UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica y Computación

ALGORITMOS DIVIDE–Y–VENCERÁS EN MALLAS DE PROCESADORES

Margarita Amor López

Diciembre, 1997

A Kiku

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas e instituciones que de alguna forma me han ayudado en este trabajo:

A los profesores D. Francisco Argüello Pedreira y D. Juan López Gómez, directores de esta tesis, por su trabajo de dirección y ayuda, así como por la confianza que han depositado en mi trabajo. En especial a D. Francisco Argüello Pedreira por su constante apoyo y su orientación en mi labor investigadora.

A mis compañeros del Departamento de Electrónica y Computación por el apoyo que me han brindado en todo momento. En especial a los integrantes del Grupo de Paralelismo y Arquitecturas Avanzadas de dicho departamento por sus sugerencias y ayuda en ciertas partes de este trabajo. En especial a Montse Bóo, amiga y constante apoyo, a María Martín, por su amistad y comprensión, a Elisardo Antelo por su amistad desinteresada, y a Vicente Blanco, por su labor como gestor de sistemas.

También quisiera mencionar a Paulo Félix por su amistad y comprensión durante estos últimos años.

A Kiku, por su sincero cariño y paciencia.

A la Xunta de Galicia, por el soporte económico durante la realización de este trabajo mediante la concesión de una Beca Predoctoral. A la Xunta de Galicia por la ayuda económica recibida a través del proyecto XUGA20606B93, XUGA20605B96; y a la Comisión Asesora de Investigación Científica y Técnica por el soporte económico a través de los proyectos TIC92-0942-C03, TIC96-1125-C03.

Por último, pero no menos importante, a mis nuevos compañeros del Departamento de Electrónica y Sistemas de la Facultad de Informática de La Coruña por la acogida dispensada.

Índice General

Índice de Tablas

Índice de Figuras

Resumen

En esta memoria se proponen varias técnicas para particionar y proyectar algoritmos *divide-y-vencerás* sobre computadores paralelos de topología malla y memoria distribuida. El trabajo parte de la observación de que durante la evaluación de un algoritmo sobre un multicomputador es necesario redistribuir los datos entre los procesadores en una gran variedad de formas, tanto regulares como irregulares. En general, esto implica complejos movimientos de datos, cuya visualización puede ser bastante difícil.

Comenzaremos considerando algoritmos divide-y-vencerás regulares. Para particionar y proyectar estos algoritmos sobre el multicomputador utilizamos una combinación de dos técnicas: la *proyección vector* y la *permutación índice–dígito*. Estas técnicas nos permiten formular de forma precisa el flujo de datos de los algoritmos y, en muchos casos, realizar importantes simplificaciones.

Como ejemplos concretos de esta clase de algoritmos consideraremos las versiones más importantes de la transformada rápida de Fourier y los algoritmos de resolución de sistemas tridiagonales más significativos de entre los propuestos en la bibliografía reciente. Los primeros presentan un flujo de datos divide-y-vencerás estándar, aunque incluyen la permutación dígito–inverso, mientras que el flujo de datos de los segundos es más variado y puede clasificarse en: normal, extendido, árbol y árbol extendido. Las técnicas propuestas nos permiten realizar una descripción unificada de todos estos algoritmos y una implementación paralela eficiente basada en subrutinas que son traducciones de las correspondientes permutaciones índice–dígito.

También abordamos un problema de gran interés tanto para la programación paralela como para la implementación VLSI: la proyección de árboles r-arios completos sobre las topologías array lineal y malla. En esta memoria presentamos una nueva metodología para realizar esta proyección que puede ser una alternativa a las técnicas usuales, basadas en árbol en "H" o en baldosas (*tiles*). En todos los casos, la distribución de los nodos del árbol entre los

procesadores es balanceada y las comunicaciones son sólo entre procesadores vecinos.

Por último, abordaremos la partición y proyección de algoritmos divide-y-vencerás irregulares sobre multicomputadores de topología malla. Tomamos como ejemplo el algoritmo Barnes–Hut del problema de los N cuerpos. A las técnicas anteriormente empleadas añadiremos la del *llenado del espacio a través de una curva* (*space–filling curve*), que nos permite mantener la localidad de los datos y reducir las comunicaciones.

El trabajo se enmarca en la línea de investigación "Proyección de Algoritmos en Arquitecturas Multiprocesador"del grupo de Arquitecturas Avanzadas de la Universidad de Santiago (Departamento de Electrónica y Computación de la Facultad de Física[1]). Otra línea general del grupo es el diseño VLSI de arquitecturas de aplicación específica en el campo del reconocimiento de formas y procesamiento de imágenes.

[1]Investigación finaciada a través de los proyectos concedidos por la CICYT, TIC92-0942-C03-03:*Computación masivamente paralela: diseño de arquitecturas VLSI*, TIC-96-1125-C03-02:*Diseño de algoritmos numéricos irregulares sobre arquitecturas masivamente paralelas*; y por la Xunta de Galicia, XUGA20606B93:*Diseño de CIAES y herramientas para la paralelización automática de algoritmos numéricos* y XUGA20605B96:*Diseño de arquitecturas y paralelizadores en computación masivamente paralela: y/o basadas en matrices dispersas.*

Capítulo 1

Computadores de topología malla con memoria distribuida

Un sistema multiprocesador es un computador compuesto por un conjunto de procesadores (PEs) que pueden comunicarse a través de una red de interconexión . El objetivo de estos sistemas es que los PEs trabajen concurrentemente para resolver un determinado problema de manera que se consiga una mayor velocidad.

En los últimos años, los computadores vectoriales y escalares convencionales fueron los dominantes en el campo de la computación. Últimamente, los sistemas multiprocesadores han dejado el campo de la investigación para introducirse en el campo comercial. Es indicativo que las grandes empresas de desarrollo de la computación, en concreto, IBM Corporation y Cray Research, hayan introducido en el mercado máquinas paralelas en respuesta a la demanda de sus clientes.

1.1 Introducción

Los dos tipos de arquitecturas paralelas más extendidas son los computadores SIMD (Simple flujo de Instrucciones, Múltiple flujo de Datos) y los computadores MIMD (Múltiple flujo de Instrucciones, Múltiple flujo de Datos) [37, 61]. En un computador SIMD todos los PEs ejecutan el mismo programa sobre diferentes conjuntos de datos bajo la supervisión de una unidad central. Los computadores MIMD se diferencian de los anteriores en que los PEs ejecutan distintos programas sobre distintos conjuntos de datos.

El factor más utilizado para clasificar los computadores MIMD es el método utilizado para compartir la información entre los PEs. Los multicomputadores de *memoria compartida* ofrecen al usuario una memoria global a la que todos los PEs puede acceder. Los PEs se comunican a través de variables compartidas en memoria, con instrucciones de carga y almacenamiento capaces de acceder a cualquier posición de memoria. La memoria está estructurada en módulos. En estos sistemas se pueden producir conflictos de memoria cuando varios PEs pretenden acceder al mismo módulo, lo que puede provocar una pérdida significativa del rendimiento. Estos conflictos se gestionan por una serie de técnicas, y se pueden disminuir incorporando caches o memorias locales. Ejemplos de estos sistemas son los servidores multiprocesador como SGI-Power Challenge o SUN Enterprise.

En los sistemas de memoria compartida también podemos distinguir entre multiprocesamiento simétrico, donde todos los procesadores tienen igual capacidad para ejecutar programas, ya sean de usuario, del sistema operativo o subrutinas de entrada/salida, y los de procesamiento asimétrico, donde un único procesador o un subconjunto de ellos pueden ejecutar las rutinas del sistema operativo o manejar los procesos de entrada/salida. Actualmente, los sistemas simétricos de memoria compartida *Symmetric Multiprocessing* (SMP) son más utilizados para las aplicaciones comerciales, porque proporcionan un rendimiento alto en la ejecución de aplicaciones ya existentes para sistemas monoprocesador, ya que el efecto de la red es transparente para las actuales aplicaciones. Aún así, todos los vendedores ahora estan presentando sistemas Massively Parallel Processors (MPP) y se han visto forzados a que, por razones tecnológicas, los sistemas por encima de un cierto tamaño sean de memoria distribuida.

En los computadores MIMD con *memoria distribuida* cada PE tiene su propio programa y memoria de datos a los que los demás no pueden acceder directamente. La comunicación de los datos compartidos se hace por pase de mensajes entre los PEs a través de una red de interconexión (paradigma del pase de mensajes).

El tipo de computador que vamos a considerar a lo largo de esta memoria es un computador MIMD con memoria distribuida y topología malla. Los PEs están organizados en una matriz bidimensional. Cada PE, excepto los situados en los bordes, tiene cuatro conexiones con los cuatro PEs vecinos inmediatos. Un toroide es similar, pero los PEs de la primera fila están conectados con los correspondientes PEs de la última fila. Y los PEs de la columna más a la izquierda están conectados con los correspodientes PEs de la columna más a la derecha, formando un toro, ver figura 1.1. Este hecho

no es crucial, pero mejora el rendimiento.

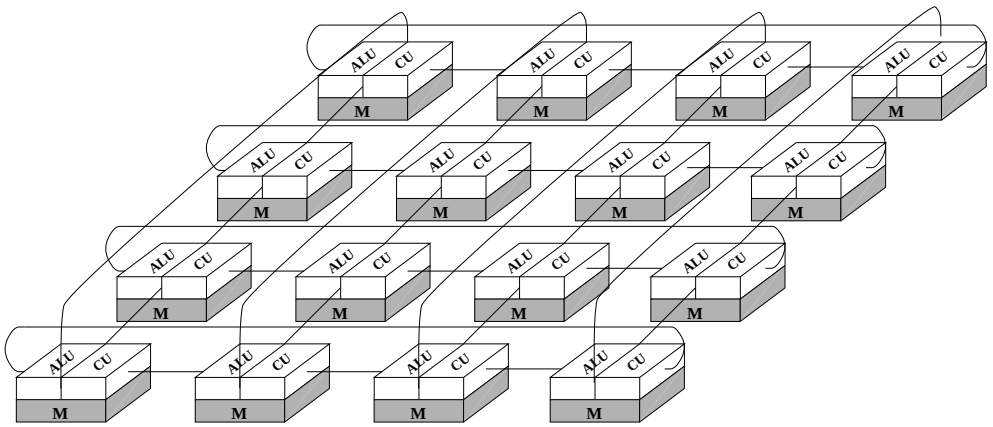


Figura 1.1: Computador de topología toroide de 4×4 PEs.

Cada PE tiene su propia memoria local y procesa sus propios datos. El espacio de los datos almacenados en la memoria local puede verse como una nueva dimensión añadida, resultando una estructura tridimensional. El acceso es aleatorio a lo largo de esta dimensión, pero el acceso en las dos restantes dimensiones implica una operación de comunicación entre dos PEs. Este acceso se realiza siempre que un PE requiera datos remotos y se realiza a través de la red de interconexión.

Una cualidad deseable en los algoritmos paralelos es la escalabilidad, es decir, que las características de dicho algoritmo no dependen del tamaño del sistema ni del tamaño del problema. Por otro lado, se debe conseguir un reparto equilibrado de la carga computacional entre los PEs. También es deseable minimizar el volumen de comunicaciones interprocesador ya que supone un consumo temporal importante, que afectará a la eficiencia de los algoritmos paralelos. En concreto, se debe minimizar la redistribución de datos entre las etapas de un algoritmo, puesto que esto implica un número importante de comunicaciones.

La asignación de datos y computaciones de un problema a los distintos PEs constituye un problema matemático conocido como mapeado (mapping). En nuestro caso el mejor mapeado es el que minimiza el tiempo de ejecución. En general, el tiempo de ejecución de un algoritmo sobre un multicomputador se puede descomponer en dos componentes: el tiempo de computación, t_c, y el tiempo de comunicación (routing), t_{com}. Normalmente, el tamaño del problema es mayor que el número de PEs. En tales casos se tiene que particionar el algoritmo y multiplexar el hardware en el tiempo.

Consideremos, por ejemplo, un problema que puede descomponerse en cuatro procesos; los procesos pueden ejecutarse concurrentemente de dos en dos, pero con respecto al otro par precisan interactuar durante la ejecución. En este caso, cada par de procesos pueden proyectarse en dos PEs diferentes. La ejecución se realizará en paralelo con los procesos 1 y 2 hasta que se precisen datos remotos. En este punto, los datos son enviados de un PE a otro, después de lo cual la ejecución continúa con los procesos 3 y 4, ver figura 1.2.A. En el caso de que los procesos 1 y 2 se mapearan al mismo procesador, se ejecutarían secuencialmente. El tiempo perdido en la comunicación interprocesador puede compensarse mediante el paralelismo, caso de la figura 1.2.B, mientras que esto no sucede en el caso 1.2.A. Esto sugiere que, un análisis previo considerando la relación entre t_c y t_{com}, puede determinar el mapeado que consigue la solución más eficiente para el problema.

En la siguiente sección trataremos algunos conceptos básicos de la teoría de grafos que serán utilizados en el resto del texto. En la sección 1.3 presentaremos el problema del mapeado y su tratamiento en la bibliografía. En las secciones 1.4 y 1.5 veremos la proyección vector y la permutación índice–dígito respectivamente. Usaremos la combinación de estas dos técnicas para mapear los datos a los PEs y definir las diferentes operaciones realizadas en el algoritmo. En la sección 1.6 presentamos los multiprocesadores utilizados en la implementación de los algoritmos y finalmente, en la sección 1.7, una serie de parámetros empleados para medir la efectividad de los algoritmos.

1.2 Conceptos de teoría de grafos

1.2.1 Definiciones básicas

Siguiendo a [97, 52] usamos la siguiente notación.

Definición 1 *Un grafo $G(V,E)$ es una estructura que consiste en un conjunto de nodos $V = \{v_1, v_2, \cdots\}$ y en un conjunto de arcos $E = \{e_1, e_2, \cdots\}$, en donde cada arco $e =< u, v >$ conecta los elementos de un par de nodos $\{u, v\}$, $u, v \in V$, que no tienen que ser necesariamente distintos. u y v son llamados los extremos de e y se dice que u y v son adyacentes.*

Designaremos V_G al conjunto de nodos de G y E_G al conjunto de arcos. En la figura 1.3 se muestra un grafo de 5 nodos y 9 arcos. En este caso,

$$V_G = \{a, b, c, d, e\}$$

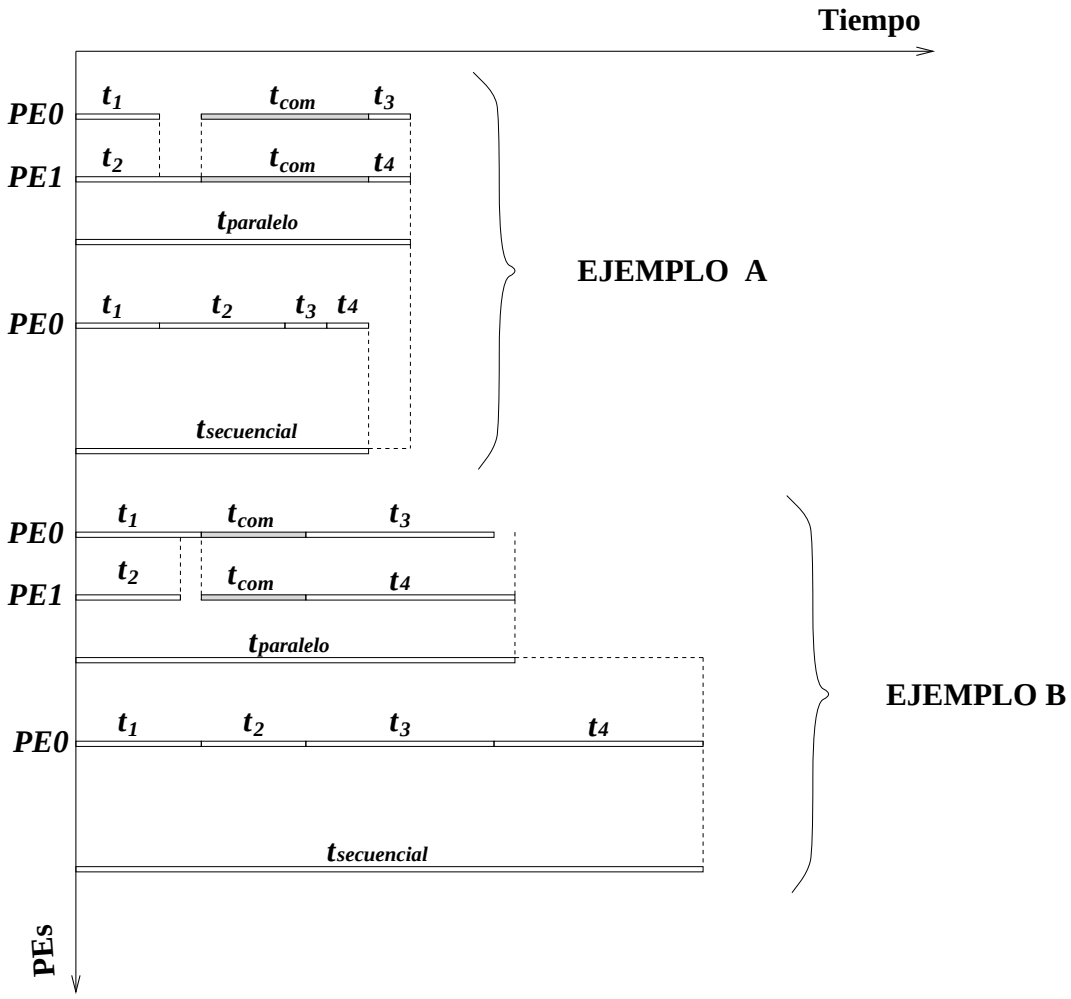


Figura 1.2: Proyección de cuatro tareas sobre dos PEs. En el ejemplo A, la distribución de las tareas entre los dos PEs es peor que la opción secuencial, $t_{paralelo} > t_{secuencial}$. En el ejemplo B, la opción paralela es la mejor, $t_{paralelo} \ll t_{secuencial}$.

y

$$E_G = \{ \quad e_1 =< a, b >, e_2 =< b, c >, e_3 =< c, d >, \\ e_4 =< a, c >, e_5 =< a, e >, e_6 =< c, e >, \\ e_7 =< b, d >, e_8 =< d, c >, e_9 =< e, b > \quad \}.$$

Dos grafos G y H se dicen *isomorfos* si existe una biyección $\theta : V_G \longrightarrow V_H$ tal que $< u, v >\in E_G$ si y sólo si $< \theta(u), \theta(v) >\in E_H$. Y un grafo H es un *subgrafo* de G si $V_H \subseteq V_G$ y $E_H \subseteq E_G$.

El número de nodos en un grafo, es decir, la cardinalidad del conjunto V, se denota usualmente por $|V|$. Análogamente, el número de arcos es la cardinalidad del conjunto E, que denotaremos por $|E|$. El grado del nodo v es el número de arcos de los cuales es extremo, que denotaremos por $d(v)$. $\triangle(G)$ denota el máximo grado de los nodos de G y $\delta(G)$ el mínimo grado de los nodos de G.

Un grafo se dice orientado si en los arcos se define una orientación, es decir, existe un nodo de entrada y un nodo de salida. El *grado de salida* del nodo v, $d_{out}(v)$, es el número de arcos que tienen a v como nodo de inicio; el *grado de entrada*, $d_{in}(v)$, se define similarmente. Claramente, para todos grafos

$$\sum_{i=1}^{|V|} d_{in}(v_i) = \sum_{i=1}^{|V|} d_{out}(v_i). \tag{1.1}$$

En el ejemplo de la figura 1.3 se muestra un grafo $G(V, E)$ donde $|V| = 5$, $|E| = 9$, $\triangle(G) = 5$ y los grados del nodo a, por ejemplo, serían $d(a) = 3$, $d_{in}(a) = 0$ y $d_{out}(a) = 3$.

Una *cadena* en G es una secuencia finita no nula $\Gamma[v_0, v_k] = \{v_o, e_1, v_1, e_2, v_2, \cdots, e_k, v_k\}$, cuyos términos son alternativamente nodos y arcos, tal que, para $1 \leq i \leq k$, los nodos extremos de e_i son v_{i-1} y v_i. Si los arcos $\{e_1, e_2, \cdots, e_k\}$ y los nodos $\{v_o, v_1, \cdots, v_k\}$ de la cadena Γ son distintos, Γ es llamado un *camino*. El entero k es la longitud del camino. Un grafo G, se dice *conectado* si $\forall(u, v) \in V$ existe un camino desde u a v.

A cada nodo $u \in V_G$ puede asociársele un entero $w(u)$, llamado *peso del nodo* u; y a cada arco $< u, v >\in E_G$ puede asociársele un entero $c(< u, v >)$, llamado el *peso del arco* $< u, v >$. Entonces G, junto con los pesos de los nodos y los arcos, es llamado un grafo *con pesos*. El camino mínimo $\Gamma[u, v]$ (obtenido sumando los pesos de los arcos) que une los nodos u y v se llamará la *distancia* entre u y v y será denotada por $d(u, v)$.

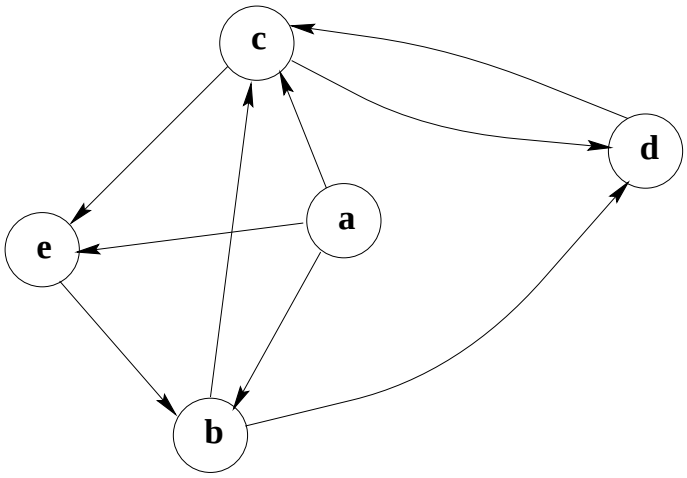


Figura 1.3: Un grafo orientado de 5 nodos y 9 arcos.

Consideremos el grafo con pesos $G(V, E)$ y el entero positivo p. Una *partición* p de V es un conjunto no vacío $\wp = \{V_1, \cdots, V_p\}$ de subconjuntos de V, tal que $\bigcup_{i=1}^{p} V_i = V$. El coste de un determinado subconjunto es la suma de los pesos de todos los nodos que pertenecen a ese subconjunto, $\sum_{i=1}^{|V_i|} w(u)$, $u \in V_i$. El coste de la partición es el máximo de los costes de cada uno de los subconjuntos. Definimos el *grafo cociente* de G con respecto a $\wp$ al grafo $Q_G = (\wp, \varepsilon)$ donde $< V_i, V_j > \in \varepsilon$ si y sólo si $\exists < u, v > \in E$ para algún $u \in V_i$ y $v \in V_j$. En la figura 1.4 mostramos una partición $p = 3$ de G y el grafo cociente asociado. Los subconjuntos son

$$\begin{aligned} V_1 &= \{c, d\} \\ V_2 &= \{e\} \\ V_3 &= \{a, b\} \end{aligned}$$

El grafo cociente asociado $Q_G = (\wp, \varepsilon)$ con respecto a la partición $p = 3$ es $\wp = \{V_1, V_2, V_3\}$ y $\varepsilon = \{< V_1, V_2 >, < V_2, V_3 >, < V_3, V_2 >, < V_3, V_1 >\}$.

Una *proyección* (embedding) de un grafo G en un grafo H es una aplicación inyectiva $\varphi : G \longrightarrow H$ de los nodos de G en los nodos de H y una asignación de cada arco $e =< u, v > \in E_G$ a un camino $\varphi(e)$ cuyos extremos son $\varphi(u)$ y $\varphi(v)$.

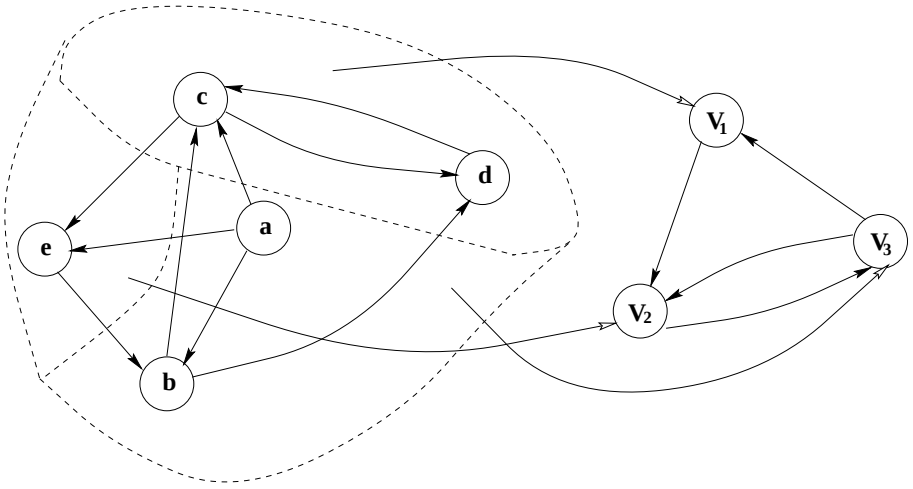


Figura 1.4: Una partición 3 de G y el grafo cociente asociado.

La *dilatación*, $d(\varphi(u), \varphi(v))$, de un arco $< u, v >$ en H es la longitud de camino $(\Gamma(\varphi(u), \varphi(v))$ y la *dilatación de una proyección* φ, denotada por Λ_∞, es

$$\Lambda_\infty = \max(d(\varphi(u), \varphi(v)),\ \forall < u, v > \in E_G. \tag{1.2}$$

La *expansión* de φ se denota por Λ_1,

$$\Lambda_1 = \left\lfloor \frac{|V_H|}{|V_G|} \right\rfloor. \tag{1.3}$$

Por último, el *factor de carga* de un arco $e' \in E_H$ es el número de arcos en G cuya imagen en H contiene el arco particular e', $\sum_{e \in E_G} \left| \{e'\} \cap E_{\varphi(e)} \right|$. El factor de carga de φ se denota por Λ_{lf},

$$\Lambda_{lf} = \max_{e' \in E_H} \left\{ \sum_{e \in E_G} |\{e'\} \cap E_{\varphi(e)}| \right\}. \tag{1.4}$$

y es el máximo factor de carga sobre todos los arcos de H. Intuitivamente, podemos ver que si proyectamos el grafo de un algoritmo sobre el grafo de un multiprocesador entonces la dilatación mide la máxima distancia en el grafo del multiprocesador de tareas vecinas en el grafo algoritmo. La expansión mide la utilización de los PEs. Y el factor de carga mide el retardo de cola de los mensajes.

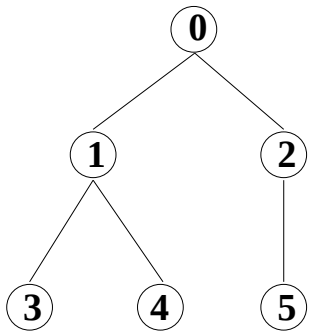


Figura 1.5: Representación de un árbol 2–ario de altura 3. El nodo 0 es la raíz.

1.2.2 Árboles

Definición 2 *Un árbol, T, es un grafo en el cual existe un nodo z que está unido a cualquier otro nodo del grafo por un único camino. El nodo z recibe el nombre de raíz y T_z denota un árbol con raíz z.*

En la figura 1.5 se muestra la representación usual de un árbol. Existen ciertas relaciones típicas entre los nodos conectados a través de un arco. Así, en el ejemplo, los nodos 1 y 2 son *hijos* de la raíz, el nodo 0. Mientras que el nodo 1 es el *padre* de los nodos 3 y 4. Una *hoja* es un nodo sin hijos, por ejemplo, los nodos $\{3, 4, 5\}$. El resto de los nodos reciben el nombre de *nodos internos*.

El camino $\Gamma[v, z]$ es la secuencia de nodos que une la raíz z con el nodo v. Como simplificación en la notación consideraremos $\Gamma[v, z] \equiv \Gamma[v]$. La longitud del camino $\Gamma[v]$ es el número de arcos de que consta ese camino. El *nivel* del nodo v, $l(v)$, es la longitud del camino $\Gamma[v]$. El nivel de la raíz es 0. El máximo nivel del árbol, $\max(l(v)),\ \forall v \in T$, es llamado la *altura* del árbol. Así, en la figura 1.5 el nodo 1 tiene nivel 1 y la altura del árbol es 3.

En un árbol r–ario cada nodo tiene como máximo r hijos. Se llamará *árbol completo* al árbol en que todos sus nodos internos tienen r hijos. Un árbol r-ario completo de altura n tiene n niveles. La raíz del árbol constituirá el nivel 1, los r hijos de la raíz constituirán el nivel 2, los r^2 hijos de estos últimos, el nivel 3, y así sucesivamente. Denominaremos niveles más bajos a los más próximos a la raíz y niveles más altos a los más alejados. En un árbol r-ario completo de n niveles, el número de nodos del nivel i es r^{i-1}, mientras que el número total del nodos es $N = \frac{r^n - 1}{r-1}$.

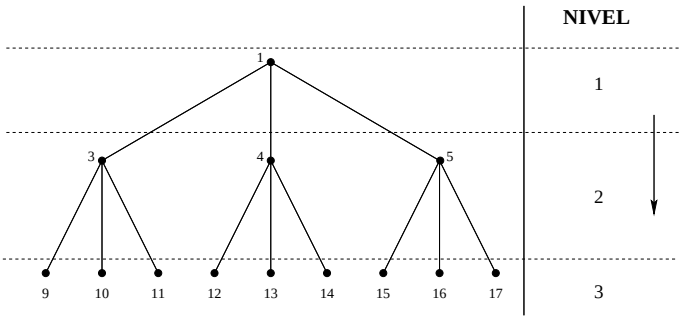


Figura 1.6: Árbol 3-ario de 3 niveles, mostrando la notación utilizada.

Identificaremos cada nodo del árbol mediante un índice entero. Al nodo que constituye la raíz del árbol le asignaremos el índice 1, mientras que a los r hijos del nodo k le asignaremos los índices $\{k \cdot r + j \mid 0 \leq j < r\}$. Así, el conjunto de nodos del segundo nivel será $\{r+j,\ 0 \leq j < r\}$, el del tercer nivel $\{r^2 + j,\ 0 \leq j < r^2\}$, y así sucesivamente. Esta notación tiene como ventaja que es muy fácil determinar el padre y los hijos de un determinado nodo. La desventaja es que los enteros utilizados no son, en general, consecutivos. En la figura 1.6 mostramos un árbol 3-ario de 3 niveles.

1.2.3 Representación de los grafos

Existen dos métodos estándar para representar un grafo en un computador. El primer método utiliza una matriz de adyacencias y el segundo método, una lista encadenada [21, 68].

Consideremos un grafo $G(V, E)$ con N nodos numerados $\{1, 2 \cdots, N\}$. La *matriz de adyacencias* de este grafo es una matriz $N \times N$, $A = (a_{i,j})$, definida de la siguiente forma:

$$a_{i,j} = \begin{cases} 1 & \text{si } (v_i, v_j) \in E_G \\ 0 & \text{en otro caso} \end{cases} \tag{1.5}$$

Las matrices de adyacencias de los grafos de las figuras 1.5 y 1.3 son, respectivamente,

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{1.6}$$

Vemos que la matriz de adyacencias de un grafo no orientado es simétrica. Esta matriz puede ser modificada fácilmente para los grafos con pesos,

$$a_{i,j} = \begin{cases} w(v_i, v_j) & \text{si } (v_i, v_j) \in E_G \\ 0 & \text{si } i = j \\ \infty & \text{en otro caso} \end{cases} \tag{1.7}$$

El espacio necesario para almacenar la matriz de adyacencias de un grafo de N nodos es $O(N^2)$. Un grafo $G(V, E)$ es *disperso* (*sparse*) si $|E| \ll O(|V^2|)$, en otro caso es *denso*. Si el grafo es disperso sería muy ineficiente almacenarlo como una matriz $N \times N$ ya que la mayoría de los elementos de la matriz son ceros y no necesitan ser almacenados explicítamente. Para matrices dispersas es muy común almacenar sólo las entradas no nulas y sus posiciones en la matriz, [64].

