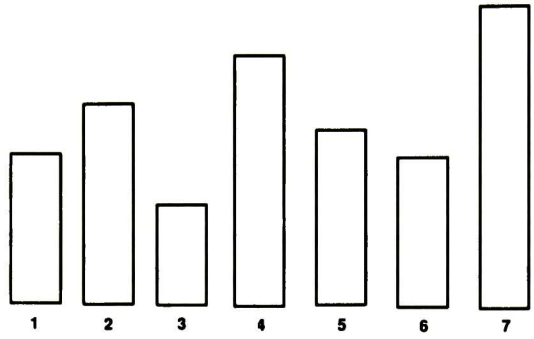
Es decir, la asignación de procesador es tal que minimiza el tiempo de iniciación de la operación o. Para calcular el agendador cuando se selecciona la operación o, primero se encuentra $p_{\min}$ usando la relación (4.13) y a la operación o se le asigna el procesador $p_{\min}$, luego se usa la relación (4.12) para calcular el tiempo de inicialización de la operación o. Note que los agendadores generados son válidos. Notamos que las siguientes estrategias:

i. en cuanto un procesador se desocupa se le asigna la operación de mayor prioridad y
ii. a la operación de mayor prioridad es asignada el procesador que le permite iniciar su ejecución al tiempo más temprano.

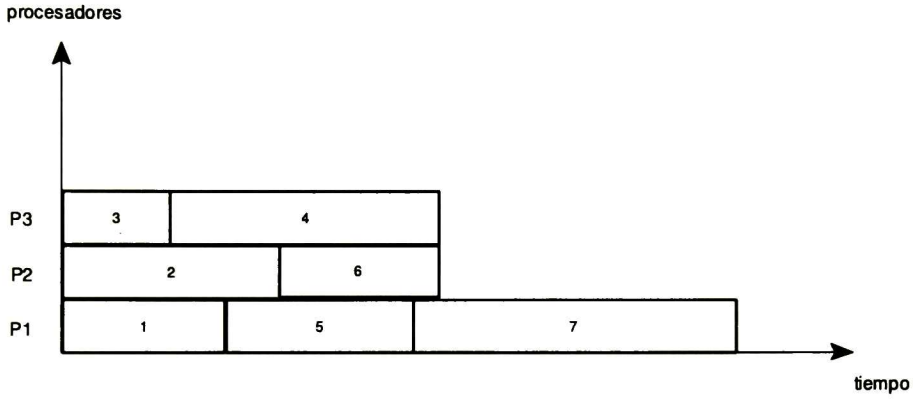
dan lugar a las mismas funciones de asignación de procesador y calendarización.

La primera estrategia se dice orientada al procesador y la segunda orientada a la operación. En la estrategia orientada a procesador se trata de tener un uso eficiente de los procesadores para la cual se balancea la carga asignada a cada uno de ellos y en la orientada a las operaciones se trata de terminar su ejecución lo mas pronto posible. En los dos casos se trata de disminuir el tiempo de ejecución del agendador. Las estrategias orientadas al procesador y a la operación son estrategias locales, esto es no usan información global para sus decisiones.

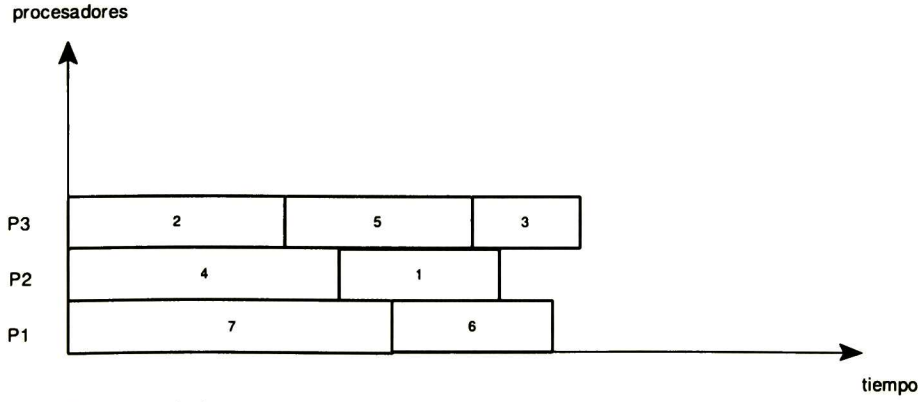
Ejemplo 4.4. Se presentan dos agendadores con lista de prioridad para un grupo de operaciones independientes. Los dos agendadores son incrementales y voraces en su segunda fase. El primero asigna las operaciones de acuerdo al número que se muestra en la figura 4.8, en el segundo agendador a las operaciones se les asigna una prioridad de acuerdo a su tiempo de computación. Mientras más largo el tiempo de computación mayor su prioridad. Observamos que el segundo agendador tiene menos tiempo de ejecución.



a) Operaciones.



b) Agendador con misma prioridad para todas las operaciones.



c) Agendador con operaciones prioritizadas de acuerdo a su tiempo de computación.

Figura 4.8: Dos agendadores con lista de prioridad.

■

Teorema 4.8. El algoritmo de agendamiento con lista de prioridad para operaciones sin restricciones de dependencia ni comunicaciones entre ellas es $\left(2-\frac{1}{m}\right)$-aproximativo. En particular $T \leq \left(2-\frac{1}{m}\right)T_{opt}$. La cota anterior es estricta.

■

Note que si todas las operaciones tienen el mismo costo de computación entonces el algoritmo de agendamiento con listas de prioridad trata que el número de operaciones asignadas a cada uno de los procesadores sea igual. El proceso que trata que todos los procesadores tengan igual carga se llama **balanceo de cargas**. Si el número de procesadores es más grande que el de las operaciones a cada procesador se le asigna a lo más una operación.

Del ejemplo 4.4 se observa que el costo de computación de la última operación en terminar tiene una influencia grande en el cálculo del tiempo de ejecución. Mientras menor el costo de computación menos es el tiempo de ejecución del agendador. Este comportamiento sugiere la siguiente heurística: calendarizar las operaciones con costo de computación grande primero que las de costo pequeño. Esta estrategia se usó en uno de los agendadores del ejemplo 4.4. Notamos que en el agendador con lista de prioridades hay dos selecciones: la de operación y la de procesador. En el caso propuesto se tiene una estrategia global para seleccionar una operación de la lista de prioridad y una estrategia local para la selección del procesador. Si las operaciones se ordenan crecientemente con respecto a su tiempo de computación y en la segunda fase se usa la misma estrategia voraz del caso anterior entonces se tiene un algoritmo que es $\left(\frac{4}{3}-\frac{1}{3m}\right)$-aproximativo. Note que este nuevo algoritmo no es incremental, dado que inicialmente es necesario ordenar las operaciones de acuerdo a su prioridad. Los siguientes dos lemas son útiles para encontrar las cotas de tiempo de ejecución del calendarizador con lista de prioridad que ordena las operaciones en orden decreciente de costo de computación.

Lema 4.4. Si a cada procesador se le asigna a lo más una operación por un agendador óptimo, el agendador con lista de prioridad con la regla de prioritizar más alto las operaciones con tiempo de computación más largos es también un agendador óptimo.

■

Lema 4.5. Si a cada procesador se le asigna a lo más dos operaciones por un agendador óptimo, el agendador con lista de prioridad con la regla de prioritizar más alto las operaciones con el tiempo de ejecución mas largo es también un agendador óptimo.

■

Teorema 4.9. El algoritmo de agendamiento con lista de prioridad donde las operaciones se ordenan en orden decreciente de costo de computación, sin restricciones de dependencia ni comunicaciones entre operaciones es $\left(\frac{4}{3}-\frac{1}{3m}\right)$- aproximativo. En particular $T \leq \left(\frac{4}{3}-\frac{1}{3m}\right)T_{opt}$. La cota anterior es estricta.

■

Sea un grafo dirigido $G=(S,R)$ donde $S=\{o_1,o_2,\ldots,o_n\}$ es el conjunto de operaciones y $o_{j_1},o_{j_2},\ldots,o_{j_r}$. $r \leq n$ es una cadena de R donde o_{j_1} es un elemento minimal de S y o_{j_r} es uno maximal. Suponemos que o_{j_r} es la operación que determina el final del tiempo de ejecución del calendario. Sea T_c los intervalos donde las operaciones o_{j_i}, $1 \leq i \leq r$. se ejecutan y $T_{\bar{c}}$ los intervalos donde no se ejecutan operaciones o_{j_i}, $1 \leq i \leq r$. Cualquier tiempo $t \in [0,T]$ esta o en T_c o en $T_{\bar{c}}$. Si $|T_c| = \sum_{1 \leq i \leq r} w(o_{j_i})$ y $|T_{\bar{c}}|$ denotan el total de tiempo respectivamente en T_c o $T_{\bar{c}}$, se tiene que $T = |T_c| + |T_{\bar{c}}|$. Notamos que para toda $t \in T_{\bar{c}}$ todos los procesadores están ocupados, de no ser así tendríamos un procesador libre y una operación activa en contradicción a la política voraz del calendarizador con listas de prioridad.

Lema 4.6. Sea un dag $G=(S,R)$ con costo de computación $w(v) \geq 0$, $v \in S$. Se tienen m procesadores. $o_{j_1},o_{j_2},\ldots,o_{j_r}$, $1 \leq r \leq n$ en una cadena en R donde o_{j_r} determina el tiempo de ejecución del calendario entonces:

$$idle \leq (m-1)\sum_{1 \leq i \leq r} w(o_{j_i}) \leq (m-1)T_{opt}$$

■

A continuación se presenta un algoritmo que efectúa el agendamiento con listas de prioridad. Se supone que existe un orden de precedencia entre operaciones. La mayor diferencia entre este programa y el anterior es en el manejo de la lista de prioridad y de su interrelación con el grafo de dependencia.

Entrada: dag $G=(S,R)$ que define relación de precedencia entre operaciones y costo de computación de los elementos de S.
Salida: funciones de calendarización $\tau(v)$ y de asignación $\pi(v)$ para todas las operaciones $v \in S$.

```
setPriority(S);

Q = {x | x is a minimal element of S};        /* lista de prioridad
L = ∅;                                        /* operaciones ejecutadas
while(Q ≠ ∅)
    {
      o = removeAndSelectHighestPrty(Q);      /* selecciona y
      L = LU{o};                              /* quita operaciones de Q
      p = assignProc(P,o);
      schedule(o,p);

      forEach(u ∈ Su(o))                      /* incluye en Q operaciones
                                              /* activadas susesoras de o
            {
              include = time;
              forEach(v ∈ P_r(v))
                    {
                      if (v ∉ L)  include = false;
                    }
              if (include)  Q = Q ∪ {u};
              }
    }
```

Listado 4.2: Agendador con lista de prioridad.

A continuación mostramos que si las condiciones del teorema 4.8 se modifican para ahora tener dependencia entre operaciones, la relación entre el tiempo de ejecución del agendador con listas de prioridad y el tiempo de ejecución del calendarizador óptimo se mantienen como lo enuncia el teorema 4.8.

Teorema 4.10. Sea un dag $G=(S,R)$ con costo de computación $w(v)\geq 0,\quad v\in S$ y τ la función de calendarización producida por agendamiento con listas de prioridad, entonces:

$$T\leq\left(2-\frac{1}{m}\right)T_{opt}$$

Donde T y T_{opt} son los tiempos de ejecución respectivamente producidos por el agendador con listas de prioridad y el óptimo. La cota anterior es estricta.

■

En el agendador con relaciones de precedencia considerado en el teorema 4.10 se tiene que el tiempo de terminación de la operación o está dado por:

$$\tau_f(o)=\tau(o)+w(o)$$

El tiempo de activación de una operación o está dado por:

$$\tau_r(o)=\max_{u\in P_r(o)}\tau_f(u)$$

El tiempo de finalización del procesador p está dado por:

$$\tau_f(p)=\max_{\substack{o\in S'\\ \pi(o)=P}}\tau_f(o)$$

donde S' es el conjunto de operaciones a las cuales ya se les ha asignado procesador y tiempo de iniciación. El tiempo de iniciación de la operación o cuando se asigna al procesador p está dado por:

$$\tau(o) = \max\{\tau_f(p), \tau_r(o)\}$$

La asignación de procesador a la operación o es tal que $\pi(o) = p_{\min}$ donde $p_{\min}$ es el procesador que minimiza $\tau_f(p)$. Es decir, la asignación de procesador es tal que se minimiza el tiempo de iniciación de la operación o sin violar la restricción de precedencia de ésta. Note que los agendadores parciales y el final son validos, esto se debe a que las operaciones se seleccionan para ejecución de la lista de prioridad cumpliendo con la restricción de dependencia y que el tiempo de iniciación de una operación es igual al tiempo de activación, es decir, es tal que todos los predecesores de la operación ya terminaron cuando ésta empieza. Aquí resulta de nueva cuenta que los criterios de mantener siempre ocupados a los procesadores y de asignar una operación al procesador que le permite ejecutar lo más temprano producen la misma función de asignación de procesador.

La lista de prioridad especifica el orden deseado en que se desean las operaciones ejecutadas. El orden en que se seleccionan las operaciones de la lista de prioridades es un orden total y debe ser tal que sea compatible con el orden de precedencia definido por el grafo de dependencias. De no cumplir con este criterio se tiene el caso de que una operación trata de ser ejecutada cuando sus datos de entrada todavía no han sido arrojados, es decir, el agendador no seria válido. Para que el agendador siempre sea válido, de las operaciones no ejecutadas sólo se consideran las activadas para ser ejecutadas. Es decir, cuando hay contradicción entre el orden especificado por las prioridades y el de precedencia éste último prevalece.

La prioridad de una operación se puede basar sólo en los parámetros de la operación o también tomar en cuenta los parámetros de todas las operaciones. La prioridad se calcula una sola vez durante toda la ejecución del calendarizador. Un calendarizador como el anterior se dice **no adaptativo** en contraposición a un **adaptativo** que puede calcular las prioridades de las operaciones por cada iteración y toma en cuenta más parámetros. Otro cálculo que un calendarizador adaptativo pudiera cambiar durante la ejecución es la política de cómo se seleccionan las operaciones activadas y los procesadores. En un calendarizador adaptativo las políticas del calendarizador cambian durante la ejecución de éste. Como es claro los agendadores adaptativos son computacionalmente más costosos.

Algunos autores llaman agendador con lista de prioridad a un agendador que usa el mismo algoritmo que el agendador con listas de prioridad excepto que la asignación de procesador no se efectúa por minimización de tiempo de inicialización de operación.

Nosotros llamamos a estos agendadores como **agendadores por listas de prioridad sin minimización** para así evitar confusiones.

Existe una gran variedad de algoritmos para asignar prioridades a las operaciones cuando se usa agendamiento con lista de prioridades aquí sólo presentamos algunos pocos. La asignación de prioridades debe ser tal que se logra que cada operación se ejecute en los tiempos determinados por su holgura. Si lo anterior no se logra el tiempo de ejecución se incrementa. Una manera de asignar prioridades es la de dar prioridad a las operaciones de acuerdo a su costo de computación, mientras mayor costo de computación mayor prioridad. Una manera muy socorrida de asignar prioridad a una operación es usar el peso de la trayectoria crítica de $v \in S$, esto es $bl(v)$. Es decir, que las operaciones con las trayectorias críticas de mayor peso se consideran más prioritarias. Cuando múltiples operaciones tienen la misma prioridad otra regla se requiere para desbaratar el empate. Esta nueva regla puede ser alguna de las reglas de asignación de prioridad que aquí explicamos. Aquí suponemos que los empates se rompen aleatoriamente. En el caso descrito el agendamiento se dice con lista de costos de trayectorias críticas. Se ha demostrado experimentalmente que este método de agendamiento (Adam et al [1974]) está dentro de 5% del óptimo en el 90% de los casos. Es decir que el comportamiento promedio es mucho mejor que el sugerido por las cotas antes presentadas.

4.4 Diseño de un Sistema con Multiprocesadores

Dos ingredientes en el proceso de implementación de un sistema paralelo son: el modelo del programa a ejecutar y el modelo de la máquina para ejecutar el programa. La figura 4.9 muestra la interacción de estos elementos para llegar eventualmente a una implementación. Durante el agendamiento los procesadores y otros recursos se asignan a las operaciones del programa y se determinan los tiempos de ejecución de las tareas. Si en la fase de diseño se tienen cifras de desempeño no deseadas se puede modificar la arquitectura de la máquina, el programa o el agendamiento, las líneas punteadas en la figura 4.9 indican esta iteración en el proceso de diseño de un sistema. La iteración se puede repetir varias veces. En cada iteración el programa y la máquina se refinan. En el proceso de agendamiento se produce un calendarizador y un asignador y en su diseño se tiene como meta optimizar un criterio predado. En el proceso de diseño del calendarizador y asignador el modelo del programa y el modelo de la máquina son entradas.

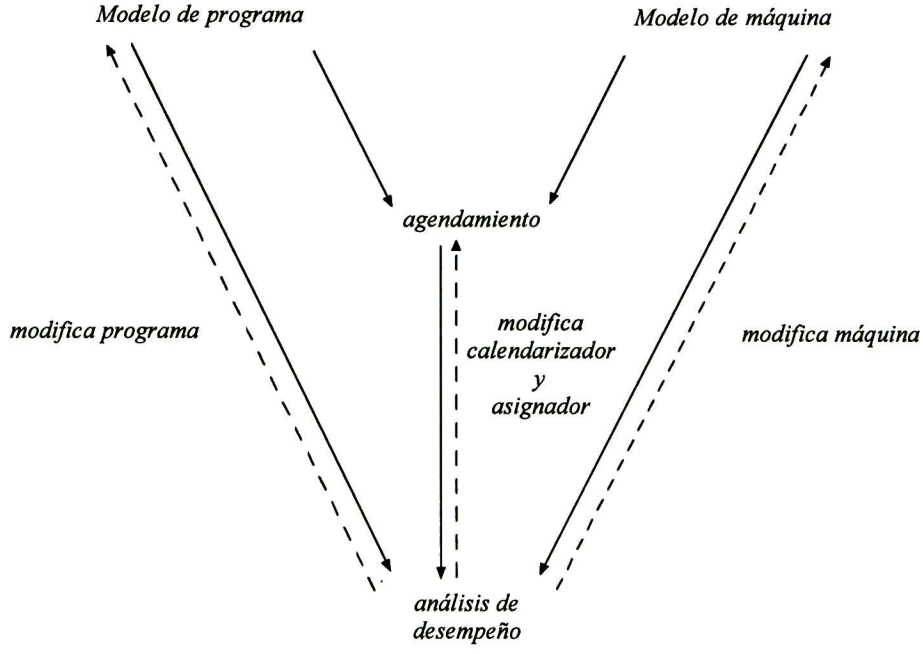


Figura 4.9: Papel del agendamiento en el diseño de un sistema.

4.4.1 Agendamiento

El modelo del programa expone el paralelismo de la aplicación y el modelo de la máquina impone restricciones al paralelismo del programa. Durante la fase de agendamiento se asignan procesadores a las operaciones y se determina al tiempo en que éstos inician su ejecución respetando las dependencias del modelo del programa y las restricciones impuestas por el modelo de la máquina y esto se hace optimizando funciones objetivo o criterios. El problema de diseño de un agendador es un problema complejo de optimización combinatoria multiobjetivo. La figura 4.10 muestra una taxonomía parcial de los métodos de calendarización.

En agendamiento local se tiene un solo procesador mientras que en agendamiento global se tienen múltiples procesadores. En **agendamiento local** es requerido ordenar la ejecución de las operaciones de tal forma que se cumple con las relaciones de precedencia y otras restricciones. En **agendamiento global** se requiere efectuar calendarización y asignación para múltiples procesadores. La característica principal de los **agendadores estocásticos** es que parte de los datos de entrada, como por ejemplo los costos de computación, están sujetos a leyes estocásticas. Es decir, que los datos de entrada nos se conocen con certeza como es en el caso de los **agendadores determínisticos**.

Existen en general dos tipos extremos de calendarizadores: **estáticos** y **dinámicos**. Un calendarizador estático supone que todos los parámetros del modelo del programa y del modelo de la arquitectura del multiprocesador se conocen al tiempo de compilación. Algunas de las características que se requieren son: costos de ejecución de operaciones, costos de comunicación y relaciones entre operaciones. Algunos calendarizadores requieren tiempos de vencimiento y lanzamiento de operaciones. En un calendarizador estático la decisión de cuando se arranca una operación se hace al tiempo de compilación. En calendarización dinámica la mayoría de las características que se necesitan del programa paralelo se recolectan al tiempo de ejecución. En un calendarizador dinámico la decisión de cuando se ejecuta una operación se efectúa al tiempo de ejecución y siempre se sabe que operaciones están listas para ser ejecutadas. Para algunas aplicaciones de tiempo real o sensibles al costo, la tarea administrativa incurrida por un calendarizador dinámico es prohibitiva. Un calendarizador dinámico optimiza el desempeño del sistema durante ejecución y trata de minimizar la tarea administrativa introducida por su presencia. Note que alguna de la información que un calendarizador dinámico necesita la puede proporcionar el compilador y se le pasa el calendarizador al inicio o durante la ejecución del programa paralelo. En el caso de calendarizadores dinámicos es necesario observar que el calendarizador para su funcionamiento requiere de recursos al tiempo de ejecución. La implementación del calendarizador constituye un controlador. El uso de calendarizadores dinámicos es obligatorio cuando la información requerida por los calendarizadores estáticos no existe de antemano. Aquí consideramos principalmente calendarizadores estáticos. La calendarización estática es aplicable cuando los parámetros de los modelos del

programa y de la arquitectura del multiprocesador son determínisticos y conocidos o si son aleatorios y se conoce el peor caso. Existen aplicaciones en el dominio del procesamiento de señales y procesadores para tratamiento de protocolos de comunicación de nodos donde la calendarización estática es la solución más natural. Muchas de las técnicas para diseñar calendarizadores estáticos pueden ser modificadas para usarse en agendadores dinámicos y a la inversa.