La segunda forma de representación de un grafo es mediante una *lista encadenada*. Para representar el grafo $G(V, E)$ se construye el array $Adj[1 \cdots |V|]$ de listas. Para cada $v \in V$, $Adj[v]$ es una lista encadenada a todos los nodos u tal que $(u, v) \in E_G$. La figura 1.7.a muestra la representación de esta lista para el caso de la figura 1.3. En el caso de un árbol la representación puede hacerse con una lista encadenada árbol como se ve en la figura 1.7.b. Esta es la representación del árbol de la figura 1.5. Cada elemento tiene un puntero que indica su padre y otros punteros que indican los hijos.

Para simplificar el almacenamiento de un árbol completo utilizaremos una *lista secuencial* [21]. El árbol se almacena en un array $As[1 \cdots N]$, secuencialmente por niveles. De esta forma la raíz será el elemento $As[0]$, el conjunto de nodos del segundo nivel será $\{1+j,\ 0 \leq j < r\}$, el del tercer nivel $\{\frac{r^2-1}{r-1} + j,\ 0 \leq j < r^2\}$ y así sucesivamente. Esta representación tiene como

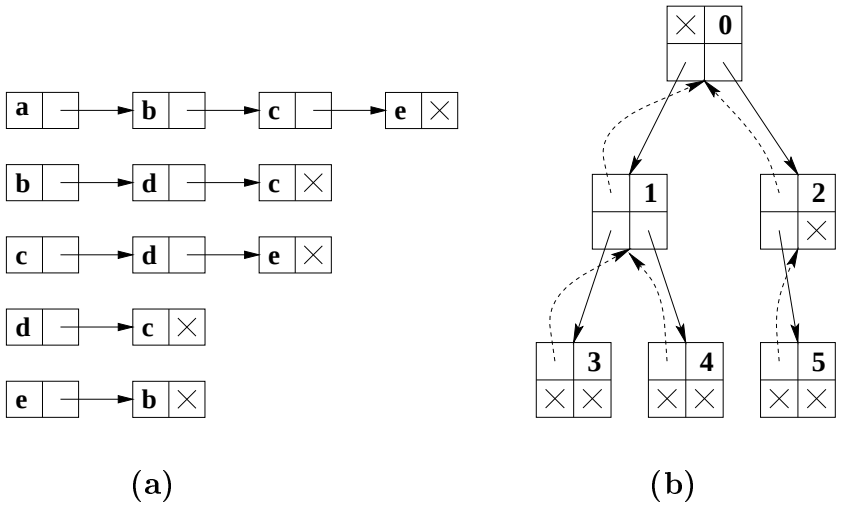


Figura 1.7: (a) Representación de lista de adyacencias del grafo de la figura 1.3. (b) Representación del árbol de la figura 1.5 con una lista encadenada árbol.

ventajas su facilidad para determinar el padre y los hijos de un determinado nodo y ahorro de memoria, pues ocupa solo $O(N)$ elementos de memoria.

1.3 Paralelización de algoritmos

La paralelización de algoritmos sobre sistemas multiprocesadores con memoria distribuida precisa de un buen balanceo de la carga computacional entre los PEs y de la reducción de las comunicaciones. Para algunas aplicaciones, conseguir un buen mapeado de los datos y de las computaciones sobre los PEs es considerablemente difícil.

1.3.1 Descripción del problema

Una aplicación paralela puede describirse mediante un grafo con pesos $G(V_G, E_G)$ llamado grafo de tareas. Los nodos del grafo representan las tareas que quizás puedan ser ejecutadas concurrentemente y los arcos representan las dependencias entre ellas. Existe un nodo $v \in V_G$ para cada tarea y un arco orientado $< u, v > \in E_G$ si la tarea u envía datos a la tarea v. El peso de un nodo de G, $w_G(u)$, es el tiempo de ejecución de la correspondiente tarea, que a su vez es dependiente del número de instrucciones o de opera-

ciones en punto flotante ejecutadas por v. El peso de un arco, $c_G(< u, v >)$, es el tiempo de las comunicaciones desde u a v, que a su vez es dependiente del número de datos transmitidos.

El sistema paralelo se representa por un grafo $P(V_P, E_P)$, llamado grafo de PEs.

El problema del mapeado consiste en encontrar una función $\Psi : V_G \longrightarrow V_P$, que asigne tareas a los PEs de manera que el tiempo de ejecución,

$$t_e = t_c + t_{com}. \tag{1.8}$$

sea mínimo. Suponiendo que el computador es homogéneo, es decir, una tarea se ejecuta idénticamente sobre cualquier PE, todos los pesos c_p y w_p son idénticos. Una técnica de mapeado eficiente debe encontrar un buen compromiso entre la localidad de los datos y el balanceo de la carga. La localidad de los datos influye en la minimización y en el ocultamiento de las latencias asociadas a la comunicación interprocesador. Una carga bien balanceada permite que todos los PEs terminen aproximadamente al mismo tiempo y minimiza el volumen de las comunicaciones.

En resumen, la solución del problema del mapeado consta de dos fases: en primer lugar, realizar una partición de los nodos V_G en P subconjuntos disjuntos y, a continuación, proyectar el grafo cociente sobre el grafo de los PEs (P PEs).

1.3.2 Trabajos previos

Encontrar una solución exacta y óptima al problema del mapeado es un problema NP–completo [19, 51] que requiere una gran cantidad de computación. Esto es así porque si consideramos que el número de nodos del grafo es igual al número de PEs, entonces el problema de buscar una proyección óptima es equivalente al problema de isomorfismo de grafos, del que se sabe que es NP-completo (es decir, que no existe ningún algoritmo de complejidad polinómica para resolverlo [42]). Algunos métodos heurísticos han aparecido en la bibliografía que no encuentran una solución exacta pero sí una solución casi óptima en un tiempo razonable.

Un algoritmo conocido como *network flow* ha sido propuesto en [102] para asignar problemas a sistemas de PEs de memoria distribuida. En el caso de un sistema de dos PEs la asignación es óptima. No es conocida una extensión de la técnica *network flow* a sistemas con más de dos PEs. En [103]

se propone una asignación para el caso de tres PEs, pero no es la solución óptima.

Kung y Stevenson [75] estudiaron la relación entre transferencia lógica y transferencia física y desarrollaron la técnica *linear mapping* en un esfuerzo por mapear algoritmos en redes de interconexión fijas. Lin y Moldovan [75] utilizan *cyclic mapping* para mapear algoritmos sobre multicomputadores de topología malla y el mapeado de algoritmos con permutaciones con barajamiento perfecto.

Bokhari [19, 20] utiliza una heurística iterativa, que combina métodos determinísticos y probabilísticos, para proyectar un grafo sobre la *Finite Element Machine* (FEM) desarrollada en el NASA Langley Research Center, y considerando el número de vértices del grafo igual al número de PEs. Este método intenta maximizar la B–*cardinalidad* de la proyección. La B–cardinalidad de una función de mapeado es una medida de la calidad del mapeado a través del número de arcos del grafo del problema que son proyectados sobre arcos del grafo de los PEs. Es decir, se intenta maximizar el número de nodos vecinos en el grafo que se proyectan sobre PEs vecinos en la FEM. Este método tiene varios problemas. En primer lugar, su complejidad es muy elevada, por lo que es ineficiente para problemas grandes. Además, al maximizar la B–cardinalidad no se tienen en cuenta los nodos vecinos asignados a PEs muy separados entre sí.

Otro método iterativo es el presentado en [51] para la proyección de un grafo no estructurado de tareas sobre un hipercubo, en concreto, la Connection Machine CM–2. Este método, denominado *Cyclic Pariwise Exchange* (CPE), intenta minimizar la función objetivo, Γ_1, que representa la suma para todas las aristas del grafo, del coste de comunicaciones de cada arista. El principal problema de este método es que, al tomar Γ_1 como función objetivo, no se tiene en cuenta que existen comunicaciones entre tareas que pueden ser llevadas a cabo en paralelo.

Otros métodos heurísticos no iterativos son, por ejemplo, la *descomposición scatter*, la *descomposición binaria* y los métodos de *distribución en tiras*.

La descomposición *scatter* (también llamada proyección modular) ha sido aplicada a la descomposición y posterior proyección de dominios altamente irregurales [38]. Este método intenta balancear la carga descomponiendo el dominio del problema en un número g grande de *clusters* rectangulares, siendo g mucho mayor que el número de PEs. Posteriormente estos bloques se distribuyen de forma cíclica entre los PEs. Discusiones detalladas de esta técnica pueden encontrarse en [77].

La *descomposición binaria recursiva* (BRD) [17] intenta maximizar la función objetivo Γ_B. Para ello, el grafo es dividido en subgrafos por una bisección ortogonal recursiva múltiple. Inicialmente, el grafo se parte a la mitad por una línea horizontal o vertical, intentando que cada mitad tenga la misma carga computacional. Cada una de las dos mitades es dividida con una línea ortogonal a la anterior, manteniendo el criterio de repartición de la carga. El proceso se repite hasta que el número de subgrafos iguala al número de PEs. Esta técnica asegura un buen balanceo de la carga pero, sin embargo, no tiene en cuenta el efecto que pueden producir las comunicaciones de longitud mayor que uno. Romero y Zapata [86] proponen una modificación que denominan *método de descomposición recursiva múltiple* para implementar el producto matriz dispersa–vector sobre un multicomputador con topología malla.

Otro grupo de técnicas de proyección es el que se engloba bajo el nombre de distribución en tiras o *strip partitioning*. Estas técnicas se basan en las descomposición del grafo en un conjunto de P tiras (*strip*), donde P es el número de PEs, todas con la misma carga computacional, y donde cada tira sólo necesita comunicarse con sus dos tiras vecinas. Estas tiras se proyectan sobre el computador paralelo de forma que tiras vecinas se sitúen en PEs adyacentes. Por tanto, esta técnica es especialmente adaptable a arquitecturas lineales, u otras arquitecturas que pueden emular a estas (hipercubos, mallas, etc.). Se pueden nombrar entre otras, *1–D Strip Partition* [87], la *distribución en tiras en un sentido (1–Way Strip Partition)* o DTS [82, 81], y la *2–D Strip Partition* [87].

Otro método de mapeado es el mapeado *basado en índices (index–based)* [114]. Este método está basado en la conversión de coordenadas multidimensionales a índices unidimensionales de forma que se mantenga la proximidad de los nodos en el espacio multidimensional. Los nodos del primer grafo se proyectan en una red de tamaño $2^{l_1} \times 2^{l_2} \times \cdots \times 2^{l_d}$. Cada nodo se representa ahora por una tupla $(coord_1, coord_2, \cdots, coord_d)$. Esta tupla se transforma en índice unidimensional usando una curva que llena el espacio (*space–filling curve*). Una vez que se han obtenido los índices de cada nodo, éstos se ordenan según el valor de su índice. A continuación se divide la lista en P subconjuntos consecutivos de igual carga computacional. Cada sublista representa una partición. Este método se ha aplicado eficientemente a algoritmos de ordenamiento [107], operaciones *quadtree* [95], imágenes dispersas [96] y simulación de N cuerpos [114, 79, 83] sobre máquinas paralelas.

Una metodología general para el mapeado de algoritmos secuenciales sobre hipercubos es el propuesto por Rivera, Zapata y col. [85]. El procedi-

miento implica las siguientes etapas:

1. Análisis del algoritmo secuencial a nivel de lazo, identificando los lazos anidados independientes. Estos lazos definen el número de dimensiones del algoritmo secuencial.

2. Particionado de las dimensiones del hipercubo en subconjuntos asociados con los lazos indepedientes del algoritmo secuencial.

3. Distribución entre los PEs de las variables que participan en el algoritmo de acuerdo a la distribución y modo de almacenamiento elegido.

4. Diseño del algoritmo paralelo. Este algoritmo paralelo es el resultado de los pasos 2 y 3, añadiendo los mensajes de comunicación para las transferencias interprocesador de los datos necesarios que no están en la memoria local.

5. Optimización del algoritmo mediante la elección de la partición en el paso 2 y la distribución de los datos del paso 3. Esta optimización se realiza utilizando parámetros como eficiencia, aceleración, redundancia de los datos y balanceo de la carga, entre otros.

La aplicación de este procedimiento ha permitido obtener algoritmos paralelos eficientes en los campos del álgebrea matricial densa y en el procesamiento de imagen y señal [22, 120, 121].

Nosotros utizaremos la proyección vector (*mapping vector*) para particionar y proyectar los algoritmos sobre máquinas paralelas de topología malla. Mostraremos que este método es extremadamente rápido, fácil de paralelizar, y produce un buen mapeado para una amplia clase de algoritmos.

1.4 La proyección vector y la representación índice–digito

Durante la evaluación de un algoritmo sobre un multiprocesador es necesario redistribuir los datos dentro y entre las memorias locales de los PEs en una gran variedad de formas, tanto regulares como irregulares. En general, esto implica complejos movimientos de datos; visualizar el movimiento de los datos y saber dónde están después de las redistribuciones puede ser bastante difícil.

El método que empleamos simplifica la tarea para un gran rango de problemas usando una representación compacta del mapeado de los datos sobre la memoria [5, 14, 7]. Se basa en la proyección vector [36] y en la representación índice–dígito [39].

Consideremos una secuencia unidimensional de datos de tamaño $N = r^n$, siendo la base r una potencia de 2. Denotaremos esta secuencia de datos usando la *representación índice-dígito* [39]. En esta representación, cada dato de la secuencia se denota por la descomposición en base r de su índice. Es decir, el dato $a(t)$ con índice $t = t_n \cdot r^{n-1} + \cdots + t_2 \cdot r + t_1$ se escribe en la forma,

$$[t_n \cdots t_2 t_1]. \tag{1.9}$$

Consideremos un multicomputador con topología malla bidimensional y con memoria distibuida. La zona de memoria en este computador puede considerarse tridimensional, donde dos de las dimensiones localizan la coordenadas del PE dentro de la malla $(fila, columna)$, y la tercera la posición en la memoria local de cada PE. Si las dimensiones de esta zona de almacenamiento son

$$r^u \times r^v \times r^w, \tag{1.10}$$

la representación índice–dígito se puede escribir de la forma,

$$[x_u \cdots x_1, y_v \cdots y_1, z_w \cdots z_1]. \tag{1.11}$$

El método de la *proyección vector* realiza el mapeado del array de datos sobre la memoria de los PEs mediante la asignación biyectiva de índices de datos a índices de las posiciones de memoria de los PEs. La asignación puede escribirse,

$$[t_n \cdots t_2 t_1] \longrightarrow [x_u \cdots x_1, y_v \cdots y_1, z_w \cdots z_1] \tag{1.12}$$

donde los índices t_i se asocian con uno de los campos x, y o z, es decir, memoria, fila o columna, según la distribución que se realice ($n = u + v + w$). Los modos de distribución más utilizados son el almacenamiento por bloques y el cíclico [119]. También se pueden utilizar los esquemas por bloques desplazados, cíclico balanceado y la distribución cíclica plegada.

Consideraremos que la malla está compuesta de $V \times W$ PEs (distribuidos en V filas y W columnas). Cada PE dispondrá de una memoria local de

$U = N/(V \times W)$ datos. En el caso particular de que V y W sean potencias de la base ($V = r^v$ y $W = r^w$) y con una distribución cíclica de los datos [119], la expresión (1.11) puede escribirse en la forma

$$[memoria, fila, columna] \tag{1.13}$$

con la asignación $memoria \equiv t_n \cdots t_{v+w+1}$, $fila \equiv t_{v+w} \cdots t_{w+1}$ y $columna \equiv t_w \cdots t_1$. Es decir, el dato de índice de la ecuación (1.13) se asigna a la posición de memoria *memoria* del PE de coordenadas (*fila, columna*). Con una distribución por bloques, la coordenada *memoria* se asignaría a los dígitos menos significativos de los índices,

$$[fila, columna, memoria] \tag{1.14}$$

con $fila \equiv t_n \cdots t_{w+u+1}$, $columna \equiv t_{w+u} \cdots t_{u+1}$, $memoria \equiv t_u \cdots t_1$. Para el caso de una malla 4×4 , una matriz de 2^8 elementos y distribución cíclica resulta,

$$[\underbrace{8\ 7\ 6\ 5}_{mem}, \underbrace{4\ 3}_{fila}, \underbrace{2\ 1}_{col}] \tag{1.15}$$

con una distribución por bloques,

$$[\underbrace{8\ 7}_{fila}, \underbrace{6\ 5}_{col}, \underbrace{4\ 3\ 2\ 1}_{mem}] \tag{1.16}$$

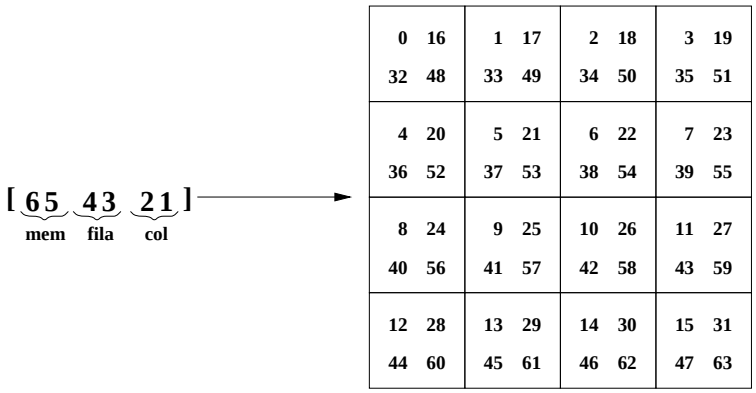
y con una distribución cíclica por filas,

$$[\underbrace{8\ 7\ 6\ 5}_{mem}, \underbrace{4\ 3}_{col}, \underbrace{2\ 1}_{fila}] \tag{1.17}$$

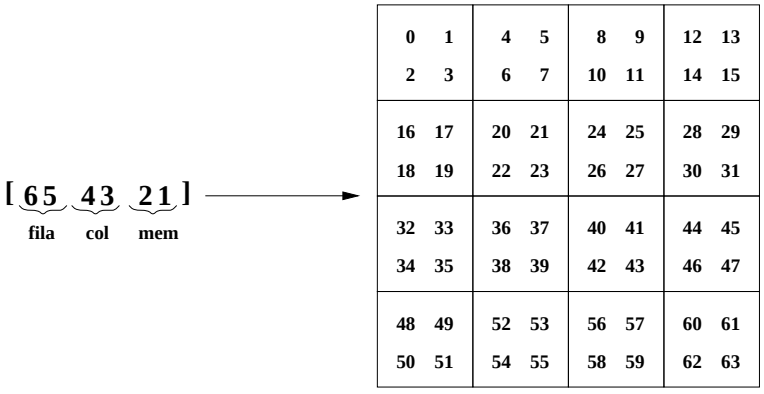
En la figura 1.8 mostramos las tres distribuciones para una malla 4×4 y una matriz de 2^6 elementos.

Para tener los datos con índices adyacentes en el mismo PE o en PEs vecinos, la distribución apropiada es seguir un ordenamiento serpiente,

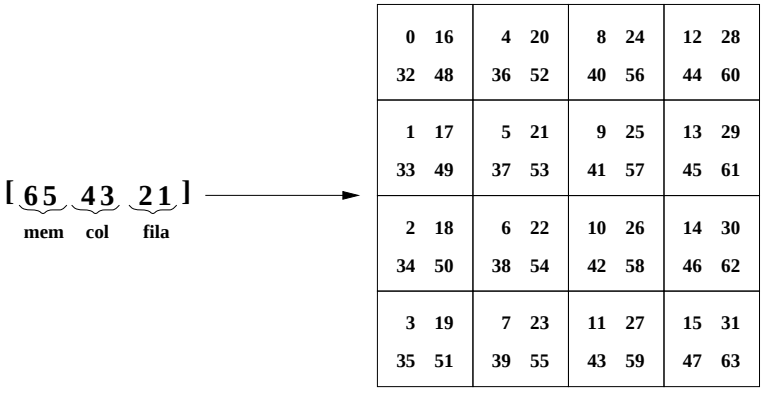
$$\begin{aligned} mem &\equiv t_u \cdots t_1 \\ col &\equiv \begin{cases} t_{w+u} \cdots t_{u+1} & \text{si} \quad t_{w+u+1} = 0 \\ \overline{t_{w+u}} \cdots \overline{t_{u+1}} & \text{si} \quad t_{w+u+1} = 1 \end{cases} \\ fila &\equiv t_n \cdots t_{w+u+1}. \end{aligned} \tag{1.18}$$



(a)



(b)



(c)

Figura 1.8: Distribución de una matriz de 2^6 elementos sobre una malla de 4×4 PEs. (a) Distribución cíclica por columnas. (b) Distribución por bloques. (c) Distribución cíclica por filas.

Para mayor claridad del texto supondremos que U, V y W son potencias de la base. Esto supone una pérdida de generalidad que puede evitarse considerando la representación binaria del índice de los datos, en vez de la representación índice–dígito. Si U, V y W no son potencias de la base, debemos repartir los bits del dígito frontera entre las dos dimensiones adyacentes (*memoria* y *fila*), o (*fila* y *columna*). Más adelante veremos en detalle este proceso.

1.5 Permutaciones índice–dígito

Para formular los algoritmos sobre un multicomputador con topología malla vamos a definir un conjunto de operadores algebraicos que nos permitirán describir las operaciones realizadas sobre los datos. Estos operadores se definen mediante distintas permutaciones *índice–dígito*. Para la escritura de las expresiones de los operadores seguiremos la convención izquierda a derecha de composición de los operadores. Por ejemplo, la expresión $\phi_1 \cdot \phi_2$ indicará que primero se generará el flujo de datos definido por el operador ϕ_1 y, a continuación, el correspodiente a ϕ_2.

Definiremos dos tipos de operadores correspondiendo, respectivamente, a computaciones y comunicaciones. Los operadores del primer tipo representan operaciones aritméticas de tipo in–situ sobre los datos contenidos en las memorias locales de los PEs. Una operación de tipo in–situ escribe los resultados de las operaciones en las mismas posiciones que los datos empleados en su cálculo. Definimos los siguientes operadores que representan las computaciones,

Definición 3 *Los operadores mariposa (butterfly), B_i, B'_i y B''_i realizan las siguientes operaciones sobre los datos,*

1. *B_i lee conjuntos de r datos que difieren en su i-ésimo dígito ($\{[t_n \cdots t_i \cdots t_1],\ 0 \leq t_i < r\}$),*
2. *B'_i es semejante a B_i, excepto que también lee los $r-1$ datos ($\{[t_n \cdots t_{i+1} 0 t_{i-1} \cdots t_1] - [0 \cdots 0 t_i 0 \cdots 0],\ 0 < t_i < r\}$),*
3. *B''_i es semejante a B'_i, excepto que también lee el dato ($[\ t_n \cdots t_{i+1}\ \{r-1\} t_{i-1} \cdots t_1] + [0 \cdots 010 \cdots 0]$),*

y escriben los r resultados en las posiciones de memoria ($\{[t_n \cdots t_i \cdots t_1],\ 0 \leq t_i < r\}$).

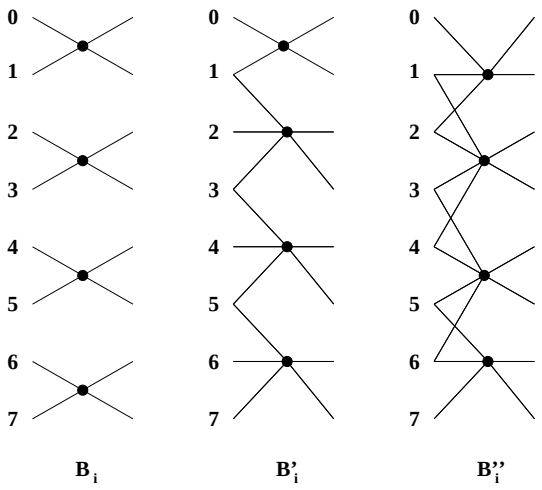


Figura 1.9: Los operadores mariposa ($B_i,\ B_i',\ B_i''$) para secuencias de $N = 8$ datos, base $r = 2$ e $i = 1$.

En estas definiciones, las operaciones $+$ y $-$ que se realizan sobre la representación índice–dígito se entienden de la siguiente manera. Si la representación índice–dígito a y b son $a = [a_n \cdots a_1]$ y $b = [b_n \cdots b_1]$, entonces $[a_n \cdots a_1] + [b_n \cdots b_1]$ es la representación índice–dígito de $a + b$.

Estos operadores se muestran en la figura 1.9. Puesto que estamos interesados sólo en el reagrupamiento de los datos, consideraremos que las computaciones que se realizan son arbitrarias. La definición de estos operadores implica que el conjunto de r datos que se combinan está contenido en la memoria local de un único PE. Este operador conmutará con las redistribuciones de datos entre PEs que mantengan este requerimiento. Nosotros realizaremos redistribuciones de este tipo con el objeto de optimizar los algoritmos. Para simplicar la notación, escribiremos $B \equiv B_n$. En la figura 1.10 mostramos un ejemplo de aplicación del operador B sobre una malla de tamaño 2×2.

Los operadores del segundo tipo representan redistribuciones de datos entre PEs. Para un operador genérico ϕ, la expresión $\phi[x, y, z] = [x', y', z']$, significa que los datos situados en la posición x–ésima de la memoria local del PE de coordenadas (y, z) se desplaza a la posición de memoria x'–ésima del PE de coordenadas (y', z'). Entre los operadores que representan comunicaciones, definimos los siguientes,

Definición 4 *El operador intercambio, $E_{i,j},\ i \geq j$, realiza el intercambio de*

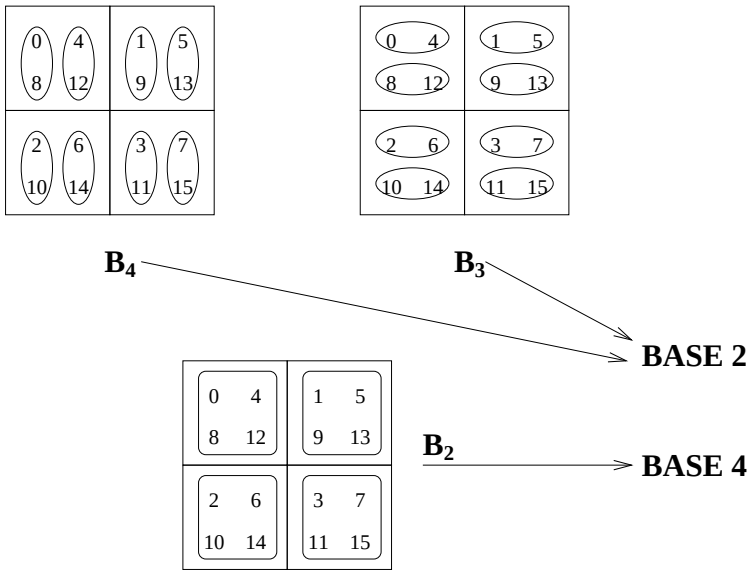


Figura 1.10: Aplicación del operador mariposa, B_4 y B_3 base 2, y B_2 base 4 sobre una secuencia de longitud 16 en una malla de tamaño 2×2. Los conjuntos de datos marcados son los que se recombinan.

los dígitos i y j–ésimo de la representación índice–dígito de los datos,

$$E_{i,j}[t_n \cdots t_i \cdots t_j \cdots t_1] = [t_n \cdots t_j \cdots t_i \cdots t_1] \tag{1.19}$$

Este operador coincide con su inverso, pues $E_{i,j}E_{i,j} = 1$.

Definición 5 *El operador barajamiento perfecto, $\sigma_{i,j}$, $i \geq j$, realiza un desplazamiento cíclico hacia la izquierda de los dígitos comprendidos entre las posiciones i y j-ésima de la representación índice-dígito de los datos,*

$$\sigma_{i,j}[t_n \cdots t_1] = [t_n \cdots t_{i+1}t_{i-1} \cdots t_j t_i t_{j-1} \cdots t_1]. \tag{1.20}$$

Definición 6 *El operador desbarajamiento perfecto, $\Gamma_{i,j}$, $i \geq j$, realiza un desplazamiento cíclico hacia la derecha de los dígitos comprendidos entre las posiciones i y j-ésima de la representación índice-dígito de los datos,*

$$\Gamma_{i,j}[t_n \cdots t_1] = [t_n \cdots t_{i+1}t_j t_i \cdots t_{j+1}t_{j-1} \cdots t_1]. \tag{1.21}$$

Este operador y el anterior son inversos, pues claramente se verifica $\sigma_{i,j}\Gamma_{i,j} = 1$.

Definición 7 *El operador dígito–inverso, $\rho_{i,j}$, $i \geq j$, invierte el orden de los dígitos comprendidos entre las posiciones i y j-ésima de la representación índice-dígito de los datos,*

$$\rho_{i,j}[t_n \cdots t_1] = [t_n \cdots t_{i+1} t_j t_{j+1} \cdots t_i t_{j-1} \cdots t_1]. \tag{1.22}$$

Este operador coincide con su inverso, pues $\rho_{i,j}\rho_{i,j} = 1$.

Por ejemplo, aplicando los operadores anteriores a una secuencia unidimensional de $N = 8$ datos y $r = 2$, resulta,

$$\begin{array}{lcl} & \overset{E_{3,1}}{\longrightarrow} & \\ (7\ 6\ 5\ 4\ 3\ 2\ 1\ 0) & & (7\ 3\ 5\ 1\ 6\ 2\ 4\ 0) \\ & \overset{\sigma_{3,1}}{\longrightarrow} & \\ (7\ 6\ 5\ 4\ 3\ 2\ 1\ 0) & & (7\ 3\ 6\ 2\ 5\ 1\ 4\ 0) \\ & \overset{\Gamma_{3,1}}{\longrightarrow} & \\ (7\ 6\ 5\ 4\ 3\ 2\ 1\ 0) & & (7\ 5\ 3\ 1\ 6\ 4\ 2\ 0) \\ & \overset{\rho_{3,1}}{\longrightarrow} & \\ (7\ 6\ 5\ 4\ 3\ 2\ 1\ 0) & & (7\ 3\ 5\ 1\ 6\ 2\ 4\ 0) \end{array} \tag{1.23}$$

En la figura 1.11 mostramos la implementación de los operadores Γ, σ, ρ y E sobre una malla de tamaño 2×2 con una representación índice–dígito base 2, $[\underbrace{t_3}_{mem}, \underbrace{t_2}_{fila}, \underbrace{t_1}_{col}]$.

Precisamos también definir los dos siguientes operadores,

Definición 8 *El operador barajamiento perfecto sobre dos subcampos de dígitos, $\sigma_{i,j,k,l}$, $i \geq j \geq k \geq l$, realiza un desplazamiento cíclico hacia la izquierda desde el dígito i al j–ésimo y desde el k al l–ésimo de la representación índice–dígito de los datos,*

$$\sigma_{i,j,k,l}[t_n \cdots t_1] = [t_n \cdots t_{i+1} t_{i-1} \cdots t_j t_k t_{j-1} \cdots t_{k+1} t_{k-1} \cdots t_l t_i t_{l-1} \cdots t_1] \tag{1.24}$$

Nótese que $\sigma_{i,j,k,l} \neq \sigma_{i,j}\sigma_{k,l}$; sin embargo $\sigma_{i,j,k,l} = \sigma_{i,j}\sigma_{k,l}E_{j,l}$.

Definición 9 *El operador desbarajamiento perfecto sobre dos subcampos de dígitos, $\Gamma_{i,j,k,l}$, $i \geq j \geq k \geq l$, realiza un desplazamiento cíclico hacia la derecha desde el dígito i al j–ésimo y desde el k al l–ésimo de la representación índice–dígito de los datos,*

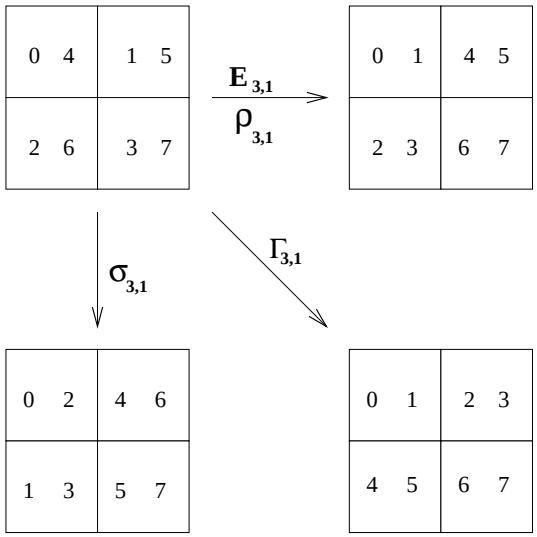


Figura 1.11: Implementación de los operadores base 2 intercambio, $E_{3,1}$, barajamiento perfecto, $\sigma_{3,1}$, desbarajamiento perfecto, $\Gamma_{3,1}$ y dígito–inverso, $\rho_{3,1}$ sobre una malla de tamaño 2×2.

$$\Gamma_{i,j,k,l}[t_n \cdots t_1] = [t_n \cdots t_{i+1} t_l t_i \cdots t_{j+1} t_{j-1} \cdots t_{k+1} t_j t_k \cdots t_{l+1} t_{l-1} \cdots t_1] \tag{1.25}$$

El objeto de estas definiciones es disponer de operadores desbarajamiento perfecto y barajamiento perfecto que modifiquen sólo las dimensiones *memoria* y *columna.*

Entre los operadores que hemos definido, se verifican una serie de relaciones que hemos incluido en la siguiente subsección. Además, podemos clasificar los operadores en cuatro categorías dependiendo del tipo de comunicaciones que precisan:

1. Si el operador modifica sólo la dimensión *memoria* no requiere comunicaciones.

2. Si el operador modifica las dimensiones *memoria* y *fila*, las comunicaciones tienen lugar en paralelo sobre las columnas.

3. Si el operador modifica las dimensiones *memoria* y *columna*, las comunicaciones tienen lugar en paralelo sobre las filas.

4. Si el operador modifica las dimensiones *fila* y *columna* las comunicaciones tienen lugar en diagonal, atravesando tanto filas como columnas.

En el ejemplo de la figura 1.11, los operadores $E_{3,1}$ y $\rho_{3,1}$ son del tipo 3, mientras que los operadores $\sigma_{3,1}$ y $\Gamma_{3,1}$ son del tipo 4.

1.5.1 Relaciones entre las permutaciones índice–dígito

En este apartado obtendremos algunas relaciones que se verifican entre los operadores definidos previamente.

El operador mariposa B_i conmuta con todos aquellos operadores que no modifican el dígito i-ésimo, por ejemplo, $B_i E_{j,k} = E_{j,k} B_i$ $(i \neq j, k)$. En cambio, si un operador modifica el dígito i-ésimo, habrá que tener esto en cuenta. Podemos establecer el siguiente lema,

Lema 1

$$B_i E_{i,j} = E_{i,j} B_j \tag{1.26}$$
$$\sigma_{n,j} B_i = B_{i-1} \sigma_{n,j}, \ i > j \tag{1.27}$$
$$B_i \Gamma_{n,j} = \Gamma_{n,j} B_{i-1}, \ i > j \tag{1.28}$$

Demostración Inmediata, por comprobación directa. □

Se verifican las siguientes expresiones relativas a los operadores intercambio:

Lema 2

$$E_{i,j} E_{i,k} = E_{i,k} E_{j,k} = E_{j,k} E_{i,j} \tag{1.29}$$
$$E_{i,j} E_{i,k} E_{i,j} = E_{j,k} \tag{1.30}$$

Demostración Inmediata, por comprobación directa. □

El lema 2 establece distintas relaciones entre las operaciones intercambio que afectan a tres dígitos.

Relativas al operador barajamiento perfecto, podemos demostrar las siguientes relaciones,

Lema 3

$$\sigma_{i,j} = \sigma_{i,j+1} E_{j+1,j} = \sigma_{i-1,j} E_{i,j} = E_{i,j} \sigma_{i,j+1} = E_{i,i-1} \sigma_{i-1,j} \tag{1.31}$$

Demostración Inmediata, aplicando la definición de los operadores. Por ejemplo, la primera igualdad de (1.31) se demuestra de la siguiente forma

$$\begin{aligned} \sigma_{i,j+1}E_{j+1,j}[t_n \cdots t_1] &= E_{j+1,j}[t_n \cdots t_{i+1}t_{i-1} \cdots t_{j+1}t_i t_j \cdots t_1] \\ &= [t_n \cdots t_{i+1}t_{i-1} \cdots t_j t_i t_{j-1} \cdots t_1] \\ &= \sigma_{i,j}[t_n \cdots t_1] \end{aligned} \tag{1.32}$$

□

El lema 3 proporciona una relación entre la permutación barajamiento perfecto y la correspondiente que desplaza un dígito más.

Lema 4

$$\sigma_{i,j} = \prod_{k=j}^{i-1} E_{i,k} = \prod_{k=j}^{i-1} E_{k+1,j} = \prod_{k=1}^{i-j} E_{i-k+1,i-k} \tag{1.33}$$

Demostración Inmediata, aplicando recursivamente el lema 3. □

El lema 4 proporciona la descomposición de la permutación barajamiento perfecto, $\sigma_{i,j}$, en $i-j$ operaciones intercambio. La teoría de grupos demuestra que cualquier permutación sobre $i-j+1$ elementos puede expresarse como composición de $i-j$ intercambios (transposiciones). En el caso de la permutación barajamiento perfecto éste es el número mínimo de intercambios que se pueden usar, aunque son posibles otras descomposiciones que utilizan $i-j$ o más intercambios.

Lema 5

$$\sigma_{i,j} = \sigma_{i,k+1}\sigma_{k,j}E_{k+1,j} = E_{i,k}\sigma_{i,k+1}\sigma_{k,j} = \sigma_{i,k+1}E_{k+1,k}\sigma_{k,j} \tag{1.34}$$

También son válidas las siguientes expresiones alternativas

$$\sigma_{i,j} = \sigma_{k,j}\sigma_{i,k+1}E_{k+1,j} = E_{i,k}\sigma_{k,j}\sigma_{i,k+1} = \sigma_{k,j}E_{i,j}\sigma_{i,k+1} \tag{1.35}$$

siendo, en ambos casos $i > k > j$.

Demostración Inmediata, aplicando el lema 3. □

El lema 5 establece que la permutación barajamiento perfecto $\sigma_{i,j}$ (que desplaza el campo de dígitos situado entre las posiciones i y j-ésima) puede descomponerse en dos subpermutaciones que desplazan los campos $\{i, \ldots, k+1\}$ y $\{k, \ldots, j\}$ más una operación de intercambio.

Se verifica el siguiente lema,

Lema 6

$$\prod_{i=1}^{s} \sigma_{n,i} = \prod_{i=1}^{s} E_{s+i,i} \prod_{i=1}^{s} \sigma_{n,2s+1,s,i} \tag{1.36}$$

Demostración Por inducción en la variable j sobre la ecuación siguiente:

$$\prod_{i=j}^{s} \sigma_{n,i} = (\sigma_{2s,j})^{s-j+1} \prod_{i=j}^{s} \sigma_{n,2s+1,s,i} \tag{1.37}$$

Esta ecuación se verifica para $j = s$, como puede comprobarse por sustitución directa. Si suponemos que se verifica para $j+1$, es decir

$$\prod_{i=j+1}^{s} \sigma_{n,i} = (\sigma_{2s,j+1})^{s-j} \prod_{i=j+1}^{s} \sigma_{n,2s+1,s,i} \tag{1.38}$$

entonces se verifica para j, pues,

$$\begin{aligned} \prod_{i=j}^{s} \sigma_{n,i} &= \sigma_{n,j} (\sigma_{2s,j+1})^{s-j} \prod_{i=j+1}^{s} \sigma_{n,2s+1,s,i} \\ &= (\sigma_{2s,j})^{s-j+1} \sigma_{n,2s+1,s,j} \prod_{i=j+1}^{s} \sigma_{n,2s+1,s,i} \end{aligned} \tag{1.39}$$

ya que $\sigma_{n,j} (\sigma_{2s,j+1})^{s-j} = (\sigma_{2s+1,j})^{s-j+1} \sigma_{n,2s+1,s,j}$ como puede comprobarse por sustitución directa, con lo que queda demostrada la expresión (1.37). Además,

$$(\sigma_{2s,1})^{s} = \prod_{i=1}^{s} E_{s+i,i} \tag{1.40}$$

como puede comprobarse por sustitución directa, con lo que queda demostrado el lema. □

Entre el operador barajamiento perfecto y el operador dígito–inverso pueden establecerse las siguientes relaciones,

Lema 7

$$\prod_{k=0}^{i-j} \sigma_{i,j+k} = \prod_{k=0}^{i-j} \sigma_{j+k,j} = \rho_{i,j} \tag{1.41}$$

con $i \geq j$.

Demostración Inmediata, por inducción. □

Relaciones similares a las de los lemas 3 a 7 pueden establecerse para la permutación desbarajamiento perfecto $\Gamma_{i,j}$. Puesto que $\Gamma_{i,j} = (\sigma_{i,j})^{-1}$, las expresiones para el operador desbarajamiento perfecto se obtienen a partir de las correspondientes expresiones para el operador barajamiento perfecto, invertiendo cada uno de los operadores y el orden en las cadenas de operadores.

El siguiente lema relaciona operadores del mismo tipo para diferentes bases.

Lema 8 *Los operadores base r^2 pueden expresarse a partir de los correspondientes operadores base r de la siguiente forma*

$$B^{r^2} = \left(\prod_{k=1}^{r} B^r_{n-k+1}\right) \rho^r_{n,n-r+1} = \rho^r_{n,n-r+1} \prod_{k=1}^{r} B^r_{n-r+k} \tag{1.42}$$

$$E^{r^2}_{i,j} = \prod_{k=1}^{r} E^r_{r(i-1)+k,r(j-1)+k} \tag{1.43}$$

$$\sigma^{r^2}_{i,j} = \prod_{k=1}^{r} \sigma^r_{r(i-1)+1,r(j-1)+1} \tag{1.44}$$

$$\Gamma^{r^2}_{i,j} = \prod_{k=1}^{r} \Gamma^r_{r(i-1)+1,r(j-1)+1} \tag{1.45}$$

en donde hemos indicado con un superíndice la base de los operadores.

Demostración Inmediata por comprobación directa. □

1.6 Multiprocesadores utilizados

En esta sección presentaremos las máquinas paralelas con las que hemos trabajado, en concreto el AP1000 de Fujitsu y el T3E de Cray. Ambas máquinas son multiprocesadores con topología malla y memoria distribuida. Utilizaremos el modelo de programación de pase de mensajes. Las rutinas que permiten realizar estas operaciones son extensiones de un lenguaje de alto nivel, como el C o FORTRAN. También están disponibles entornos estándard como MPI [50].

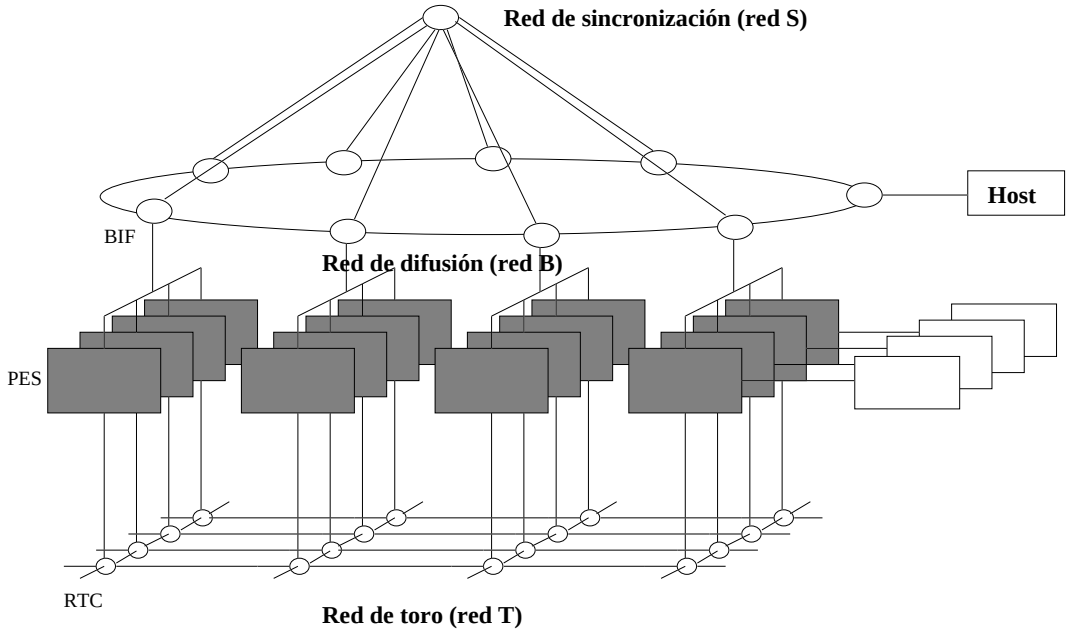


Figura 1.12: Estructura del computador multiprocesador AP1000.

1.6.1 El AP1000 de Fujitsu

El AP1000 es un computador multiprocesador de memoria distribuida comercializado por Fujitsu. Este computador tiene una topología de toroide 2–D. Consta de 64 a 1024 PEs o celdas los cuales se encuentran interconectados mediante tres redes de comunicación independientes, como puede observarse en la figura 1.12. Estas son:

- Red de difusión o red B, para comunicaciones de 1 a N, entre el procesador host y los PEs, así como para distribución y recolección de los datos.
- Red toroide o red T, para comunicaciones punto a punto entre los PEs.
- Red de sincronización o red S, empleada en las sincronizaciones de barrera.

Los PEs del AP1000, cuya configuración se muestra en la figura 1.13, constan de: una unidad de enteros (IU), una unidad de punto flotante (FPU), un controlador de mensajes (MSC), un controlador de encaminamiento (RTC), una interface con la red B (BIF) y 16 Mbytes de memoria RAM dinámica (DRAM). La IU, la FPU y una cache de 128 Kbytes están conectadas al

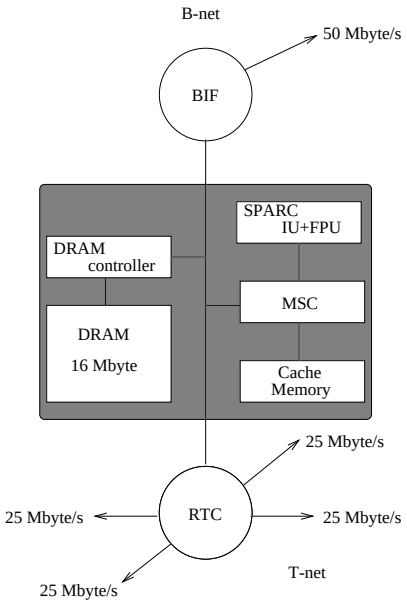


Figura 1.13: Configuración de los PEs del computador AP1000.

MSC. Durante la operación normal, los MSCs trabajan como controladores de memoria cache con un esquema de ubicación directa y una estrategia de postescritura (*copy–back*) de actualización de la memoria principal, siendo el tamaño de los bloques de cuatro palabras.

El MSC, RTC, BIF y el controlador DRAM de cada PE están conectados a través de un bus local (LBUS en la figura 1.13), que es un bus síncrono de 32 bits. Además, cada PE tiene un conector al LBUS externo, que permite varias opciones *hardware* tales como interface de E/S de alta velocidad, interface de disco y memoria adicional.

Una estación Sun–4/330 actúa como un computador *host* que efectúa tareas de control. Los interfaces del host están constituidos por una interfaz VME–bus, una interfaz B–net y una memoria local de 32 Mbytes.

1.6.2 El Cray T3E

El Cray T3E [93] implementa un espacio de direcciones almacenadas sobre una memoria distribuida (sobre 2GB por PE). Cada PE contiene un DEC Alpha 21164. Puede constar de 16 a 2048 PEs conectados mediante una red bidireccional en toro tridimensional, ver figura 1.14, con un gran ancho de banda. La velocidad de comunicación entre PEs en cualquier dirección a través del toro es de 480 Mbytes/s.

Cada celda del T3E, como se puede ver en la figura 1.15, incluye: un microprocesador DEC Alpha 21164, una memoria local, un *router* de comunicación y una lógica de control.

El sistema de memoria es lógicamente compartido y físicamente distribuido, con lo que todos los PEs tienen su memoria local, pero pueden acceder a la memoria de los otros PEs sin necesidad de utilizar un protocolo de paso de mensajes. Así, este multiprocesador se puede programar utilizando un modelo de pase de mensaje (MPI o PVM), o utilizando un modelo de programación de memoria compartida (HPF).

El T3E aumenta el interface de memoria del microprocesador DEC 21164 con un conjunto de registros externos (E-registros). Estos registros se utilizan como fuente o destino para las comunicaciones remotas. Todas las comunicaciones remotas y las sincronizaciones se realizan entre los registros y la memoria.

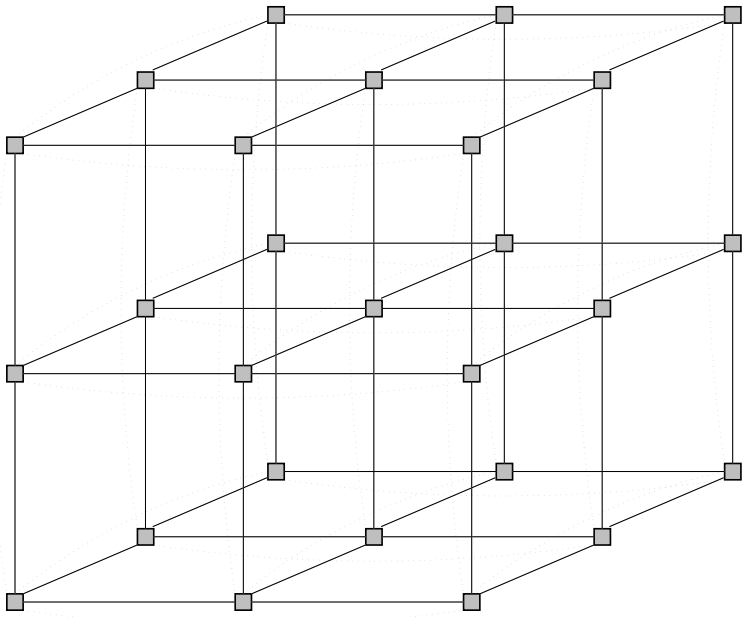


Figura 1.14: Red 3D toro.

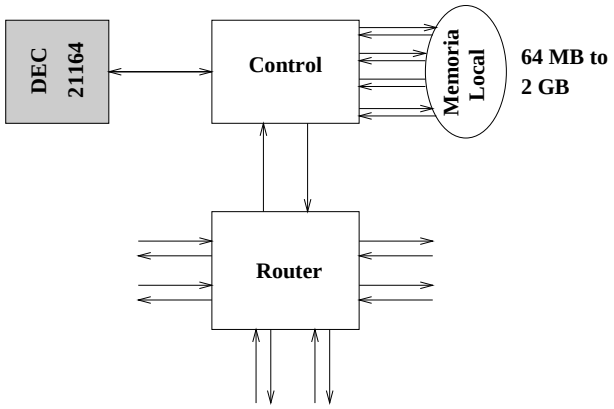


Figura 1.15: Diagrama de bloque del PE del Cray T3E.

1.7 Rendimiento de un algoritmo paralelo

En esta memoria describiremos una serie de algoritmos paralelos con ayuda del álgebra de operadores que hemos presentado. Para medir la efectividad de los algoritmos se han propuesto una serie de paramétros [75] entre los que seleccionamos los siguientes:

- **Aceleración**, (S_p). El factor de aceleración (*speed–up*) de un programa paralelo que emplea P PEs se define mediante la expresión

$$S_p = \frac{T_S}{T_P} \tag{1.46}$$

donde T_S es el tiempo de ejecución del mejor programa secuencial que resuelve el problema sobre un PE y T_P es el tiempo de ejecución sobre P PEs. En otras palabras, la aceleración muestra la ganancia de velocidad de computación paralela; cuanto mayor sea S_p, mejor. El factor de aceleración, como se muestra en la figura 1.16.a es normalmente menor que el número de PEs ($1 \leq S_p \leq P$), que sería el caso ideal. Esto se debe al tiempo perdido debido a las dependencias de datos, a las comunicaciones, sincronizaciones, y otros gastos requeridos por la computación paralela.

Aceleraciones más grandes que el número de PEs, llamadas *sobreaceleraciones*, se pueden obtener si el programa paralelo elimina computaciones innecesarias y caminos erróneos del programa secuencial, como puede ocurrir en casos particulares de problemas de búsqueda. También pueden ocurrir sobreaceleraciones cuando la estructura de datos

del algoritmo paralelo pueda distribuirse completamente en las caches de los PEs, mientras que en el programa secuencial deba utilizarse memoria principal.

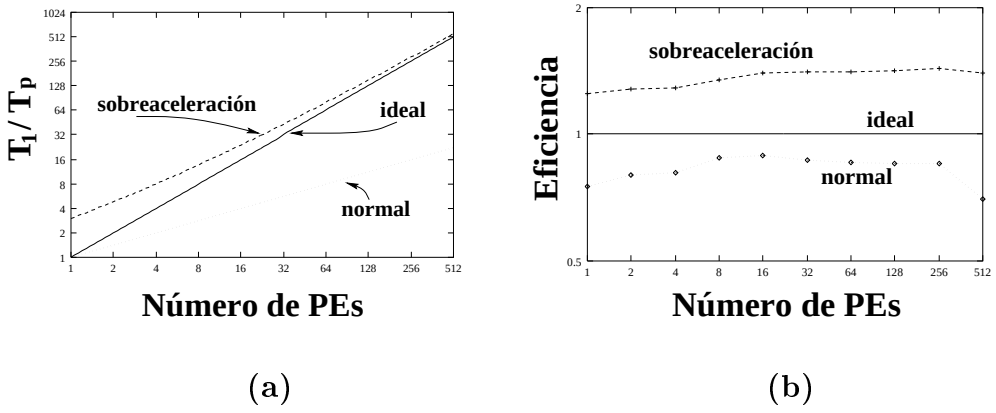


Figura 1.16: (a) Gráficas de aceleraciones típicas. (b) Gráficas de eficiencias típicas.

- **Eficiencia**, (E_p). La eficiencia de una computación paralela se define como la relación entre el factor de aceleración y el número de PEs

$$E_p = \frac{S_p}{P} = \frac{T_S}{P \cdot T_P} \tag{1.47}$$

Es una medida normalizada (toma valores comprendidos entre 0 y 1) de la efectividad de una computación paralela. En el caso ideal $E_p = 1$, (ver figura 1.16.b). Pero, normalmente, $E_p < 1$ por los mismos motivos que el factor de aceleración.

Se define un *algoritmo escalable*, como el algoritmo que es N veces más rápido cuando se ejecuta sobre N PEs. La escalabilidad de un sistema paralelo es una medida de su capacidad para aumentar la aceleración en proporción al número de PEs.

Capítulo 2

La transformada rápida de Fourier

2.1 Introducción

La transformada de Fourier es una herramienta básica en áreas científicas y de ingeniería tan diversas como la medicina, la acústica, el procesado de imágenes, el diseño de sistemas y muchos otros campos. Durante años, todos estos campos han estado limitados en su desarrollo debido a que los algoritmos utilizados para calcular la transformada de Fourier empleaban un tiempo prohibitivo.

En 1965 Cooley y Tukey [27] desarrollaron un algoritmo para acelerar el cálculo de la transformada discreta de Fourier (DFT). Este algoritmo conocido como *Fast Fourier Transform (FFT) DET base 2*, está basado en la aplicación del método de doblamiento sucesivo (DS). El método DS se basa en la estrategia *divide y vencerás* (DV), la metodología de diseño paralelo más ampliamente utilizada. Algoritmos paralelos basados en la estrategia DV se utilizan en casi todas las áreas científicas. Algunos ejemplos son el cálculo de la suma de N elementos, la resolución de sistemas tridiagonales, el cálculo de transformadas rápidas, la determinación del camino mínimo entre dos nodos, etc.

El ejemplo más sencillo de aplicación del método DS es la suma definida

$$X = \sum_{i=0}^{N-1} x_i. \tag{2.1}$$

La ejecución secuencial de esta suma se muestra en la figura 2.1.a. La apli-

cación del método de DS consiste en descomponer la expresión 2.1 en dos subsecuencias del mismo tipo que operan sobre la mitad de los elementos. Reiterando el proceso sobre cada subsuma obtenemos el algoritmo de la figura 2.1.b.

La aplicación del método DS consigue un flujo de datos muy regular que permite su proyección eficiente sobre sistemas paralelos y, en muchas ocasiones, también elimina las redundancias del problema, construyendo algoritmos de mucha menor complejidad y más rápidos que los de partida. Así, la complejidad algorítmica de la DFT de una secuencia de N elementos es de orden $O(N^2)$, mientras que la de la FFT es $O(N\lceil \log_2 N \rceil)$. Esta reducción puede ser de varios órdenes de magnitud en determinadas aplicaciones. Por la regularidad de acceso a los datos en la FFT, se puede optimizar su rendimiento mediante un mapeado óptimo y una adecuada temporización de las computaciones.

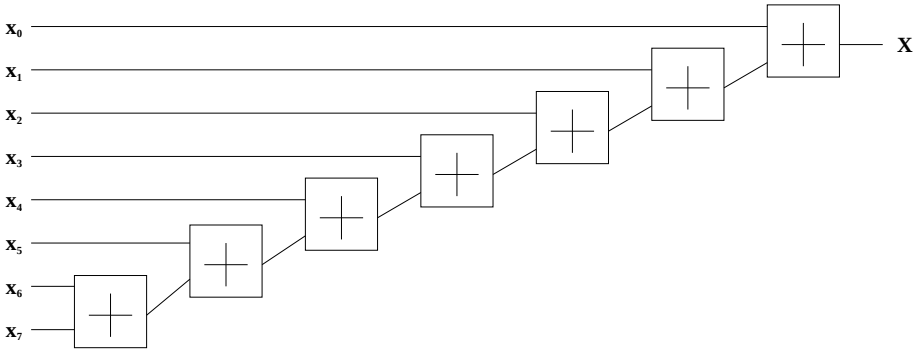
El algoritmo FFT permitió resolver problemas hasta entonces inabordables. Con el fin de probarlo se efectuó el análisis de un temblor de tierra sucedido en Alaska en 1964. El algoritmo clásico requirió más de 26 minutos, mientras bastaron menos de dos segundos y medio con el nuevo para realizar la tarea.

Desde la aparición del algoritmo de Cooley y Tukey para la FFT en 1965 han surgido un gran número de algoritmos alternativos. Todos los algoritmos obtienen el mismo resultado pero varían el cálculo y el flujo de datos de los resultados intermedios.

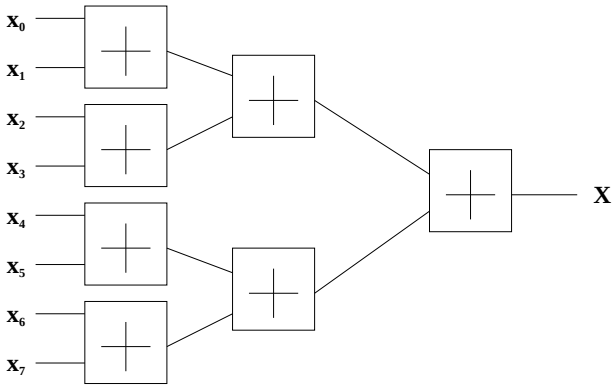
No revisaremos los algoritmos secuenciales de la bibliografía. Sólo citaremos aquí algunos de los numerosos libros y monografías que han ido apareciendo a lo largo de los años. Destacan el libro de Rabiner y Gold [84] y la recopilación de trabajos [105] como una guía bibliográfica de algoritmos secuenciales y vectoriales. Han sido presentada implementaciones de diferentes versiones de la FFT sobre sistemas multiprocesadores [78, 13, 119, 31], sobre computadores de topología hipercubo [108], sobre computadores de topología malla [36, 11, 14], sobre computadores vectoriales [15, 25, 6, 53], y sobre multiprocesadores con memoria distribuida [23], entre otros.

2.2 La transformada rápida de Fourier

La FFT es un algoritmo obtenido a partir de la DFT aplicando la estrategia DV [27]. La DFT de una secuencia $\{x(m),\ 0 \leq m < N\}$ de N elementos, se define a través de la siguiente ecuación,



(a)



(b)

Figura 2.1: Recurrencia del tipo suma sobre una secuencia de $N = 8$ datos, (a) realizada secuencialmente, (b) realizada en paralelo aplicando el método de DS.

$$X(k) = \sum_{m=0}^{N-1} x(m) W_N^{mk},\ k = 0, 1 \cdots, N-1. \tag{2.2}$$

donde $W_N^{mk} = e^{-j(2\pi/N)mk}$.

La estrategia DV se aplica a una DFT de tamaño $N = r^n$ dividiendo la secuencia inicial de datos en r subsecuencias de longitud N/r (algoritmo base r). Cada una de las subsecuencias obtenidas se subdivide en r subsecuencias. Dependiendo del criterio de formación de las subsecuencias se obtienen los algoritmos Decimación en Tiempo (DET) y Decimación en Frecuencia (DEF). A continuación presentamos el proceso de obtención de ambos algoritmos para el caso de una secuencia $x(m)$ de N datos, donde N es potencia de dos ($r = 2$).

2.2.1 Decimación en el tiempo

La secuencia se divide en dos subsecuencias de $N/2$ datos, $x_1(m)$ y $x_2(m)$, tomando los términos pares e impares de $x(m)$ respectivamente,

$$\begin{aligned} x_1(m) &= x(2m) \qquad m = 0, 1, \cdots, N/2-1 \\ x_2(m) &= x(2m+1) \quad m = 0, 1, \cdots, N/2-1. \end{aligned} \tag{2.3}$$

La DFT de la secuencia N datos puede escribirse como

$$X(k) = \sum_{m=0,\, m\, par}^{N-1} x(m) W_N^{mk} + \sum_{m=0,\, m\, impar}^{N-1} x(m) W_N^{mk} \tag{2.4}$$

y obtenemos la expresión

$$X(k) = \sum_{m=0}^{N/2-1} x(2m) W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1) W_N^{(2m+1)k} \tag{2.5}$$

renombrando

$$W_N^2 = [e^{-j(2\pi/N)}]^2 = e^{-j(2\pi/(N/2))} = W_{N/2} \tag{2.6}$$

podemos reescribir $X(k)$ como

$$\begin{aligned} X(k) &= \underbrace{\sum_{m=0}^{N/2-1} x_1(m) W_{N/2}^{mk}}_{X_1(k)} + W_N^k \underbrace{\sum_{m=0}^{N/2-1} x_2(m) W_{N/2}^{mk}}_{X_2(k)} \\ &= X_1(k) + W_N^k X_2(k) \end{aligned} \tag{2.7}$$

donde las secuencias $X_1(k)$ y $X_2(k)$ son las DFTs de las secuencias de $N/2$ datos $x_1(m)$ y $x_2(m)$ respectivamente. $X(k)$ está definida para $0 \leq k < N$ y $X_1(k)$ y $X_2(k)$ están definidas para $0 \leq k < N/2$. La secuencia transformada de los valores $k \geq N/2$ se obtiene por la periodicidad de la DFT y por la igualdad $W_N^{k-N/2} = -W_N^k$.

$$X(k) = \begin{cases} X_1 + W_N^k X_2(k), & 0 \leq k < \frac{N}{2} \\ \\ X_1(k - \frac{N}{2}) + W_N^{k-N/2}(k - \frac{N}{2}), & \frac{N}{2} \leq k < N \end{cases} \tag{2.8}$$

Así, las ecuaciones que relacionan una transformada de longitud N con dos subtransformadas de longitud $N/2$ son:

$$X(k) = \begin{cases} X_1(k) + W_N^k X_2(k), & 0 \leq k < \frac{N}{2} \\ \\ X_1(k - \frac{N}{2}) - W_N^k X_2(k - \frac{N}{2}), & \frac{N}{2} \leq k < N \end{cases} \tag{2.9}$$

En la figura 2.2 se muestra la evaluación de una DFT sobre una secuencia de 8 datos usando dos DFTs de longitud 4. Utilizaremos la notación gráfica que se muestra en la figura 2.3: la flecha define una multiplicación por el factor colocado sobre ella, mientras que el círculo indica una adición–sustracción, apareciendo la suma siempre en la parte superior y la diferencia en la parte inferior. En general, todas las variables son números complejos.