Entre el agendador totalmente estático y el calendarizador totalmente dinámico hay agendadores intermedios, por ejemplo, el agendador que produce una asignación estática y la calendarización es dinámica, otro agendador que es similar al anterior excepto que el ordenamiento entre las operaciones se efectúa estáticamente y la determinación de la temporización final es dinámica. En este caso se requiere sincronización implícita o explicita para esperar que los datos arriben a una operación. El primer agendador se llama **asignación estática** y el segundo **autotemporizado**. En la práctica puede suceder que aunque se pueda tener un agendador estático se opte por uno autotemporizado. Esto es para prevenir cambios en los tiempos que se suponen cuando se diseña el agendador. El agendador autotemporizado es inmune a estos cambios.

En un **algoritmo incremental** se supone que no todos los datos de entrada están disponibles a la vez. El algoritmo incremental computa con los datos disponibles iniciales y conforme más datos le son proporcionados continúa su solución. Los resultados del cómputo inicial son irrevocables y no son desechados nunca. La solución siempre se construye sobre soluciones parciales anteriores, es decir, que la solución final se construye como una secuencia de soluciones parciales. Donde una solución parcial se construye de la solución parcial anterior y los nuevos datos de entrada. Un algoritmo incremental se dice **voraz** si cada vez que construye una solución parcial en base a la solución parcial anterior y la nueva entrada se efectúa una decisión que optimiza un criterio sin considerar que nuevas entradas se tienen más tarde, es decir, optimiza como si la entrada actual fuese la última entrada. Como consecuencia de su operación resulta que no hay garantía que los algoritmos voraces produzcan una solución óptima. Los algoritmos voraces por lo general son algoritmos computacionalmente eficientes.

Un **algoritmo en línea** es un algoritmo incremental y sus entradas se le proporcionan por el medio donde está embebido conforme se ejecuta. En contraste a un algoritmo en línea, un **algoritmo fuera de línea** puede considerar todas sus entradas. Observamos que un calendarizador en línea puede ser estático o dinámico. Note que la clasificación de en línea y fuera de línea se refiere a como se usa y la de estático y dinámico se refiere a que información se conoce de antes de empezar a calendarizar.

Una calendarización con **restricciones de tiempo** debe de calendarizar de tal forma que el sistema del cual es parte responde a ciertos estímulos dentro de un tiempo corto. Las restricciones de tiempo pueden ser **estrictos** o **relajados**. En el primer caso, si el sistema no responde como esperado su funcionamiento no es aceptable y puede tener

consecuencias funestas. En el segundo caso las restricciones pueden ser violadas hasta por cierto límite. Hay diferentes definiciones contradictorias en la literatura de lo que es un sistema real, aquí presentamos una que nos parece razonable.

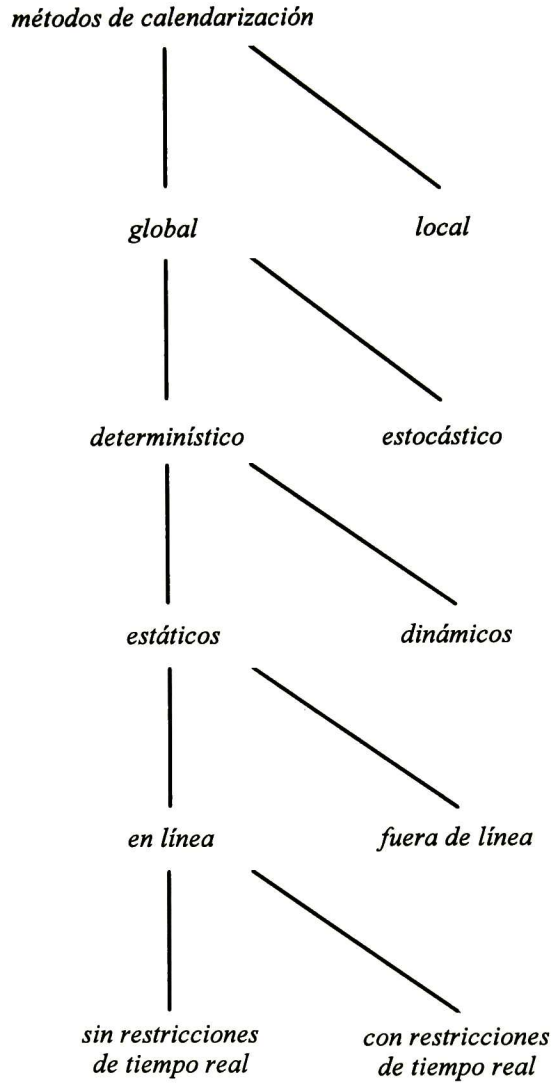


Figura 4.10: Taxonomía parcial de métodos de calendarización.

4.4.2 Modelo de Programa

El modelo del programa debe facilitar la tarea de agendamiento. Los programas que consideramos se pueden modelar como un conjunto de operaciones y una relación de orden parcial estricto que indica el orden de precedencia en la ejecución de las operaciones. El problema de descomposición de un programa en sus operaciones y el grado de paralelismo de éstas no es objeto de estudio en este capítulo. Si $G=(S,R)$ es el dag que representa el programa P S es el conjunto de operaciones de P esto es $S=\{o_1,\ldots,o_n\}$ y R es tal que si $o_1 R o_2$ entonces la operación o_2 requiere de resultados producidos por la operación o_1, como consecuencia o_1 se debe ejecutar primero que o_2. Las restricciones de precedencia especificadas por la relación R indican que una operación no se puede empezar a ejecutar si todos sus predecesores inmediatos no se han ejecutado y sus resultados han sido comunicados a la operación. El modelo de programa puede o no imponer un orden en la ejecución de sus operaciones. En general el diseño de calendarizadores con operaciones independientes es menos complejo que cuando existen dependencias entre operaciones. En el caso de operaciones independientes éstas se pueden ejecutar en orden arbitrario y el problema de agendamiento se reduce a problema de asignación donde el propósito es balancear la carga de los elementos procesadores de la máquina. Llamamos **grafo de dependencia** al grafo que representa al programa. El grafo de dependencia expone la concurrencia y dependencia de las operaciones. El grafo de dependencia se puede anotar de tal forma que cada nodo se la asocian los recursos, además de un procesador, que se requiere para la ejecución de la operación del nodo. Asociado a cada $o \in S$ hay un costo de computación que es una indicación del tiempo que toma la ejecución de la operación o y se denota por $w(o)$. Con respecto al modelo del programa tenemos las variantes mostradas en la figura 4.11.

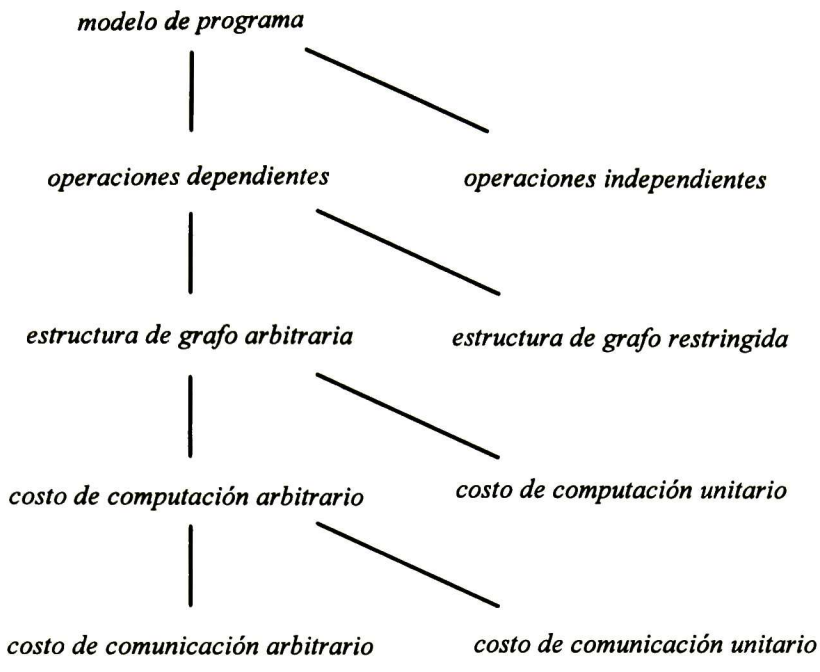


Figura 4.11: Variantes del modelo del programa.

Asociado al arco formado por (s_1, s_2) hay un costo de comunicación que es una indicación del volumen de datos para enviar a o_2 los datos que requiere de o_1, este costo lo denotamos por $c(o_1, o_2)$. Suponemos que las funciones $w(o)$ y $c(o_1, o_2)$ toman valores no negativos. Los costos asociados a las operaciones y los arcos del grafo pueden ser obtenidos al tiempo de compilación o durante ejecución del programa o de perfiles de ejecución del programa.

El modelo con cero costo de comunicación es adecuado para bucles anidados, para compactación de microcódigo horizontal y para sistemas digitales secuenciales sincrónicos.

4.4.3 Modelo de la Arquitectura del Sistema Multiprocesador

Si el tiempo de ejecución de una operación no depende en que elemento procesador se ejecuta se dice que tenemos un **sistema de elementos procesadores homogéneos**, de otra forma se dice que el **sistema es de elementos procesadores heterogéneos**. Los elementos procesadores pueden tener diferentes consumos de potencia, área, capacidades y velocidades de procesamiento. Si un procesador ejecuta una operación sin interrupción se dice que tenemos un **procesador sin desalojo**, por otro lado si la ejecución de la operación puede ser suspendida para retomarla más tarde se dice que se tiene un **procesador con desalojo**. Cuando un procesador permite desalojo se debe

contar con un mecanismo eficiente para cambio de contexto. Si se permite desalojo más de una operación puede estar activada en el procesador. Procesadores que permiten desalojo son convenientes para calendarizadores de sistemas que requieren respuesta a ciertos eventos en un tiempo muy corto. Los sistemas multiprocesador que consideramos se pueden modelar como un dag donde los nodos corresponden a los procesadores de comunicación y las aristas corresponden a los enlaces de comunicación. Sea el grafo $A=(P,C)$ donde $P=\{p_1,\ldots,p_n\}$ es el conjunto de procesadores de comunicaciones y $C\subseteq P\times P$ es el conjunto de enlaces dirigidos de comunicación e indican que procesadores que están físicamente interconectados. El **tamaño del arreglo de procesadores,** $|P|=m$, es el número de procesadores donde se ejecuta el programa. Siempre suponemos que $(p_i,p_i)\in C$. Si existen múltiples enlaces de comunicación entre procesadores entonces A es un multígrafo. La topología de la red de comunicación la determina el conjunto C. El grafo A modela adecuadamente un sistema de comunicaciones donde los procesadores de comunicación no comparten memoria. Un bus conectado de múltiples procesadores se modela como un grafo $G=(P,C)$ que representa una red totalmente conectada. Los enlaces entre los procesadores de comunicación en general introducen retardos.

Si dos operaciones entre las cuales existe dependencia se asignan a diferentes procesadores entonces en la red de comunicaciones se debe poder establecer una trayectoria de comunicaciones, usando quizá múltiples canales, que conectan los dos procesadores. Si a las dos operaciones dependientes se les asigna el mismo procesador entonces las dos operaciones se pasan información por medio de la memoria local del elemento procesador. En este último cado se considera que el costo de comunicación es cero.

Ingredientes del modelo de la máquina son: recursos de cómputo y recursos de comunicación. Entre los recursos de cómputo se tienen: procesadores y memorias. Entre los recursos de comunicación se tienen: canales de comunicación y enrutadores. No todo elemento procesador puede ejecutar cualquiera de las operaciones del programa. El tiempo de ejecución de una operación depende de en que elemento procesador se ejecuta. El costo de computación junto con el procesador determinan el tiempo de ejecución. Se supone que el sistema meta en el cual se ejecutará el programa es una red de elementos procesadores cada uno con memoria local. Las memorias se caracterizan por su tamaño y tiempos de acceso. En general cada operación requiere memoria para su ejecución. La memoria local asociada a cada procesador debe ser suficiente para poder ejecutar todas las operaciones asignadas al elemento procesador. El sistema formado es un sistema de paso de mensajes donde los procesadores de comunicación no comparten memoria. Los elementos procesadores están conectados por una red con una topología dada como por ejemplo un hipercubo, una red de malla o una red totalmente conectada. Los elementos procesadores pueden o no estar involucrados en la comunicación. Si lo están entonces no existe paralelismo completo entre procesamiento

y comunicación. Esto ocurre cuando la red de comunicaciones usa los procesadores como procesadores de comunicaciones. Cuando el procesador no está involucrado en la comunicación se supone la existencia de un procesador de comunicaciones por cada procesador el cual es encargado de todo el procesamiento relacionado a la comunicación con otros procesadores. La figura 4.13 muestra los efectos de los procesadores de comunicación en la comunicación. Los enlaces se caracterizan por su ancho de banda y latencia. Si todos los enlaces tienen el mismo ancho de banda y latencia se dicen homogéneos de otra forma son heterogéneos. Con respecto al modelo de la arquitectura del multiprocesador tenemos las variantes mostradas en la figura 4.12. Es posible modificar el grafo de dependencia del programa con información del modelo de la máquina para así obtener un grafo de dependencia del sistema completo. En este grafo no sólo los procesadores son recursos sino los canales de comunicación también. Mientras los procesadores ejecutan operaciones, los canales de comunicación ejecutan interacciones entre operaciones. Diferentes interacciones pueden desear el mismo canal de comunicación produciendo contención por su uso e incrementando el retardo para transferencia de datos.

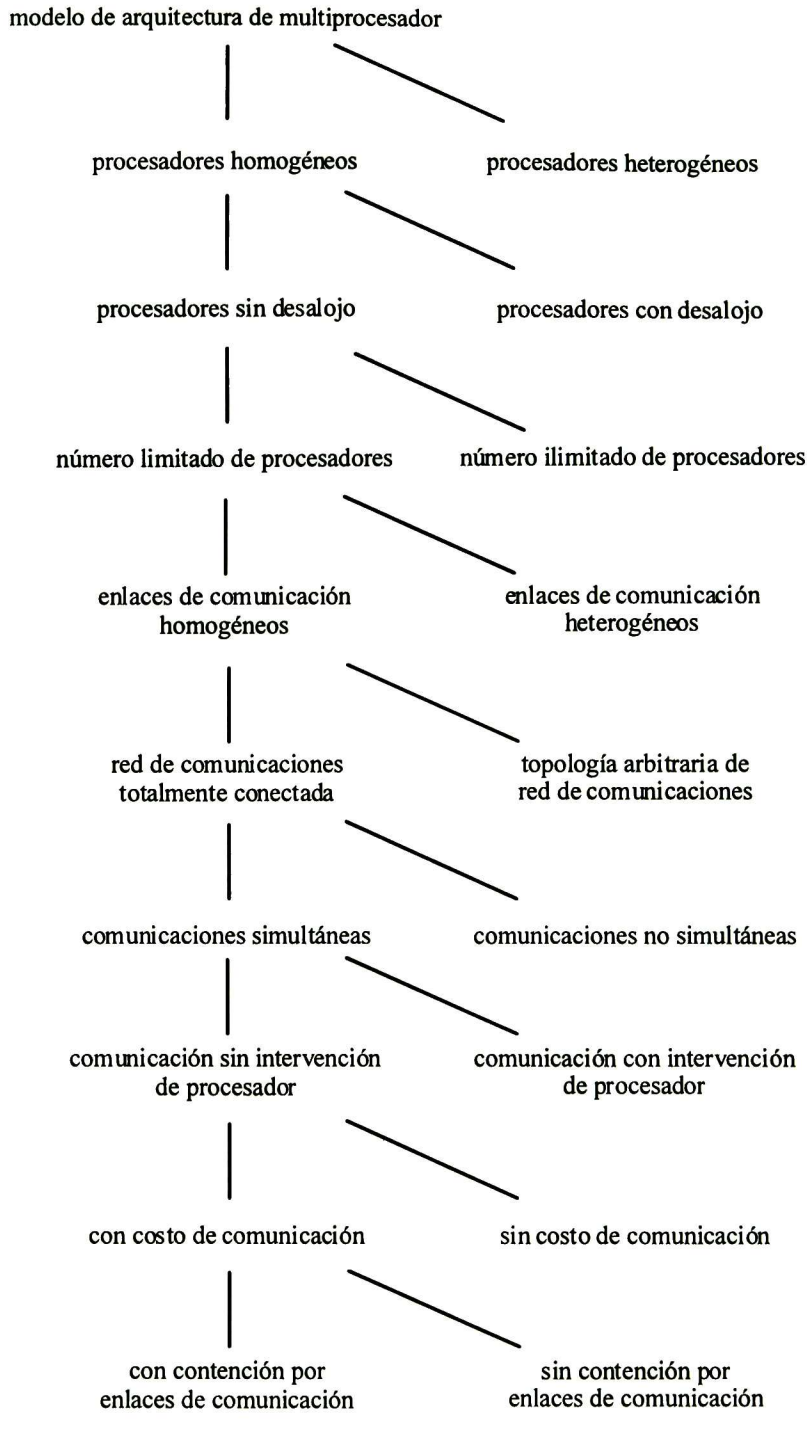
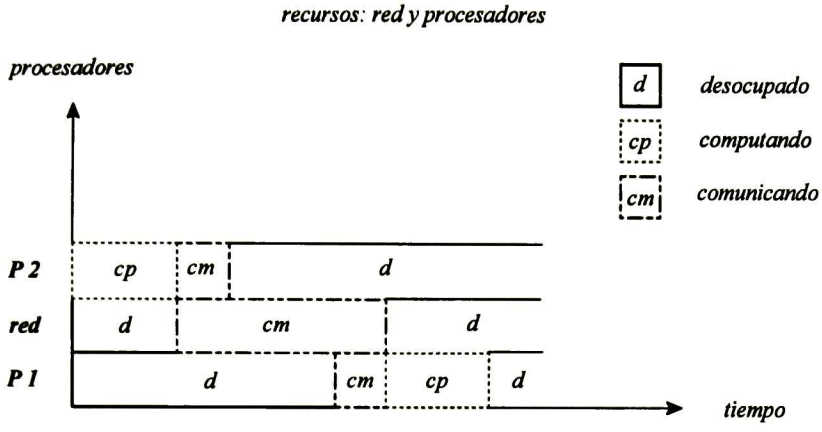
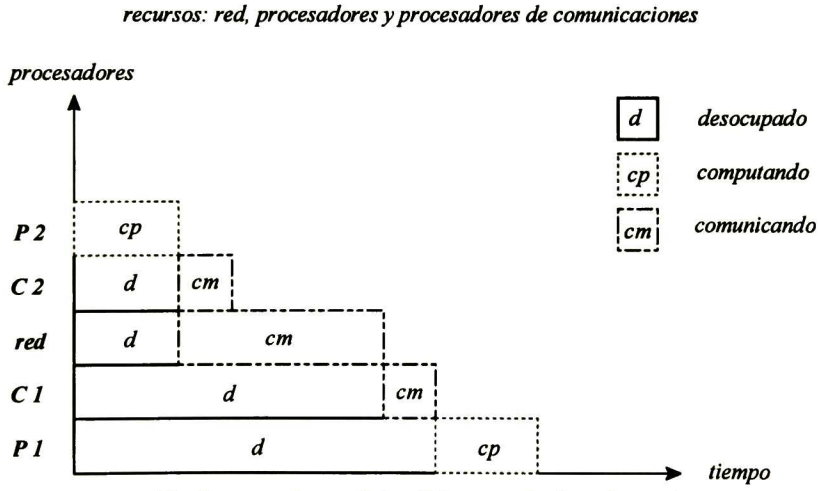


Figura 4.12: Variantes del modelo del multiprocesador.



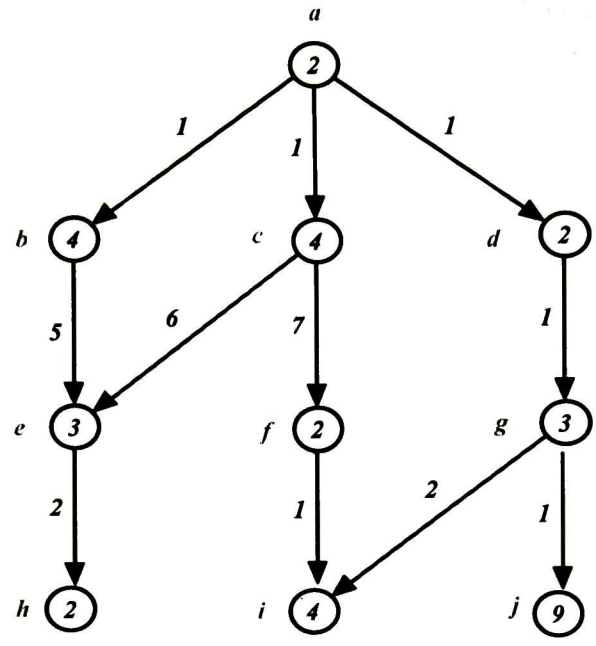
a) Procesadores *P1* y *P2* involucrados en comunicación.