Si reiteramos el proceso de tal forma que cada una de las secuencias $x_1(m)$ y $x_2(m)$ se dividen a su vez en dos subsecuencias (compuestas por los términos pares e impares de las secuencias), entonces las DFTs de longitud $N/2$ datos pueden obtenerse como una combinación de DFTs de longitud $N/4$. Este proceso se reitera hasta obtener subsecuencias de tamaño mínimo (r datos), para las que el cálculo de la DFT es inmediato. Por ejemplo, si $N = 8$, $r = 2$ y se realiza otro desdoblamiento sobre las subsecuencias de longitud $N/4$ se llega a subsecuencias de dos datos (base de la transformada). El cálculo de estas DFTs se puede ver en la figura 2.4. Una DFT de dos datos, $F(k),\ k = 0, 1$ puede ser evaluada por

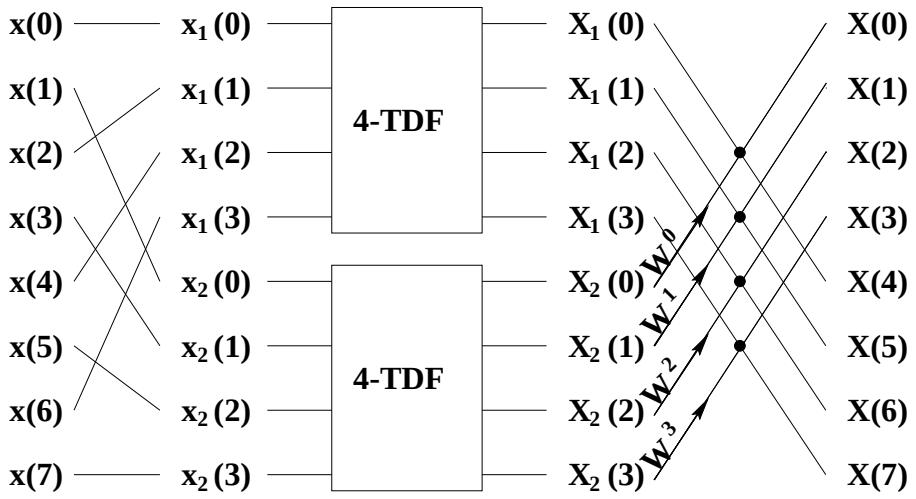


Figura 2.2: Evaluación de una DFT de ocho datos a partir de dos DFTs de cuatro datos.

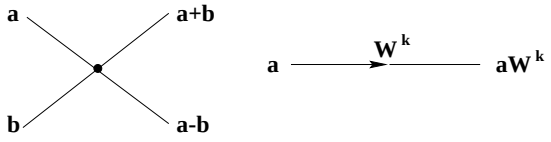


Figura 2.3: Notación gráfica empleada para indicar las operaciones que se realizan sobre los datos en la FFT base 2.

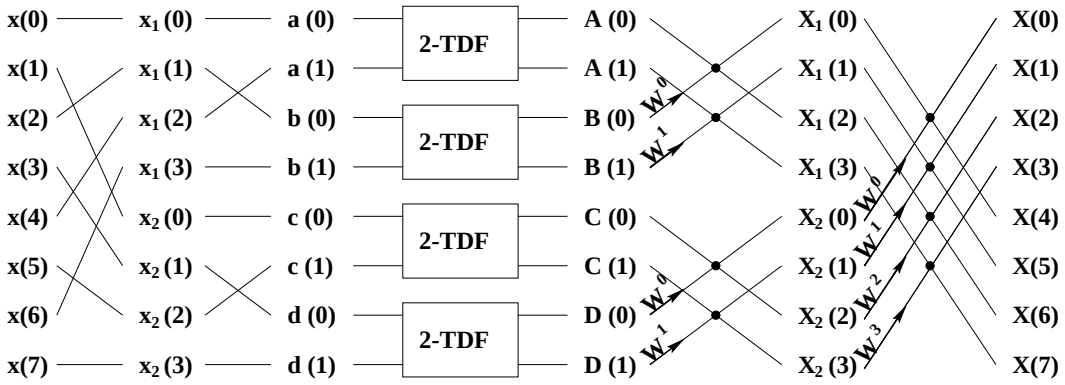


Figura 2.4: Evaluación de una DFT de una secuencia de 8 datos a partir de cuatro DFTs de dos datos (algoritmo FFT).

$$\begin{aligned} F(0) &= f(0) + f(1)W_8^0 \\ F(1) &= f(0) + f(1)W_8^4 \end{aligned} \tag{2.10}$$

donde $f(n)$ es la secuencia de entrada de dos datos que se transforman. Como $W_8^0 = 1$ y $W_8^4 = -1$, ninguna multiplicación es necesaria para evaluar la ecuación (2.10). Así, la DFT de 8 datos de las figuras 2.2 y 2.4 se pueden reducir al flujo de datos de la figura 2.5.

El algoritmo que hemos descrito se llama *decimación en el tiempo*, (DET), porque en cada paso del algoritmo la secuencia de entrada, dominio del tiempo, se divide en subsecuencias menores. La secuencia de entrada es decimada en el tiempo. La operación básica de la DET se llama *mariposa*. Esta operación se realiza sobre dos datos A y B que dan dos salidas X e Y mediante la expresión,

$$\begin{aligned} X &= A + W_N^k B \\ Y &= A - W_N^k B \end{aligned} \tag{2.11}$$

en la figura 2.8.a se muestra el flujo para una mariposa DET.

La complejidad algorítmica de la DFT de una secuencia de N datos computada a partir de la definición es de orden $O(N^2)$, mientras que la del algoritmo rápido, FFT, es de orden de $O(N \log_r N)$.

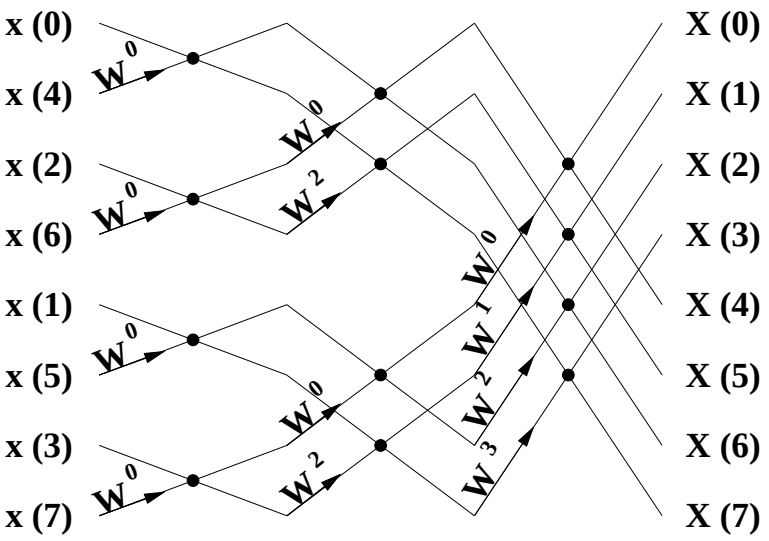


Figura 2.5: FFT de una secuencia de 8 datos obtenida por la técnica DV en base 2.

2.2.2 Decimación en frecuencia

Otra versión de la FFT es la llamada *decimación en frecuencia*, DEF. En este caso, la secuencia de entrada $x(m)$, se parte en dos subsecuencias, cada una de longitud $N/2$. La primera secuencia $x_1(m)$ consiste en los primeros $N/2$ datos de $x(m)$, y la segunda, $x_2(m)$ consiste en los últimos $N/2$ datos de $x(m)$,

$$\begin{aligned} x_1(m) &= x(m) \qquad\quad m = 0, 1, \cdots, N/2-1 \\ x_2(m) &= x(N/2+m) \quad m = 0, 1, \cdots, N/2-1 \end{aligned} \tag{2.12}$$

La DFT de longitud N de la secuencia $x(m)$ puede ahora escribirse

$$\begin{aligned} X(k) &= \sum_{m=0}^{N/2-1} x(m) W_N^{mk} + \sum_{m=N/2}^{N-1} x(m) W_N^{mk} \\ &= \sum_{m=0}^{N/2-1} x_1(m) W_N^{mk} + \sum_{m=0}^{N/2-1} x_2(m) W_N^{(m+N/2)k} \end{aligned} \tag{2.13}$$

y usando el hecho de que $W_N^{kN/2} = e^{-j\pi k}$, se llega a la siguiente relación,

$$X(k) = \sum_{m=0}^{N/2-1} [x_1(m) + e^{-j\pi k} x_2(m)] W_N^{mk} \tag{2.14}$$

Si ahora consideramos separadamente los términos pares e impares de la transformada obtenemos las siguientes expresiones:

$$\begin{aligned} X(2k) &= \sum_{m=0}^{N/2-1} (W_N^2)^{mk} [x_1(m) + x_2(m)] \\ &= \sum_{m=0}^{N-1} W_{N/2}^{mk} [x_1(m) + x_2(m)] \\ X(2k+1) &= \sum_{m=0}^{N/2-1} W_N^{m(2k+1)} [x_1(m) - x_2(m)] \\ &= \sum_{m=0}^{N/2-1} W_{N/2}^{mk} (W_N^{mk} [x_1(m) - x_2(m)]) \end{aligned} \tag{2.15}$$

Estas ecuaciones muestran que los términos pares e impares de la transformada pueden obtenerse desde dos transformadas de longitud $N/2$ de las secuencias $f(m)$ y $g(m)$ siguientes:

$$\begin{aligned} f(m) &= x_1(m) + x_2(m), \qquad m = 0, 1, \cdots, N/2-1 \\ g(m) &= W_N^m [x_1(m) - x_2(m)], \quad m = 0, 1, \cdots, N/2-1 \end{aligned} \tag{2.16}$$

En la figura 2.6 se muestra la evaluación de una DFT con esta aproximación sobre una secuencia de 8 datos usando dos DFT de longitud 4. Si reiteramos el proceso como hemos hecho en el caso de la DET se obtiene el algoritmo FFT de la figura 2.7.

Comparando los dos algoritmos obtenidos se observan dos diferencias. En primer lugar, las mariposas DEF y DET son distintas, (figura 2.8). En las mariposas del algoritmo DET la multiplicación por el factor complejo se realiza antes que la operación de adición-sustracción, mientras que en las mariposas DEF esto sucede al revés. Una segunda diferencia es que en el algoritmo DET la entrada es en orden dígito–inverso y la salida en orden

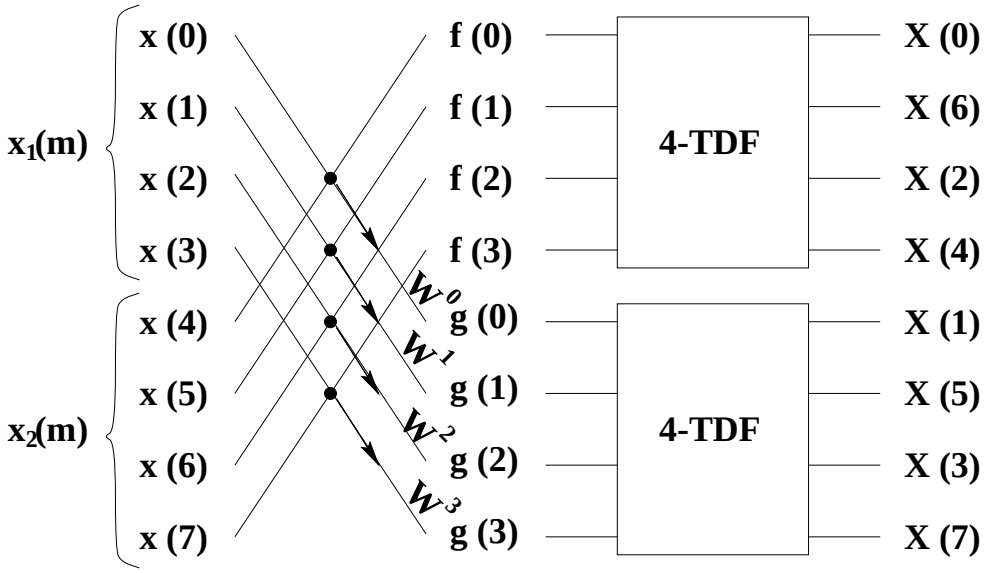


Figura 2.6: Evaluación de una DFT de ocho datos a partir de dos DFTs de cuatro datos usando DEF.

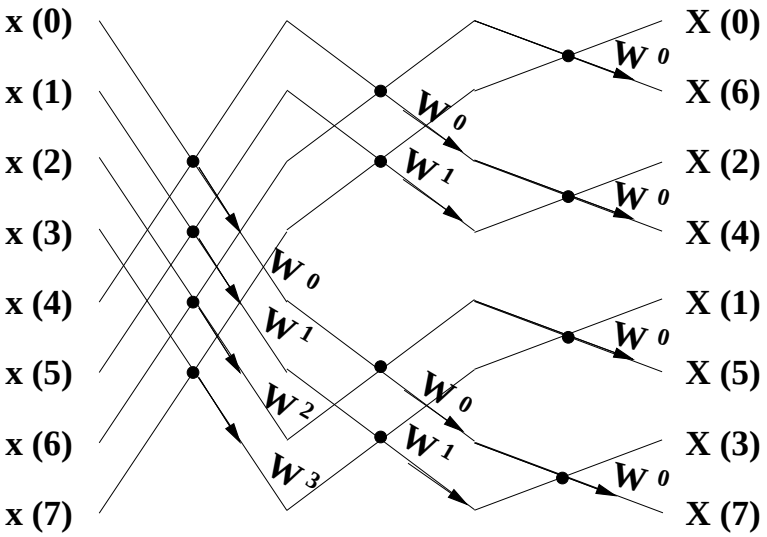


Figura 2.7: Evaluación de una DFT de ocho datos a partir de cuatro DFTs de dos datos usando DEF.

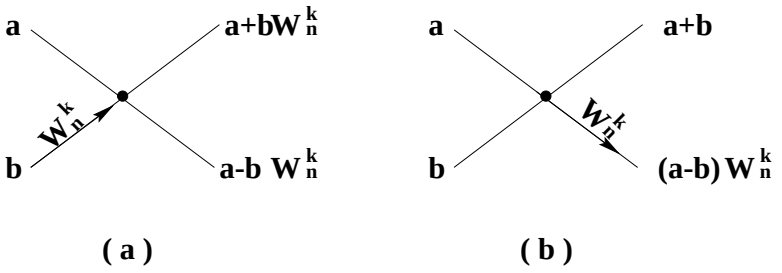


Figura 2.8: Mariposa de los algoritmos DET (a) y DEF(b).

natural, mientras que para la DEF la entrada es en orden natural y dígito-inverso para la salida. Sin embargo, esta última diferencia es sólo aparente, pues se pueden construir algoritmos DEF y DET con cualquiera de los dos tipos de ordenamiento.

2.2.3 Algoritmos para distintas bases

En los apartados anteriores hemos desarrollado los algoritmos FFT, DET y DEF para base 2. Pero se pueden construir algoritmos similares para cualquier base. En general, un algoritmo base r divide una transformada de longitud N en N/r subtransformadas de longitud r donde las mariposas obtenidas operan sobre r datos.

Un algoritmo de especial importancia es la FFT base 4. En las figuras 2.9 y 2.10 se muestran los algoritmos DET y DEF base 4 para una secuencia de $N = 16$ datos. En estas figuras se utiliza para la representación de las mariposas la misma notación que en los esquemas correspondientes a base 2. En este caso la notación se ha generalizado: el círculo con r entradas y r salidas representa un nodo aritmético que computa una transformada de r datos. Esta operación se realiza sobre cuatro datos A, B, C y D y da cuatro salidas X, Y, Z y T por la expresión,

$$\begin{array}{rcccccccc} X & = & A & + & BW^{k} & + & CW^{2k} & + & DW^{3k} \\ Y & = & A & + & BjW^{k} & - & CW^{2k} & - & DjW^{3k} \\ Z & = & A & - & BW^{k} & + & CW^{2k} & - & DW^{3k} \\ T & = & A & - & BjW^{k} & - & CW^{2k} & - & DjW^{3k} \end{array} \tag{2.17}$$

Frente a la flexibilidad del algoritmo base 2 (la secuencia a transformar es potencia de 2), los algoritmos base 4 presentan la ventaja de un menor número de multiplicaciones y referencias a memoria, lo cual supone un incremento en la velocidad de computación.

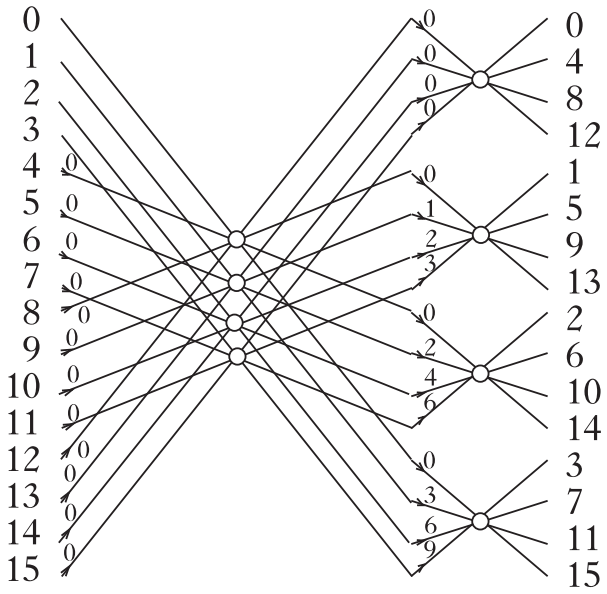


Figura 2.9: Algoritmo FFT base 4 en una transformada de 16 datos usando DET.

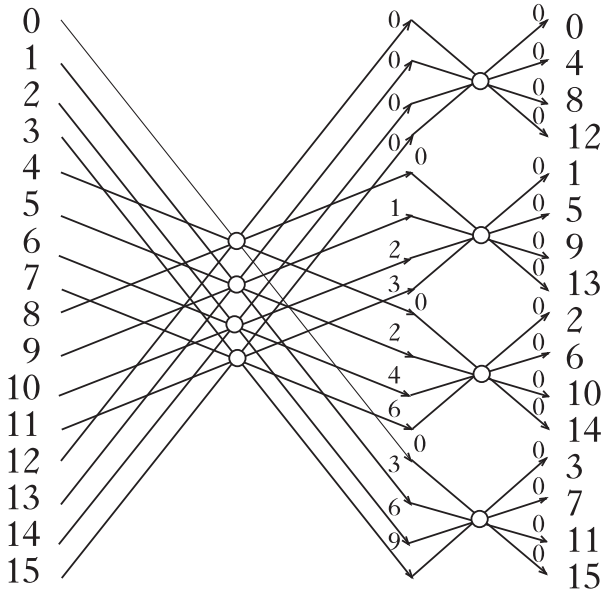


Figura 2.10: Algoritmo FFT base 4 en una transformada de 16 datos usando DEF.

Para bases mayores que 4 no se obtienen algoritmos eficientes. La figura 2.11 muestra una transformada base 8 para $N = 64$ datos. Una transformada de 8 datos se realiza sobre ocho datos A, B, C, D, E, F, G y H y da ocho salidas X, Y, Z, T, P, Q, R, y S por la expresión,

$$
\begin{aligned}
X &= [A + EW^{4k} + CW^{2k} + GW^{6k}] + \\
&\quad [BW^k + FW^{5k} + DW^{3k} + HW^{7k}] \\
Y &= [A - EW^{4k} + j(CW^{2k} - GW^{6k})] + \\
&\quad (1+j)[BW^k - FW^{5k} + DW^{3k} - HW^{7k}] \\
Z &= [A + EW^{4k} - (CW^{2k} + GW^{6k})] + \\
&\quad j[BW^k + FW^{5k} - (DW^{3k} + HW^{7k})] \\
T &= [A - EW^{4k} - j(CW^{2k} - GW^{6k})] + \\
&\quad (1-j)[BW^k - FW^{5k} - j(DW^{3k} - HW^{7k})] \\
P &= [A + EW^{4k} + CW^{2k} + GW^{6k}] - \\
&\quad [BW^k + FW^{5k} + DW^{3k} + HW^{7k}] \\
Q &= [A - EW^{4k} + j(CW^{2k} - GW^{6k})] - \\
&\quad (1+j)[BW^k - FW^{5k} + DW^{3k} - HW^{7k}] \\
R &= [A + EW^{4k} - (CW^{2k} + GW^{6k})] - \\
&\quad j[BW^k + FW^{5k} - (DW^{3k} + HW^{7k})] \\
S &= [A - EW^{4k} - j(CW^{2k} - GW^{6k})] - \\
&\quad (1-j)[BW^k - FW^{5k} - j(DW^{3k} - HW^{7k})]
\end{aligned}
\tag{2.18}
$$

El cálculo de una mariposa de 8 datos precisa dos multiplicaciones complejas; en cambio el cálculo de las mariposas de 2 y 4 datos, no introduce multiplicaciones. Por lo tanto, en el algoritmo base 8 las multiplicaciones se deben tanto al cálculo de las transformadas discretas (representadas por círculos en los esquemas), como a los productos por los coeficientes (factores sobre las flechas).

De lo anterior parece deducirse que el algoritmo base 4 es óptimo. Esto es cierto para una FFT de 64 datos. Sin embargo, no es cierto que la transformada base 4 sea siempre óptima, sino que para una determinada transformada es aconsejable probar distintas bases para encontrar el algoritmo más eficiente. Además, no es lo mismo un algoritmo óptimo secuencial

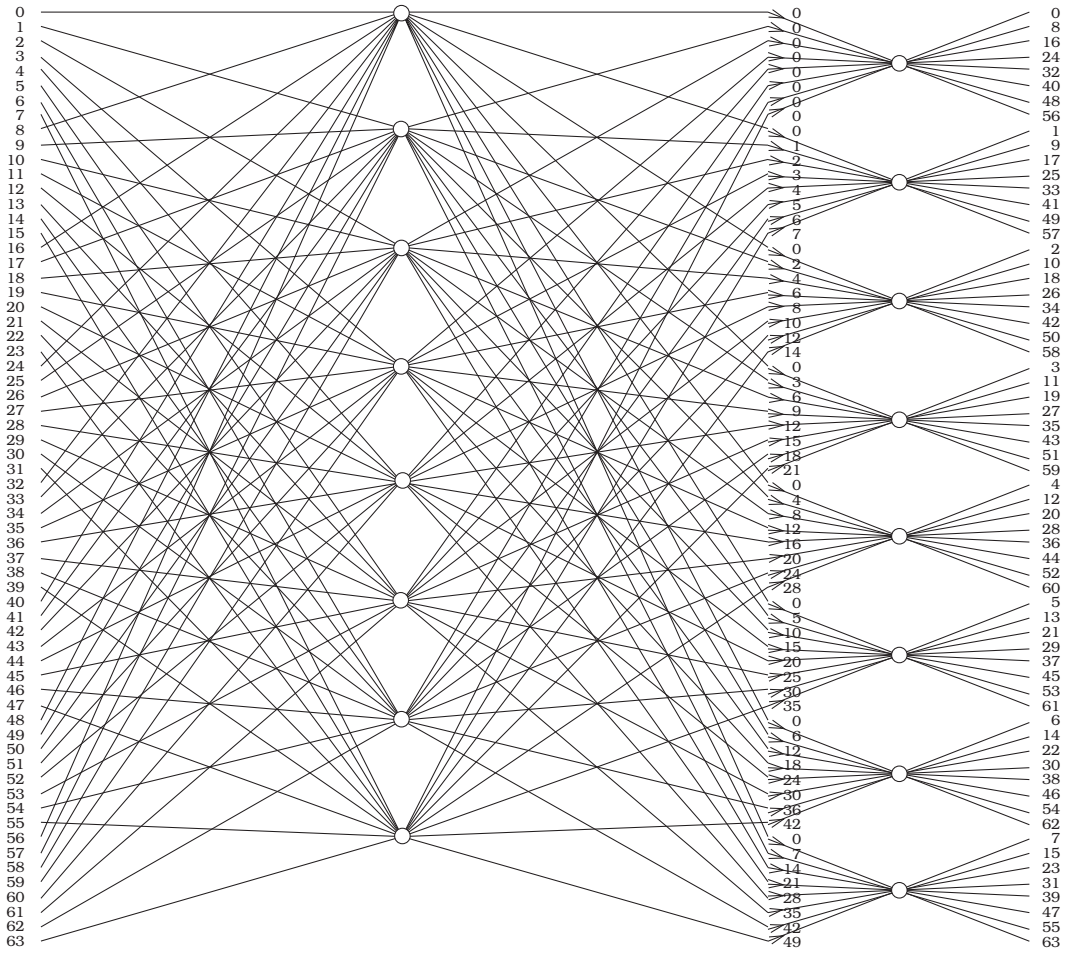


Figura 2.11: Algoritmo FFT base 8 en una transformada de 64 datos usando DEF.

que uno paralelo o vectorizado.

Un algoritmo que conjuga las ventajas de los algoritmos base 2 (flexibilidad) y base 4 (reducido número de multiplicaciones y acceso a memoria) es la FFT *división–base*, inicialmente desarrollada por Duhamel y Hollman [32].

2.3 Los algoritmos FFT

El proceso de descomposición de la transformada de una secuencia en múltiples transformadas sobre secuencias de menor tamaño es la base de todos los algoritmos FFT. Existen, sin embargo, muchos algoritmos FFT para computar la DFT de una determinada secuencia, aunque el resultado de todos ellos sea el mismo. Dependiendo de cómo se organicen las computaciones y el flujo de datos se han diseñado un gran número de algoritmos FFT [84]. Estas diferencias son de especial importancia, pues proveen algoritmos específicos con propiedades atractivas para cada aplicación particular. Clasificaremos los algoritmos FFT según los distintos tipos de flujos de datos y consideraremos algoritmos *in–situ*, *geometría constante* y *en–orden*.

2.3.1 El algoritmo in–situ

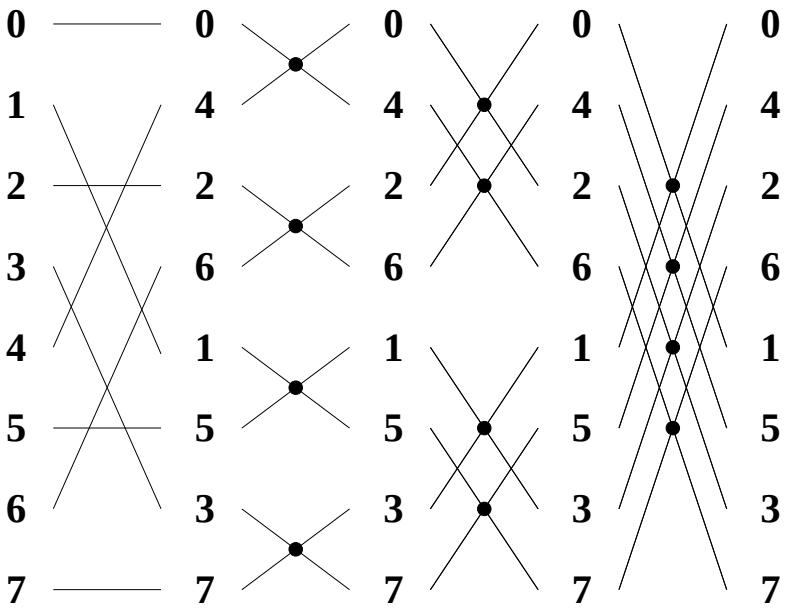
El algoritmo de Cooley–Tukey de FFT [27] es un algoritmo in–situ (ver figura 2.12). Los algoritmos in–situ se caracterizan porque los resultados parciales se escriben sobre los datos empleados. Por esta razón no se necesitan elementos de almacenamiento adicionales y, por tanto, se minimizan los requerimientos de memoria. Pero estos algoritmos FFT necesitan como primer paso una operación dígito–inverso, $\rho_{n,1}$, de la secuencia de entrada y a continuación n etapas de computación, como expresa el siguiente lema

Lema 9 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la cadena de operadores*

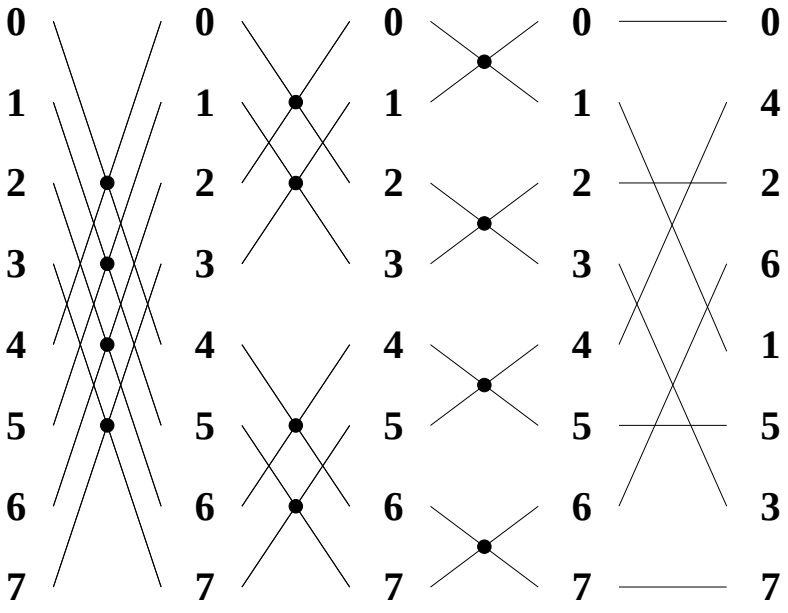
$$\rho_{n,1} \prod_{i=1}^{n} B_i \tag{2.19}$$

o, alternativamente, mediante la cadena,

$$\left[\prod_{i=1}^{n} B_{n-i+1} \right] \rho_{n,1} \tag{2.20}$$



(a)



(b)

Figura 2.12: Flujo de datos del algoritmo FFT in–situ de una transformada de 8 datos, base 2, entrada dígito–inverso (a) y salida dígito–inverso (b).