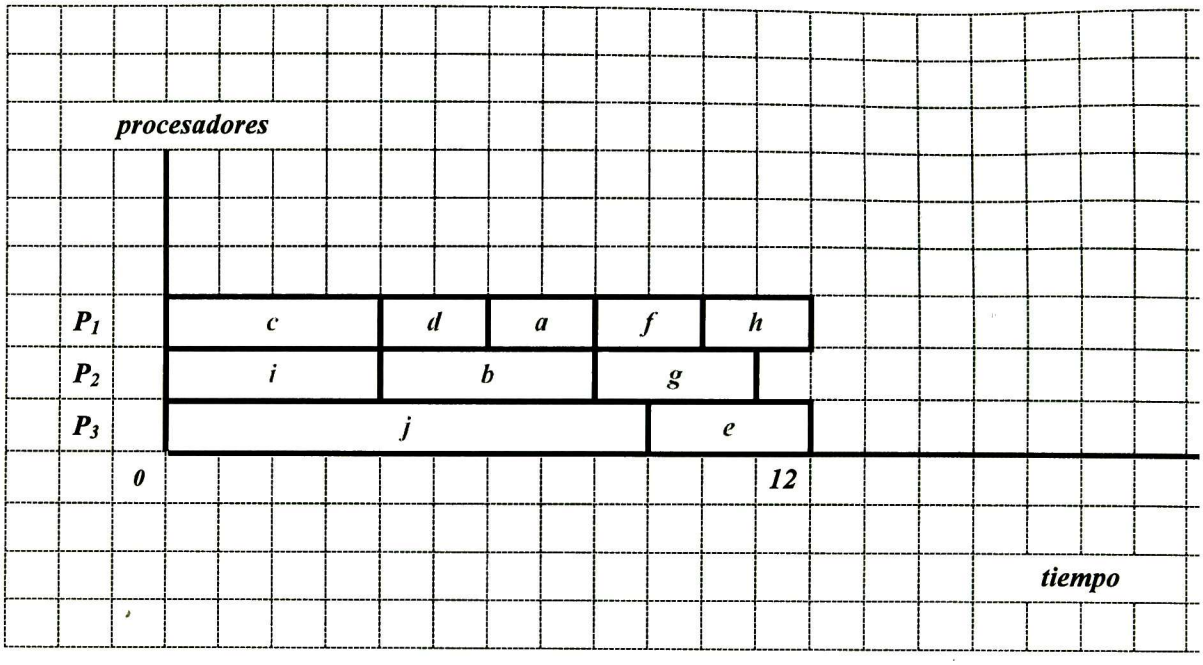
b) Procesadores *P1* y *P2* no están involucrados.

Figura 4.13: Efecto de involucramiento de procesador en comunicación.

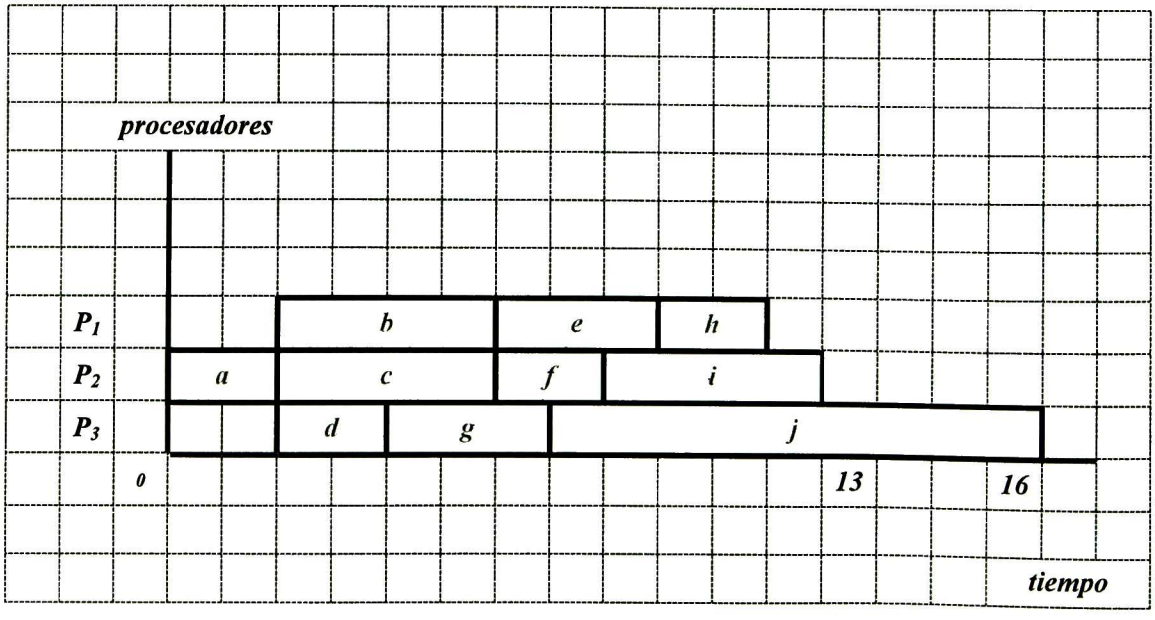
Ejemplo 4.5. Considere el dag y los diagramas de Gantt en la figura 4.14. Los diagramas de Gantt corresponden a diferentes variantes de comunicación y de dependencia, en todos los casos los procesadores son homogéneos y sin desalojo. Todos los calendarizadores presentados son óptimos en tiempo. El tiempo de comunicación entre operaciones que se asignan al mismo procesador es cero en todos los casos. En el primer diagrama de Gantt se supone una red de comunicación totalmente conectada, las diferentes comunicaciones pueden suceder simultáneamente, con costo de comunicación cero, sin intervención del procesador en las comunicaciones y no hay dependencia entre operaciones. El proceso de generar un calendarizador óptimo en tiempo es prácticamente acomodar las operaciones tomando en cuenta su tiempo de ejecución en tres silos. De la figura 4.14 b debe ser claro que el calendarizador presentado es óptimo en tiempo. En el segundo diagrama de Gantt se supone igual que el caso anterior excepto que si hay dependencia entre operaciones y está dada por el dag. En el tercer diagrama se suponen las mismas condiciones que el anterior excepto que se supone que si hay retardos de comunicación y es igual a los costos de comunicación. En el diagrama se tienen las mismas condiciones que en el diagrama anterior excepto que se supone un bus que interconecta todos los procesadores, por lo tanto se tiene contención en el uso del bus en vez de una red totalmente conectada. En este caso no se pueden tener comunicaciones simultáneas, tienen que ser serializadas por los accesos al bus. Se supone que la comunicación que se envía del nodo a los otros nodos es diferente y por lo tanto no se pueden usar las características de difusión de un bus. El quinto diagrama es bajo las mismas hipótesis que el anterior excepto que ahora se supone intervención del procesador. Se supone que un procesador toma una unidad de tiempo en transmitir y una unidad de tiempo en recibir información de otro procesador esto es independiente del tamaño de la información. El efecto de la intervención del procesador en las comunicaciones lo hace inaccesible para la computación durante el tiempo que se comunica. Los diagramas de Gantt muestran el aumento de tiempo de ejecución conforme más factores se toman en cuenta en el sistema de comunicaciones. El alargamiento del tiempo de ejecución se debe a los retardos introducidos por el canal y a la contención por el uso de los recursos en el caso mostrado del bus.



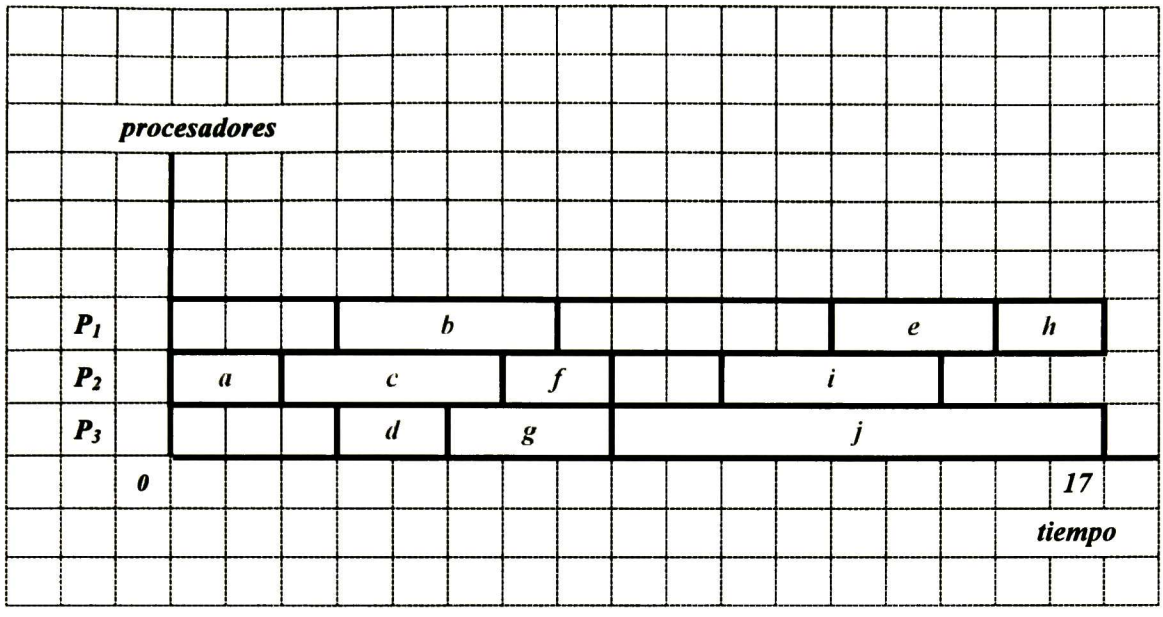
a) Dag.



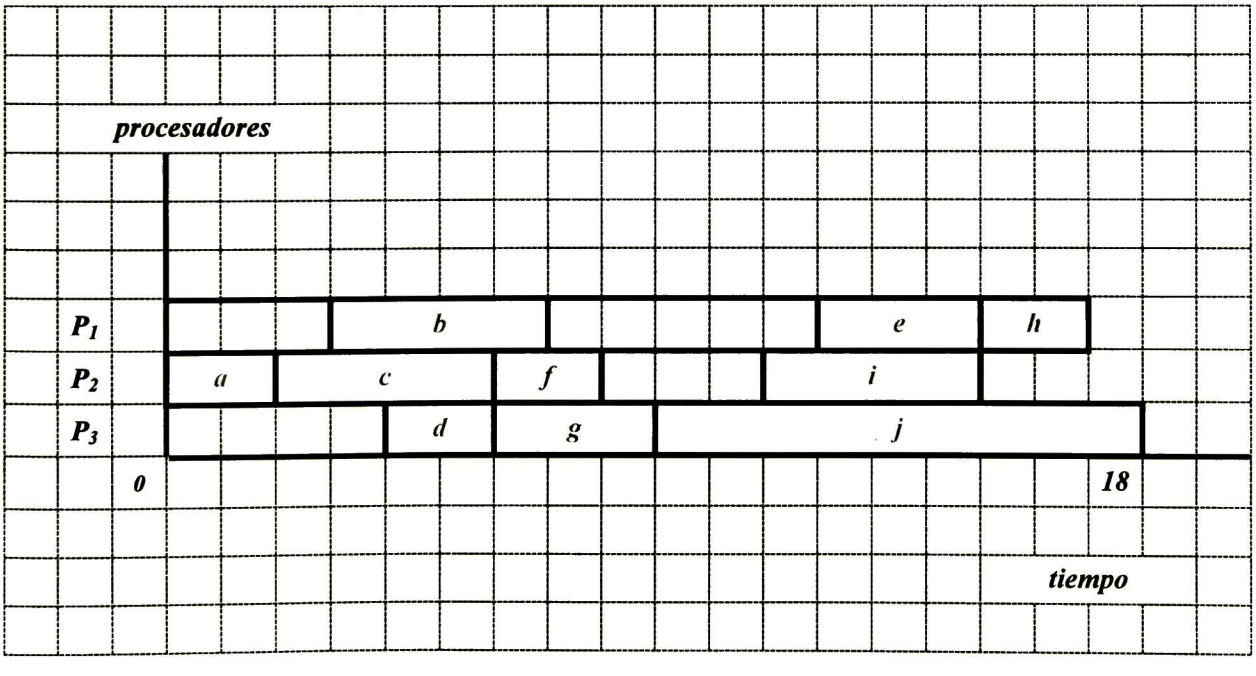
b) Operaciones independientes.



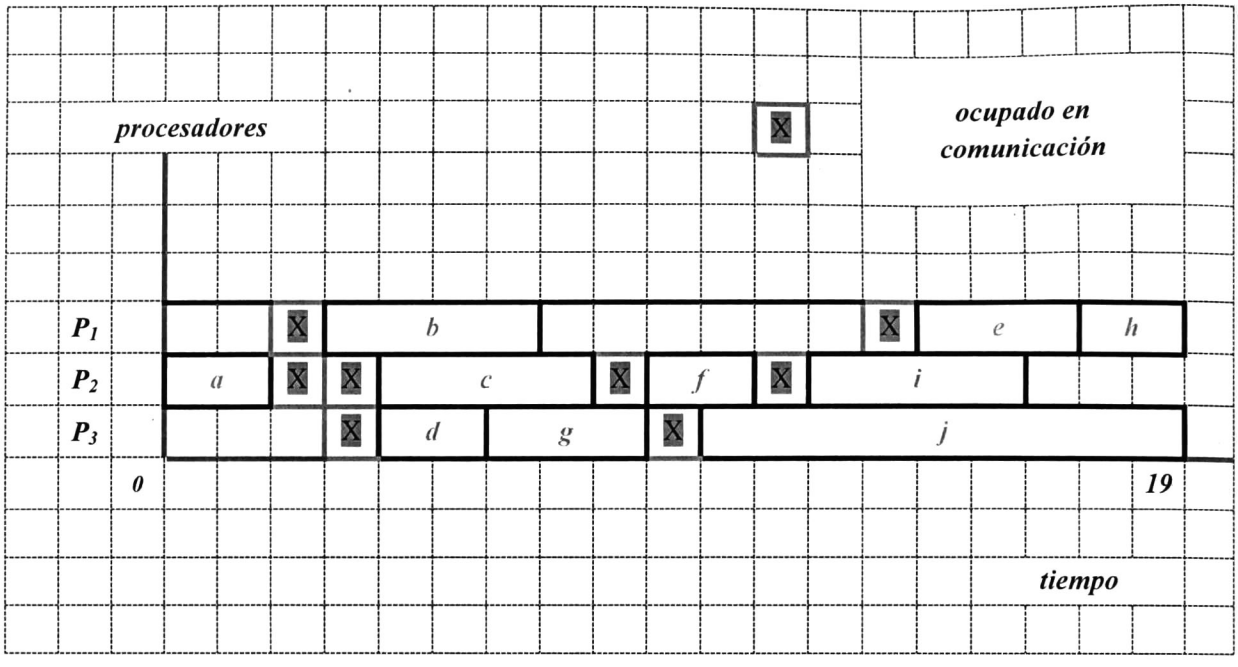
c) Operaciones dependientes.



d) Con retardo de comunicaciones.



e) Con contención por bus de comunicaciones.



f) Con intervención de procesador.

Figura 4.14: Calendarizadores para diferentes variantes del sistema de comunicaciones.

■

4.4.4 Funciones Objetivo

Como resultado del agendamiento se espera diseñar un buen calendarizador y un buen asignador. El **criterio de bondad, función objetivo** u **objetivo de optimización** se define de acuerdo al problema a resolver. Algunas de las funciones objetivo que comúnmente se optimizan, cuando se diseñan el calendarizador y el asignador, son: tiempo de ejecución del programa, el promedio de los tiempos de terminación de las operaciones, la máxima tardirez, número de operaciones tarde.

El **tiempo de ejecución** T de un programa es el intervalo de tiempo entre la primera operación y la última operación.

$$T = \max_{v \in S}\{\tau(v) + w(v)\} - \min_{v \in S}\tau(v) + 1$$

Sin pérdida de generalidad suponemos que la primera tarea se despacha al tiempo cero, el tiempo de ejecución se reduce entonces a:

$$T = \max_{v \in S}\{\tau(v) + w(v)\} = \max_{v \in S}\tau_f(v)$$

Sea la trayectoria $\mu = v_0 v_1 \ldots v_k, \quad k \geq 0$ en G El peso de la trayectoria μ satisface que:

$$w(\mu) = \sum_{i=0}^{k-1}\left(w(v_i) + c(v_i, v_{i+1})\right) + w(v_k)$$

La minimización del tiempo de ejecución es equivalente a maximizar el factor de aceleración en el caso de procesadores homogéneos. El **factor de aceleración** se define como:

$$\sigma = \frac{\sum_{o \in S} w(o)}{T}$$

La **eficiencia de los procesadores** de define como:

$$\eta = \frac{\sigma}{n}$$

El **promedio de los tiempos de terminación** se define como:

$$\bar{T} = \frac{1}{n}\sum_{i=1}^{n} \tau_f(o_i)$$

donde $n = |S|$ es el número de operaciones del programa y $o_i \in S$. Optimizar $\bar{T}$ es equivalente a optimizar $\bar{T}_f$ donde:

$$\bar{T}_f = \frac{1}{n}\sum_{i=1}^{n} \tau_f(o_i)$$

dado su simplicidad se prefiere optimizar $\bar{T}_f$.

El **promedio ponderado de los tiempos de terminación** se define como:

$$\bar{T}_{f_p} = \sum_{i=1}^{n} w_i \tau_f(o_i)$$

donde $w_i > 0$ para $i = 1,\ldots,n$. Los pasos w_i le dan diferente ponderación o importancia a las diferentes operaciones. En ocasiones se asocia un tiempo de vencimiento d_j con la operación o_j. El tiempo de vencimiento es el tiempo de terminación esperado de la operación o_j. La tardanza de la operación o_j se define como:

$$L_j = \tau_f(o_j) - d_j$$

y la máxima tardanza como:

$$L = \max_{j=1,\ldots,n} L_j$$

Si definimos $U_j = 0$ si $\tau_f(o_j) \leq d_j$ y $U_j = 1$ de otra forma entonces el **número de operadores tardíos ponderados** es:

$$U = \sum_{j=1}^{n} w_i U_j$$

donde $w_j \geq 0$ son los factores de ponderación.

4.4.5 Modelado de Recursos y Restricciones

En lo que sigue presentamos un modelo para los recursos que la ejecución de una operación pueden requerir, también se presentan las restricciones más comunes que se pueden requerir.

A menudo es necesario complementar los modelos del programa y de la máquina con restricciones que limitan el espacio de definición del calendarizador y el asignador. Es importante que el algoritmo que diseña el calendarizador y el asignador sea extensible en el sentido que estas restricciones se puedan modificar o añadir de acuerdo con el problema que se resuelve. Las restricciones pueden tener diferentes orígenes, entre otros: sugeridas por el ingeniero para guiar el algoritmo diseñador en alguna dirección, impuestos por limitaciones del algoritmo diseñador, requerimientos no capturados por los modelos del programa y la arquitectura del sistema multiprocesador. Las restricciones en general varían de sistema a sistema.

En ocasiones las operaciones requieren, además del procesador, de más recursos para su ejecución. Ejemplo de recursos son las memorias de datos y de instrucciones. Cuando el número de recursos disponibles es finito se requiere tomar en cuenta éstos para poder definir un calendarizador y un asignador. En particular múltiples operaciones se pueden ejecutar en paralelo sólo si el número consolidado y tipo de recursos es tal que no excede, durante la ejecución, el número y tipo de recursos disponibles. Un modelo simple del uso de los recursos indica que recursos requiere cada operación y en que orden y a que tiempos se usa cada una de ellas. Dos operaciones están en conflicto si requieren el mismo recurso al mismo tiempo, las operaciones en conflicto no se pueden ejecutar simultáneamente. Una restricción del calendarizador es evitar que dos operaciones estén en conflicto.

Cuando el vector $\rho_0(t)$ se representa por medio de una tabla se llama **tabla de reservación de recursos**. A continuación presentamos una forma de modelar el hecho de que las operaciones requieren recursos para su ejecución. Sea $R \in \mathbb{Z}^n$ R es llamado **vector de configuración de recursos**. n es el número de diferentes tipos de recursos disponibles en el sistema. Si $n=1$ se tiene que todos los recursos son iguales y se dice que los recursos son **homogéneos** de otra manera se dice que los recursos son **heterogéneos**. La componente i de R, $R(i)$, denota el número de recursos del tipo i disponibles, $1 \le i \le n$. $\rho_0(t) \in \mathbb{Z}^n$ es el **vector de uso de los recursos** de la operación $o \in S$. Por definición para todo tiempo t $0 \le \rho_0(t) \le R$. $\rho_0(t)$ es un vector cuyas componentes denotan los tipos de recursos que la operación o usa al tiempo t. Si $w(o)$ es el tiempo que dura la operación o entonces $\rho_0(t) = 0$ cuando $t \notin [0, w]$. El diseño del calendarizador es tal que siempre se satisface la relación:

Si tomamos en cuenta que cada recurso ejecuta una función requerida por la operación es posible interpretar la relación (4.14) como un calendarizador para los recursos.

$$\text{para toda } t \quad \sum_{o \in S} \rho_0 \left(t - w(o)\right) \leq R \tag{4.14}$$

Esta relación establece que en cualquier tiempo no se puede usar más recursos que los disponibles. En el diseño del calendarizador y del asignador se debe considerar la relación (4.14).