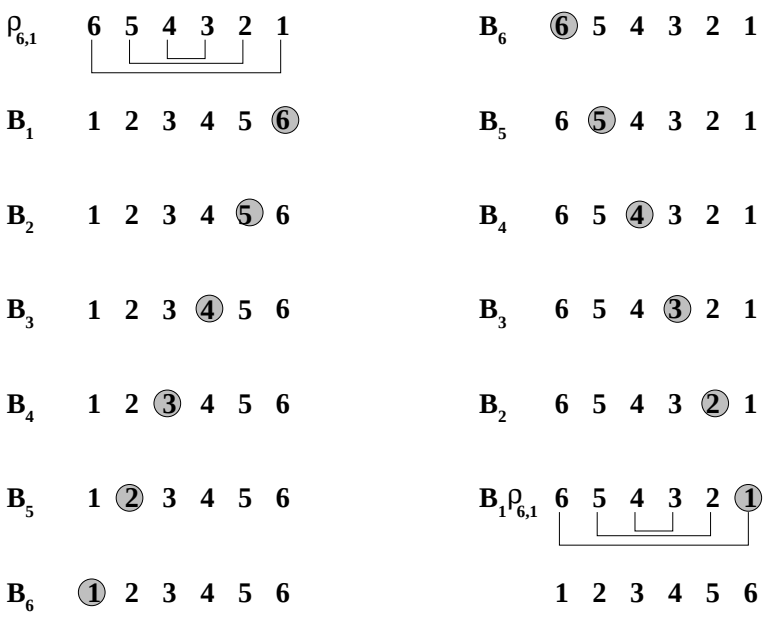


Figura 2.13: Esquema índice-dígito de las redistribuciones de datos en los algoritmos in–situ (lema 9) de longitud $N = r^6$. Los círculos representan al operador mariposa y las líneas al operador dígito–inverso.

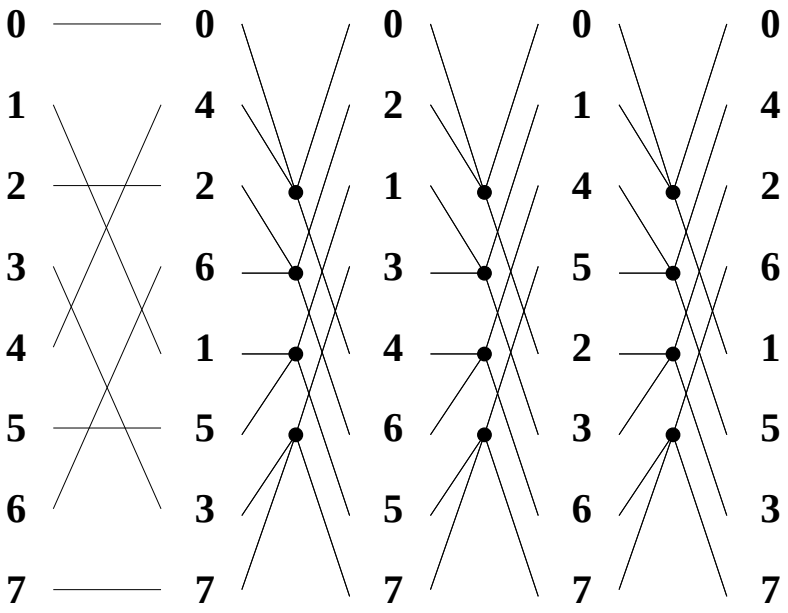
En la figura 2.13 se ilustra esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice–dígito.

2.3.2 El algoritmo con geometría–constante

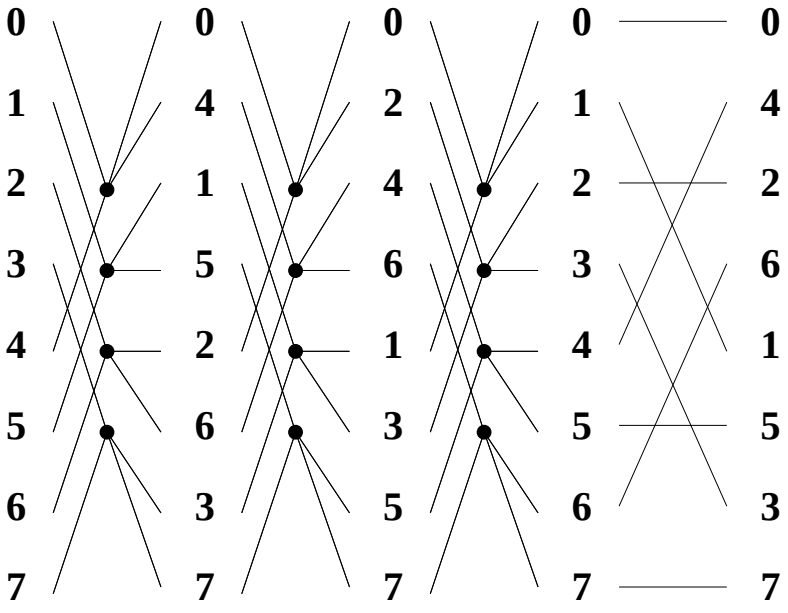
El algoritmo con geometría–constante (ver figura 2.14) propuesto por Pease [80], también necesita realizar la permutación índice–dígito de la secuencia de entrada, pero el resto de sus etapas son idénticas. Su principal desventaja es que no se puede implementar in–situ. Este algoritmo es particularmente adecuado para su implementación en redes de PEs. Al ser las n etapas de computación idénticas, el elemento básico de la arquitectura es una columna de PEs; esta columna puede repetirse n veces para implementar la transformada completa.

El algoritmo se puede expresar por el siguiente lema,

Lema 10 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la*



(a)



(b)

Figura 2.14: Flujo de datos del algoritmo FFT con geometría–constante de una transformada de 8 datos, base 2, entrada dígito–inverso (a) y salida dígito–inverso (b).

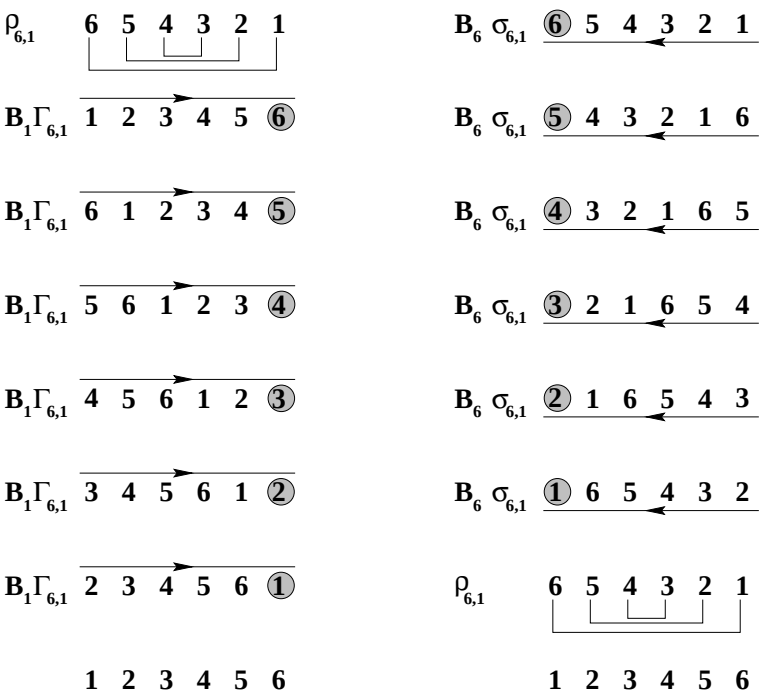


Figura 2.15: Esquema índice-dígito de las redistribuciones de datos en los algoritmos con geometría constante (lema 10) de longitud $N = r^6$. Los círculos representan al operador mariposa, las flechas operadores barajamiento perfecto (flecha a la izquierda) y desbarajamiento perfecto (flecha a la derecha) y las líneas al operador dígito–inverso.

cadena de operadores

$$\rho_{n,1} \prod_{i=1}^{n} B_1 \Gamma_{n,1} \tag{2.21}$$

o, alternativamente, mediante la cadena,

$$\left[\prod_{i=1}^{n} B_n \sigma_{n,1} \right] \rho_{n,1} \tag{2.22}$$

Todas las etapas se describen como $\Gamma_{n,1} B_1$ o $B_n \sigma_{n,1}$ dependiendo de cómo se organicen las entradas. En la figura 2.15 se ilustra esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice–dígito.

2.4 Los algoritmos en-orden. Paralelización

Estos algoritmos, al contrario que las restantes versiones de la FFT, proporcionan la secuencia de salida en el orden correcto (dígito-inverso) respecto a la secuencia de entrada y no requieren barajamiento adicionales. En la figura 2.16.a y 2.16.b se muestra la aplicación de este algoritmo al cálculo de la transformada de una secuencia de 8 datos.

Mostramos a continuación dos versiones de este tipo de algoritmos,

Algoritmo 1 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la cadena de operadores*

$$\prod_{i=1}^{n} B\sigma_{n,i} \tag{2.23}$$

o, alternativamente, mediante la cadena,

$$\prod_{i=1}^{n} \Gamma_{n,n-i+1} B, \tag{2.24}$$

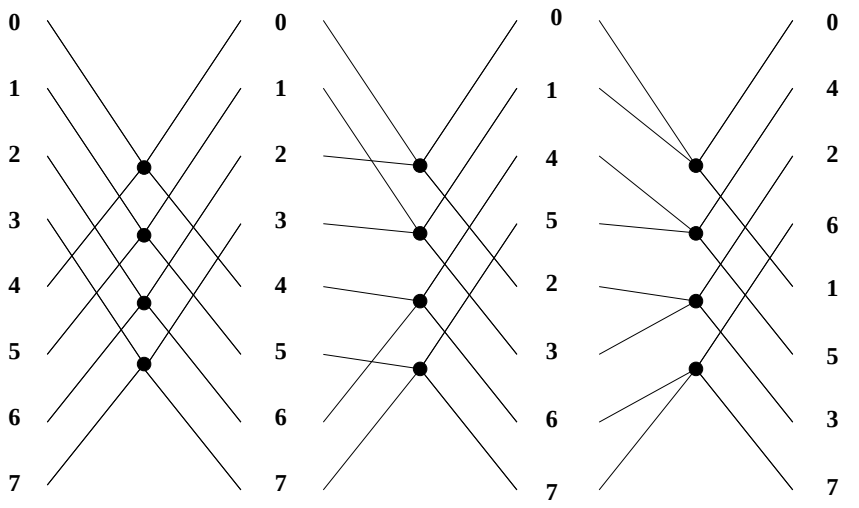
donde $B \equiv B_n$.

En la figura 2.17 ilustramos esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice-dígito.

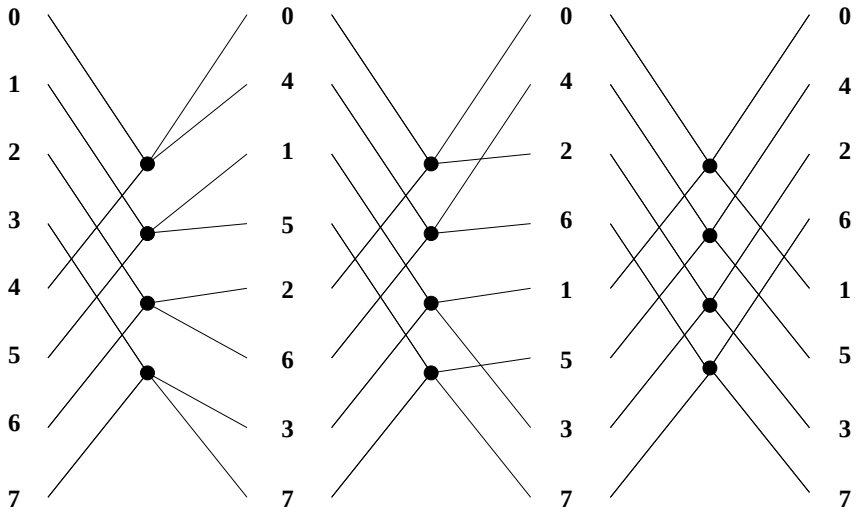
Como ya se indicó, los operadores mariposa representan las operaciones aritméticas que requieren los algoritmos mientras que los restantes operadores, barajamiento y desbarajamiento perfecto, etc. representan las comunicaciones

Los algoritmos en-orden no requieren barajamientos adicionales de los datos. Sin embargo, cuando se implementan sobre una malla, los operadores que aparecen en la ecuación 2.23 implican comunicaciones en diagonal sobre filas y columnas y son difíciles de computar in-situ. Por tanto, estos algoritmos no son muy eficientes. Las comunicaciones pueden mejorarse modificando las redistribuciones de los datos y empleando los operadores de las definiciones 8 y 9 del capítulo 1. Los algoritmos resultantes son los siguientes,

Algoritmo 2 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la cadena de operadores*



(a)



(b)

Figura 2.16: Flujo de datos del algoritmo FFT en-orden para una transformada de 8 datos, base 2, DET (a) y DEF (b).

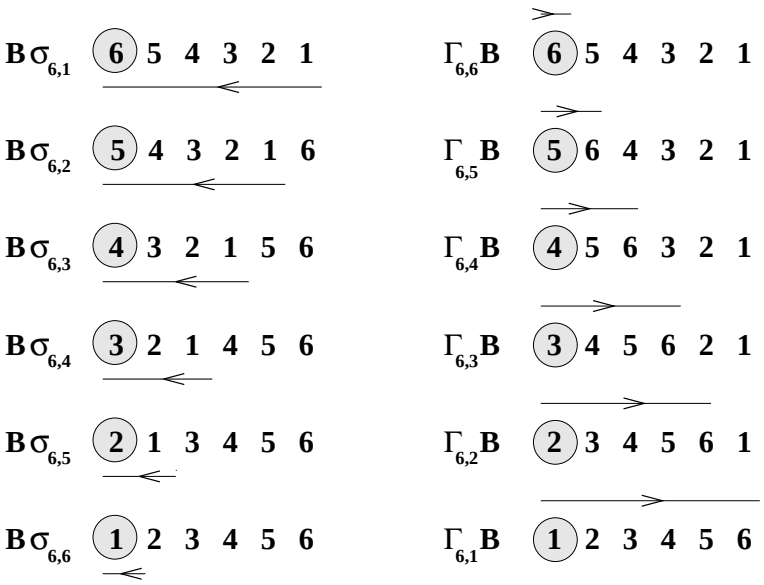


Figura 2.17: Esquema índice-dígito de las redistribuciones de datos en los algoritmos de tipo 1 de longitud $N = r^6$. Los círculos representan al operador mariposa y las flechas a los operadores barajamiento perfecto (flecha a la izquierda) y desbarajamiento perfecto (flecha a la derecha).

$$\prod_{i=1}^{w} E_{w+i,i} \prod_{i=1}^{w} B\sigma_{n,2w+1,w,i} \prod_{i=w+1}^{2w} B\sigma_{n,i} \prod_{i=2w+1}^{n} B\sigma_{n,i} \tag{2.25}$$

o, alternativamente, mediante la cadena,

$$\prod_{i=1}^{n-2w} \Gamma_{n,n-i+1} B \prod_{i=n-2w+1}^{n-w} \Gamma_{n,n-i+1} B \prod_{i=n-w+1}^{n} \Gamma_{n,2w+1,w,n-i+1} B \prod_{i=1}^{w} E_{i,w+i} \tag{2.26}$$

Demostración Demostraremos sólo la primera expresión. Partiendo del primer algoritmo, ecuación (2.23) y usando la relación obtenida en el lema 6 (Capítulo 1), tenemos

$$\begin{aligned} \prod_{i=1}^{n} B\sigma_{n,i} &= \prod_{i=1}^{w} B\sigma_{n,i} \prod_{i=w+1}^{2w} B\sigma_{n,i} \prod_{i=2w+1}^{n} B\sigma_{n,i} \\ &= \prod_{i=1}^{w} E_{w+i,i} \prod_{i=1}^{w} B\sigma_{n,2w+1,w,i} \prod_{i=w+1}^{2w} B\sigma_{n,i} \prod_{2w+1}^{n} B\sigma_{n,i} \end{aligned} \tag{2.27}$$

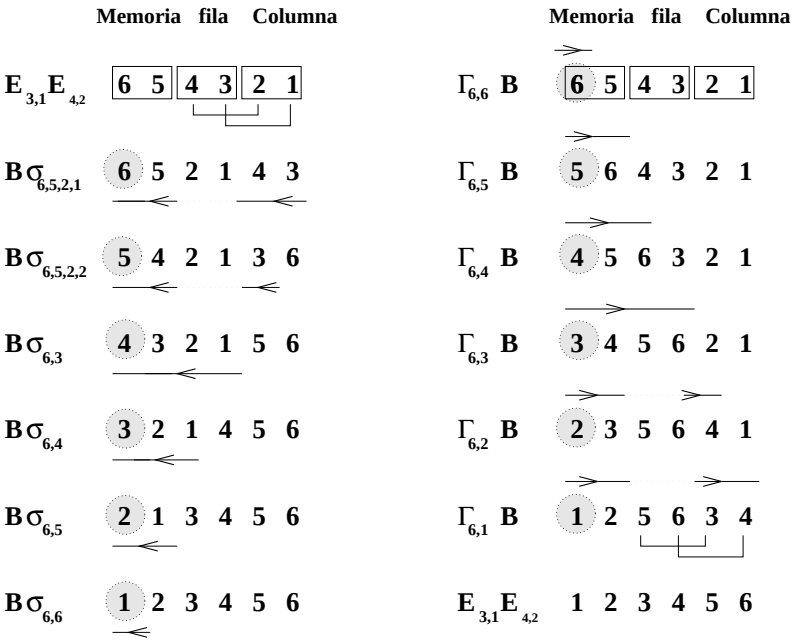


Figura 2.18: Esquema índice-dígito de las redistribuciones de datos en los algoritmos de tipo 2 de longitud $N = r^6$. Los círculos representan al operador mariposa y las flechas a los operadores barajamiento perfecto y desbarajamiento perfecto.

en donde hemos tenido en cuenta que B conmuta con $E_{w+i,i}$. □

Estos algoritmos sólo están definidos cuando la malla es cuadrada (suponemos $v = w$). Requieren calcular la transpuesta de la matriz de datos y, a continuación, n etapas de computación/comunicación. Considerando la ecuación (2.25), el primer término producto representa la transposición de los datos entre filas y columnas de la malla y requiere comunicaciones en diagonal. Alternativamente, esta transposición puede conseguirse escribiendo los datos en la malla por filas y leyendo los resultados por columnas. De las n etapas de computación/comunicación, w etapas (segundo término producto) implican comunicaciones en paralelo sobre las filas. Las siguientes $v = w$ etapas (tercer término producto), implican comunicaciones en paralelo sobre las columnas y, por último, las u etapas restantes (cuarto término producto), son barajamientos de datos dentro de la memorias locales y no implican ningún tipo de comunicaciones. En la figura 2.18 ilustramos esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice–dígito.

En general, las permutaciones barajamiento perfecto y desbarajamiento perfecto requieren bastantes comunicaciones y son difíciles de computar en-

orden. El intercambio de datos involucra a varios PEs ya que varios dígitos de la dimensión *memoria* se desplazan a la dimensión *fila* o a la dimensión *columna*. A continuación veremos otros algoritmos FFT basados en la permutación intercambio, permutación más fácil de implementar.

Algoritmo 3 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la cadena de operadores*

$$\rho_{n-1,1} \prod_{i=1}^{n} BE_{n,i} \tag{2.28}$$

o, alternativamente, mediante la cadena,

$$\left(\prod_{i=1}^{n} E_{n,n-i+1} B \right) \rho_{n-1,1} \tag{2.29}$$

Demostración Demostraremos sólo la primera expresión. Partiendo del primer algoritmo, ecuación (2.23) y usando la relación $\sigma_{n,i} = \sigma_{n-1,i} E_{n,i}$ obtenida en el lema 3 (Capítulo 1), tenemos,

$$\begin{aligned} \prod_{i=1}^{n} B\sigma_{n,i} &= \prod_{i=1}^{n} B\sigma_{n-1,i} E_{n,i} \\ &= \prod_{i=1}^{n-1} \sigma_{n-1,i} \prod_{i=1}^{n} BE_{n,i} \\ &= \rho_{n-1,1} \prod_{i=1}^{n} BE_{n,i} \end{aligned} \tag{2.30}$$

en donde hemos usado el lema 7 (Capítulo 1) y tenido en cuenta que B conmuta con $\sigma_{n-1,i}$. □

En estos algoritmos podemos distinguir n etapas de computación/comunicación y un barajamiento de datos según la permutación $\rho_{n-1,1}$. El flujo de datos en estas n etapas se define mediante el operador intercambio: u de estas etapas no requieren comunicaciones, v etapas requieren comunicaciones en paralelo a través de las columnas y w etapas requieren comunicaciones en paralelo a través de las filas. Dependiendo del algoritmo, el barajamiento de datos $\rho_{n-1,1}$ puede realizarse sobre la secuencia de entrada o sobre la secuencia de salida. Este barajamiento implica comunicaciones en

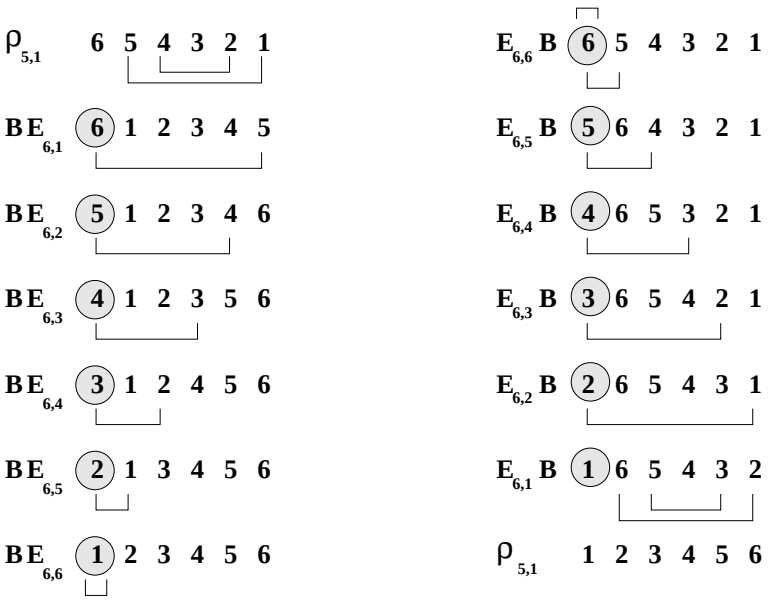


Figura 2.19: Esquema índice-dígito de las redistribuciones de datos en los algoritmos de tipo 3 de longitud $N = r^6$. Los círculos representan al operador mariposa y las líneas al operador intercambio.

diagonal sobre filas y columnas. El algoritmo dado por la ecuación (2.29) es equivalente al propuesto en [36].

En general, las comunicaciones basadas en el operador intercambio son más eficientes que las basadas en el operador barajamiento perfecto. El operador intercambio sólo transfiere un dígito de la dimensión *memoria* a la dimensión *fila* o a la dimensión *columna*, por lo que el número de PEs que intercambian datos entre sí es menor (r PEs para un algoritmo base r). Además el algoritmo puede computarse en-orden fácilmente. En la figura 2.19 ilustramos esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice-dígito.

El siguiente algoritmo, también basado en la permutación intercambio, no requiere ningún barajamiento adicional de los datos que implique comunicaciones en diagonal entre filas y columnas,

Algoritmo 4 *La FFT base r y longitud $N = r^n$ puede expresarse mediante la cadena de operadores*

$$\prod_{i=1}^{p} BE_{n,i}E_{n,n-i} \prod_{i=p+1}^{n} BE_{n,i} \tag{2.31}$$

o, alternativamente, mediante la cadena,

$$\prod_{i=1}^{n-p} E_{n,n-i+1}B \prod_{i=n-p+1}^{n} E_{n,i-1}E_{n,n-i+1}B \tag{2.32}$$

en ambos casos $p = n/2 - 1$ *si* n *es par o* $p = \lfloor n/2 \rfloor$ *si* n *es impar.*

Demostración Demostraremos sólo la primera expresión. Si n es par, entonces $\rho_{n-1,1} = E_{n-1,1}E_{n-2,2}\cdots E_{n/2+1,n/2-1}$; y si n es impar, $\rho_{n-1,1} = E_{n-1,1}E_{n-2,2}\cdots E_{\lfloor n/2\rfloor+1,\lfloor n/2\rfloor}$. Suponiendo que n es par y partiendo del algoritmo 3,

$$\begin{aligned}
\rho_{n-1,1}\prod_{i=1}^{n} BE_{n,i} &= \prod_{i=1}^{n/2-1} E_{n-i,i}\prod_{i=1}^{n} BE_{n,i} \\
&= \prod_{i=1}^{n/2-1} BE_{n-i,i}E_{n,i}\prod_{i=n/2}^{n} BE_{n,i} \\
&= \prod_{i=1}^{n/2-1} BE_{n,i}E_{n,n-i}\prod_{i=n/2}^{n} BE_{n,i}
\end{aligned} \tag{2.33}$$

en donde hemos tenido en cuenta que B conmuta con $E_{n-i,i}$ y que $E_{n-i,i}E_{n,i} = E_{n,i}E_{n,n-i}$ (lema 2, Capítulo 1). Si n es impar, la demostración es similar. □

Estos algoritmos tienen las dos características deseables, pues son en-orden e in-situ. Son en-orden puesto que requieren realizar únicamente n etapas de computación/comunicación, sin necesidad de barajamientos adicionales. Son in-situ puesto que los operadores que definen las redistribuciones de datos son intercambio. Algunas de las etapas incluyen dos operadores intercambio que implican respectivamente comunicaciones en paralelo sobre filas y columnas. En la figura 2.20 ilustramos esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice–dígito. Un algoritmo similar al de la ecuación (2.32) se implementa en [108] sobre un computador hipercubo, aunque puede adaptarse fácilmente para una malla.

A continuación presentamos otro algoritmo de la FFT basado en la permutación intercambio que no necesita ninguna permutación adicional entre PEs.

Algoritmo 5 *La FFT base* r *y longitud* $N = r^n$ *puede expresarse mediante la cadena de operadores*

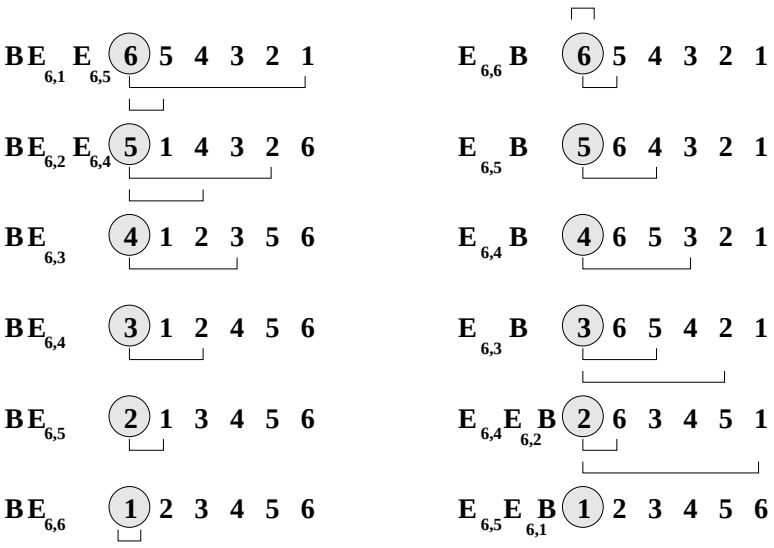


Figura 2.20: Esquema índice-dígito de las redistribuciones de datos en los algoritmos de tipo 4 de longitud $N = r^6$. Los círculos representan al operador mariposa y las líneas al operador intercambio.

$$\left[\prod_{i=1}^{v+w} B_{n-i+1}E_{n-i+1,i}\right] \rho_{n-v-w,v+w+1} \left[\prod_{i=v+w+1}^{n} B_i\right] \tag{2.34}$$

o, alternativamente, mediante la cadena,

$$\left[\prod_{i=1}^{n-v-w} B_{n-i+1}\right] \rho_{n-v-w,v+w+1} \left[\prod_{i=n-v-w+1}^{n} E_{i,n-i+1}B_{n-v-w+i}\right] \tag{2.35}$$

en ambos casos $n-v-w > v+w$*, es decir,* $n > 2(v+w)$*.*

Demostración

Demostraremos sólo la primera expresión. Suponiendo que n es par y que $v + w = n/2 - 1$ y partiendo del algoritmo 3.

$$\begin{aligned} \rho_{n-1,1}\prod_{i=1}^{n} BE_{n,i} &= \prod_{i=1}^{n/2-1} E_{n-i,i}\prod_{i=1}^{v+w} BE_{n,i}\prod_{i=v+w+1}^{n} BE_{n,i} \\ &= \prod_{i=1}^{v+w} BE_{n,i}E_{n,n-i}E_{n,n-i}\prod_{i=v+w+1}^{n}\prod_{i=v+w+1}^{n} B_i \end{aligned} \tag{2.36}$$

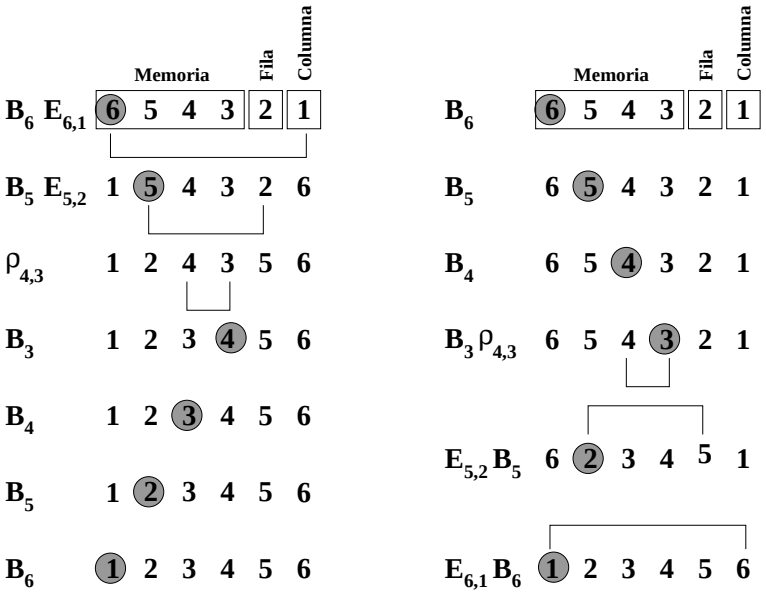


Figura 2.21: Esquema índice-dígito de las redistribuciones de datos en los algoritmos de tipo 5 de longitud $N = r^6$. Los círculos representan al operador mariposa y las líneas al operador intercambio.

$$= \left[\prod_{i=1}^{v+w} B_{n-i+1} E_{n-i+1,i}\right] \rho_{n-v-w,v+w+1} \prod_{i=v+w+1}^{n} B_i \tag{2.37}$$

en donde he tenido en cuenta los lemas 1 y 2 de Capítulo 1 y que B conmuta con $E_{n-i,i}$. Si n es impar, la demostración es similar. □

En estos algoritmos podemos distinguir n etapas de computación/comunicación y una permutación $\rho_{n-v-w,v+w+1}$ que implica realizar una permutación índice–dígito entre los datos locales de cada PE sin ninguna comunicación entre PEs. El flujo de datos en estas n etapas se define mediante el operador intercambio: u de estas etapas no requieren comunicaciones, v etapas requieren comunicaciones en paralelo a través de las columnas y w etapas requieren comunicaciones en paralelo a través de las filas. El barajamiento de datos $\rho_{n-v-w,v+w+1}$ se realiza sólo con datos locales en cada PE sin necesidad de ninguna comunicación entre PEs. En la figura 2.21 ilustramos esquemáticamente el flujo de datos de estos algoritmos utilizando la representación índice–dígito. El único problema que presenta este tipo de algoritmos es que sólo son válidos si $n > 2(v + w)$.