A continuación presentamos una lista de restricciones:

Tiempo de vencimiento de operaciones. Como parte de los requerimientos del calendarizador a cada operación o se le asocia un tiempo de vencimiento d_o que indica el tiempo deseado para que la operación termine. Esta restricción se expresa como:

$$\tau_f(o) \leq d_o$$

Tiempo de lanzamiento de una operación. Como parte de los requerimientos del calendarizador a cada operación o se le asocia un tiempo de lanzamiento r_o que indica el tiempo deseado para que la operación empiece su ejecución. Esta restricción se expresa como:

$$r_o \leq \tau(o)$$

Tiempo relativo entre lanzamiento de dos operaciones. Dadas dos operaciones u y v se requiere que el calendarizador despache tal que se cumple que:

$$\tau(u) - \tau(v) \left\{ \begin{matrix} \geq \\ \leq \\ = \end{matrix} \right\} d_{uv}$$

donde d_{uv} es la distancia en tiempo entre los inicios de ejecución de las operaciones u y v

Serialización de ejecución de operaciones. Dadas las operaciones u y v los tiempos de ejecución de u y v no se sobrelapan. Esta restricción se expresa como:

$$\pi(u) = \pi(v) \quad \text{o} \quad \text{si} \quad \pi(u) \neq \pi(v) \Rightarrow \tau_f(u) \geq \tau(v) \quad \text{o} \quad \tau_f(v) \geq \tau(u)$$

Aglutinamiento de operaciones. Dado el conjunto de operaciones $V_o = \{o, o_1, o_2, \ldots, o_k\}$, todas sus operaciones se asignan al mismo procesador. Esta restricción se expresa como:

$$\forall v \in V_o - \{o\} \qquad \pi(o) = \pi(v)$$

Esta restricción es conveniente, cuando las operaciones de V_o comparten mucha información y es costoso enviar la información a través de la red de comunicación.

Segregación de operaciones. Dado el conjunto de operaciones $V_o = \{o, o_1, o_2, \ldots, o_k\}$ ninguna de sus operaciones pueden ser asignadas al mismo procesador. Esta restricción se expresa como:

$$v_i, v_j \in V_o, \qquad v_i \neq v_j \Rightarrow \pi(v_i) \neq \pi(v_j)$$

Esta restricción es útil cuando los procesadores no pueden contener las operaciones porque requieren más recursos que los disponibles.

Pertenencia de procesadores. Dado el conjunto de procesadores P_o y la operación o, el asignador debe cumplir que:

$$\pi(o) \in P_o \subseteq P$$

Es decir, la operación o se debe asignar a un procesador en P_o. Este requerimiento es útil cuando se tienen procesadores heterogéneos, la restricción garantiza que el procesador puede ejecutar la operación.

En el caso del calendarizador libre, N_t se llama el **perfil de paralelismo** del programa. Note que N_t está determinado por el grafo de dependencia del algoritmo y por el calendarizador libre; en particular nos dice cuantos procesadores requiere el calendarizador libre en cada tiempo para poder efectuar los cómputos indicados en el grafo de dependencia. N_t es una medida del paralelismo disponible en el dag al tiempo t. El calendarizador libre proporciona el tiempo mínimo para ejecutar un algoritmo y el perfil de paralelismo nos permite calcular, $N_{\max}$, una cota superior al número de elementos procesadores que se requiere para obtener el tiempo mínimo, cuando se usa el calendarizador libre. $N_{\max}$ se llama el grado de paralelismo del programa. Obsérvese que el tiempo de ejecución logrado por un calendarizador libre no se puede mejorar por ningún otro calendarizador y es lo que llamamos tiempo óptimo, T_{opt}.

Ejemplo 4.6. En la figura 4.15 se aplica el calendarizador libre al grafo de dependencia del programa del ejemplo 4.3, con la restricción de elementos procesadores, para este caso particular dos elementos procesadores. En la figura 4.15, 0,1,...,8 denotan los tiempos a los cuales inicia la ejecución de las operaciones del programa.

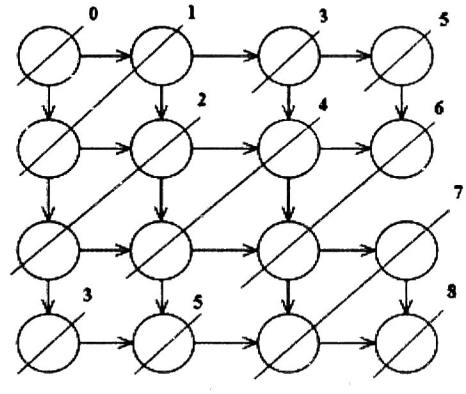


Figura 4.15: Grafo de dependencia.

Con la restricción al número de elementos procesadores, a partir del grafo de dependencia, se obtiene la información contenida en la tabla 4.3.

Tiempo	Número de procesadores ocupados
0	1
1	2
2	2
3	2
4	2
5	2
6	2
7	2
8	1

Tabla 4.3: Activación de elementos procesadores.

A partir del contenido de la tabla 4.3, se grafica el perfil de paralelismo de los elementos procesadores contra el tiempo, dada la restricción de dos elementos procesadores.

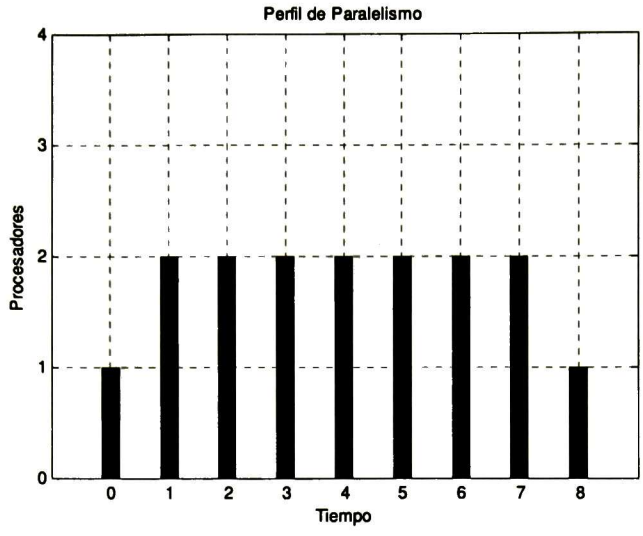


Figura 4.16: Perfil de paralelismo.

Calculamos la eficiencia del uso de los elementos procesadores con la ecuación (2.1):

$$\eta_{opt} = \frac{\sum_{t=0}^{T_{opt}} N_t}{T_{opt} * N_{opt}} = \frac{16}{9*2} = 0.888$$

■

En el caso de un perfil de paralelismo con número limitado de elementos el calendarizador libre no usa más que el número máximo de procesadores disponibles. Aquí al igual que en el caso sin restricción en el número de elementos procesadores un algoritmo con un número limitado de procesadores no se puede correr en un tiempo menor al indicado al perfil de paralelismo con un número de elementos procesadores restringidos. Cualquier algoritmo que tome el mismo tiempo no puede usar los procesadores con una mejor eficiencia que la obtenida del perfil de paralelismo con restricción en el número de elementos procesadores.

Ejemplo 4.7. En la figura 4.17, se aplica un calendarizador no libre, para este ejemplo no existe restricción en el número de elementos procesadores, 0,1,...,9 denotan los tiempos a los cuales inicia la ejecución de las operaciones del programa.

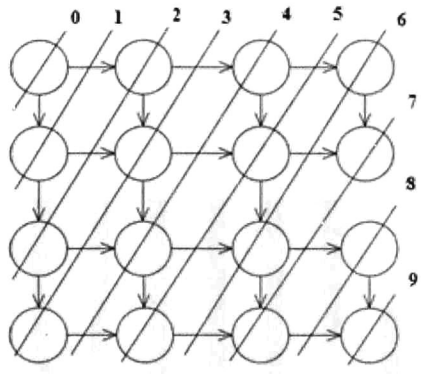


Figura 4.17: Grafo de dependencia.

A partir del grafo de dependencia, se obtiene la información contenida en la tabla 4.4.

Tiempo	Número de procesadores ocupados
0	1
1	1
2	2
3	2
4	2
5	2
6	2
7	2
8	1
9	1

Tabla 4.4: Activación de elementos procesadores, al aplicar un calendarizador no libre.

En base al contenido de la tabla 4.4, se grafica el perfil de paralelismo de los elementos procesadores contra el tiempo.

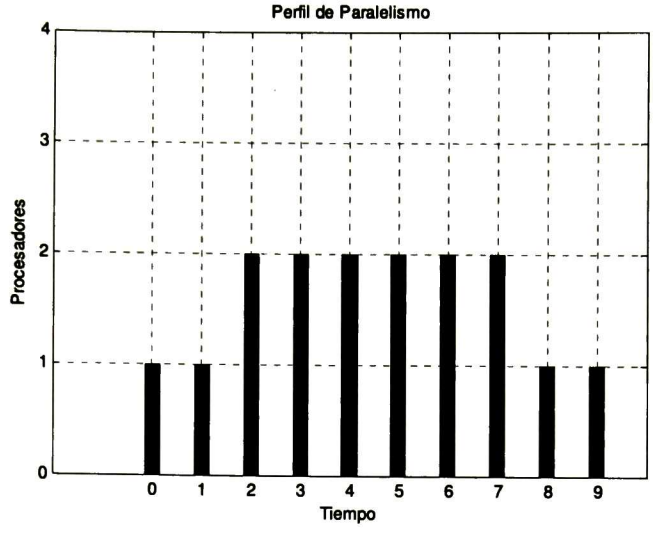


Figura 4.18: Perfil de paralelismo.

Calculamos la eficiencia del uso de los elementos procesadores con la ecuación (2.1):

$$\eta_{opt} = \frac{\sum_{t=0}^{T_{opt}} N_t}{T_{opt} * N_{opt}} = \frac{16}{10*2} = 0.8$$

■

Los resultados anteriores son útiles cuando se trata de dimensionar un arreglo de procesadores o para efectuar los cómputos de un grafo de dependencia bajo un calendarizador no libre. Los resultados proporcionan cotas a lo que se puede obtener usando calendarizadores no libres. Un calendarizador no libre será mejor mientras esté más cerca de los tiempos de ejecución y de uso eficiente de recursos que se obtienen con el calendarizador libre.

Ejemplo 4.8. Para el caso de análisis de una implementación en hardware, del programa presentado en los ejemplos anteriores observamos de la tabla 4.5, lo siguiente:

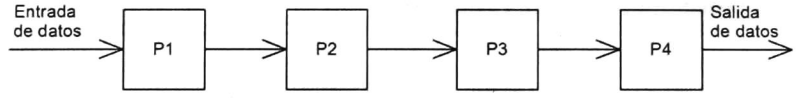


Figura 4.19: Arreglo de procesadores físicos.

	P1	P2	P3	P4
0	1			
1	1			
2	1	1		
3		1	1	
4		1	1	1
5			1	1
6				1

Tabla 4.5: Activación de elementos procesadores físicos.

En base al contenido de la tabla 4.5, se grafica el perfil de paralelismo de los elementos procesadores contra el tiempo:

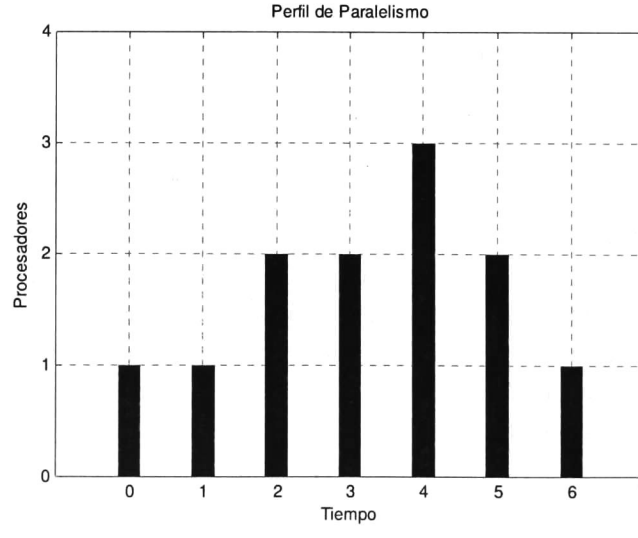


Figura 4.20: Perfil de paralelismo.

Calculamos la eficiencia del uso de los elementos procesadores con la siguiente ecuación, donde P representa el número de procesadores físicos implementados en el arreglo:

$$\eta_{opt} = \frac{\sum_{t=0}^{T_{opt}} N_t}{T_{opt} * P} = \frac{12}{7*4} = 0.42$$

En esta implementación de hardware se hace evidente, la necesidad de mejorar la eficiencia en el uso de los procesadores físicos, determinada en el ejemplo 4.2, el cual nos indica que la eficiencia se puede incrementar dada la cota mínima al usar un calendarizador libre.

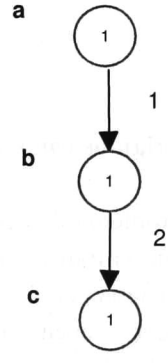
■

4.5 Agendamiento de Grafos Dirigidos con Comunicaciones con Retardo

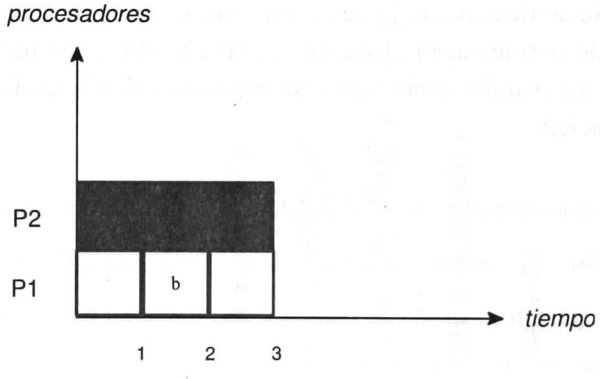
En esta sección estudiamos el problema de agendamiento suponiendo que existe un costo de comunicación, es decir, que la transferencia de datos de un procesador a otro toma tiempo, mientras que la transferencia de datos entre operaciones asignadas al mismo procesador se suponen que tienen costo de comunicación iguala cero. Suponemos que la red de comunicaciones es completamente conectada y que la comunicación y el procesamiento toman lugar en paralelo, como consecuencia se tiene que los procesadores no intervienen en la comunicación, lo cual implica la existencia de procesadores de comunicaciones. Los procesadores y enlaces de comunicación son homogéneos. No se tiene desalojo de operaciones en ejecución ni contención por el uso de los canales de comunicación. Es decir, que la comunicación puede ocurrir tan pronto como los datos a transmitir están listos. Se requieren calendarizadores que minimicen el tiempo de ejecución.

El grafo de dependencias $G=(S,R)$ del programa ahora está extendido con costos de computación y comunicaciones denotados respectivamente por $w(u) \geq 0$ y $c((u,v)) \geq 0$ para u y $v \in S$. El costo de computación es una medida del número de instrucciones que se ejecutan y el costo de comunicación una medida del volumen de datos que se transfieren. El modelo que presentamos es crudo sin embargo puede ser refinado para obtener modelos más realistas.

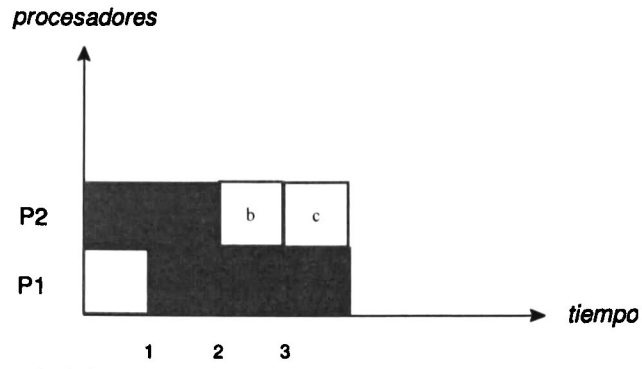
Ejemplo 4.9. Sea el grafo de dependencia aumentado con costo de computación y de comunicación mostrado en la figura 4.21. Como se puede comprobar si construimos varios calendarizadores, la figura 4.21 muestra dos calendarizadores, el tiempo de ejecución siempre es menor si se usa un solo procesador. Es decir, el calendarizador óptimo usa un sólo procesador. Es decir, se ilustra que el paralelismo no necesariamente conduce a un calendarizador óptimo en tiempo cuando se tienen costos de comunicación. Es decir, que la estrategia de mantener todos los procesadores ocupados siempre que sea posible sin tomar en cuenta los costos de comunicación no es lo mejor. El tipo de comportamiento descrito ocurre cuando los costos de comunicación son altos con respecto a los de computación. Notamos que a diferencia del caso con grafos dirigidos extendidos con sólo costo de computación, si la asignación de procesador se efectúa con la estrategia de orientado al procesador no resulta en el mismo calendarizador que con la estrategia orientada a la operación.



a) Grafo de dependencia.



b) Diagrama de Gantt con un sólo procesador usado.

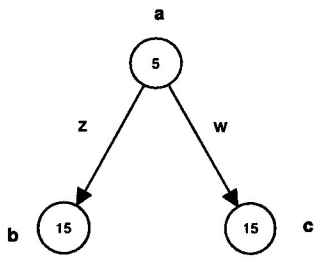


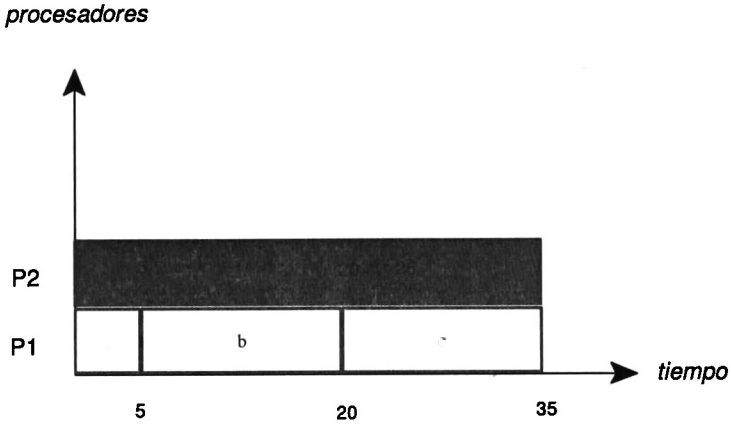
c) Diagrama de Gantt con dos procesadores usados.

Figura 4.21: Grafo de dependencia extendido y dos agendadores.

■

Ejemplo 4.10. Sea el grafo dirigido mostrado en la figura 4.22. Se supone que los costos de comunicación satisfacen que $w < z$. Es claro que el tiempo óptimo de ejecución es $T_{opt} = \min(35, 5+w+15)$ y el tiempo de ejecución de un agendador voraz es $T = 5+w+15$. Note que no existe una constante c válida para toda w tal que $T \leq cT_{opt}$. Es decir, no existe cota superior que límite el tiempo de ejecución cuando se usa un agendador con lista de prioridad cuando existen retardos de comunicación. Se ilustra que usar más procesadores no necesariamente implica tiempos de ejecución más cortos. La estrategia para seleccionar procesador orientada a procesador no siempre es buena.





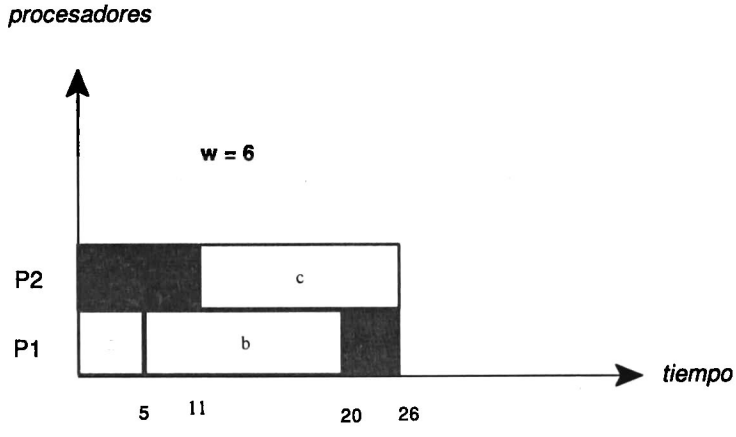


Figura 4.22: Grafo de dependencia y dos agendadores.

■

En general nos interesa maximizar el paralelismo y disminuir la comunicación que aquí es gasto administrativo con objeto de acortar el tiempo de ejecución. Como los ejemplos anteriores lo muestran, la computación paralela y la comunicación están íntimamente interrelacionados y en general son factores en tensión, lo cual hace el problema de agendamiento más difícil. Un sistema que minimiza comunicación y no tiene paralelismo es un sistema uniprocesador. Por otro lado si se tienen varios procesadores, el paralelismo incrementa aunque éste debe no ser a costa de incrementar el tiempo de ejecución del sistema. En particular con agendadores con lista de prioridad no es posible garantizar tiempos de ejecución más cortos que los que se obtienen con un solo procesador. Sin embargo estudios experimentales han mostrado buenos agendadores con lista de prioridad cuando los costos de comunicación no son grandes comparados a los de computación. Si el agendador con listas de prioridad se aplica bajo los mismos supuestos que en el caso de comunicaciones sin retardo se puede caer en el caso en que en aras de aumentar el paralelismo se incremente el tiempo de ejecución en forma no deseada debido a los costos de comunicación. Se requiere introducirle cambios al agendador con listas de prioridad para que maximice paralelismo y minimice costo de comunicación, de tal forma que se tiene tiempo de ejecución mínimo.

El **tiempo de terminación de la operación** u se define como:

$$\tau_f(u) = \tau(u) + w(u)$$

El **tiempo de terminación de la comunicación** e_{ij} del procesador p_s al procesador p_d se define como:

$$\tau_f(e_{ij}) = \tau_f(o_i) + \begin{cases} 0, & \text{si} \quad p_s = p_d \\ c\big((o_i, o_j)\big), & \text{si} \quad p_s \neq p_d \end{cases} \tag{4.15}$$

donde $e_{ij} = (o_i, o_j)$, $\pi(o_i) = p_s$ y $\pi(o_j) = p_d$. De la relación (4.15) se observa que el tiempo de terminación de comunicación depende de en que procesadores se ejecutan las operaciones que se comunican. Si las dos operaciones están en el mismo procesador el costo de la comunicación es cero mientras que si están en diferente procesador el costo de la comunicación es $c\big((o_i, o_j)\big)$. En el primer caso se dice que se tiene **nulificación de arista**. Para que el agendamiento sea válido se tienen que cumplir las restricciones de precedencia y de procesador. La restricción de precedencia se expresa como:

$$u, v \in S, \quad u \neq v, \quad uRv, \quad \tau(u) = p_s, \quad \tau(v) = p_d \quad \text{y} \quad p_s \neq p_d \Rightarrow \tau_f\big((u,v)\big) \leq \tau(v)$$

La restricción de procesador se expresa como:

$$u,v \in S, \quad u \neq v, \quad \text{y} \quad \pi(u) = \pi(v) \Rightarrow \tau_f((u,v)) \leq \tau(v) \quad \text{o} \quad \tau_f((v,u)) \leq \tau(u)$$

Al cumplirse las relaciones anteriores se satisface la relación de precedencia del grafo de dependencia. Las restricciones anteriores suponen paralelismo entre computación y comunicación es decir, mientras la comunicación de un procesador esta sucediendo, el procesador puede estar enfrascado en otra computación. De aquí en adelante sólo nos referiremos a agendamientos válidos.

Dado que a las operaciones $u \in P_r(o)$ se les ha asignado procesador y que $\pi(0) = p_d$, el **tiempo de activación de una operación** o se define como:

$$\tau_r(o) = \max_{u \in P_r(o)} \tau_f((u,o))$$

y es el tiempo al cual la operación o tiene todas sus entradas listas por lo cual está activada. Note que $\tau_r(o)$ puede ser evaluada para diferentes valores de p_d, es decir, que depende a que procesador se asigna la operación. En el caso de comunicación sin retardo $\tau_r(o)$ no depende de a que procesador se asigna la operación. Cuando se usa la estrategia orientada a operación es necesario examinar cual es el tiempo de inicialización de la operación para cada procesador, para así seleccionar el que da tiempo mínimo. Usando la definición anterior, la restricción de precedencia se puede reformular como:

$$v \in S, \quad u \in P_r(v), \quad u \neq v \quad \text{y} \quad \pi(v) \neq \pi(u) \Rightarrow \tau_r(u) \leq \tau(v)$$

dado que a las operaciones $u \in P_r(v)$ se les ha asignado procesador y que $\pi(v) = p_d$.

Conforme se construye el agendador, las operaciones se asignan a los procesadores y si la comunicación entre dos operaciones es local, esto es las operaciones están asignadas al mismo procesador, el costo de comunicación es cero. Es requerido por los algoritmos que producen un agendador poder calcular trayectorias y longitudes de éstas de grafos de dependencia parciales. Un **grafo de dependencia parcial** se construye a partir de un grafo de dependencia haciendo el costo de comunicación cero de las aristas cuyas operaciones se asignan al mismo procesador. El costo de comunicación de una arista que no se le ha asignado procesador a alguno de sus procesadores permanece igual al que tiene en el grafo de dependencia inicial. Un grafo de dependencia con todos los costos de comunicación igual a cero se llama **grafo de dependencia computacional**.

La **longitud de una trayectoria** μ del grafo dirigido acíclico $G=(S,R)$ aumentados con costos de computación y comunicación dados respectivamente por $w(v)\geq 0,\ v\in S$ y $c((u,v))\geq 0,\ u,v\in S$. Se define como la suma de los costos de computación asociados a los nodos y los de comunicación asociados a los arcos, más formalmente:

$$l(\mu)=\sum_{o\in\mu} w(o)+\sum_{\substack{u,v\in\mu\\(u,v)\in R}} c((u,v)) \tag{4.16}$$

$l(v)$ se puede interpretar como el tiempo que toma en ejecutar todas las operaciones que forman μ cuando las operaciones se ejecutan en procesadores diferentes. En este contexto la **longitud computacional de la trayectoria** v, $l_c(v)$, es el tiempo que toma ejecutar todas las operaciones de la trayectoria cuando se asignan al mismo procesador.

De la misma forma que se definen trayectorias, longitudes de éstas y trayectorias críticas para grafos de dependencia computacionales y grafos de dependencias expandidos con costo de computación y comunicación se pueden definir para grafos de dependencia parciales.

En general una trayectoria de un grafo de dependencia parcial tiene aristas cuyo costo de comunicación es cero, porque sus nodos fueron asignados al mismo procesador, o conservan el costo de comunicación del grafo de dependencia original. La longitud de una trayectoria de un grafo de dependencia parcial se denota por $l_p(\mu)$ y se calculo usando la relación (4.16) excepto que ahora algunos de los costos de comunicación de la trayectoria pueden ser cero. En general se tiene que:

$$l_c(\mu)\leq l_p(\mu)\leq l(\mu)$$

donde $l_c(\mu)$ se ha usado para denotar la longitud de una trayectoria del grafo de dependencia computacional obtenido haciendo cero todos los costos de comunicación de grafo de dependencias. Los conceptos basados en trayectorias de grafos son los mismos para los grafos de dependencia, de dependencia parcial y de dependencia computacional, lo único que varía es el grafo que se usa.

Dado un grafo de dependencia G uno de dependencia parcial derivado de G denotado por G_p y uno de dependencia computacional G_c también derivado de G tenemos las siguientes definiciones de trayectorias críticas μ_c :

$$l_c(\mu_c^c) = \max_{\mu} l_c(\mu), \quad \mu \text{ es trayectoria de } G_c,$$
$$l_p(\mu_c^p) = \max_{\mu} l_p(\mu), \quad \mu \text{ es trayectoria de } G_p \text{ y}$$
$$l(\mu_c) = \max_{\mu} l(\mu), \quad \mu \text{ es trayectoria de } G.$$

En general se tiene que:

$$\mu_c \neq \mu_c^p \neq \mu_c^c$$

Una medida de la cantidad de comunicación en relación a la cantidad de computación en un grafo está dada por el grado del grafo el cual se define como:

$$\textit{grado de grafo} = \frac{\textit{número de aristas}}{\textit{número de nodos}}$$

Otra medida es el cociente C/P el cual se define como:

$$(C/P) = \frac{\frac{1}{e}\sum_{i,j \in S,\, iRj} c(i,j)}{\frac{1}{m}\sum_{i \in S} w(i)} \tag{4.17}$$

donde e es el número de aristas del grafo.

Otra definición del cociente C/P es:

$$C/P = \frac{\sum_{i,j \in S\ iRj} c(i,j)}{\sum_{i \in S} w(i)} \tag{4.18}$$

La definición dada por la relación (4.17) es insensible al número de aristas y no refleja el volumen de datos comunicado, sin embargo es usada por algunos autores. La relación (4.18) es usada en algunas comparaciones experimentales de agendadores y nos proporciona información global acerca del impacto de los costos de comunicación, los cuales su vez determinan el comportamiento del agendador.

En la construcción de un agendador existen dos estrategia: paralelizar y secuencializar. Estas dos estrategias están en tensión según se ha visto antes. El punto de balance entre las dos estrategias está relacionado con la noción de granularidad que es una medida de la relación entre la computación y comunicación de un grafo dirigido extendido con costos de computación y comunicación. En general si la comunicación es baja se alienta

la paralelización y si es alta se alienta la serialización. El concepto de granularidad captura la relación entre los costos de computación y comunicación de un grafo dirigido. La granularidad depende del grafo de dependencia extendido y del sistema multiprocesador en el cual el programa se ejecuta, lo anterior es claro cuando consideramos procesadores y enlaces de comunicación heterogéneos. Existen varias definiciones de este concepto.

En cada operación del grafo de dependencia $G=(S,R)$ existe una ramificación y una confluencia. Las operaciones predecesoras inmediatas confluyen en la operación y las sucesoras inmediatas se ramifican de la operación, véase figura 4.23. La **granularidad débil de una operación** o_x se define como:

$$g_w(o_x)=\min\left\{\frac{\min\limits_{o\in iP_r(o_x)} w(o)}{\max\limits_{\substack{(o,o_x)\in\mathbb{R}\\ o\in iP_r(o_x)}} c((o,o_x))},\frac{\min\limits_{o\in S_u(o_x)} w(o)}{\max\limits_{\substack{(o_x,o)\in\mathbb{R}\\ o\in S_u(o_x)}} c((o_x,o))}\right\}$$

No se define la granularidad para una operación aislada, la cual es claro que no tiene ni sucesores ni predecesores. La **granularidad débil del grafo** G se define como:

$$g_w(G)=\min_{o\in S} g_w(o)$$

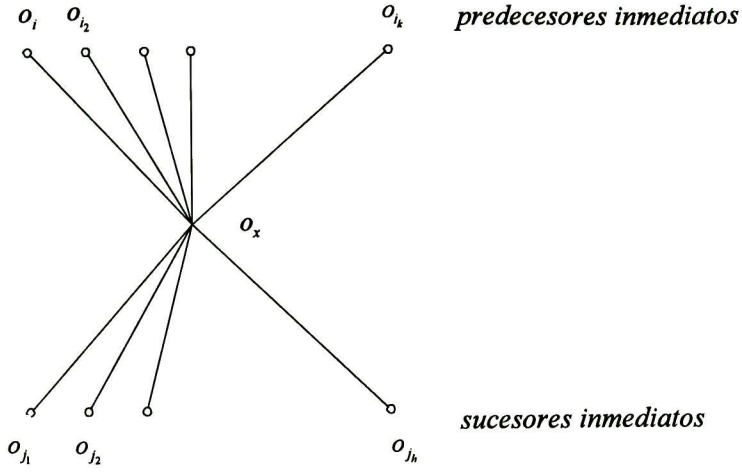


Figura 4.23: Predecesores y sucesores de una operación.

Directo de las definiciones anteriores se tiene que:

$$g_w(G) \leq g_w(o_x) \leq \frac{w(o_x)}{c((o_x, v))} \tag{4.19}$$

donde $o_x, v \in S$ y $(o_x, v) \in \mathbb{R}$.

La granularidad del grafo G se dice **gruesa** si $g_w(G) \geq 1$ y **fina** si no cumple la relación anterior. Para el caso de un grafo de granularidad gruesa se tiene que el costo de comunicación es pequeño con respecto al costo de computación. Un grafo con granularidad gruesa es preferible a uno de granularidad fina debido a que es menos difícil agendador un grafo de granularidad gruesa. Una manera de lograr un grafo con mayor granularidad es por aglutamiento de varias operaciones en un cúmulo al cual eventualmente se les asigna un procesador para su ejecución. El aglutamiento incrementa la granularidad disminuyendo a cero los costos de comunicación de las operaciones que están en el mismo cúmulo. Observamos que el aglutamiento de operaciones está en tensión con el objetivo de ejecutar tantas operaciones en paralelo como es posible para acotar el tiempo de ejecución de un grafo.

Otra definición de granularidad de un grafo $G=(S,R)$ extendido con costos de computación y comunicación es:

$$g(G)=\frac{\min_{o\in S} w(o)}{\max_{\substack{(u,v)\in \mathbb{R}\\ u,v\in S}} c((u,v))}$$

directo de las definiciones se tiene que:

$$g(G)\le g_w(G) \qquad \text{y} \qquad g(G)\le \frac{w(o)}{c((u,v))} \tag{4.20}$$

donde u y $v\in S$ de la relación anterior se ve que es más fácil en general cumplir con el criterio de granularidad débil. Más adelante se presentan algunos resultados donde los dos conceptos de granularidad son clave.

Teorema 4.11. Sea el grafo dirigido $G=(S,R)$ extendido con costos de computación y comunicación respectivamente dados por $w(o)\ge 0$, $c(u,v)\ge 0$, u y $v\in S$. μ_c y μ_c^c son respectivamente las trayectorias criticas de G y G_c donde G_c es el grafo computacional de G $\mu_c=v_0,\ldots,v_k$ donde $v_i\in S$, $i=0,\ldots,k$ y $\mu_c^c=v_0,v_1,\ldots,v_k$ donde $v_i\in S, i=0,\ldots,j$. Se tiene que:

$$l(\mu_c)\le\left(1+\frac{1}{g_w(G)}\right)l(\mu_c^c)$$

■

Ejemplo 4.11. Sea el grafo dirigido extendido mostrado en la figura 4.24. Note que conforme se avanza en la ejecución el tiempo de computación y comunicación se incrementan.

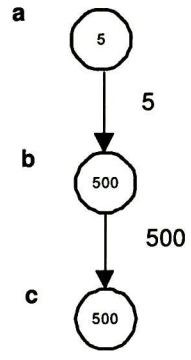


Figura 4.24: Grafo dirigido extendido.

La granularidad está dada por:

$$g(G) = \frac{\min\{5,500\}}{\max\{1,500\}} = \frac{5}{500} = \frac{1}{100}$$

y la granularidad débil es:

$$g_w(G) = \min\{g_w(a), g_w(b), g_w(c)\} = \{1,1,1\} = 1$$

observamos que a pesar de que las dos definiciones tratan de capturar la misma noción sus valores pueden ser muy diferentes. Para el caso tratado, la granularidad débil parece ser más adecuada.

■

Si la granularidad es muy grande se reduce el paralelismo dado que se agrupan operaciones para ser ejecutadas secuencialmente por un procesador. Si la granularidad es muy pequeña, el costo de comunicación incrementa el tiempo de ejecución del agendador.

4.5.1 Aglutinamiento

En esta sección estamos interesados en encontrar agendadores que minimizan el tiempo de ejecución cuando el número de procesadores es ilimitado y se tienen costos de computación y de comunicación. Es decir, hay retardo de comunicación entre procesadores. Se supone una red de interconexión totalmente conectada donde no hay contención ni intervención de los procesadores en la comunicación. Los agendadores encontrados son útiles para construir agendadores donde el número de procesadores es limitado. El problema es NP completo.

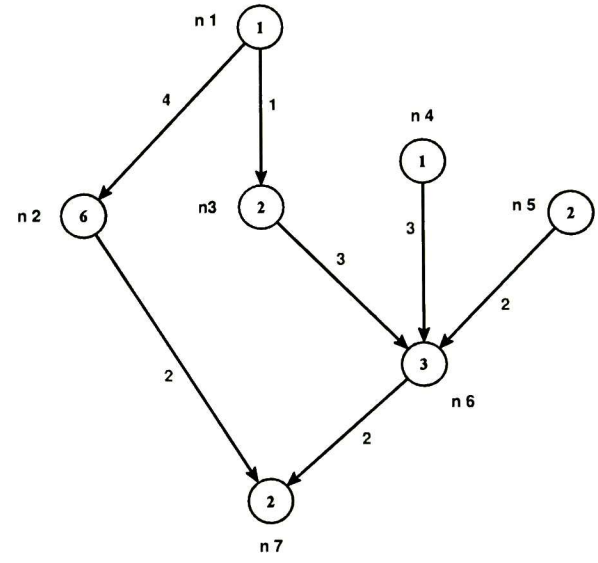
En este contexto el proceso de asignar procesadores a las operaciones se llama **aglutinamiento**. Un **cúmulo** es un conjunto de operaciones. Los algoritmos heurísticos para aglutinamiento presentados son incrementales y en cada paso refieren el aglutinamiento actual de cúmulos. La idea es que cada cúmulo del aglutinamiento final se ejecute en un solo procesador. Durante el aglutinamiento se nulifican aristas que tienen alto costo de comunicación. Las operaciones unidas por la arista nulificada se asocian al mismo cúmulo.

Sea un dag $G=(S,R)$ extendido con funciones de costo de computación y de comunicación dadas respectivamente por $w(i)\geq 0$ y $c(i,j)\geq 0$ para $i,j\in S$. Un aglutamiento C es una función $C:S\rightarrow N$ que asigna cada operación a un cúmulo identificado por un número natural. La función C induce una partición de las operaciones del grafo G y un cúmulo contiene los elementos de una clase de equivalencia. El diseño de C es tal que minimiza el tiempo de ejecución del agendador.

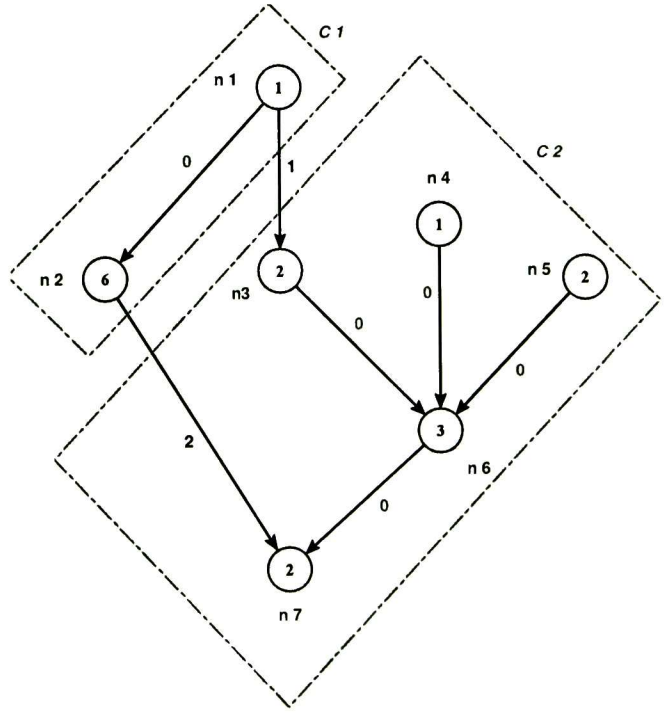
Los algoritmos de aglutinamiento que presentamos son iterativos. Se empieza con un aglutamiento C_0 donde cada cúmulo sólo contiene una sola operación, es decir, se empieza con máximo paralelismo. En la iteración i se mejora el aglutamiento C_{i-1} para obtener el aglutamiento C_i. Se espera que el nuevo aglutamiento no tenga peor tiempo de ejecución que su predecesor. Entre los criterios que se usan para parar la iteración están: no hay más vértices o arcos que procesar o el tiempo de ejecución del nuevo aglutamiento de peor que el previo.

Ahora introducimos terminología y conceptos básicos que nos ayudan a la explicación de los algoritmos heurísticos para encontrar aglutinamientos. Un cúmulo se dice **lineal** si sus operaciones están totalmente ordenadas de otra forma se dice **no lineal**. Un **aglutinamiento lineal** es un aglutinamiento con solo cúmulos lineales. Un aglutinamiento es **no lineal** si contiene al menos un cúmulo no lineal. El problema de encontrar un agendador con tiempo de ejecución mínimo para un aglutinamiento lineal es de complejidad polinomial, sin embargo si el aglutinamiento es no lineal el problema es NP completo. Un aglutinamiento lineal equivale a una asignación de procesadores a las operaciones donde el orden en que cada procesador ejecuta las operaciones está

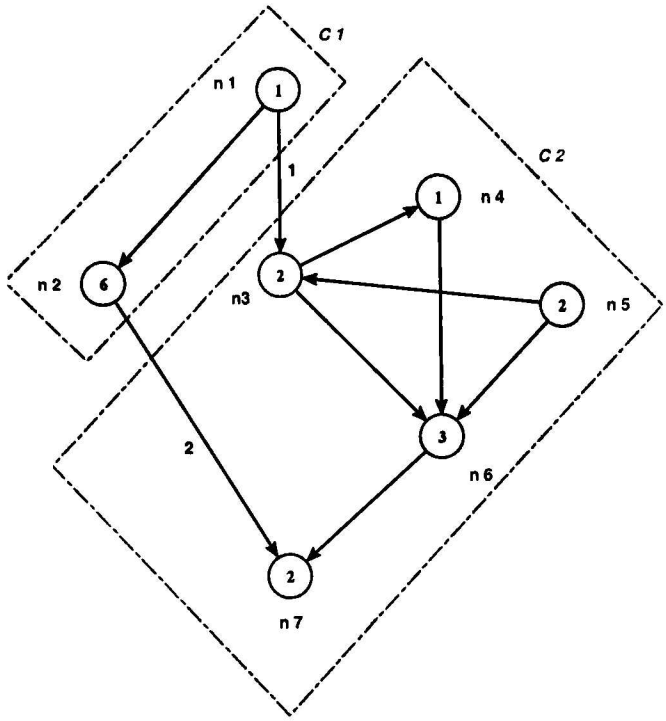
determinado por un orden total que es el mismo que el del cúmulo lineal asociado al procesador.