Los algoritmos 1 a 5 se basan en la realización de una secuencia de etapas

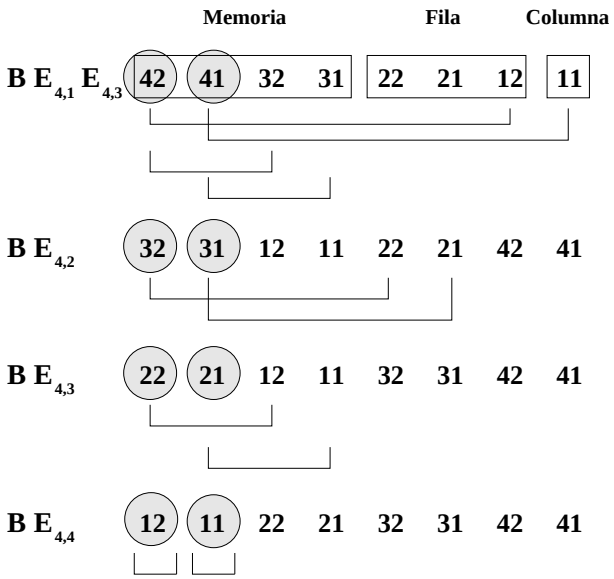


Figura 2.22: Esquema índice-dígito de las redistribuciones de datos en un algoritmo de tipo 4 base 4 en función de operaciones base 2 de longitud 256 sobre una malla de tamaño 8×2. Los círculos representan al operador mariposa y las líneas a los operadores intercambio.

de computación/comunicación. Esta no es la única forma posible de organizar la FFT. Otra posibilidad es realizar un conjunto de etapas de computación hasta agotar los datos disponibles en las memorias locales de los PEs (bloques de tamaño $N/(V \times W)$) y realizar a continuación un conjunto de etapas de comunicación. Este esquema se repite hasta completar el conjunto de etapas de la FFT. Esta técnica es equivalente a considerar un algoritmo base $r = N/(V \times W)$.

Incrementar la base no implica complicar los algoritmos. El lema 8 establece la expresión de los operadores base r^2 en función de los operadores base r. Este lema nos permitirá implementar algoritmos sobre mallas en las que el número de filas y columnas de PEs no es múltiplo de la base. En la figura 2.22 mostramos esquemáticamente el flujo de datos de un algoritmo de tipo 4 base 4 en función de operaciones base 2 sobre una malla de tamaño 8×2.

2.5 Evaluación e implementación de la transformada rápida de Fourier

En el apartado anterior hemos formulado diez algoritmos FFT que hemos agrupado en grupos de dos. El segundo algoritmo de cada grupo se obtiene mediante la reversión (implementa las mismas operaciones en orden inverso) de cada una de las operaciones realizadas en el primero y, por tanto, sus características son similares. Estos diez algoritmos son los siguientes,

1. Dos algoritmos basados respectivamente en las permutaciones barajamiento perfecto y desbarajamiento perfecto, de tipo no in-situ, en–orden, y que requieren comunicaciones en diagonal sobre filas y columnas, (Algoritmo 1).

2. Dos algoritmos basados respectivamente en las permutaciones barajamiento perfecto y desbarajamiento perfecto, de tipo no in-situ, que requieren la transposición de las filas y columnas de datos, y con comunicaciones en paralelo sobre filas y columnas, (Algoritmo 2).

3. Dos algoritmos basados en la permutación intercambio, de tipo in-situ, que requieren la permutación de datos de acuerdo a la permutación dígito–inverso, bien sobre la secuencia de entrada, bien sobre la secuencia de salida, y con comunicaciones en paralelo sobre filas y columnas, (Algoritmo 3).

4. Dos algoritmos basados en la permutación intercambio, de tipo in-situ, en–orden, y con comunicaciones en paralelo sobre filas y columnas, (Algoritmo 4).

5. Dos algoritmos basados en la permutación intercambio, de tipo in–situ, en–orden, que requieren una permutación índice–dígito entre los datos locales de cada PE, y con comunicaciones en paralelo sobre filas y columnas, (Algoritmo 5).

En la tabla 2.1 resumimos el número de comunicaciones y de posiciones de memoria requeridas por los algoritmos. En esta tabla hemos supuesto algoritmos base 2 ($N = 2^n$) y mallas cuadradas ($W \times W$). El número de comunicaciones lo hemos obtenido sumando las transferencias celda a celda requeridas por cada dato en las etapas de computación/comunicación. De aquí se deduce que los algoritmos 4 y 5, en general, serán los más eficientes.

Algoritmo	*Comunicaciones*	*Permutaciones adicionales*	*Memoria*
1	$(2v+w)\frac{U}{2}$	–	$2N$
2	$2w\frac{U}{2}$	$\prod_{i=1}^{w} E_{w+i,i}$	$2N$
3	$(u+v)2^{n-1}$	$\rho_{n-1,1}$	N
4	$(u+v)2^{n-1}+$ $\frac{U}{2}(2w-n-p-1)^*$	–	N
5	$(u+v)2^{n-1}$	–	N

Tabla 2.1: Número de comunicaciones y de celdas de memoria requeridas por los algoritmos base 2 ($N = 2^n$) sobre mallas cuadradas ($W \times W$, $W = 2^w$). (*) Este término sólo existe si $2w > n - p - 1$ ($p = n/2 - 1$ si n es par o $p = \lfloor n/2 \rfloor$ si n es impar).

2.5.1 Los coeficientes trigonométricos

Existen varios métodos para el cálculo de los coeficientes trigonométricos: cálculo directo, tabla look–up, permutación y el método de Tong–Swarztrauber, entre otros.

- *Cálculo directo.* Los coeficientes trigonométricos se calculan directamente mediante la ecuación $W_N^k = e^{-j(2\pi/N)k}$. El cálculo de las funciones trigonométricas en cada paso consume mucho tiempo, particularmente cuando la FFT sólo requiere unas pocas operaciones.

- *Tabla look–up.* Los coeficientes trigonométricos son precomputados y almacenados en cada PE. Esto tiene la ventaja de que los coeficientes trigonométricos son utilizados sin recálculo en cada uno de los pasos de la FFT. Este esquema tiene una implementación sencilla pero requiere una gran cantidad de memoria (proporcional a $r^{n-1}\log_r N$) en cada uno de los P PEs. Se puede mejorar la utilización de la memoria a $r^{n-v-w-1}\log_r N$, observando que no todos los PEs necesitan todos los coeficientes trigonométricos. Por ejemplo, para $N = 16$ y $P = 4$, los coeficientes trigonométricos necesarios en cada PE se muestran en la figura 2.23.

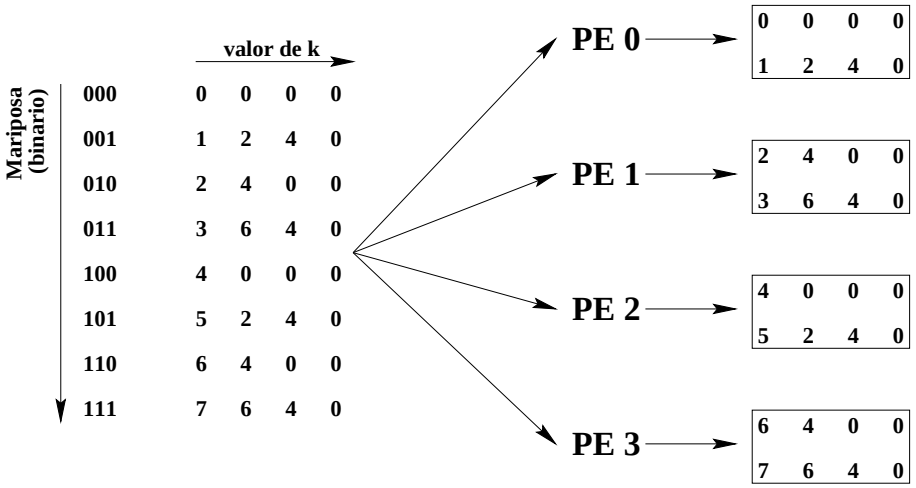


Figura 2.23: Se muestran las tablas look–up necesaria en cada uno de los PEs. Los enteros en cada columna corresponden al factor trigonométrico de W^k en cada una de las etapas.

- *Permutación.* Inicialmente, se distribuyen los coeficientes entre los PEs de acuerdo con los cálculos necesarios para el primer paso. En los

subsiguientes pasos, la mitad de los coeficientes son permutados como se ve en la figura 2.24 para el caso $N = 32$ y $P = 4$.

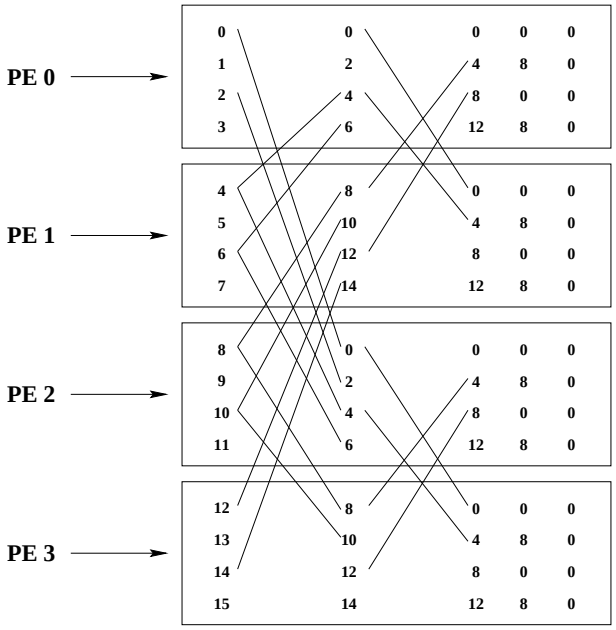


Figura 2.24: Los factores trigonométricos para DEF de un FFT de $N = 32$ datos sobre 4 PEs. Las líneas indican los factores trigonométricos que necesitan transferirse a otro PE.

A partir de la etapa $\log_r P$ no hace falta realizar más comunicaciones para la permutación de los coeficientes. Este esquema puede ser muy ineficiente sobre máquinas paralelas debido al coste de las comunicaciones.

Ninguno de estos métodos es completamente sastifactorio sobre computadores paralelos si la memoria es limitada y la comunicación costosa. La solución que propone [108] es añadir unas pocas operaciones adicionales en cada etapa de forma que los coeficientes trigonométricos puedan computarse en paralelo sin ninguna comunicación. Si consideramos el ejemplo de una FFT de longitud $N = 16$ y suponemos que la mariposa i es mapeada al PE i, entonces los factores trigonométricos necesarios son los mostrados en la figura 2.23. Se puede ver que los enteros en cada columna son dos veces $(mod\ N/2j)$ los enteros de la columna previa con lo que estos coeficientes pueden computarse de la siguiente forma

$$\cos 2\theta \quad = \quad \cos^2\theta - \sin^2\theta$$

$$\sin 2\theta \quad = \quad 2\cos\theta \cdot \sin\theta \tag{2.38}$$

con la salvedad de que los ángulos comprendidos en el rango $\pi \leq 2\theta < 2\pi$ se obtienen sumando π para que funcione dicha regla. Se pueden calcular los coeficientes trigonométricos de la etapa actual a través de los correspondientes del paso previo mediante cuatro multiplicaciones y una adición (o tres multiplicaciones y dos adiciones). Este método puede aplicarse sólo a la DEF, no funciona con la DET. Con este método se disminuye el tiempo de computación de los coeficientes trigonométricos y la memoria necesaria para su almacenamiento.

Nosotros hemos utilizado este método para calcular los coeficientes. Los coeficientes trigonométricos que requieren las computaciones son precomputados y almacenados en cada PE. La tabla de coeficientes se genera utilizando el método de [108] visto previamente.

2.5.2 Resultados experimentales

Para comprobar el funcionamiento de los algoritmos hemos implementado los códigos FFT en el multiprocesador AP1000 de Fujitsu [40]. Hemos utilizado distintos tamaños de la secuencia de entrada, considerando datos de tipo punto flotante y simple precisión. En la figura 2.25 se muestran los tiempos de ejecución para diferentes algoritmos en–orden. En esta figura se puede ver los tiempos de ejecución para diferentes tamaños de la secuencia de entrada para 4 PEs. Se muestra el algoritmo 4 (ecuación (2.32)) con cálculo directo de los coeficientes trigonométricos, que llamaremos algoritmo 4.1. El mismo algoritmo pero calculando los coeficientes trigonométricos por el método de Tong–Swarztrauber, algoritmo 4.2. También se muestra los tiempos de ejecución para los algoritmos 2 (ecuación (2.26)) y 5 (ecuación (2.35)), que son los algoritmos más significativos.

Vemos que los algoritmos 5 y 4.2 son, como era de esperar, los que mejores resultados proporcionan. El algoritmo 2 es el que peor resultados ofrece con el inconveniente añadido de que no es in–situ, necesitando $2N$ posiciones de memoria. Por lo comentado en el apartado anterior es preferible el algoritmo 4.2 al 5.

En la tabla 2.2 se resumen los tiempos medidos para distintos tamaños de la FFT tipo 4 (ecuación (2.32)) y de la malla. Este algoritmo es de los más eficientes puesto que es in–situ, con lo que evita duplicar la cantidad de memoria que requiere el almacenamiento de los datos, y en–orden, lo que evita la necesidad de operaciones adicionales de reordenamiento.

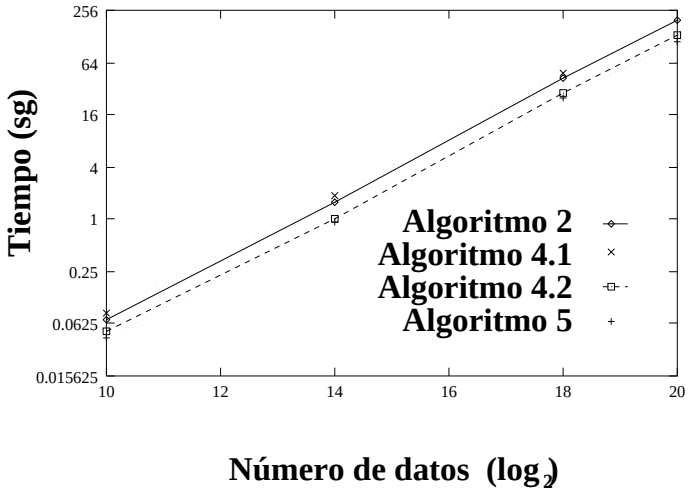


Figura 2.25: Tiempos de ejecución para 4 PEs de los algoritmos en–orden de tipo 4 (con cálculo directo de los coeficientes (4.1) y con cálculo de los coeficientes por el método de Tong–Swarztrauber (4.2)) y de los algoritmos tipo 2 y 5.

	Número de datos			
Número de PEs	2^{14}	2^{16}	2^{18}	2^{20}
1	4.46	25.42	116.36	–
2	2.01	12.66	57.87	–
4	1.00	5.04	28.83	130.14
8	0.50	2.27	14.22	64.76
16	0.25	1.13	5.63	32.26
32	0.13	0.56	2.54	15.83
64	0.07	0.28	1.27	6.25
128	0.03	0.14	0.64	2.83
256	0.02	0.08	0.32	1.41
512	0.01	0.04	0.16	0.71

Tabla 2.2: Tiempos de ejecución (en segundos) medidos sobre el AP1000.

En la figura 2.26.a vemos los tiempos de ejecución para $N = 2^{14}$, $N = 2^{16}$ y $N = 2^{18}$ datos. En la figura 2.26.b mostramos la aceleración obtenida para estas secuencias de datos. Observamos que para $N = 2^{16}$ y para $N = 2^{18}$ se consigue una sobreaceleración para todos los PEs llegando a alcanzar un valor de 627.76 y 710.37 sobre 512 PEs respectivamente. El mayor aumento de aceleración relativa, ver figura 2.26.c, se consigue cuando cada PE contiene 2^{14} datos, valor que coincide con el tamaño de la cache contenida dentro de cada PE ($2^{14} \cdot 8$ bytes = 128 Kbytes). Las eficiencias de este algoritmo se muestran en la figura 2.26.d. El algoritmo para $N = 2^{14}$ no obtiene resultados tan excelentes como para $N = 2^{16}$ y $N = 2^{18}$ ya que todavía se tiene poca granularidad por PE. Se alcanza la mayor eficiencia en este algoritmo cuando el volumen de comunicación es menor de 2^{12} datos lo que se alcanza para $N = 2^{16}$ y $N = 2^{18}$ con 8 y 32 PEs respectivamente. A partir de esa configuración de la malla, la eficiencia permanece constante con una leve tendencia decreciente sobre un valor de 1.40 hasta que la granularidad comienza a disminuir por PE y el tiempo de comunicación aumenta. Estos resultados muestran que el algoritmo implementado proporciona un excelente rendimiento sobre el computador AP1000.

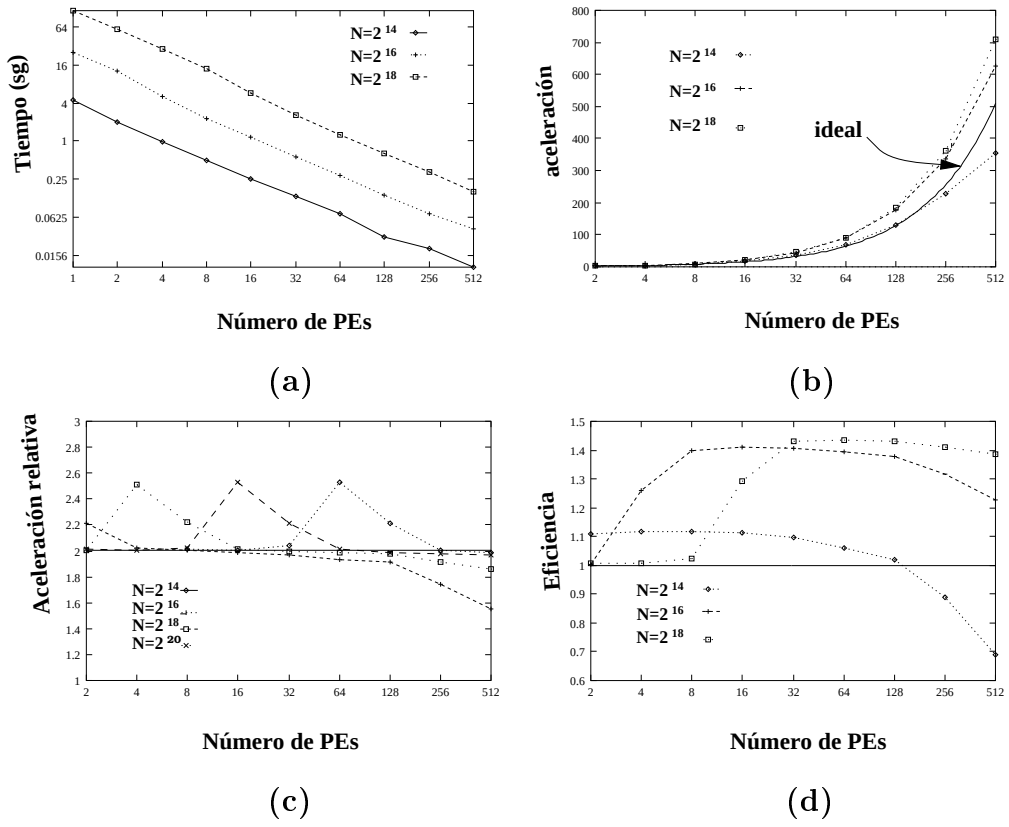


Figura 2.26: Resultados experimentales del algoritmo FFT paralelo de tipo 4 (ecuación (2.32)) sobre el AP1000. (a) Tiempos en segundos. (b) Aceleración. (c) Aceleración relativa. (d) Eficiencia.

Capítulo 3

Sistemas tridiagonales

3.1 Introducción

Los fenómenos que ocurren en la naturaleza vienen descritos por modelos matemáticos. Normalmente estos modelos matemáticos son sistemas complejos de ecuaciones diferenciales ordinarias o en derivadas parciales. Además, la mayoría de los fenómenos físicos son no lineales por lo que vienen descritos por ecuaciones no lineales. La resolución de estas ecuaciones muy a menudo puede reducirse a la evaluación de un sistema de tipo tridiagonal.

A modo de ejemplo veamos como el método de las diferencias finitas para la resolución de ecuaciones en derivadas parciales genera un sistema tridiagonal. El método de diferencias finitas involucra la discretización de ecuaciones diferenciales y posteriormente resolver el sistema de ecuaciones que genera. El espacio–tiempo continuo se sustituye por una malla espacio–tiempo de paso espacial h y paso temporal τ. Una ecuación en diferencias presenta como incógnitas la función $F(x,t)$ definida en los puntos de la malla $(x_i^h, t_n) = (ih, n\tau)$, donde x representa el espacio y t representa el tiempo. Las ecuaciones en diferencias dan lugar a sistemas de ecuaciones tridiagonales (ST). Este método es usado para resolver ecuaciones diferenciales en derivadas parciales tales como las ecuaciones de Helmholtz, Poisson, Laplace, Schrödinger y las ecuaciones de difusión.

En concreto vamos a ver un problema de enorme relevancia física cuyo modelo propuesto es *la ecuación de Schrödinger*:

$$i\frac{\partial F}{\partial t} + \frac{\partial^2 F}{\partial x^2} + V(F) = 0 \tag{3.1}$$

donde $V(F)$ es una función que puede ser lineal o no lineal en F. El estudio

de esta ecuación y su posterior simulación es muy importante en muchos campos de la física y la ingeniería. A modo de ejemplo, la simulación de la propagación de un solitón en una fibra óptica ($V(F) = a|F|^2F$) puede encontrarse en [72].

De los numerosos métodos de discretización o esquemas numéricos existentes, usamos el esquema en diferencias finitas de Zhang–Vazquez [73]. Aplicando este esquema sobre la ecuación no lineal de Schrödinger en una dimensión, se obtiene la siguiente ecuación en diferencias finitas:

$$\begin{array}{l} \frac{-i}{}\frac{F_i^{n+1}-F_i^{n-1}}{2\tau} = \\ -\frac{1}{2h^2}\left(F_{i+1}^{n+1} - 2F_i^{n+1} + F_{i-1}^{n+1} + F_{i+1}^{n+1} - 2F_i^{n-1} + F_{i-1}^{n-1}\right) \\ -\frac{a}{2}\left|F_i^n\right|^2\left(F_i^{n+1} + F_i^{n-1}\right) \end{array} \tag{3.2}$$

donde h y τ son los intervalos espacial y temporal respectivamente y $F_i^n \equiv F(n\tau, ih)$ [35]. Este esquema es implícito lineal. Esto significa que en cada instante temporal $n+1$ es necesario resolver un sistema de ecuaciones lineales para calcular F_i^{n+1}, $i = 0, \cdots, N-1$. Para calcular la función en el instante $n+1$ es necesario conocer la función en los instantes n y $n-1$. En particular, para calcular la función en el segundo instante son necesarios los valores de la función en los instantes primero e inicial. Para simular el primer paso temporal se usa otro esquema numérico: el esquema de Crank y Nicholson [106]. Ordenando adecuadamente, la expresión anterior se transforma en la ecuación tridiagonal:

$$F_{i-1}^{n+1} - a_i F_i^{n+1} + F_{i+1}^{n+1} = d_i \tag{3.3}$$

y el sistema de ecuaciones a resolver en el instante $n+1$ es

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdot & 0 \\ 1 & -a_1 & 1 & 0 & \cdot & 0 \\ 0 & 1 & -a_2 & 1 & \cdot & 0 \\ & & & \ddots & & \\ 0 & 0 & 0 & 1 & -a_{N-2} & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} F_0^{n+1} \\ F_1^{n+1} \\ F_2^{n+1} \\ \vdots \\ F_{N-2}^{n+1} \\ F_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-2} \\ d_{N-1} \end{pmatrix} \tag{3.4}$$

Es decir, en cada paso temporal $n+1$ hay que resolver un sistema de ecuaciones $A\mathbf{u} = \mathbf{d}$, donde $\mathbf{u}$ es el vector de los valores de la función solución en cada punto de la malla en el instante temporal actual y las matrices A y $\mathbf{d}$

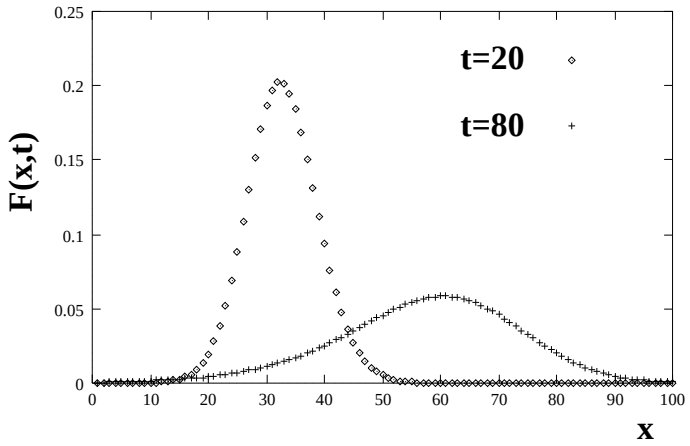


Figura 3.1: Evolución en el tiempo de un paquete de onda según la ecuación de Schrödinger por el método de diferencias finitas.

son funciones de h, τ y de los valores de F en los pasos n y $n-1$. En la figura 3.1 vemos la evolución de un paquete de ondas en el tiempo.

El método de las diferencias finitas se generaliza cuando se consideran varias dimensiones espaciales. En este caso se obtienen sistemas pentadiagonales o heptadiogonales, pero existen métodos para transformar estos sistemas en tridiagonales.

Existen muchos algoritmos directos para resolver ST, como la eliminación gaussiana, que es el algoritmo secuencial más eficiente sobre computadores escalares, la factorización LU o la reducción cíclica (RC). Con el desarrollo de la computación paralela, han surgidos nuevos algoritmos para resolver ST convenientes para estos computadores paralelos. Estos algoritmos han sido ampliamente discutidos en la bibliografía. Encontramos nuevos métodos para incrementar el paralelismo de los algoritmos típicamente secuenciales, como el PARA-RC [55], doblamiento recursivo [34], el método de Lin y Chung [66] y el de Lin y Chen [67]. Por otra parte, se han creados nuevos algoritmos que exploten el paralelismo de la estrategia divide–y–vencerás: el método del doblamiento sucesivo de Wang y Mou [113, 2], el método de desacoplamiento recursivo de Spaletta y Evans [101], el método de Particionamiento Solapado (OPM) de Larriba y col. [65, 62] y un grupo de algoritmos llamados algoritmos híbridos. Los algoritmos híbridos están basados en el particionamiento del sistemas en bloques de ecuaciones, usando un algoritmo local para reducir el subsistema en cada bloque y un algoritmo global para resolver el sistema reducido. Los algoritmos híbridos propuestos en la bibliografía difieren en

el algoritmo que usan localmente o en el algoritmo empleado en la solución global del sistema reducido. En este grupo incluimos los algoritmos de Krechel, Plum y Stüben [63], Cox y Knisley [28], Müller y Scheerer [76], Matton, Williams y Hewett [74] y el método de Amodio y Brugnano [3], que resuelven ST cuyas matrices de coeficientes son no singulares pero sin necesidad de dominancia diagonal (ver [45]).

El estudio de ST nos permite clasificarlos en términos de su diagrama de flujo. En este capítulo presentamos una versión unificada de los algoritmos para resolver los ST sobre computadores con topología malla y memoria distribuida. Para este fin usamos una combinación de las técnicas "proyección vector" [36] y permutación "índice–dígito" [39], que nos permite definir la distribución de los datos en la memoria de los PEs y el movimiento de datos en los algoritmos [10, 7, 9]. Estas técnicas han sido usadas en [71, 70] para diseñar una arquitectura sistólica unificada con geometría constante.

3.2 Sistemas tridiagonales

Consideramos un conjunto de N ecuaciones lineales con N incógnitas

$$A\mathbf{u} = \mathbf{d}, \tag{3.5}$$

donde A es una matriz tridiagonal $N \times N$ de la forma

$$A = \begin{pmatrix} b_0 & c_0 & & & & & \\ a_1 & b_1 & c_1 & & & & \\ & a_2 & b_2 & c_2 & & & \\ & & . & . & . & & \\ & & & . & . & . & \\ & & & & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & & & a_{N-1} & b_{N-1} \end{pmatrix}. \tag{3.6}$$

La ecuación i–ésima es del tipo

$$E_i^{(0)} = \{a_i^{(0)} x_{i-1} + b_i^{(0)} x_i + c_i^{(0)} x_{i+1} = d_i^{(0)}\}, \tag{3.7}$$

para $i = 0, ..., N-1$, donde $a_0 = c_{N-1} = x_{-1} = x_N = 0$. Sin pérdida de generalidad suponemos que el número de ecuaciones es $N = 2^n$.

Los coeficientes b_i constituyen la diagonal de la matriz del sistema, A, mientras los coeficientes a_i y c_i son denominados subdiagonales y superdia-

gonales, respectivamente. La *dominancia diagonal*, δ, de una matriz A se define,

$$\delta = \min_i \left\{ |a_{ii}| / \left(\sum_{\substack{j=0 \\ j \neq i}}^{N-1} |a_{ij}| \right) \right\}, \; i = 0, \cdots, N-1 \tag{3.8}$$

En particular, para el ST (3.6) resulta la expresión

$$\delta = \min_i \left\{ |b_i| / \left(|a_i| + |c_i| \right) \right\}, \; i = 0, \cdots, N-1. \tag{3.9}$$

Si $\delta \geq 1$, el sistema se llama diagonalmente dominante. La definición dada en [55] considera el mínimo de los coeficientes $(|b|/|a|, |b|/|c|)$. Si la dominancia de un sistema según este criterio es mayor que 2, entonces δ definida en (3.9) es mayor que 1. La dominancia diagonal de un ST asegura la estabilidad de casi todos los algoritmos propuestos en la bibliografía. Si la dominancia diagonal es fuerte, $\delta \gg 1$, se puede reducir el tiempo de computación necesario para obtener la solución del sistema. Los métodos de particionamiento solapado [65, 62] y diagonalmente dominante [104] utilizan este hecho para dividir el sistema en bloques de sistemas que pueden considerarse tridiagonales y resolverse simultánea e independientemente en PEs distintos, o con mínimas comunicaciones entre los PEs.

Los algoritmos para ST pueden clasificarse en términos de su flujo de datos en los siguientes grupos:

- Árboles directos e inversos.
- Árbol extendido.
- Divide–y–vencerás.
- Divide–y–vencerás extendido.

Existen otros algoritmos para ST que pueden ser incluidos en uno de los casos anteriores. El algoritmo de desacoplamiento recursivo es un caso particular del divide–y–vencerás.

A continuación formularemos estos algoritmos con los operadores que estamos utilizando y realizaremos la implementación de un algoritmo típico de cada grupo.

3.3 Árboles directos e inversos

La operación de cada nodo (mariposa) de un árbol binario directo completo tiene dos entradas y una o dos salidas, pero en el último caso, sólo una de las salidas progresa en el siguiente paso. El nodo almacena la otra. La fase de eliminación del *sistema reducido* en los algoritmos de Cox–Knisely [28], Krechel, Plum y Stüben [63] y Müller y Scheerer [76] presentan este tipo de flujo de datos.

En general, en un árbol directo r-ario, el número de mariposas computadas en cada paso se reduce en un factor r a medida de que se atraviesa el árbol. Esta reducción la representamos por el operador Ω, el cual elimina los dígitos menos significativos de los índices de los datos en cada paso del árbol.

Consideraremos la implementación de árboles r-arios sobre los multicomputadores de topología malla. Utilizando las técnicas de la proyección vector y de las permutaciones índice–dígito que hemos visto en el capítulo uno, distribuimos los datos entre los PEs del multicomputador. El índice de los datos se dividen en tres campos: ($fila$, $columna$, $memoria$).

Como consecuencia de aplicar el operador reducción Ω, los n pasos del árbol están divididos en tres grupos.