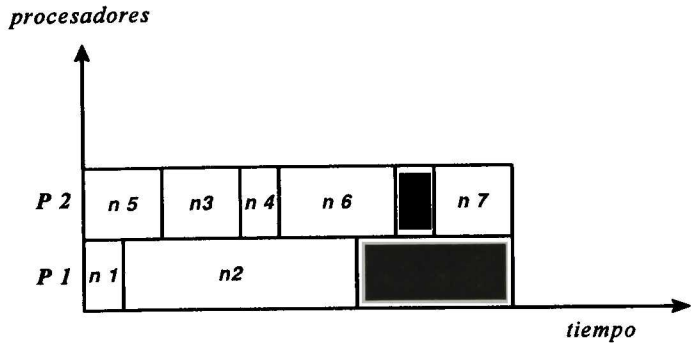
a) Grafo de dependencias.



b) Grafo aglutinado.



c) Grafo calendarizado.



d) Diagrama de Gantt

Figura 4.25: Grafos aglutinado y calendarizado.

Cuando en un grafo dirigido se reflejan los cúmulos del aglutinamiento se dice que tenemos un **grafo aglutinado**. La figura 4.25 muestra un grafo dirigido y un grafo aglutinado que se obtiene de él. Los costos de comunicación de las aristas dentro de un cúmulo son cero, mientras los enlaces que conectan cúmulos conservan su costo de comunicación inicial. El grafo que se obtiene de un grafo aglutinado una vez que todos los cúmulos no lineales se convierten en lineales se llama **grafo calendarizado**. La

figura 4.25 muestra un grafo calendarizado y el grafo dirigido del cual se obtiene. El grafo calendarizado es un supergrafo del grafo original donde las aristas añadidas son las que convierten los cúmulos no lineales en lineales.

Se presentan algoritmos heurísticos para aglutinar operaciones de un grafo dirigido en cúmulos bajo el supuesto que se tiene un número ilimitado de procesadores. El propósito es minimizar el tiempo de ejecución del grafo. Dado que no es posible calcular el tiempo de ejecución de manera exacta se usa el estimado de éste que antes se introdujo. Los algoritmos heurísticos que se presentan son:

i. aglutinamiento lineal,
ii. aglutinamiento voraz y
iii. aglutinamiento de secuencia dominante

Los tres algoritmos son iterativos y examinan todas las aristas del grafo dirigido y terminan cuando se han examinado todas las aristas del grafo. En el paso i del algoritmo se construye el aglutamiento C_i a partir del aglutamiento C_{i-1}, todo esto disminuyendo, o no incrementando, el estimado del tiempo de ejecución de C_{i-1}. El aglutinamiento C_i o es igual a C_{i-1} o se obtiene del aglutinamiento C_{i-1} al unir algunos de los cúmulos de C_{i-1} en un nuevo cúmulo. La decisión de unir o no varios cúmulos depende del comportamiento del estimado del tiempo de ejecución del agendador, si este se incrementa con la unión, entonces ésta no se efectúa. La unión de dos cúmulos nulifica el costo de las aristas que las unen.

El tiempo estimado de ejecución no se mejora uniendo cúmulos que están conectados por aristas de costo de comunicación cero. Al contrario, puede suceder que la unión aumente el tiempo estimado de ejecución, esto se debe a que las operaciones de un cúmulo se ejecutan de inicialmente por un solo procesador, eliminando así cierto paralelismo que existía antes de la unión. C_0 es un aglutinamiento unitario esto es todos sus cúmulos son unitarios y un **cúmulo unitario** sólo contiene una operación. Un aglutinamiento unitario es lineal. El algoritmo de aglutinamiento lineal usa el hecho que la unión de dos cúmulos lineales que resulta en un cúmulo lineal no incrementa el tiempo estimado de ejecución.

La idea básica del algoritmo de **aglutinamiento lineal** es obtener el aglutinamiento C_i decreciendo el estimado del tiempo de ejecución del aglutamiento C_{i-1}. Por esto se aglutina en un cúmulo los cúmulos de una trayectoria crítica de C_{i-1}. El paso i es como sigue: se identifica una trayectoria crítica en C_{i-1}. C_i se obtiene de C_{i-1} al eliminar todos los cúmulos de C_{i-1} en la trayectoria crítica y formar un nuevo cúmulo con los cúmulos eliminados. Los arcos en la trayectoria crítica de C_{i-1} se marcan como examinados.

La idea básica del **aglutinamiento voraz** es unir en un cúmulo los cúmulos de C_{i-1} que están unidos por el arco de costo de comunicación más alto, lo anterior se efectúa sólo si esta acción no incrementa el estimado del tiempo de ejecución del aglutamiento C_{i-1}.

La idea básica del algoritmo de **secuencia dominante** es construir una trayectoria crítica de C_{i-1}, ésta es la **secuencia dominante** del paso i, seleccione una arista de la secuencia dominante, elimine los dos cúmulos unidos por el arista seleccionada e introduzca un nuevo cúmulo que se obtiene de unir dos cúmulos eliminados. Se marcan como examinados y se nulifican los pesos de comunicación a todas las aristas que unen los cúmulos que se unen. Existen las siguientes formas de seleccionar la arista:

i. el que decrece más el estimado del tiempo de ejecución
ii. el que mayor costo tiene,
iii. el primero y
iv. al azar

El listado 4.3 muestra un algoritmo que es un esqueleto para implementar cualquiera de los algoritmos de aglutamiento antes mencionados.

Entrada: grafo dirigido $G=(S,R)$ aumentado con costos de computación y comunicaciones respectivamente dados por $w(u)\geq 0$, $C((u,v))\geq 0$, y u y $v\in S$.
Salida: C_i un aglutamiento con estimado de tiempo de ejecución subóptimo.

```
C_o = S;                                   /* aglutamiento inicial
for(j = 0; j++)
  {
    C = selectClustersToMerge(C_{j-1});    /* selecciona cúmulos a fusionar
    c = mergeClusters(C);                  /* fusiona cúmulos
    C = C_{j-1} - C ∪ {c};
    if (estimado de tiempo de ejecución de C > estimado de tiempo de ejecución de C_{j-1})
      C_j = C_{j-1};
    else
      C_j = C;
    }
```

Listado 4.3: Algoritmo general de aglutinamiento.

La característica que diferencia a los algoritmos de aglutamiento antes tratados es como se seleccionan los cúmulos que posteriormente se fusionan para ser un solo cúmulo. El algoritmo de aglutinamiento lineal selecciona todos los cúmulos de una trayectoria crítica del grafo. El algoritmo de aglutinamiento voraz selecciona los cúmulos unidos por la arista de mayor costo que no ha sido examinada y el algoritmo de aglutinamiento de secuencia dominante selecciona una arista de la secuencia dominante y los cúmulos unidos por ésta se fusionan. La fusión de cúmulos puede desencadenar la fusión de otros cúmulos que están unidos por aristas con los cúmulos que se fusionan.

El algoritmo de aglutinamiento se puede usar para aumentar la granularidad de un grafo de dependencia. Se inicia con el grafo que tiene granularidad más baja y a éste se le aplica aglutinamiento. La mejora de la granularidad la ocasiona la nulificación de las aristas y no la creación de nuevas operaciones con más costo de computación. Para nulificar las aristas se supone que las operaciones de un mismo cúmulo se asignan eventualmente a un mismo procesador. Durante el proceso de aglutinamiento cierto paralelismo es eliminado al asignar diferentes operaciones independientes al mismo procesador. Sin embargo si el paralelismo eliminado se trata de aprovechar degrada el tiempo de ejecución del agendador, es decir la idea es eliminar paralelismo que no contribuye a disminuir el tiempo de ejecución del agendador. La granularidad es tal que minimiza el tiempo de ejecución del agendador. Remarcamos que en la estrategia propuesta las operaciones no pierden su individualidad, no se forma una gran operación, dentro de un cúmulo o las operaciones conservan un ordenamiento y su estructura de comunicación con otras operaciones fuera del cúmulo.

4.5.2 Agendador con Lista de Prioridad

En esta sección extendemos los agendadores con lista de prioridad antes estudiados al caso donde se tienen retardos de comunicación. En este caso el problema también es de complejidad NP completo.

En los agendadores con lista de prioridad se asigna priòridad a todas las operaciones del grafo de dependencia y se construye una lista de prioridad con las operaciones activadas. En los agendadores con lista de prioridad la estrategia consiste en asignar procesador a la operación de mayor prioridad cuyos predecesores ya han sido ejecutados. La diferencia entre los agendadores con lista de prioridad se debe a como asignan prioridad a las operaciones y como se selecciona un procesador para asignación de operación activada. Como ya se vió las estrategias orientadas a procesador y orientada a operación producen resultados equivalente cuando no se tienen retardos de comunicación. En el ejemplo 4.9 y el ejemplo 4.10 se mostró que para el caso con retardos de comunicación éste no es el caso. La estrategia orientada a procesador que es la que procura balancear las cargas de los procesadores produce resultados muy pobres, esto es muy alejado del valor óptimo. De hecho se da el caso que un procesador tiene mejor tiempo de ejecución que un sistema multiprocesador. La estrategia voraz de sólo considerar la utilización de los procesadores sin considerar los costos de comunicación no resulta adecuada.

En el caso de agendadores con grafos de dependencia aumentados con costos de comunicación puede ser conveniente usar **agendadores adaptativos** donde las prioridades se recalculan para cada grafo de dependencias parcial. Las nuevas prioridades se pueden calcular para todas las operaciones no ejecutadas o únicamente para las no ejecutadas y activadas.

Dos técnicas para mejorar el desempeño de agendadores con lista de prioridad son: en duplicación de operaciones e inserción en medio.

En **duplicación de operación** para reducir el costo de comunicación, una misma operación se puede ejecutar en múltiples procesadores evitando así transferencia de datos entre procesadores. La duplicación de operaciones es una estrategia para reducir la granularidad de un grafo de dependencia.

Dada una operación activada se asigna el procesador que permite la ejecución más temprana de aquélla. La operación se puede calendarizar al final de las operaciones ya calendarizadas o antes entre algunas de las ya calendarizadas. En el primer caso tenemos **inserción al final** y en el segundo **inserción en medio**. La inserción al final siempre es posible mientras que la inserción en medio en ocasiones no es posible. Para que la inserción en medio sea posible debe existir un período ocioso del procesador

entre dos operaciones y la operación a la que se le quiere asignar procesador debe estar activada en este período. La inserción en medio crea posibilidades para acortar el tiempo de ejecución del calendarizador. En la inserción al final, el tiempo de iniciación de ejecución de la operación o esta dado por:

$$\tau(o) = \min_{p_i \in P} \max\left(\tau_r(o), \tau_f(p_i)\right) \tag{4.21}$$

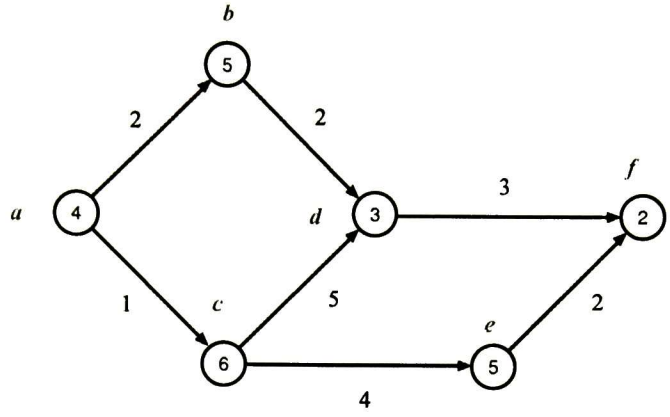
y la asignación es $\pi(o) = p_{\min}$ donde o es la operación a la que se le asigna procesador. La operación de max supone la asignación $\pi(o) = p_i$. P es el conjunto de procesadores. $p_{\min}$ es el procesador que minimiza la expresión (4.21). max se evalúa al tiempo de iniciación de ejecución de la operación o cuando la asignación es $\pi(o) = p_i$.

En la descripción anterior el sistema de comunicaciones entre procesadores se ha supuesto completamente conectado donde se pueden tener múltiples comunicaciones simultáneamente y que los procesadores no están involucrados en el proceso de comunicación. En consecuencia cada enlace de comunicación se modela como retardo. Nos interesa poder modelar implementaciones del sistema de comunicaciones más complejas que las descritas.

Una de las formas de producir un agendador con número limitado de procesadores es usar como entrada el aglutinamiento producido por los algoritmos heurísticos ya mencionados, para esto cada cúmulo del aglutinamiento se considera como una operación.

Como un primer paso modelamos la comunicación entre dos operaciones por medio de un recurso, canal de comunicación, cuya operación es comunicar y que tiene asociado un peso que es el costo de comunicación. Con este cambio los nodos del dag representan computaciones y comunicaciones, y los arcos representan orden de procedencia o dependencias. La figura 4.26 muestra un grafo con retardos y un grafo equivalente donde el canal de comunicación se modela como un nodo. Como posteriormente veremos este cambio nos permite modelar sistemas de comunicaciones más complicados y sólo se necesitan los grafos con los pesos asociados a los nodos y no a las aristas. En este caso modelamos el programa por un dag $G = (S, R)$ extendido con funciones de costo de computación y de comunicación dados respectivamente por $w(i) \geq 0$ y $c(i,j) \geq 0, \quad i,j \in S$. Se supone que la red que interconecta los procesadores es totalmente conectada y que hay un retardo cuando dos procesadores se comunican y esta dado por el costo de comunicación que en este caso lo interpretamos como tiempo de comunicación y es el tiempo que toma desde que se envía el primer bit hasta que se recibe el último bit. Los procesadores son homogéneos y no intervienen en la comunicación y múltiples procesadores se pueden comunicar ala vez.

Con el cambio introducido no sólo las operaciones pueden ser calendarizadas sino las comunicaciones también. Una diferencia, la cual tratamos más adelante, es que mientras una operación es ejecutada por un procesador, una comunicación puede requerir de múltiples canales de comunicación y de enrutamiento.



a) Grafo con retardos no presentados en los enlaces.

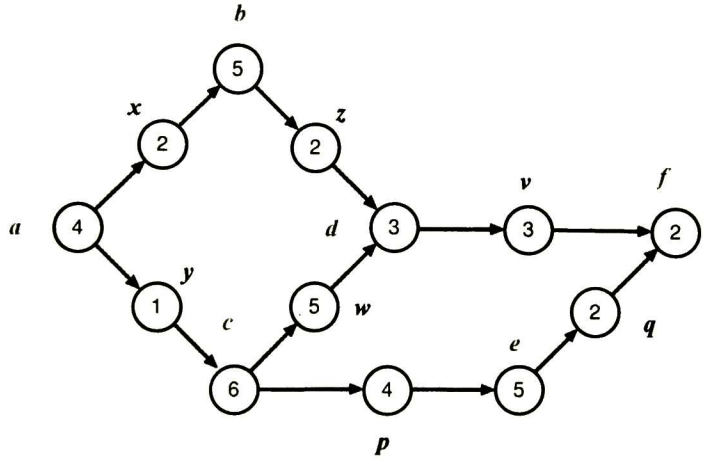


Figura 4.26: Grafos equivalentes con enlaces anotados con retardos y nodos que son canales de comunicación.

En el análisis de sistemas digitales secuenciales la complejidad de computo dada en término de tiempo y espacio es lo común. En el caso de sistemas paralelos es necesario incluir el tiempo de comunicación y sincronización entre operaciones. La comunicación y la sincronización son facturas que fuertemente afectan el factor de aceleración que se puede alcanzar con los múltiples procesadores y se deben incluir en el análisis de algoritmos paralelos.

Teorema 4.12. Sea un grafo dirigido de dependencia $G=(S,R)$ extendido con costos de computación y comunicación expresados respectivamente por $w(u)$ y $C((u,v)), v \text{ y } u \in S$ Sean τ y π respectivamente funciones de calendarización y asignación para el grafo G Sean $\tau^{(l)}$ y $\pi^{(l)}$ respectivamente las funciones de calendarización y de asignación de un agendador con lista de prioridad que usa la siguiente regla para calcular el tiempo de iniciación de la operación o dado que $\pi^{(l)}(o)=p$.

$$\tau^{(l)} = \max\left\{\tau_f^{(l)}(p),\ \tau_r^{(l)}(o)\right\} \tag{4.22}$$

que usa la siguiente asignación de procesador

$$\pi(o)=\pi^{(l)}(o)$$

y que usa $\tau(o)$ como la prioridad de operación o. Entonces se tiene que los tiempos de ejecución de los calendarizadores satisfacen que:

$$T^{(l)} \leq T$$

donde $T^{(l)}$ y T son respectivamente los tiempos de ejecución del calendarizador con lista de prioridad y el calendarizador original.

■

En el teorema 4.12, el calendarizador definido por τ podría ser el óptimo dado por τ_{opt} en cuyo caso el calendarizador producido por el agendador con listas de prioridad es óptimo. Lo cual implica que un agendador óptimo está definido por el orden entre operaciones y la función de asignación. El calendarizador óptimo se obtiene en este caso del uso de un agendador con lista de prioridad de la forma que se describe por la relación (4.22) del teorema 4.12.

El **agendamiento** abarca *agendadores* que modelan el *multiprocesador* como un conjunto de procesadores homogéneos donde no se permite la interrupción y desalojo de la operación que se ejecuta. Sobre el sistema de comunicaciones se supone que es completamente conectado y que los procesadores no intervienen en la comunicación para esto se cuenta con procesadores de comunicación. Se pueden tener múltiples comunicaciones simultáneas y no hay contención en el acceso a un enlace de comunicaciones. Los agendadores estudiados con profundidad son los llamados con **listas de prioridad** y que forman la base para agendadores más complejos. Los sistemas paralelos considerados se pueden modelar por un grafo de dependencias

extendido con un costo de computación asociado a cada nodo del grafo y un costo de comunicación asociado a cada arista del grafo. Es decir, los agendadores estudiados son útiles para agendar cualquier proceso que se puede describir por un grafo extendido.

Capítulo 5

Conclusiones y Trabajo Futuro

Ninguna investigación humana se puede calificar como realmente científica si no puede ser demostrada matemáticamente.
Leonardo Da Vinci (1452-1519)

5.1 Conclusiones

El programa secuencial para el algoritmo que efectúa la descomposición QR se modela como un poliedro el cual tiene embebido un grafo de dependencia. El poliedro nos revela eparalelismo existente en el programa secuencial, con objeto de obtener un paralelismo más eficiente se transforma el poliedro original en un poliedro destino. Del poliedro destino se obtiene el programa paralelo.

Las transformaciones iniciales al programa son para volverlo un programa con localidad espacial y temporal. Las transformaciones se reducen a una transformación ***espacio-temporal*** y su objetivo es la extracción de paralelismo del programa secuencial. El programa paralelo obtenido contiene toda la información para implementar la trayectoria de datos y controlador de ésta del circuito que implementa la factorización QR.

En este trabajo se encuentran matemáticamente las ecuaciones que permiten transformar los índices ***espaciales*** en índices ***espacio-temporales***.

La transformación de ***embaldosado*** presentada en este trabajo permitirá la implementación en hardware de un arreglo de procesadores virtuales con un gran número de éstos con una cantidad menor de procesadores físicos en el arreglo final, permitiendo así la ejecución de problemas que requieren un gran número de procesadores virtuales en un hardware que tiene menor cantidad de procesadores físicos. Con la implicación de tiempos más largos en la ejecución del algoritmo.

La transformación de ***multiproyección*** es una transformación espacio-temporal, la cual nos permite mapear muchos procesadores virtuales a unos pocos procesadores físicos. El programa paralelo obtenido contiene toda la información para implementar la trayectoria de datos y controlador de ésta del circuito que implementa la factorización QR en un arreglo lineal de procesadores.

La contribución en el capítulo de agendamiento es de resumir y presentar en una forma clara los resultados más básicos de la teoría del agendamiento, además se presenta la teoría y pseudocódigo para construir agendadores con *lista de prioridad* bajo las hipótesis antes mencionadas.

5.2 Trabajo Futuro

El algoritmo que efectúa la *descomposición QR* se ha llevado a una forma directa, para su aplicación en los *Sistemas de Comunicación*. Dada la tendencia de particionar un programa y usar múltiples procesadores simultáneamente para su ejecución, este enfoque se conoce como **procesamiento en paralelo**.

Los cimientos proporcionados a lo largo de este trabajo proporcionan una base sólida para profundizar en el estudio, de modelar un programa *secuencial*, como un *poliedro* y realizar con el toda una serie de *transformaciones*, para la extracción de *paralelismo*. Los resultados obtenidos son aplicables a una amplia gama de algoritmos presentes en los campos de la ingeniería.