1. En el primer grupo de pasos es eliminado un dígito del campo $memoria$. Después de u pasos, el campo $memoria$ desaparece de los índices de los datos.

2. En el segundo grupo de pasos el campo eliminado es el $columna$. Con objeto de implementar eficientemente estos pasos en el multicomputador su implementación se realiza de una manera ligeramente diferente. En el comienzo de la fase realizamos un intercambio de los campos $columna$ y $memoria$, (es decir, vaciamos el campo $columna$ y rellenamos el $memoria$). Por tanto, el operador reducción también se aplica al campo $memoria$ en este conjunto de w pasos.

3. Finalmente, en el tercer grupo, la reducción afecta al campo $fila$ pero, como antes, primero comenzamos intercambiando los campos $fila$ y $memoria$. También, una vez más, la reducción afecta al campo $memoria$ en v nuevos pasos.

El flujo de datos implementado de esta forma implica primero la computación de u pasos que pueden ser realizados con los datos almacenados en

cada PE. Al final de este conjunto de pasos hay sólo un dato por PE. Ahora, todos los datos de la misma fila son reducidos por el primer PE de la fila (intercambio entre *memoria* y *columna*). En cada uno de los siguientes w pasos, el número de elementos en cada PE de la primera fila es reducido por un factor r hasta que sólo queda un dato por PE. Entonces todos los datos contenidos en los PEs son reducidos a un único PE: el primero en la primera fila. Finalmente, el número de datos en este PE es reducido en los últimos v pasos, hasta que se alcanza la raíz.

En la figura 3.2.a presentamos un ejemplo para el caso de $N = 64$ datos sobre una malla de 4×4 PEs y en la figura 3.2.b presentamos su representación índice–dígito.

Algoritmo 6 *El árbol directo de longitud $N = r^n$ se puede implementar por la siguiente cadena de operadores con una distribución por bloques*

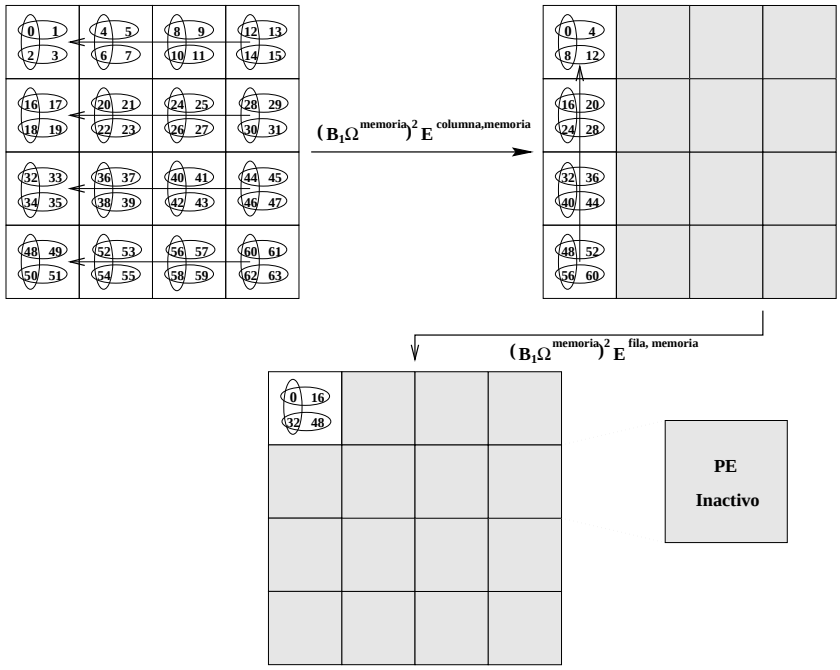
$$(\prod_{i=1}^{u} B_1\Omega^{mem})E^{col,mem}(\prod_{i=1}^{w} B_1\Omega^{mem})E^{fila,mem}(\prod_{i=1}^{v} B_1\Omega^{mem}). \tag{3.10}$$

La fase de sustitución del algoritmo de reducción cíclica (RC) y del *sistema reducido* en los algoritmos Cox y Knisely [28], Krechel, Plum y Stüben [63] y Müller y Scheerer [76] presentan un árbol binario inverso en flujo de datos, mientras la fase de sustitución del algoritmo r-RC [29, 30] es un árbol inverso r-ario. La mitad derecha de la figura 3.5 visualiza la fase de sustitución del algoritmo binario RC. Notar que el número de mariposas computadas incrementa por un factor 2 en cada paso. El operador ε añade el mismo número de datos de la secuencia de la fase de eliminación a los datos procesados en el paso previo.

Siguiendo una línea de argumentación similar a la usada en el árbol directo, un árbol inverso puede formularse por la cadena de operadores inversa de (3.10),

$$(\prod_{i=1}^{v} \varepsilon^{mem} B_1)E^{fila,mem}(\prod_{i=1}^{w} \varepsilon^{mem} B_1)E^{col,mem}(\prod_{i=1}^{u} \varepsilon^{mem} B_1). \tag{3.11}$$

En la figura 3.2.c mostramos el flujo de datos de este algoritmo usando la representación índice–dígito.



(a)

fila		col		mem		
6	5	4	3	2	1	$B_1\Omega^{memoria}$
6	5	4	3	2	⊗	$B_1\Omega^{memoria}$
6	5	4	3	⊗	–	$E^{columna, memoria}$
6	5	–	–	4	3	$B_1\Omega^{memoria}$
6	5	–	–	4	⊗	$B_1\Omega^{memoria}$
6	5	–	–	⊗	–	$E^{fila, memoria}$
–	–	–	–	6	5	$B_1\Omega^{memoria}$
–	–	–	–	6	⊗	$B_1\Omega^{memoria}$
–	–	–	–	⊗	–	

(b)

fila		col		mem		
–	–	–	–	–	–	$\varepsilon^{memoria}B_1$
–	–	–	–	⊙	–	$\varepsilon^{memoria}B_1$
–	–	–	–	6	⊙	$E^{fila, memoria}$
6	5	–	–	–	–	$\varepsilon^{memoria}B_1$
6	5	–	–	⊙	–	$\varepsilon^{memoria}B_1$
6	5	–	–	4	⊙	$E^{columna, memoria}$
6	5	4	3	–	–	$\varepsilon^{memoria}B_1$
6	5	4	3	⊙	–	$\varepsilon^{memoria}B_1$
6	5	4	3	2	⊙	

(c)

Figura 3.2: (a) Flujo de datos de un árbol directo con $N = 64$ datos sobre una malla de 4×4 PEs (Algoritmo 6). (b) Esquema índice–dígito de la redistribución de datos en la fase de eliminación. (c) Esquema índice–dígito de la redistribución de datos en la fase de sustitución. Los círculos representan el operador mariposa y las cruces el operador reducción.

3.3.1 El método de Müller y Scheerer

El algoritmo de Müller y Scheerer [76] es un buen ejemplo de un algoritmo donde las fases de eliminación y sustitución de su sistema reducido muestran una estructura de árbol directo e inverso respectivamente. Si el sistema original (3.5) es particionado en $V \times W$ sistemas, entonces cada PE trabaja sobre U ecuaciones consecutivas. El sistema k-ésimo tiene la siguiente forma:

$$\begin{pmatrix} a_j & b_j & c_j & & & & & \\ & a_{j+1} & b_{j+1} & c_{j+1} & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & a_{p-1} & b_{p-1} & c_{p-1} & \\ & & & & & a_p & b_p & c_p \end{pmatrix} \times \begin{pmatrix} x_{j-1} \\ x_j \\ \cdot \\ \cdot \\ \cdot \\ x_p \\ x_{p+1} \end{pmatrix} = \begin{pmatrix} d_j \\ d_{j+1} \\ \cdot \\ \cdot \\ d_{p-1} \\ d_p \end{pmatrix}, \tag{3.12}$$

donde $j = (k-1)U$, $p = kU - 1$ y $a_0 = c_{N-1} = x_{-1} = x_N = 0$.

Si en las ecuaciones primera y última del sistema (3.12) pasamos al segundo miembro los términos $a_j x_{j-1}$ y $c_p x_{p+1}$ respectivamente y consideramos x_{j-1} y x_{p+1} como parámetros, en cada PE k tenemos el ST,

$$\underbrace{\begin{pmatrix} b_j & c_j & & & & \\ a_{j+1} & b_{j+1} & c_{j+1} & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & a_{p-1} & b_{p-1} & c_{p-1} \\ & & & & a_p & b_p \end{pmatrix}}_{A^k} \times \underbrace{\begin{pmatrix} x_j \\ x_{j+1} \\ \cdot \\ \cdot \\ x_{p-1} \\ x_p \end{pmatrix}}_{x^k} =$$

$$= \underbrace{\begin{pmatrix} d_j \\ d_{j+1} \\ \cdot \\ \cdot \\ d_{p-1} \\ d_p \end{pmatrix}}_{d^k} - x_{j-1} \underbrace{\begin{pmatrix} a_j \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{pmatrix}}_{f^k} - x_{p+1} \underbrace{\begin{pmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ c_p \end{pmatrix}}_{g^k}. \tag{3.13}$$

Entonces, en cada PE se resuelven tres sistemas lineales independientes,

$$A^k u^k = d^k, \quad A^k y^k = f^k, \quad A^k z^k = g^k, \tag{3.14}$$

y la solución x^k es obtenida como una combinación lineal de las tres soluciones parciales,

$$x^k = u^k - x_{j-1} y^k - x_{p+1} z^k. \tag{3.15}$$

Los subsistemas de la ecuación (3.14) son cuadráticos (es decir, tienen U ecuaciones y U incógnitas) y pueden resolverse sin ninguna comunicación entre PEs. Antes de la computación de la solución global x^k, los parámetros x_{j-1} y x_{p+1} que conectan los subsistemas tienen que ser determinados. Tomando la primera y la última ecuación de cada sistema (sólo la última del primer sistema y la primera del último sistema) tenemos un ST de $2(V \times W) - 2$ ecuaciones (*sistema reducido*).

$$\begin{pmatrix} z_U & 1 & & & & & \\ 1 & y_{U+1} & z_{U+1} & & & & \\ & y_{2U} & z_{2U} & 1 & & & \\ & & & \cdots & & & \\ & & & y_{U(V\times W-1)} & z_{U(V\times W-1)} & 1 & \\ & & & & 1 & & y_{U(V\times W-1)+1} \end{pmatrix} \times \begin{pmatrix} x_{U+1} \\ x_U \\ x_{2U+1} \\ \vdots \\ x_{U(V\times W-1)+1} \\ x_{U(V\times W-1)} \end{pmatrix} = \begin{pmatrix} d_U \\ d_{U+1} \\ d_{2U} \\ \vdots \\ d_{U(V\times W-1)} \\ d_{U(V\times W-1)+1} \end{pmatrix}. \tag{3.16}$$

Las dos ecuaciones del PE k son:

$$\begin{aligned} x_j^k &= u_j^k - x_{j-1} y_j^k - x_{p+1} z_j^k, \\ x_p^k &= u_p^k - x_{j-1} y_p^k - x_{p+1} z_p^k \end{aligned} \tag{3.17}$$

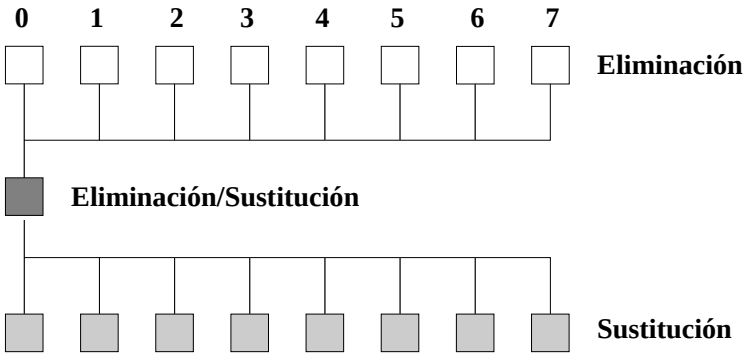
Una aproximación directa para la solución del sistema podría ser congregarlos en un único PE. Esto produciría un gran desbalanceo entre los PEs, pues un único nodo tendría la carga de resolver el sistema global, mientras

el resto de los nodos estarían inactivos, figura 3.3.a. La solución propuesta en [76] consiste en resolver el sistema reducido por un árbol directo (fase de eliminación) y un árbol inverso (fase de sustitución), ver figura 3.3.b.

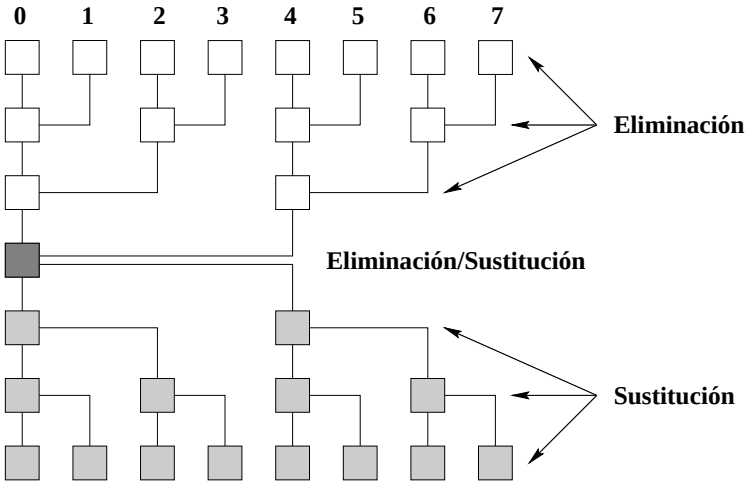
En nuestra implementación del método de Müller y Scheerer usamos la cadena del Algoritmo 6, que congrega las dos ecuaciones de todos los nodos de una fila en el primer PE de cada fila, resultando un sistema de $2W$ ecuaciones en cada uno excepto en los PEs primero y último de la primera columna, donde los sistemas serán de $2W-1$ ecuaciones. Después resolvemos estos sistemas usando el método que acabamos de describir. Las ecuaciones primera y última dependerán de las incógnitas de los PEs superiores e inferiores a cada uno. Enviaremos estas dos ecuaciones de cada PE al PE 0 y se resolverá el sistema de $2V-2$ ecuaciones. Calculados los parámetros, las restantes incógnitas son obtenidas a través del flujo inverso. De esta forma, cada PE consigue dos parámetros con los cuales puede completar la solución del sistema.

El método de Müller y Scheerer [76] es un método general de paralelización de algoritmos para resolver ST. Si se utiliza el método de Gauss en las fases primera y segunda se obtiene el método de descomposición de Wang [112]. Si se utiliza el algoritmo de reducción cíclica, se obtiene el método descrito por Krechel y col. [63]. Nosotros utilizamos la descomposición de Wang, [112].

Como un ejemplo, consideremos el método para un sistema de 8 PEs y 16 ecuaciones. El sistema de 16 ecuaciones se parte en cuatro subsistemas como en la ecuación 3.12, ver la figura 3.4. Resolveríamos el sistema reducido que le corresponde a cada PE por el método de Gauss, sin necesidad de realizar comunicaciones. Los ST resultantes para el PE 2 serían,



(a)



(b)

Figura 3.3: Método de Müller y Scheerer. (a) Se realiza una fase local de eliminación en los PEs para resolver la solución del sistema en un único PE. La fase de sustitución local también se realiza en cada uno de los PEs. (b) Las fases de eliminación y de sustitución se realizan en un árbol directo e inverso respectivamente.

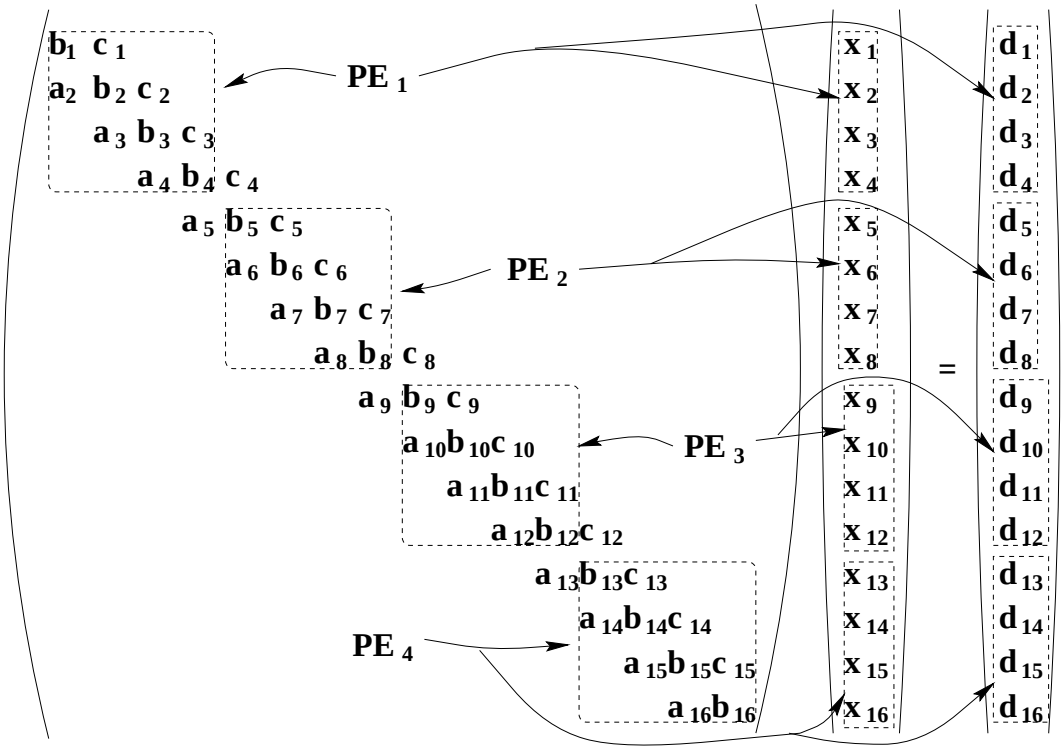


Figura 3.4: Particionamiento de un sistema de 16 ecuaciones en 4 PEs por el método de Müller y Scheerer.

$$
\begin{aligned}
A^2u^2 = d^2 \quad &\longrightarrow \quad \begin{pmatrix} b_5 & c_5 & & \\ a_6 & b_6 & c_6 & \\ & a_7 & b_7 & c_7 \\ & & a_8 & b_8 \end{pmatrix} \begin{pmatrix} u_5 \\ u_6 \\ u_7 \\ u_8 \end{pmatrix} = \begin{pmatrix} d_5 \\ d_6 \\ d_7 \\ d_8 \end{pmatrix} \\
A^2y^2 = f^2 \quad &\longrightarrow \quad \begin{pmatrix} b_5 & c_5 & & \\ a_6 & b_6 & c_6 & \\ & a_7 & b_7 & c_7 \\ & & a_8 & b_8 \end{pmatrix} \begin{pmatrix} y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix} = \begin{pmatrix} a_5 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad (3.18) \\
A^2z^2 = g^2 \quad &\longrightarrow \quad \begin{pmatrix} b_5 & c_5 & & \\ a_6 & b_6 & c_6 & \\ & a_7 & b_7 & c_7 \\ & & a_8 & b_8 \end{pmatrix} \begin{pmatrix} z_5 \\ z_6 \\ z_7 \\ z_8 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ c_8 \end{pmatrix}
\end{aligned}
$$

la solución de cada PE se obtiene como una combinación lineal de estas tres soluciones parciales. Quedan dos ecuaciones (ver (3.15)) por PE salvo en el primero y en el último,

$$
\begin{aligned}
PE_1 \quad &\rightarrow \quad x_4 = u_4 - x_5 z_4 \\
PE_2 \quad &\rightarrow \quad \begin{cases} x_5 = u_5 - x_4 y_5 - x_9 z_5 \\ x_8 = u_8 - x_4 y_8 - x_9 z_8 \end{cases} \\
PE_3 \quad &\rightarrow \quad \begin{cases} x_9 = u_9 - x_8 y_9 - x_{13} z_9 \\ x_{12} = u_{12} - x_8 y_{12} - x_{13} z_{12} \end{cases} \qquad (3.19) \\
PE_3 \quad &\rightarrow \quad x_{13} = u_{13} - x_{12} y_{13}
\end{aligned}
$$

Consideramos que reducimos todas las ecuaciones a un único PE. El sistema lineal que tenemos que resolver es

$$
\begin{pmatrix} z_4 & 1 & & & & \\ 1 & y_5 & z_5 & & & \\ & y_8 & z_8 & 1 & & \\ & & 1 & y_9 & z_9 & \\ & & & y_{12} & z_{12} & 1 \\ & & & & 1 & y_{13} \end{pmatrix} \begin{pmatrix} x_5 \\ x_4 \\ x_9 \\ x_8 \\ x_{13} \\ x_{12} \end{pmatrix} = \begin{pmatrix} u_4 \\ u_5 \\ u_8 \\ u_9 \\ u_{12} \\ u_{13} \end{pmatrix}. \qquad (3.20)
$$

Encontramos las incógnitas $\{x_4, x_5, x_8, x_9, x_{12}, x_{13}\}$ por el método de la GE, por ejemplo, y luego, por el proceso inverso, calculamos las incógnitas restantes del sistema global de la figura 3.4.

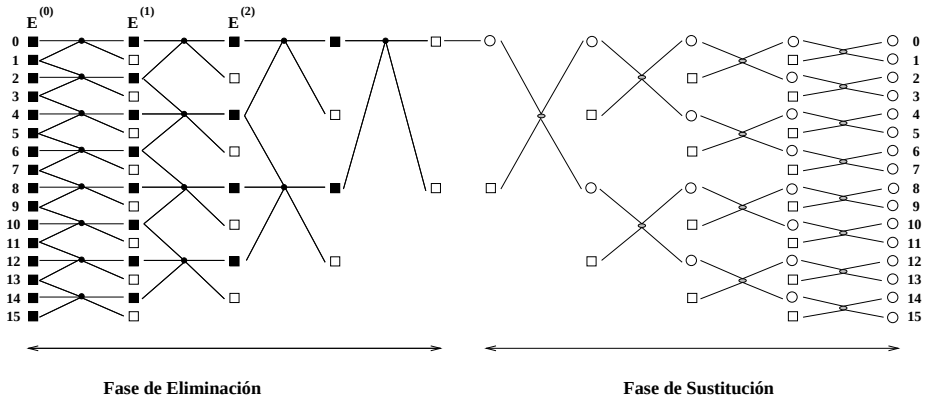


Figura 3.5: El flujo de datos del algoritmo RC para un sistema de $N = 16$ ecuaciones. La parte izquierda es un árbol directo extendido y la derecha un árbol inverso. En la fase de eliminación los cuadrados negros representan las ecuaciones que serán procesadas en el siguiente paso mientras que los cuadrados blancos representan las ecuaciones que no serán usadas hasta la fase de sustitución. En esta fase los cuadrados blancos representan las ecuaciones y los círculos dos incógnitas ya calculadas. Los nodos representan las operaciones que se realizan sobre ecuaciones e incógnitas.

3.4 Árbol extendido

En este tipo de árbol, la salida de algunos nodos son entradas para dos nodos del siguiente paso. Este tipo de flujo se encuentra en la fase de eliminación del algoritmo r-RC [29, 33, 30] ($r \geq 2$). El caso $r = 2$ se presenta en la parte izquierda de la figura 3.5. Las mariposas (operador B') presentan $2r - 1$ entradas y r salidas.

Para una distribución por bloques de los datos existen entradas del operador B' que están localizadas en diferentes PEs. Por ejemplo, en la figura 3.6 vemos que el dato de índice 3 está en el PE $(fila, columna) = (0, 0)$ y es necesario también en el PE de su derecha para computar la mariposa de entrada (3,4,5) (ver también figura 3.5). Para tener todos los datos necesarios para la computación de las mariposas de un paso en el mismo PE o en PEs vecinos, la distribución apropiada es seguir un ordenamiento serpiente.

Además, es necesario definir un operador que realice la transmisión entre los PEs de los datos que se necesitan para el operador B'.

Definición 10 *El operador η^{col} envía $r-1$ datos localizados en las posiciones*

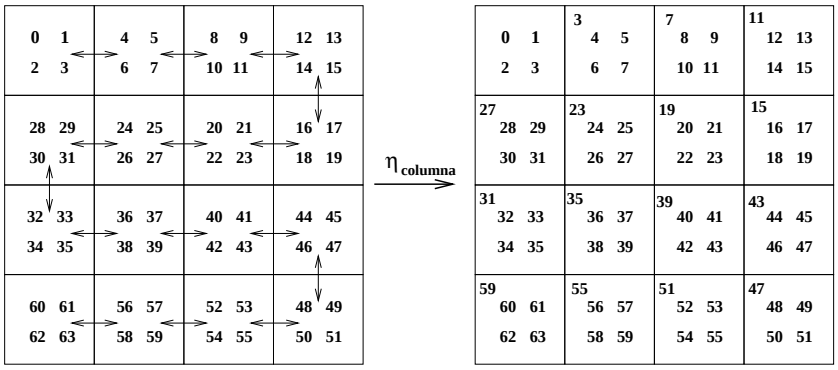


Figura 3.6: Flujo de datos del operador η^{col}.

más significativas del campo memoria del PE k al PE k+1. El operador η^{fila} *envía los* $r-1$ *datos al PE* $k+V$.

De la notación que hemos introducido en el primer capítulo podemos escribir el árbol extendido,

Algoritmo 7 *El árbol extendido de longitud* $N = r^n$ *se puede implementar por la siguiente cadena de operadores*

$$(\prod_{i=1}^{u} \eta^{col} B_1' \Omega^{mem}) E^{col,mem} (\prod_{i=1}^{w} \eta^{fila} B_1' \Omega^{mem}) E^{fila,mem} (\prod_{i=1}^{v} B_1' \Omega^{mem}). \quad (3.21)$$

Los pasos de computación y comunicación determinados por la cadena (3.21) son similares a los descritos en el Algoritmo 6, pero añadimos las comunicaciones producidas por los operadores η^{fila} y $\eta^{columna}$. En la implementación del Algoritmo 7 según la cadena de operadores, la primera operación es una reducción por filas y por columnas como en la figura 3.2.a.

3.4.1 El método de la reducción cíclica

Un algoritmo típico con flujo de datos de árbol extendido es el algoritmo RC [55] para ST. El RC fue aplicado por primera vez a la resolución de los ST por Hockney y Golub en 1965 en el computador secuencial IBM 7090. Realiza

más operaciones aritméticas que el método de Gauss, que es el algoritmo secuencial más eficiente, pero el paralelismo inherente de su flujo de datos lo hace especialmente indicado para su computación en procesadores vectoriales y máquinas paralelas.

El algoritmo RC consta de dos fases, eliminación y sustitución, de n etapas cada una, siendo $N = 2^n$ el tamaño del sistema. La fase de eliminación tiene flujo de datos tipo árbol directo (de las hojas a la raíz) y finaliza proporcionando una ecuación con una sola incógnita. La fase de sustitución resuelve esa ecuación y calcula las restantes soluciones siguiendo un árbol binario inverso. La versión que expondremos aquí esta basada en Evans [33], que es un poco diferente de las versiones presentadas en [29]. En la fase de eliminación sólo modificamos las ecuaciones que se utilizarán en las etapas siguientes, dejando inalteradas las que no progresan. La fase de eliminación se simplifica a costa de la fase de sustitución. En [29] se modifican las ecuaciones que no progresan en la fase de eliminación para facilitar el cálculo de las incógnitas en la fase de sustitución.

Recientemente se han propuesto variantes del algorimo RC que aumentan su paralelismo o la base de las mariposas. En el primer caso está el algoritmo Reducción Cíclica Paralela (PARA-RC) [55], que resuelve el sistema en una sola fase de n etapas computando N/2 mariposas en cada una de ellas. En el segundo caso se encuentran los algoritmos Reducción Tricíclica ([30]) y p-Cíclica ([29]), que utilizan mariposas con base impar $r > 2$ con el objetivo de minimizar los conflictos de memoria cuando se implementan en procesadores vectoriales.

Sin pérdida de generalidad consideraremos el caso de $r = 2$. En la figura 3.5 mostramos el flujo de datos del algoritmo RC para un sistemas de ecuaciones de $N = 2^4$. El flujo de datos de la fase de eliminación es un árbol extendido. La fase de eliminación se realiza en n pasos. En cada paso se obtiene un nuevo ST con la mitad del número de ecuaciones que en el paso previo. Las 2^{n-t} ecuaciones obtenidas en el paso t-ésimo ($t = 1, \cdots, n$) presenta incógnitas a una distancia 2^t,

$$E^{(t)}(i) = a_i^{(t)} x_{i-2^t} + b_i^{(t)} x_i + c_i^{(t)} x_{i+2^t} = d_i^{(t)}, \tag{3.22}$$

donde $i = k2^t$, $0 \leq k < 2^{n-t}$. Cualquier ecuación cuyo índice i esté fuera del rango $[0, N-1]$ tiene de coeficientes $\{0, 1, 0, 0\}$. La denominamos ecuación identidad y la denotamos por I. Cualquier incógnita con subíndice fuera de ese mismo rango tiene valor 0. Tres ecuaciones que son computadas en el paso t-ésimo son:

$$\begin{array}{lllllllll} a_{i-2^t}x_{i-2\cdot 2^t} & + & b_{i-2^t}x_{i-2^t} & + & c_{i-2^t}x_i & & & = & d_{i-2^t} \\ & + & a_i x_{i-2^t} & + & b_i x_i & + & c_i x_{i+2^t} & = & d_i \\ & & & + & a_{i+2^t}x_i & + & b_{i+2^t}x_{i+2^t} & + & c_{i+2^t}x_{i+2\cdot 2^t} & = & d_{i+2^t}, \end{array} \tag{3.23}$$

Si la primera de estas ecuaciones es multiplicada por $\eta_i = -a_i/b_{i-2^{t-1}}$, y la última por $\gamma_i = -c_i/b_{i+2^{t-1}}$, y sumamos las tres ecuaciones, todas las referencias a las variables x_{i-2^t} y x_{i+2^t} son eliminadas. Los superíndices $(t-1)$ de los coeficientes a_i, c_i, etc..., son omitidos por simplicidad. Así, en el paso t-ésimo de la fase de eliminación obtenemos la ecuación $E^{(t)}(i)$,

$$E^{(t)}(i) = \gamma^{(t-1)}E^{(t-1)}(i-2^{t-1}) + E^{(t-1)}(i) + \eta_i^{(t-1)}E^{(t-1)}(i+2^{t-1}). \tag{3.24}$$

El segundo término de (3.24) se notará abreviadamente por $[E(i-2^{t-1}),\ E(i),\ E(i+2^{t-1})]^{(t-1)}$.

En este algoritmo el operador mariposa B' admite entradas de tres ecuaciones, $[E(i-2^{t-1}),\ E(i),\ (i+2^{t-1})]^{(t-1)}$. Ejecutando la ecuación (3.24) obtenemos como salidas las ecuaciones $E^{(t)}(i)$ y $E^{(t-1)}(i+2^{t-1})$ (representadas por cuadrados blancos y negros en la figura 3.5). El primer sumando de (3.24) elimina $i-2^{t-1}$ en $E^{(t-1)}(i)$ e introduce en ella la incógnita $i-2\cdot 2^{t-1} = i-2^t$. Análogamente, el último sumando elimina la incógnita $i+2^{t-1}$ e introduce $i+2\cdot 2^{t-1} = i+2^t$ en la ecuación $E^{(t-1)}(i)$, con lo cual resulta $E^{(t)}(i)$,

$$\begin{array}{llll} (i-2\cdot 2^{t-1}, & i-2^{t-1}, & i) & \\ & (i-2^{t-1}, \quad i, & i+2^{t-1}) & \longrightarrow \quad (i-2^t, i, i+2^t) \\ & & (i, \quad i+2^{t-1}, & i+2\cdot 2^{t-1}) \end{array} \tag{3.25}$$

En el paso n-ésimo, último paso de la fase eliminación, sólo se tiene una ecuación $E^{(n)}(0)$, cuyas incógnitas son x_{-2^n}, x_0 y x_{2^n}. Como $x_{-2^n} = x_{2^n} = 0$, la única incógnita de $E^{(n)}(0)$ es x_0. La fase de sustitución comienza con el cálculo del valor de la incógnita x_0 y tiene un flujo de datos de tipo árbol binario inverso. En cada nodo se sustituyen en la ecuación $E^{(n-t)}(i)$ los valores de las incógnitas $i \pm 2^{n-t}$, calculadas en las etapas anteriores y se calculan las incógnitas x_i $(i = 2^{n-t} + k2^{n-t+1})$, donde $0 \le k < 2^{t-1}$.