La teoría presentada en el capítulo de agendamiento es una plataforma sobre la cual se pueden desarrollar teoría y agendadores para sistemas paralelos más *complejos*. Los agendadores que consideran modelos más realistas de los sistemas paralelos, incluyen *procesadores heterogéneos*, *enlaces de comunicación heterogéneos* con *redes de comunicación complicada* y que pueden incluir *intervención de los procesadores en la comunicación*.

APÉNDICE A

Prueba de las expresiones de $s = f(a,b,r)$ y $c = g(a,b,r)$ a partir de la ecuación (2.3):

$$(ca+sb)^2 = r^2 = c^2a^2 + s^2b^2 + 2casb \qquad \text{(A. 1)}$$

$$(-sa+cb)^2 = 0 = s^2a^2 + c^2b^2 - 2casb \qquad \text{(A. 2)}$$

Usando el hecho de que $c^2 + s^2 = 1$ al sumar las relaciones (A. 1) y (A. 2) se tiene que:

$$r^2 = a^2 + b^2 \qquad \text{(A. 3)}$$

Que es uno de los resultados requeridos. De la relación (A. 3) tenemos que:

$$\frac{a^2}{b}s + bs = r = \frac{s}{b}\left(a^2 + b^2\right) = \frac{sr^2}{b}$$

De donde se tiene que:

$$s = \frac{b}{r}$$

De igual manera obtenemos que:

$$c = \frac{a}{r}$$

■

APÉNDICE B

Relaciones y Grafos

B.1 Introducción

Se presentan las bases de la teoría de relaciones y grafos. Las relaciones de orden parcial y su representación por medio de grafos se usan como modelos de programas paralelos, de allí nuestro interés. Los grafos también se usan como modelos del sistema de comunicaciones de máquinas paralelas. Se estudian los algoritmos más básicos relacionados con los grafos. Para detalles de los *teoremas* y *lemas* presentados consultar en la bibliografía Banerjee, U., *Loop Transformation for Restructuring Compilers: The Foundations*. Intel Corporation, Ed. Kluwer Academic Publishers, U.S.A., 1993. [3].

La notación O se refiere a una cota superior asintótica a la complejidad. Una función $f(x)$ se dice de orden $g(x)$, denotado por $O(g(n))$, si existen constantes c y N tal que $cg(n) \geq f(n)$ para $n \geq N$ Un algoritmo es de complejidad en tiempo polinomial si tiene tiempo de ejecución $O(g(n))$ donde $g(n)$ es un polinomio y n el tamaño de los datos de entrada al algoritmo. Mientras más pequeño el grado del polinomio $g(n)$ más eficiente es el algoritmo. Un algoritmo es de **complejidad exponencial, intratable o no eficiente**, si $g(n) = e^{+kn}$ para $k > 0$, es decir la complejidad en tiempo es enorme para tamaños regulares de n.

B.2 Rudimentos de la Teoría de Relaciones

En lo que sigue se hace una recapitulación sobre la teoría de las relaciones que es pertinente para el estudio de sistemas paralelos.

Una relación es un concepto fundamental de las matemáticas y fue inventada para capturar asociaciones entre objetos que aparecen en la realidad o en las mismas matemáticas. El estudio de relaciones se requiere para el entendimiento del paralelismo y el orden de ejecución de operaciones definidas por un programa.

El **producto cartesiano** de los conjuntos S y T denotado por $S \times T$ se define como:

$$S \times T = \{(s,t) \mid s \in S \text{ y } t \in T\}$$

Es decir, el producto cartesiano de dos conjuntos es un conjunto de pares ordenados donde el primer elemento del par es un elemento de primer conjunto y el segundo elemento del par es un elemento de segundo conjunto. Una **relación binaria** R entre el conjunto S y T es un subconjunto del producto cartesiano $S \times T$ cuando $S = T$ decimos que R es una relación en S S es el **dominio de la relación** y T el **contradominio**. Sean las relaciones $R_1 \subseteq A \times B$ y $R_2 \subseteq A \times B$. La relación R_1 es una **extensión** de R_2 si se cumple que $R_2 \subset R_1$. La **inversa de la relación** R se denota por R^{-1} y se define como un conjunto de pares ordenados tal que:

$$(t,s) \in R^{-1} \Leftrightarrow (s,t) = R$$

La **imagen** de A bajo la relación $R \subseteq Z \times W$. $R(A)$, se define como:

$$R(A) = \{b \in W \mid zRb,\ z \in A \subseteq Z\}$$

y la **imagen inversa** de B, $R^{-1}(B)$, se define como:

$$R^{-1}(B) = \{a \in Z \mid aRz,\ z \in B \subseteq W\}$$

La relación $R \subseteq Z \times W$ es **sobreyectiva** o **sobre** si $R(Z) = W$ y es **total** si $R^{-1}(W) = Z$. Note que si una relación no es total entonces existe al menos un z tal que zRx no se satisface para toda $x \in W$ Una relación que no es total se dice **parcial** si $R \subseteq Z \times Z$, $x \in Z$, $y \in Z$ entonces x y y son comparables si xRy o yRx.

Sean las relaciones $R_1 \subseteq A \times B$ y $R_2 \subseteq B \times C$ la **composición**, de R_1 y R_2 se define como:

$$R_2 \circ R_1 = \{(a,c) | \exists b((a,b) \in R_1 \text{ y } (b,c) \in R_2)\}$$

Es decir, que $(a,c) \in R_2 \circ R_1$ si existe una b tal que $(a,b) \in R_1$ y $(b,c) \in R_2$. Note que en general la aplicación de composición entre relaciones no conmuta, esto es $R_2 \circ R_1 \neq R_1 \circ R_2$. No todos los autores definen la composición como se hizo antes. Definimos R^n como:

$$R^n = \underbrace{R \circ (R \circ (\ldots \circ (R)) \ldots)}_{R \text{ aparece } n \text{ veces}}$$

Sea R una relación sobre el conjunto A y $B \subset A$. La **restricción** de R a B es la relación dada por $R' = R \cap (B \times B)$.

Sea R una relación binaria definida en el conjunto S.

i. R es **reflexiva** $\Leftrightarrow \forall x \in S \quad xRx$.
ii. R es **irreflexiva** $\Leftrightarrow \forall x \in S \sim (xRx)$.
iii. R es **antisimétrica** $\Leftrightarrow \forall x, y \in S \quad xRy \wedge yRx \rightarrow x = y$.
iv. R es **asimétrica** $\Leftrightarrow \forall x, y \in S \quad xRy \rightarrow \sim yRx$.
v. R es **simétrica** $\Leftrightarrow \forall x, y \in S \quad xRy \rightarrow yRx$.
vi. R es **transitiva** $\Leftrightarrow \forall x, y, z \in S \quad xRy \wedge yRz \rightarrow xRz$.

Una consecuencia de la propiedad antisimétrica es: $\forall x, y \in S$ si $x \neq y$ y xRy entonces no es el caso que yRx, es decir, que no se puede dar simultáneamente que xRy y yRx cuando $x \neq y$.

Dada una relación R, la **operación de cerradura** con respecto a una propiedad aumenta los elementos de la relación, tal que la propiedad deseada se cumple. El número de elementos que se le agrega a la relación debe ser el mínimo necesario.

La **cerradura transitiva** de R denotada por R^+ se define como: $R^+ = \{(x,y) | xRy \text{ o } \exists n : xR^n y\}$. Note que $R \subseteq R^+$ y R^+ es el conjunto más pequeño que satisface esta relación. R^+ es una extensión de R.

Un **grafo dirigido**, también llamado **digrafo**, es formalmente una relación. Si $R \subseteq A \times B$, los nodos del grafo son los elementos de A y B que satisfacen $R \subseteq A \times B$.

Existe un arco dirigido de x a y donde $x \in A$, $y \in B$ si xRy. Es usual representar los dígrafos de manera geométrica. Los nodos o vértices esto es los elementos de A y B se representan por pequeños círculos en el plano. Un círculo por cada nodo. aRb se representa por una flecha que va del nodo que representa a el nodo que representa b. Los arcos que tienen como origen y destino el mismo nodo se llaman **autolazos**.

Del grafo de la relación se pueden comprobar fácilmente algunas propiedades.

i. reflexividad: todos los nodos tienen autolazos.
ii. irreflexibilidad: no existen autolazos.
iii. simetría: todos los arcos son bidireccionales.
iv. transitividad: dada una trayectoria, existe un arco que va del origen al destino de la trayectoria.
v. antisimetría: dado un arco (a,b) de la relación, donde a y b son nodos diferentes, el arco (b,a) no pertenece a la relación.

Ejemplo B.1. Sea $R=\{(1,1),(2,2),(3,3),(1,2),(1,3),(4,1)\}$. La figura b.1 muestra el dígrafo de la relación anterior.

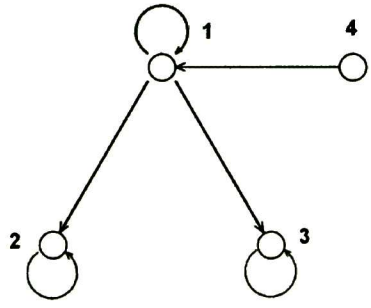


Figura B.1: Digrafo de la relación R.

La relación del ejemplo anterior es reflexiva dado que todos los nodos tienen autolazo. No es simétrica dado que no hay arcos que unan dos nodos y vayan en dirección contraria. La relación tampoco es transitiva dado que para arcos consecutivos no hay un arco que una el primer nodo con el último.

■

B.2.1 Relación de Equivalencia

Las relaciones de equivalencia y las relaciones de orden son dos clases de relaciones que aparecen en el estudio de paralelismo.

Sea S un conjunto y $S_i \subset S$, $i=1,...,m$. Los conjuntos S_i, $i=1,...,m$ son una **partición** de S si $\bigcup_{i=1}^{m} S_i = S$ y $S_i \cap S_j = \varnothing$ para $i \neq j$. Sean $\{S_i\}$ y $\{S_i'\}$ dos particiones del conjunto S. Si para toda S_i existe S_j' tal que $S_i \subseteq S_j'$ entonces decimos que $\{S_i\}$ es una **partición más fina** que $\{S_i'\}$ de S y que $\{S_i'\}$ es una **partición más tosca** que $\{S_i\}$ de S.

Ejemplo B.2. Sean $S=\{1,2,3,4,5,6,7,8\}$, $S_1=\{1,3,5\}$, $S_2=\{7,8\}$ y $S_3=\{2,4,6\}$. Se tiene que $S=S_1 \cup S_2 \cup S_3$ y $S_1 \cap S_2 = \varnothing$, $S_1 \cap S_3 = \varnothing$, y $S_2 \cap S_3 = \varnothing$ De lo anterior se tiene que S_1, S_2 y S_3 son una partición de S

■

Una **relación de equivalencia** es una relación que es reflexiva, simétrica y transitiva. Sean S un conjunto y R una relación de equivalencia en S Definimos $[a]_R = \{x \mid aRx\}$, $[a]_R$ es la **clase de equivalencia** de a bajo R. a es un representante de $[a]_R$. Los siguientes lemas nos indican que una partición de un conjunto S induce una relación de equivalencia y viceversa, una relación de equivalencia en un conjunto S define una partición de éste. Estos lemas tienen múltiples aplicaciones en ciencia e ingeniería.

Lema B.1. Sea R una relación en el conjunto S. Los conjuntos $[a]_R$ para $a \in S$ constituyen una partición del conjunto S

■

Lema B.2. Dada una partición S_i, $i=1,...,n$ del conjunto S y la relación R tal que xRy si y sólo si $x,y \in S_i$ se tiene que la relación R es una relación de equivalencia.

■

Sean R y R' dos relaciones de equivalencia definidas en el conjunto A; Si $\forall a,b \in A \ \ (aRb \Rightarrow aR'b)$ se dice que la **relación** R está **contenida en la relación** R'

Lema B.3. Sean R y R' relaciones de equivalencia definidas en A. Si $\forall a,b \in A \ \ (aRb \Rightarrow aR'b)$ entonces la partición formada por las clases de equivalencia de R es más fina que la clase de equivalencia de R'

■

B.2.2 Relación de Orden

Una relación de **orden parcial** definida sobre un conjunto es una relación binaria que es antisimétrica y transitiva. Es parcial porqué no todos los elementos del dominio son comparables. Las relaciones de orden parcial son de suma importancia en el estudio de concurrencia y paralelismo en base de datos, sistemas distribuidos y en pruebas de terminación de programas. Aquí se está especialmente interesado en el uso de ordenes parciales para modelar ciertos programas paralelos y como fundamento para construir calendarizadores.

Dos tipos de orden parcial que es convenientemente distinguir son: **orden parcial reflexivo** y **orden parcial irreflexivo**. Una relación es un **orden parcial reflexivo** si y sólo si es reflexiva, antisimétrica y transitiva. Una relación es un **orden parcial irreflexivo** si y sólo si es irreflexiva y transitiva. Una relación de orden parcial reflexiva también se llama relación de **orden parcial débil** y una de orden parcial irreflexiva se dice de **orden parcial estricto**. Note que si una relación es irreflexiva y transitiva entonces es antisimétrica y que existen relaciones de orden parcial que no son ni irreflexivas ni reflexivas. Un **orden total** es un orden parcial reflexivo donde ocurre que, todos los elementos de S son **comparables** $x, y \in S$ son comparables si sucede que $xRy \vee yRx \vee x = y$ En un orden parcial no todos los elementos de S son comparables.

Ejemplo B.3. A continuación se define una relación entre vectores de dimensión n. Sean los vectores $\boldsymbol{i} = (i_1,...,i_n)$ y $\boldsymbol{j} = (j_1,...,j_n)$, la relación $\boldsymbol{i} \prec \boldsymbol{j}$ se define como:

$$\boldsymbol{i} \prec \boldsymbol{j} \Leftrightarrow i_k < j_k \text{ para } k = 1,...,n$$

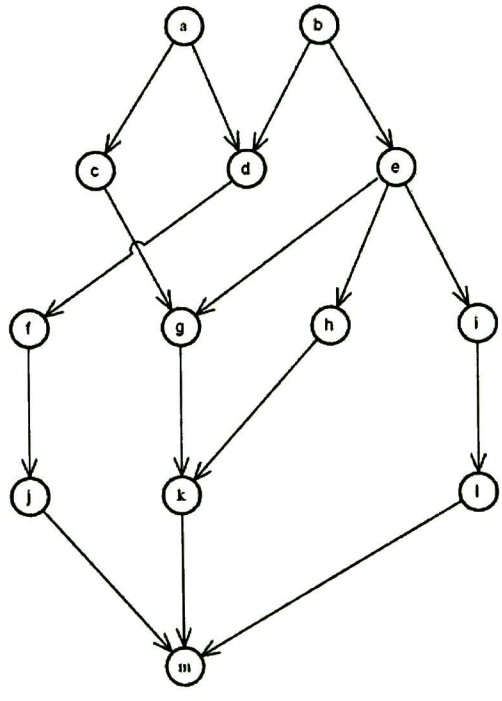
La relación es un orden parcial y no es reflexiva. De manera similar se pueden definir $\preceq$, $\succ$ y $\succeq$.

■

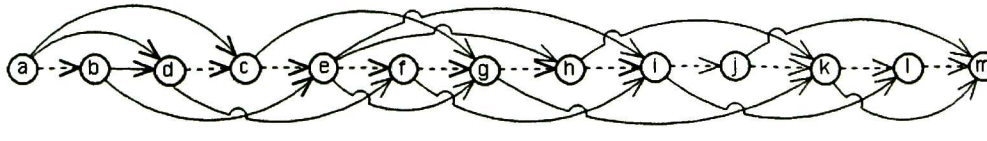
Sea R **una relación de orden parcial en el conjunto** S, un conjunto parcialmente ordenado, llamado también **copo**, es la pareja ordenada (S,R). Un copo es estricto o débil si la relación que lo define es respectivamente estricta o débil.

La **clasificación topológica** de un copo finito irreflexivo (S,R) es una relación de orden total, R_T, en el conjunto S tal que si xRy entonces xR_Ty. Observe que la clasificación topológica de un copo irreflexivo preserva el orden parcial del copo, es decir, que la clasificación topológica es una extensión del orden parcial del copo. Lo anterior se expresa diciendo que el orden parcial está **embebido** en el orden total. Notamos que al clasificar topológicamente los vértices V de un conjunto sobre el que se define la relación R implícitamente se construye una biyección $\tau: V \to \{0,1,\ldots,|V|-1\}$ tal que para todo uRv se satisface que $\tau(u) < \tau(v)$. No se requiere que la relación R sea transitiva. Se dice que un copo irreflexivo se clasifica topológicamente cuando se obtiene una clasificación topológica de él. Dado un copo irreflexivo, su orden topológico en general no es único. Cuando los elementos del conjunto de un copo irreflexivo son operaciones que se requieren realizar y la relación de orden es una relación de precedencia, un orden topológico de la relación proporciona una manera de ejecutar secuencialmente las operaciones sin violar las relaciones de precedencia. Por otro lado las operaciones del copo estricto que no son comparables se pueden ejecutar en paralelo. Por ahora suponemos que las operaciones toman un tiempo unitario en ejecutarse.

Ejemplo B.4. Se ilustra el concepto de clasificación topológica. Consideramos el grafo de la figura b.2 que muestra dos grafos, el segundo es una clasificación topológica del primero. El segundo grafo se ha dibujado tal que todos están colocados horizontalmente. Note que todos los arcos van dirigidos de izquierda a derecha. Los arcos punteados se le agregan al primer grafo para obtener el segundo.



a) Grafo dirigido.



b) Orden topológico del grafo.

Figura B.2: Grafo y su orden topológico.

■

Sea $G=(V,E)$ un grafo dirigido. Una **trayectoria** de G es una secuencia $e_0,e_1,...,e_k$, $k\geq 0$ tal que $e_i\in V$ y $(e_i,e_{i+1})\in E$, $i=0,\ k-1$. k es la **longitud de la trayectoria**. La trayectoria más larga también se llama **trayectoria crítica**. Un grafo puede tener más de una trayectoria crítica. El **tamaño de una trayectoria** es igual a su longitud más uno. Un **ciclo** es una trayectoria que se inicia y termina en el mismo nodo. Una **trayectoria simple** es una trayectoria que no tiene vértices repetidos. Un grafo dirigido sin ciclos se llama **grafo dirigido acíclico** o **dag**. De la definición se tiene que un dag es una relación irreflexiva. Más adelante usamos los dags como modelos de programas. Sea el grafo dirigido $G=(V,E)$ su **grafo inverso** es el grafo $G^{-1}=\left(V,\left\{(u,v)\middle|(v,u)\in E\right\}\right)$ Es decir, que G^{-1} se obtiene de G invirtiendo la dirección de sus aristas.

Lema B.4. El grafo de una relación de orden parcial estricto es un grafo dirigido acíclico.

Demostración. Supongamos que el grafo de la relación R de orden parcial estricto tiene un ciclo el cual denotamos como $e_0e_1,...,e_ke_0$. De la propiedad de transitividad se tiene que e_0Re_k, e_kRe_0 y e_0Re_0, lo cual viola la propiedad de irreflexibilidad y es una contradicción.

■

En general un grafo dirigido acíclico no es el grafo de una relación de orden parcial estricto, para que esto suceda el grafo debe ser transitivo. Así se tiene:

Lema B.5. Un grafo dirigido acíclico y transitivo es el grafo de una relación de orden parcial estricto.

Demostración. Por hipótesis el grafo dirigido acíclico es irreflexivo y transitivo así que sólo resta demostrar que es antisimétrico. Supongamos que aRb, bRa y $a\neq b$, es decir, no es antisimétrica. De aquí se sigue que hay una trayectoria de a hacia b y una de b hacia a, estas dos trayectorias forman un ciclo lo cual está en contradicción con el hecho de que el grafo es acíclico. Es decir, que el grafo es antisimétrico.

■

De los resultados anteriores se tiene que la cerradura transitiva de un dag es una relación de orden parcial estricto. De ahora en adelante todos los dags que consideramos se suponen transitivos y no lo enunciamos explícitamente. En las figuras no dibujamos los arcos que son una consecuencia de la transitividad de la relación. El grafo de una relación de orden total es acíclico y se ve como una recta. Dado un dag (S,R) y $x\in S$.

$P_r(x)=\{y|yRx \text{ y } x\neq y\}$ y $S_u(x)=\{y|xRy \text{ y } x\neq y\}$ se llaman respectivamente el conjunto de los **predecesores** de x y el conjunto de los **sucesores** de x. Si yRx y no existe w tal que yRw y wRx entonces y es un predecesor inmediato de x y x un inmediato sucesor de y. x es elemento **minimal** de (S,R) si $\forall y\in R$ se tiene que xRy o x y y no son comparables, es decir, x es minimal si no existe $z\in S$ tal que zRx y $z\neq x$. Otra manera de expresar lo mismo es: x es elemento minimal si $P_r(x)=\varnothing$. **Nodo de entrada** o **nodo superior** son otros nombres que se usan para designar un elemento minimal. x es un **elemento maximal** de (S,R) si $\forall y$ se tiene que yRx o x y y no son comparables, es decir, que x es maximal si no existe z tal que xRz y $z\neq x$. Otra manera de expresar lo mismo es: x es un elemento maximal si $S_u(x)=\varnothing$. **Nodo de salida** o **nodo inferior** son otros nombres para designar un elemento maximal.

Un copo puede o no tener maximales y minimales y si existen ambos pueden ser múltiples. Se puede dar el caso que un minimal también es máximal. Si la relación de orden de un copo es total a lo máximo se puede tener un máximal y un minimal.

Lema B.6. Todo dag finito $G=(S,R)$ tiene al menos un elemento minimal y un elemento maximal.

Demostración. Si existe un $x\in S$ tal que $P_r(x)=\varnothing$ entonces x es un elemento minimal. Sea x tal que $P_r(x)\neq\varnothing$, si yRx por la propiedad de transitividad de R se tiene que $P_r(y)\subseteq P_r(x)$. Dado que R es antisimétrica se tiene que de yRx y $x\neq y$ entonces $x\notin P_r(y)$ y por lo tanto $P_r(y)\subset P_r(x)$. Si $P_r(x)\neq\varnothing$ entonces existe y tal que $P_r(y)\subset P_r(x)$ y dado que S es finito se tiene que $|P_r(y)|<|P_r(x)|$. Del principio del buen ordenamiento existe $z\in S$ tal que $P_r(z)$ tiene tamaño mínimo. No existe $P_r(y)$ tal que $|P_r(y)|<|P_r(z)|$ lo cual implica que z es un elemento minimal.

■

Lema B.7. Sea $G=(S,R)$ un dag finito. Toda trayectoria crítica μ_C de G empieza en un nodo minimal y termina en un nodo maximal. La demostración del lema b.7 por contradicción es directa.

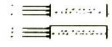
■

x es un elemento **mínimo** del copo (S,R) si para todo $y\in S$, $x\neq y$ se tiene xRy. x es un elemento **máximo** del copo (S,R) si para todo $y\in S$, $x\neq y$ se tiene que yRx.

Teorema B.1. Un grafo dirigido finito $G=(S,R)$ es acíclico si y sólo si existe al menos una clasificación topológica de él.

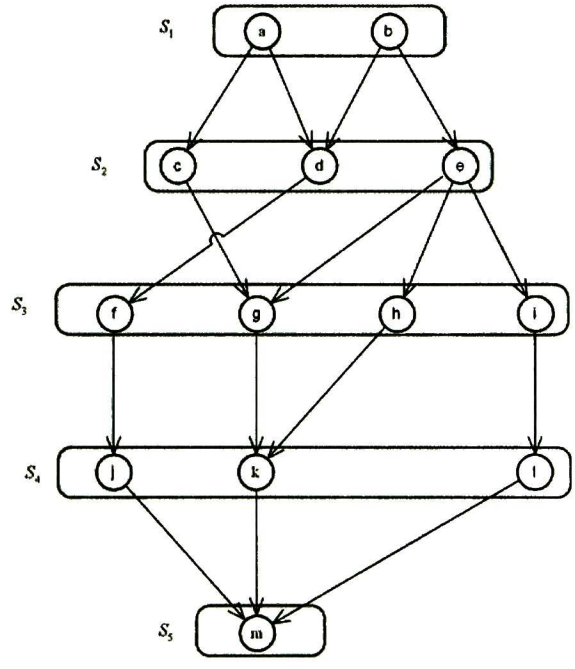
Demostración. Inicialmente demostramos la condición de suficiencia. Es por inducción en, n, el número de elementos del conjunto S. Para $n=1$ la clasificación topológica de S es la relación de orden original. Ahora suponemos que la hipótesis es cierta para n y efectúamos la demostración para $n+1$. Dado que G es acíclico existe al menos un elemento minimal de S, sea s_0 uno de éstos y definamos el conjunto $S' = S-\{s_0\}$. Por hipótesis para S' existe un orden topológico, el cual denotamos por R' Ahora definimos $aR''b$ si y sólo si $aR'b$ o $a=s_0$. R'' es una extensión de R. R'' como se puede comprobar es un orden topológico. Es decir, la hipótesis es cierta para conjuntos con $n+1$ elementos. Ahora demostramos la condición de necesidad y es por contradicción. Suponemos que G es un grafo cíclico. Sea $o_0,o_1,o_2,\ldots,o_k,o_0$ un ciclo de G En un ordenamiento topológico o_{i+1} aparece después de o_i, $i=0,\ldots,k-1$, esto se debe a que o_iRo_{i+1}. Lo anterior implica que o_0Ro_k, es decir, que en un ordenamiento topológico o_0 precede a o_k, también se tiene que o_kRo_0, es decir, que en el mismo ordenamiento o_k precede a o_0 lo que es una contradicción.

■

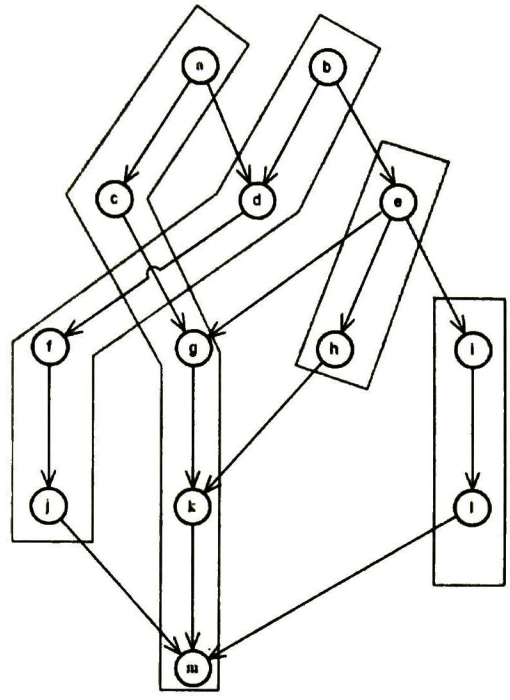
Notamos que la primera parte de la demostración puede ser usada para implementar un algoritmo recursivo que encuentra un orden topológico a partir de un orden parcial irreflexivo de un copo irreflexivo. Un subconjunto C del conjunto S del copo irreflexivo (S,R) es una **cadena** si para $u,v\in C$ se tiene que u y v son comparables, es decir, todos los elementos de una cadena son comparables. Más formalmente a_i, $i=1,\ldots,k$ es una cadena de (S,R) si $a_i\in S$ y $a_i\neq a_j$, $a_i<a_j$ para  . Note que una cadena es una trayectoria del copo. La restricción de la relación R a los elementos de la cadena C es un orden total. Los nodos de cualquier trayectoria en el copo irreflexivo constituyen una cadena. Dada una trayectoria en un copo irreflexivo, una cadena puede contener un subconjunto de los nodos de la trayectoria, el subconjunto no es necesariamente propio. Un subconjunto  del conjunto S del copo irreflexivo (S,R) es una **anticadena** si para $u,v\in C$ se tiene que u y v no son comparables, es decir, los elementos de una cadena no son comparables. La cardinalidad de una cadena o anticadena se llama su **longitud**. Una **cadena**, o **anticadena**, de un conjunto es **maximal** si no es subconjunto propio de otra cadena, o anticadena, del conjunto. El **ancho de un grafo** G es la longitud de su anticadena más grande, el **largo de un grafo** G es la longitud de su cadena más grande. Si interpretamos los nodos como operaciones y la relación R como una relación de precedencia entre ellas entonces, el tiempo más corto de ejecución tiene al menos el

largo del grafo. Si el tiempo de ejecución fuése menor que la longitud de una cadena entonces dos operaciones que están en relación de precedencia se tendrían que ejecutar simultáneamente violando así el orden de la relación. El máximo número de procesadores que se necesitan para ejecutar las operaciones de un grafo no exceden su ancho.

Ejemplo B.5. Los siguientes grafos muestran las anticadenas maximales de un grafo y algunas cadenas del mismo. $S_1 = \{a,b\}$, $S_2 = \{c,d,e\}$, $S_3 = \{f,g,h,i\}$, $S_4 = \{j,k,l\}$ y $S_5 = \{m\}$ son anticadenas maximales y son una partición del conjunto de vértices del grafo. $S_1' = \{a,c,g,k,m\}$ es una cadena maximal y $S_2' = \{b,d,f,j\}$, $S_3' = \{e,h\}$ y $S_1' = \{i,l\}$ son cadenas. Las cadenas son una partición del conjunto de vértices del grafo. El ancho del grafo es cuatro y el largo es cinco.



a) Anticadenas.



b) Cadenas.

Figura B.3: Anticadenas y cadenas de un grafo.

■

A continuación mostramos que las anticadenas de un conjunto parcialmente ordenado no vacío y finito constituyen una partición del conjunto. Si los nodos representan operaciones entonces todas las operaciones de una anticadena se pueden ejecutar en paralelo.

Teorema B.2 (de la descomposición en anticadenas). Sea S un conjunto no vacío y finito que es parcialmente ordenado por la relación irreflexiva R. Existe una partición $S_1, S_2, ..., S_n$ de S donde n es el largo del grafo (S,R) las S_i satisfacen que:

i. cada S_i es una anticadena maximal, $i=1, ..., n$,

ii. para $1 \le i < j \le n$, no existen elementos en S_i con un predecesor en S_j,

iii. para $1 < i \le n$, todo elemento en S_i tiene al menos un predecesor inmediato en S_{i-1}.

Demostración. Sea S_1 el conjunto de elementos minimales de S, S_2 el conjunto de elementos minimales de $S - S_1$ y así para los otros S_i. Por construcción es directo que los S_i, $i-1, ..., n$ cumplen con la propiedad *ii*. Ahora demostramos por contradicción que S_i cumple con la propiedad *iii*. Si un elemento de S_j no tiene predecesor en S_{j-1} entonces por construcción el elemento debe pertenecer a uno de S_k para $k \le j-1$ lo cual contradice la hipótesis de que pertenece S_j. Ahora demostramos que los elementos S_j son una anticadena, sean s y $t \in S_j$ y s y t son comparables tal que se cumple sRt, por lo tanto s y t no pueden ser elementos minimales de S_j en contradicción con la hipótesis de que S_j está formado por elementos minimales. Ahora de la construcción se sigue que las S_i son conjuntos maximales. Note que por construcción S_1 no tiene predecesores en S y los predecesores de S_i están en $S_1 \cup ... \cup S_{i-1}$.
Otra manera de definir S_i es:

$$S_i = \{x \in S \mid \text{una trayectoria de tamaño } i \text{ termina en } x\}$$

■

De los conjuntos S_i se pueden construir una clasificación topológica como sigue: los elementos de S_i preceden los de S_j si $i < j$ y los elementos de S_i se ordenan arbitrariamente.

Sea el dag $G=(S,R)$. El conjunto S es un conjunto de operaciones y R es una relación de precedencia entre operaciones, también corresponde a una trayectoria de datos entre las operaciones. Este dag modela el comportamiento de un secuenciador que

controla múltiples unidades funcionales o elementos procesadores. Una clasificación topológica proporciona una ejecución secuencial de las operaciones del dag. El problema de decidir a que tiempo se ejecuta cada operación respetando las restricciones del orden parcial se llama **calendarización de operaciones**. La clasificación topológica es un **calendarizador secuencial** de las operaciones donde se tiene un solo elemento procesador y se ejecuta una sola operación a la vez. Un calendarizador que logra el tiempo más corto de ejecución y el máximo paralelismo, es el **calendarizador libre**. El calendarizador libre despacha como sigue:

```
while(S ≠ ∅)
        {
          ejecuta los elementos minimales de S;
          S = S − elementos minimales de S;
        }
```

Listado B.1: Calendarizador libre.

El calendarizador libre supone que no hay restricción en el número de elementos procesadores. Notamos que para el S inicial, los elementos iniciales de S vienen dados por la anticadena S_1 del teorema b.2; de la misma manera los elementos minimales de $S - S_{i-1} - S_{i-2} - \ldots - S_1$ viene dado por la anticadena S_i. Es decir, que el programa calendariza las operaciones de la anticadena S_i al tiempo $i-1$. En situaciones reales puede suceder que no se tenga el número de elementos procesadores que requiere el calendarizador libre. En este caso se requiere diseñar un calendarizador que sólo use los elementos procesadores disponibles y que tenga el tiempo de ejecución de las operaciones más corto. El diseño del calendarizador es equivalente a encontrar una relación de equivalencia R' donde R' es una extensión de R. Para la ejecución de (S, R') se requiere a lo más los elementos procesadores existentes.

Corolario B.1. El tiempo necesario para ejecutar todas las operaciones de $G = (S, R)$ un copo estricto es igual a la longitud de la trayectoria más larga de G. todo bajo el supuesto que se tiene un número no limitado de procesadores.

■

En la demostración del *teorema de la descomposición en anticadenas* se muestra un algoritmo, llamado **calendarizador libre**, que nos permite encontrar una secuencia de anticadenas maximales que constituyen una partición de un conjunto parcialmente ordenado y que no se requiera más que n pasos del calendarizador libre si la trayectoria crítica tiene tamaño n. Todos los elementos de una anticadena se despachan a ejecución al mismo tiempo. El calendarizador así construido despacha una operación tan pronto como es posible, esto es en cuanto las operaciones predecesoras de la operación han terminado su ejecución.

Bibliografía

[1] Aart, J.C. and Harry, A.G., Wikshoff, *Implementation of Fourier-Motzkin Elimination.* Reporte Tecnico, High Performance Computing Division, Department of Computer Science, Universidad de Leiden, 1994.

[2] Aho, Alfred V., Lam Monica S., Sethi, R., Ullman Jeffrey D., *Compilers: Principles, Techniques, & Tools.* Second Edition, Addison Wesley, U.S.A., 2007.

[3] Banerjee, U., *Loop Transformation for Restructuring Compilers: The Foundations.* Intel Corporation, Ed. Kluwer Academic Publishers, U.S.A., 1993.

[4] Bruno, J., Coffman, E., Jr., and Sethi, R., *Scheduling Independent Tasks to Reduce Mean Finishing Time.* Commun. ACM 17, 7, 382-387. 1974.

[5] Darte, A., and Y. Robert, *Scheduling and Automatic Parallelization.* Ed. Birkhäuser, New York, U.S.A., 2000.

[6] Dutta, H., Hanning, F. and Teich, J., *Controller Synthesis for Mapping Partitioned Programs on Array Architectures.* Department of Computer Science 12, Hardware-Software-Co-Design, University of Erlangen-Nuremberg, Am Weichselgarten, 3 D-91058 Erlangen, Germany. Co-Design-Report 03-2005, June 18, 2005.

[7] Eager, Derek L., Zahorjan, J., Lazowska Edward D., *Speedup Versus Efficiency in Parallel Systems.* IEEE Transactions on Computers, Vol. 38, No. 3, March 1989.

[8] Feautrier, P., *Automatic Parallelization in the Polytope Model.* In G.-R. Perrin and A. Darte, eds., The Data Parallel Programing Model, Vol. 1132 of *LNCS*, pages 79-103. Springer-Verlag, June 1996.

[9] Garey, Michael R., and Johnson, David S., *Computer and Intractibility: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[10] Gomez, R., Castillo, Alejandro A., and Torres, D., *Control Generation for QR Decomposition based on the Polytope Model.* The 6th International Conference Electrical Engineering, Computing Science and Automatic Control: CCE 2009. CINVESTAV, Toluca, México, Noviembre, 2009.

[11] González, Raúl E., *Notas de Algebra Moderna.* Departamento de Matemáticas y Física ITESO, Guadalajara, México, 1997.

[12] Guzmán, Manuel E., *Introducción al Paralelismo de Datos.* Notas CINVESTAV, Guadalajara, México, 2008.

[13] Hen, Y., *CORDIC-Based VLSI Architectures for Digital Signal Processing.* IEEE Signal Processing Magazine, July, 1992.

[14] Jimenez, M., *Multilevel Tiling for Non-Rectangular Iteration Spaces.* P.hD. Tesis, Universidad Politecnica de Catalunya, 1999, pp. 46-58.

[15] Kim, D., and Rajopadhye, Sanjay V., *An Improved Systolic Architecture for LU Decomposition.* Application-specific Systems, Architectures and Processors (ASAP'06), IEEE Computer Society, 2006.

[16] Kung, Sun Y., *VLSI Array Processors.* Ed. Prentice Hall, New Jersey, U.S.A., 1988.

[17] Kwok, Y., and Ahmad, I., *Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors.* ACM Computing Surveys, Vol. 31, No. 4, December 1999.

[18] Lenguager, C., *Loop Parallelization in the Polytope Model.* In E. Best, ed., *CONCUR'93*, LNCS 715, pages 398-416. Springer-Verlag, 1993.

[19] Liu, Z., LightBody, G., Walke, R., Hu, Y., and McCanny J., *Generic Scheduling Methods for a Linear QR Array SoC Processor.* DSiP Laboratory, School of Electrical and Electronics Engineering, The Queen's University of Belfast, Belfast BT 9 5AH, N. Ireland, 2001.

[20] Ma, L., Dickson, K., McAllister, J., and McCanny, J., *Modified Givens Rotations and their Application to Matrix Inversion.* ICASSP 2008, Institute of Electronics, Communications and Information Technology, Queen's University, Belfast, 2008.

[21] Maltsev, A., Pestretsov, V., Maslennikov, R., and Khoryaev, A., *Triangular Systolic Array with Reduced Latency for QR-decomposition of Complex Matrices.* ISCAS 2006, Broadband Wireless Division, Intel Corporation, Nizhny Novgorod, Russia, 2006.

[22] Martínez, Manuel O., *Herramienta para Visualización de Grafos de Dependencia en Matlab.* Borrador de Tesis de Maestría, CINVESTAV, Guadalajara, México, 2009.

[23] Moldovan, Dan I., *Parallel Processing, From Applications to Systems.* Ed. Morgan Kaufmann Publishers Inc., 1993.

[24] Moreno, Jaime, H., *Matrix Computations on Systolic Type.* Kluwer Academic Publisher, U.S.A., 1992.

[25] Myllylä, M., Hintikka, J., Cavallaro, Joseph R., and Juntti M., *Complexity Analysis of MMSE Detector Architectures for MIMO OFDM Systems.* P.O. Box 4500, FIN-90014 University of Oulu, Finland, 2005.

[26] Parhi, Keshab K., *VLSI Digital Signal Processing Systems.* Ed. John Wiley & Sons, New York, U.S.A., 1999.

[27] Petkov, N., *Systolic Parallel Processing.* Ed. North-Holland, Vol. 5, Netherlands, 1993.

[28] Pizano, Jose L., *Igualador de Canal para Sistemas en 60Ghz.* Borrador de Tesis de Maestría, CINVESTAV, Guadalajara, México, 2009.

[29] Ravindran, K., *Task Allocation and Scheduling of Concurrent Applications to Multiprocessor Systems*. Ph.D. Thesis, Electrical Engineering and Computer Sciences University at Berkeley, 2007.

[30] Rodríguez, R., *Modelo de un sistema de comunicaciones WiMax*. Borrador de Tesis de Maestría, CINVESTAV, Guadalajara, México, 2009.

[31] Sih, Gilbert C., *Multiprocessor Scheduling to Account for Interprocessor Communication*. Ph.D. Thesis, Electronics Research Laboratory. College of Engineering University of California, Berkeley, April, 1991.

[32] Sinnen, O., Sousa, Leonel A., Sandnes, Frode E., *Toward a Realistic Task Scheduling Model*. IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 3, March 2006.

[33] Song, Siang, W., *Systolic Algorithms: Concepts, Synthesis y Evolution*. University of Sao Paulo Institute of Mathematics and Statistics, Department of Computer Science, Brazil.

[34] Wolfe, M., *Advanced loop interchange*. In Proceedings of the 1986 International Conference on Parallel Processing, August 1986.

[35] Wolfe, M., and Lam, M., *A Loop Transformation Theory and an Algorithm to Maximize Parallelism*. IEEE Transactions on Parallel and Distributed System, Vol. 2, No. 4, pp. 452-471, October 1991.

[36] Wolfe, M., *High Performance Compilers for Parallel Computing*. Addison-Wesley Publishing Company, 1996.

[37] Wolfe, M., *More Iteration Space Tiling*. International Conference on Supercomputing, pp. 655-664, 1989.

[38] Wolfe, M., *Optimizing Supercompilers for Supercomputers*. Ph.D. Thesis, University of Illinois, October 1982.

[39] Xue, J., *On Tiling as a Loop Transformation*. University of New England, Department of Mathematics, Statistics and Computing Science, University of New England, Parallel Processing Letters, 7(4):409-424, 1997.

[40] Xue, J., *Unimodular Transformation of Non-Perfectly Nested Loops*. University of New England, Department of Mathematics, Statistics and Computing Science, University of New England, 1996.

[41] Yaacoby, Y., and Cappello P., *Bounded Broadcast in Systolic Arrays*. International Journal of High Speed Computing, 6(2):223-237, 1994.

[42] Yang, T., *Scheduling and Code Generation for Parallel Architectures*. PhD. Thesis, Computer Science, The State University of New Jersey, May, 1993.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Transformación del Algoritmo Matricial que Efectúa la
Descomposición QR de su Forma Secuencial a su Forma Paralela

del (la) C.

Ricardo GOMEZ KU

el día 17 de Diciembre de 2009.

Dr. Yuriy Shkvarko Sosnoff
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Deni Librado Torres Román
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Ramón Parra Michel
Investigador CINVESTAV 2C
CINVESTAV Unidad Guadalajara

Dr. Mariano Aguirre Hernández
Científico Investigador
Intel Tecnología de Mexico S. A. de C.V.