3.5 Los algoritmos divide–y–vencerás

La estrategia divide–y–vencerás consiste en dividir un problema en algunos subproblemas de un tamaño más pequeño, resolviendo cada subproblema independientemente y combinando los resultados obtenidos para construir la solución del problema original. Se genera un flujo de datos muy regular y eficiente cuando se proyecta sobre arquitecturas paralelas, ver figura 3.7. Vemos que a diferencia del flujo de datos de la FFT no necesita ninguna operación adicional de dígito–inverso. Un ejemplo de este tipo de algoritmo es el de doblamiento sucesivo de Wang y Mou [113].

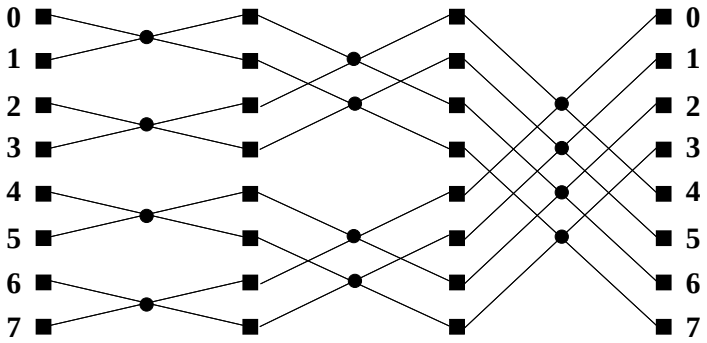


Figura 3.7: Flujo de datos del algoritmo divide–y–vencerás para un sistema de ecuaciones de $N = 8$. En el algoritmo de Wang y Mou los cuadrados representan una tríada de ecuaciones y los nodos las operaciones que se realizan sobre estas ecuaciones. Los nodos de este último algoritmo se muestran en la figura 3.9 con más detalle.

El flujo de datos del algoritmo sobre la malla puede realizarse por dos caminos alternativos:

1. Combinando operaciones de computación (operador mariposa) y operaciones de comunicación (operador intercambio) en cada uno de los n pasos. Es decir, en los primeros u pasos no se requieren comunicaciones; w pasos requieren comunicación paralela en las filas y v pasos necesitan comunicación paralela en las columnas.

2. Agrupando las operaciones de computación por un lado y las de comunicación por otro. Como en el caso previo, los primeros u pasos no necesitan comunicaciones. Después se intercambian los campos *memoria* y *columna* (si no son del mismo tamaño, sólo se intercambia una

parte del campo *memoria*) y se realizan w pasos de computación. Finalmente, intercambiamos los campos *memoria* y *fila* y realizamos v pasos de computación.

El algoritmo puede escribirse de la siguiente forma:

Algoritmo 8 *El algoritmo divide–y–vencerás de longitud $N = r^n$ DS se puede implementar por la siguiente cadena de operadores:*

1.

$$\prod_{i=1}^{u} B_i \prod_{i=u+1}^{n} E_{i,u} B_u, \tag{3.26}$$

2.

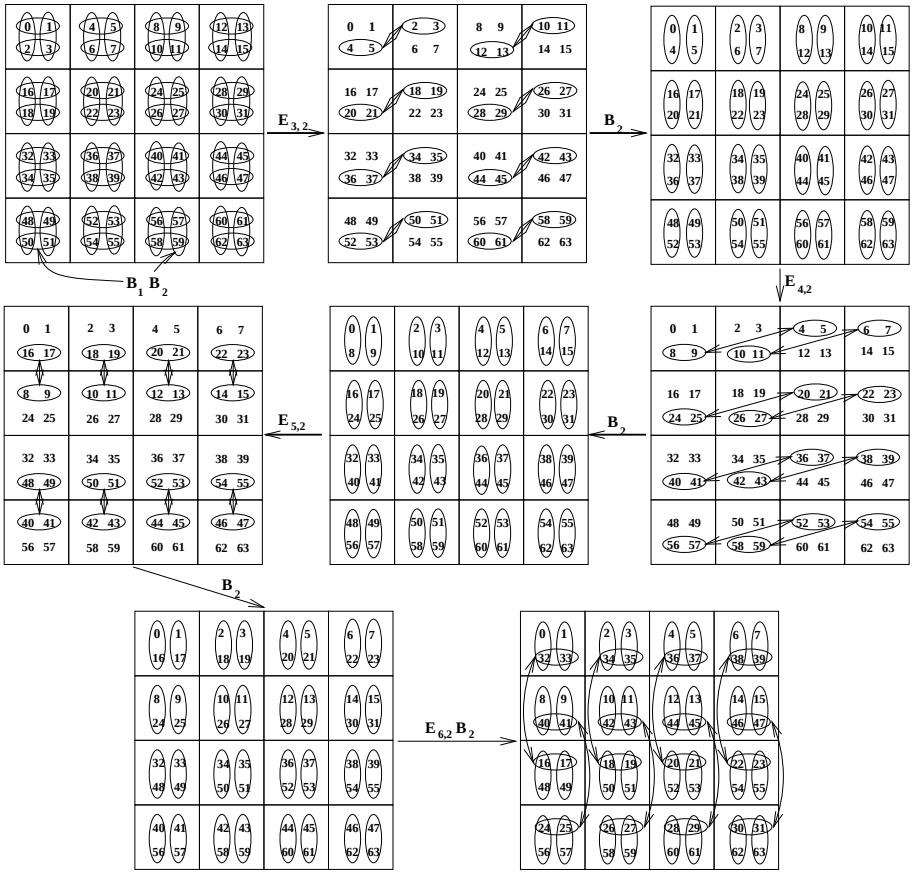
$$(\prod_{i=1}^{u} B_i)\ E^{col,mem}\ (\prod_{i=1}^{w} B_i)\ E^{fil,mem}\ \prod_{i=1}^{v} B_i, \tag{3.27}$$

Estas formulaciones del algoritmo no necesitan ningún barajamiento adicional de los datos que impliquen comunicaciones diagonales entre filas y columnas. La cadena de operadores de la Proposición (3.26) es la que obtiene un mejor rendimiento, permitiendo solapar computaciones y comunicaciones. Además, posibilita que todas las computaciones y comunicaciones ocurran in–situ. En la figura 3.8.a se muestra el flujo de datos de la cadena (3.26) para un sistema de ecuaciones $N = 64$ sobre una malla de 4×4 PEs y en las figuras 3.8.b y 3.8.c la representación índice–dígito de las cadenas (3.26) y (3.27).

3.5.1 El método del doblamiento sucesivo de Wang y Mou

El algoritmo de Wang y Mou [113] actúa sobre tríadas de ecuaciones. En la secuencia de tríadas inicial, denotada por $[i]^{(0)},\ 0 \leq i < N$, cada tríada está construída por tres ecuaciones iguales, $E^{(0)}(i)$. El algoritmo se ejecuta en n pasos. En el paso t-ésimo, la secuencia inicial está formada por las tríadas $[i]^{(t-1)},\ 0 \leq i < N$, constituidas por las ecuaciones

$$[i]^{(t-1)} = [E^{(t-1)}(s2^{t-1}),\ E^{(t-1)}(i),\ E^{(t-1)}((s+1)2^{t-1}-1)], \tag{3.28}$$



(a)

fila	col	mem	
6 5	4 3	2 ①	B_1
6 5	4 3	② 1	B_2
6 5	4 2	③ 1	$E_{3,2}B_2$
6 5	3 2	④ 1	$E_{4,2}B_2$
6 5	3 2	⑤ 1	$E_{5,2}B_2$
5 4	3 2	⑥ 1	$E_{6,2}B_2$

(b)

fila	col	mem	
6 5	4 3	2 ①	B_1
6 5	4 3	② 1	B_2
6 5	4 3	2 1	$E^{\text{columna, memoria}}$
6 5	2 1	4 ③	B_1
6 5	2 1	④ 3	B_2
6 5	2 1	4 3	$E^{\text{fila, memoria}}$
4 3	2 1	6 ⑤	B_1
4 3	2 1	⑥ 5	B_2

(c)

Figura 3.8: (a) Flujo de datos del algoritmo de Wang y Mou para un sistema de ecuaciones de $N = 64$ (Algoritmo 8). (b) Representación índice–dígito de la cadena (3.26). (c) Representación índice–dígito de la cadena (3.27).

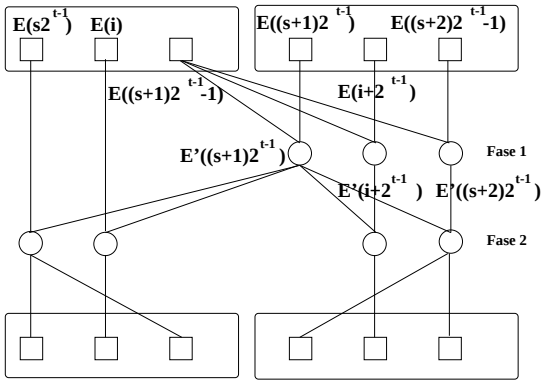


Figura 3.9: Bosquejo del operador B en el algoritmo de Wang y Mou. En este caso, cada nodo del algoritmo contiene tres ecuaciones.

donde $s = \lfloor i/2^{t-1} \rfloor$. La ecuaciones $E^{(t-1)}(i)$ son de la forma

$$E^{(t-1)}(i) = a_i^{(t-1)} x_{s2^{t-1}-1} + b_i^{(t-1)} x_i + c_i^{(t-1)} x_{(s+1)2^{t-1}} = d_i^{(t-1)}. \qquad (3.29)$$

El operador mariposa B consta de dos fases y está esbozado en la figura 3.9. En la primera fase, la ecuación $E^{(t-1)}((s+1)2^{t-1}-1)$ se usa para eliminar la incógnita $(s+1)2^{t-1}-1$ de la tercera ecuación de la tríada $[i+2^{t-1}]^{(t-1)}$. Denotaremos las ecuaciones de la tríada $[i+2^{t-1}]^{(t-1)}$ modificadas después de la primera fase como $E'((s+1)2^{t-1})$, $E'(i+2^{t-1})$ y $E'((s+2)2^{t-1})$. En la segunda fase, emplearemos las ecuación $E'((s+1)2^{t-1})$ para eliminar la incógnita $(s+1)2^{t-1}$ de las ecuaciones $E^{(t-1)}(s2^{t-1})$, $E^{(t-1)}(i)$, $E'(i+2^{t-1})$ y $E'((s+2)2^{t-1})$. Esta fase produce cuatro ecuaciones, $E^{(t)}(s2^{t-1})$, $E^{(t)}(i)$, $E^{(t)}(i+2^{t-1})$ y $E^{(t)}((s+2)2^{t-1}-1)$, que nos permitirán construir las tríadas $[i]^{(t)}$ y $[i+2^{t-1}]^{(t)}$.

En la última etapa del algoritmo cada tríada $[i]^{(n)}$ contiene la ecuación $E^{(n)}(i)$ con incógnitas x_{-1}, x_i y x_{2^n}. Dado que $x_{-1} = x_N = 0$, una simple división permite calcular cada incógnita x_i.

En resumen, el algoritmo propuesto por Wang y Mou [113] para la resolución de ST, presenta un flujo de datos muy regular conocido como doblamiento sucesivo. Cada dato consiste en una tríada de ecuaciones (doce coeficientes). La operación (mariposa) realizada por el operador B sobre cada pareja de tríadas implica 41 operaciones aritmétricas.

La principal ventaja del algoritmo de Wang y Mou es la regularidad de su flujo de datos que se traduce en comunicaciones poco complejas cuando se proyecta en hipercubos, mallas o en la Connection Machine [113].

3.5.2 El método de desacoplamiento recursivo de Spaletta y Evans

El algoritmo de Spaletta y Evans [101] es un buen ejemplo de un algoritmo divide–y–vencerás con base distinta de 2. Para ilustrar el algoritmo, suponemos que N es una potencia de 2 con $m = N/2 = 2^{n-1}$. La matriz de coeficientes A (3.6) es reorganizada de la siguiente forma

$$\begin{pmatrix} e_0 & c_0 & & & & & \\ a_1 & e_1 & & & 0 & & \\ & & e_2 & c_2 & & & \\ & & a_3 & e_3 & & & \\ & & & & \ddots & & \\ & 0 & & & e_{N-2} & c_{N-2} \\ & & & & a_{N-1} & e_{N-1} \end{pmatrix} + \sum_{j=1}^{m-1} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{pmatrix}^{(j)} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \end{pmatrix}^{(j)T}, \tag{3.30}$$

donde

$$\begin{aligned} e_0 &= b_0 \\ e_{2j} &= b_{2j} - c_{2j-1} \quad \text{donde } j = 1, \ldots, m-1 \\ e_{2j-1} &= b_{2j-1} - a_{2j} \quad \text{donde } j = 1, \ldots, m-1 \\ e_{N-1} &= b_{N-1} \end{aligned} \tag{3.31}$$

y donde los vectores $\mathbf{x}^{(j)}$, $\mathbf{y}^{(j)}$ tienen solo dos elementos no nulos en las posiciones $2(j-1)$–ésima y $2j$–ésima,

$$\begin{cases} x_k = 1 & k = 2j-1,\ 2j \\ x_k = 0 & \text{otros casos} \end{cases}$$

$$\begin{cases} y_k = a_{2j} & \text{si } k = 2j-1 \\ y_k = c_{2j-1} & \text{si } k = 2j \\ y_k = 0 & \text{otros casos} \end{cases}, \tag{3.32}$$

es decir,

$$\mathbf{x}^{(j)} = (0,\cdots,0,1,1,0,\cdots,0)^T \tag{3.33}$$

$$\mathbf{y}^{(j)} = (0,\cdots,0,a_{2j},c_{2j-1},0,\cdots,0)^T \tag{3.34}$$

En notación matricial, la partición de la matriz A puede ser representada por

$$A = J + \sum_{j=1}^{m-1} \mathbf{x}^{(j)}\mathbf{y}^{(j)T}, \tag{3.35}$$

donde J es una matriz diagonal de bloques de la forma

$$\begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_m \end{pmatrix}. \tag{3.36}$$

Cada bloque en J es una submatriz 2×2, J_i, del siguiente tipo

$$J_i = \begin{pmatrix} e_{2(i-1)} & c_{2(i-1)} \\ a_{2i-1} & e_{2i-1} \end{pmatrix}, \quad i = 1,2,\cdots,m. \tag{3.37}$$

Los elementos de J_i están definidos en (3.31).

Esta particular partición tiene como objeto aplicar el método de Sherman–Morrison para calcular la inversa de A [45]. Suponemos que hemos computado la matriz inversa J^{-1} de una matriz J de dimensiones $N\times N$. Sea A la matriz

$$A = J + \mathbf{x}\mathbf{y}^T, \tag{3.38}$$

donde $\mathbf{x}$, $\mathbf{y}$ son vectores N–dimensionales. Sherman–Morrison han demostrado que la matriz inversa A^{-1} se puede computar como

$$A^{-1} = J^{-1} - \alpha(J^{-1}\mathbf{x})(\mathbf{y}^T J^{-1}), \quad \alpha = \tfrac{1}{1+\mathbf{y}^T J^{-1}\mathbf{x}}. \tag{3.39}$$

El cómputo directo de la inversa de A tiene un coste de $O(N^3)$ operaciones aritméticas, mientras que con el uso de la expresión (3.39) el coste es sólo $O(N^2)$ operaciones.

La fórmula de Sherman–Morrison implica también que, una vez calculada la inversa J^{-1} de la matriz J y denotando $\mathbf{u}' = J^{-1}\mathbf{d}$ la solución de $A\mathbf{u} = \mathbf{d}$, se obtiene a partir de la expresión (3.38) como sigue,

$$\begin{aligned} \mathbf{u} \;= A^{-1}\mathbf{d} \;&= (J^{-1} - \alpha(J^{-1}\mathbf{x})(\mathbf{y}^T J^{-1}))\mathbf{d} \\ &= \mathbf{u}' - \alpha(J^{-1}\mathbf{x})(\mathbf{y}^T J^{-1})\mathbf{d} \end{aligned} \tag{3.40}$$

A continuación vamos a explicar el método de desacoplamiento recursivo basado en las ideas ya expuestas en (3.40).

Cuando la matriz A de coeficientes es particionada como en (3.30), tenemos que particionar los vectores $\mathbf{u}$ y $\mathbf{d}$ de forma análoga a la matriz J. Por lo que consideramos los vectores $\mathbf{u}$ y $\mathbf{d}$ escritos de la siguiente forma

$$\mathbf{u} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} \mathbf{u_0} \\ \mathbf{u_1} \\ \vdots \\ \mathbf{u_{m-1}} \end{pmatrix}, \; \mathbf{d} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{N-2} \\ d_{N-1} \end{pmatrix} = \begin{pmatrix} \mathbf{d_0} \\ \mathbf{d_1} \\ \vdots \\ \mathbf{d_{m-1}} \end{pmatrix}. \tag{3.41}$$

Podemos encontrar la solución del sistema (3.6) aplicando el método de Sherman–Morrison recursivamente a (3.35),

$$\begin{aligned} \mathbf{M_m} \;&= \left(J + \sum_{j=1}^{m-1} \mathbf{x}^{(j)}\mathbf{y}^{(j)T} \right)^{-1} \\ &= \left(I - \alpha_m \mathbf{M}_{m-1}\mathbf{x}^{(m)}\mathbf{y}^{(m)T}J^{-1} \right)\mathbf{M}_m - 1 \\ \alpha_m \;&= 1/\left(1 + \mathbf{y}^{(m)T}\mathbf{M}_{m-1}\mathbf{x}^{(m)} \right). \end{aligned} \tag{3.42}$$

La recurrencia obtenida permite calcular la inversa de la matriz A definida en (3.35) a partir de $\mathbf{M}_0 = J^{-1}$. Denotamos $J^{-1}\mathbf{d} = \mathbf{u}'$ y $J^{-1}\mathbf{x}^{(j)} = \mathbf{g}^{(j)}$. A la partición de la matriz A y al cálculo de los vectores $\mathbf{u}$ y $\mathbf{d}$ los conoceremos como el *paso preliminar* del método. En la figura 3.10 vemos el flujo de datos del algoritmo de desacoplamiento recursivo para un sistema de $N = 8$ datos. Cada dato inicial está formado por los elementos

$$\left(a_i, b_i, c_i, d_i, u_i, g_i^{(0)}, \cdots, g_i^{(m-1)} \right). \tag{3.43}$$

Si realizamos la distribución de los datos entre dos PEs, el dato 3 de la figura 3.10 está en el PE $(fila, columna) = (0,0)$ y es necesario también en el PE de su derecha para computar la mariposa $(3,4)$. Con lo que esta mariposa se realiza redundantemente en los dos PEs. Para tener todos los datos necesarios para la computación de las mariposas de un paso en el mismo PE o en PEs vecinos, una distribución apropiada es seguir un ordenamiento serpiente (1.18).

Es necesario definir el operador que realiza la transmisión de los datos necesarios en dos nodos del operador mariposa en PEs diferentes.

Definición 11 *El operador η_i^j envía r^j datos localizados en la parte más significativa del campo memoria del PE k al PE $k+r^i$, y transmite $(r-1)r^j$ datos localizados en la parte menos significativa del campo memoria del PE k al PE $k-r^i$.*

De lo que podemos escribir el paso preliminar como

$$\eta_0^0 B_1 \left([t_n \cdots t_1] - [0 \cdots 1] \right). \tag{3.44}$$

Después de este paso, el algoritmo es formulado dentro de tres secciones diferentes, respectivamente llamadas *paso1, paso2* y *paso3.*

Paso 1 El primer paso del algoritmo consiste en encontrar la solución de $J\mathbf{u} = \mathbf{d}$, que se obtiene de

$$\mathbf{u} = \begin{pmatrix} J_1^{-1} & & & \\ & J_2^{-1} & & \\ & & \ddots & \\ & & & J_m^{-1} \end{pmatrix} \mathbf{d}. \tag{3.45}$$

Es equivalente a resolver m sistemas de la forma,

$$\begin{pmatrix} u_{2(i-1)} \\ \\ u_{2i-1} \end{pmatrix} = J_i^{-1} \begin{pmatrix} d_{2(i-1)} \\ \\ d_{2i-1} \end{pmatrix}. \tag{3.46}$$

Cada uno de los bloques de la matriz diagonal de bloques J es de dimensión 2×2 por lo que el cálculo de su inverso es inmediato

$$J_i^{-1} = \tfrac{1}{\Delta_i} \begin{pmatrix} e_{2i-1} & -c_{2(i-1)} \\ \\ -a_{2i-1} & e_{2(i-1)} \end{pmatrix}, \quad \Delta_i = e_{2(i-1)} e_{2i-1} - a_{2i-1} c_{2(i-1)}. \tag{3.47}$$

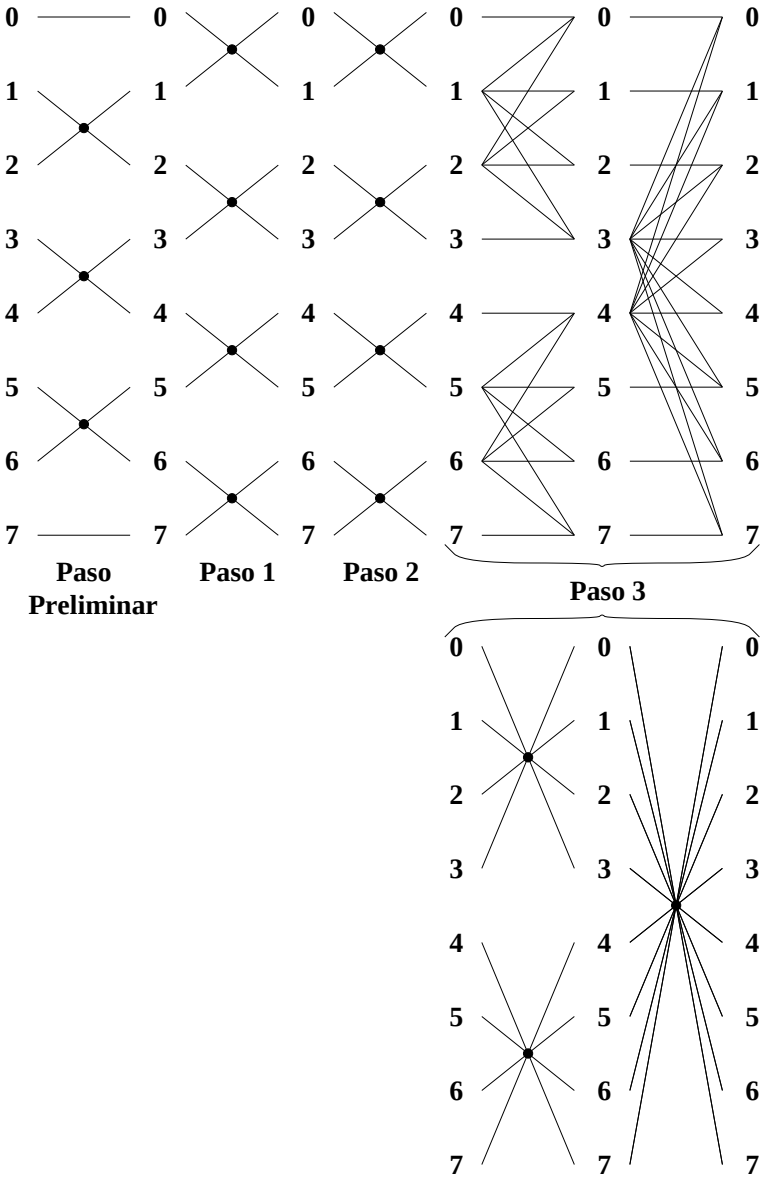


Figura 3.10: Flujo de datos del algoritmo de desacoplamiento recursivo de Spaletta y Evans para un sistema de ecuaciones con $N = 8$.

y la solución $\mathbf{u}$ del sistema se puede expresar como

$$\mathbf{u} = \begin{pmatrix} (e_1 d_0 - c_0 d_1)/\Delta_1 \\ (-a_1 d_0 + e_0 d_1)/\Delta_1 \\ \vdots \\ (e_{2i-1} d_{2(i-1)} - c_{2(i-1)} d_{2i-1})/\Delta_i \\ (-a_{2i-1} d_{2(i-1)} + e_{2(i-1)} d_{2i-1})/\Delta_i \\ \vdots \\ (e_{2m-1} d_{2m-2} - c_{2m-2} d_{2m-1})/\Delta_m \\ (-a_{2m-1} d_{2m-2} + e_{2m-2} d_{2m-1})/\Delta_m \end{pmatrix}. \tag{3.48}$$

Se puede ver que cada uno de los m subsistemas de (3.46) se puede resolver independientemente en cada uno de los PEs del multicomputador, ver figura 3.10.

Paso 2 De una forma análoga a la sección previa, el segundo paso consiste en encontrar la solución de $J\mathbf{g}^{(j)} = \mathbf{x}^{(j)}$ para $j = 1, \cdots, m-1$, que se obtiene

$$\mathbf{g}^{(j)} = \begin{pmatrix} J_1^{-1} & & & & \\ & J_2^{-1} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & J_m^{-1} \end{pmatrix} \mathbf{x}^{(j)},\ j = 1, 2, \cdots, m-1. \tag{3.49}$$

Es equivalente a resolver m sistemas de la forma

$$\begin{pmatrix} g_{2(i-1)}^{(j)} \\ \\ g_{2i-1}^{(j)} \end{pmatrix} = J_i^{-1} \begin{pmatrix} x_{2(i-1)}^{(j)} \\ \\ x_{2i-1}^{(j)} \end{pmatrix} \tag{3.50}$$

donde $j = 1, \cdots, m-1$. Como las matrices J_i^{-1} son conocidas, podemos expresar la solución de $\mathbf{g}^{(j)}$ de cada uno de los $m-1$ sistemas de (3.49)

como sigue:

$$\mathbf{g}^{(j)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -c_{2(i-1)}/\Delta_{2i-1} \\ e_{2(i-1)}/\Delta_{2i-1} \\ e_{2i-1}/\Delta_{2i} \\ -a_{2i-1}/\Delta_{2i} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{3.51}$$

donde los únicos elementos no nulos están en las posiciones $(2(j-1))$–ésima, $(2j-1)$–ésima, $(2j)$–ésima y en la $(2j+1)$–ésima por la forma que tiene el vector $\mathbf{x}^{(j)}$. Los $m-1$ sistemas puede ser también evaluados en paralelo, independientemente en cada PE sin ninguna comunicación. Algunos vectores $\mathbf{g}^{(j)}$ están divididos entre dos PEs. En el ejemplo de la figura 3.11 $\mathbf{g}^{(4)}$ tiene sus componentes $\{6,7\}$ en el PE 0 y las $\{8,9\}$ en el PE1. Este paso se puede resolver concurrentemente en todos los PEs sin ninguna comunicación.

Paso 3 Durante el último paso, los vectores $\mathbf{u}$ y $\mathbf{g}^{(j)}$ son actualizados, usando la fórmula recursiva de Sherman–Morrison. Este procedimiento puede ser escrito de la siguiente forma:

```
for k = 1, 2, ··· , n − 1
    for j = 2^{k−1}, 2^{n−1} − 2^{k−1}, 2^k
        α_j = 1/(1 + y^{(j)T} g^{(j)})
        u = (I − α_j g^{(j)} y^{(j)T}) u
        for i = 2^k, 2^{n−1} − 2^k, 2^k
            g^{(j)} = (I − α_j g^{(j)} y^{(j)T}) g^{(j)}
        end
    end
end
```

La solución final se obtiene y almacena en $\mathbf{u}$. Durante los primeros $n-(v+w)-1$ pasos no es necesario realizar ninguna comunicación mientras que en los siguientes $v+w$ pasos se necesita enviar los datos de una ecuación al resto de los PEs que tienen esa mariposa, ver figura

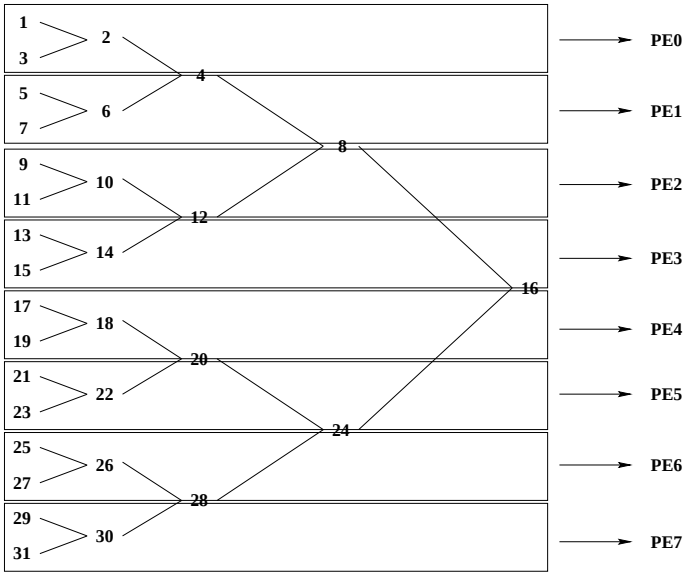


Figura 3.11: Los vectores $\mathbf{g}^{(j)}$ de un sistema de $N = 64$ ecuaciones sobre 8 PEs. En el primer paso los $g^{(j)}$ tienen elementos no nulos en los PEs que vemos en la figura. Algunos, como en el caso de $g^{(16)}$, tienen elementos no nulos entre dos PEs, el 3 y 4.

3.10. Por lo tanto, es necesario definir un nuevo operador que realice la transmisión de los datos necesarios entre los PEs adecuados,

Definición 12 *El operador ξ_j envía el dato localizado en las posiciones más significativas del campo memoria del PE k a los PEs $k+i$, $i = \{-2^j - 1, \cdots, -1, 1, \cdots, 2^j\}$ y los datos en las posiciones menos significativas del campo memoria del PE $k+1$ a los PEs $k+i$, $i = \{-2^j, \cdots, -1, 1, \cdots, 2^j - 1\}$.*

De lo que podemos expresar el algoritmo de desacoplamiento recursivo de Spaletta y Evans,

Algoritmo 9 *El algoritmo de desacoplamiento recursivo de Spaletta y Evans de longitud $N = r^n$ se puede implementar por la siguiente cadena de operadores*

$$\eta_0^0 B_1([t_n \cdots t_1] - [0 \cdots 1]) B_1 B_1 \prod_{i=1}^{u-1} B_1^{2^{i+1}} \prod_{i=u}^{n-1} \xi_{i-u+1} B_1^{2^u} \tag{3.52}$$

3.6 Divide–y–vencerás extendido

En la figura 3.12 mostramos el flujo de datos del algoritmo divide–y–vencerás extendido para un sistema de $N = 8$ ecuaciones. Cada nodo es una ecuación (cuatro coeficientes). Este algoritmo tiene la misma formulación que el algoritmo divide–y–vencerás, pero cada dato es entrada a dos mariposas (representadas por el operador B''). Hay N/r mariposas presentando $2r$ entradas y r salidas. Alguna entrada puede ser la ecuación identidad I.

El flujo de datos del algoritmo en la malla puede realizarse por tres caminos alternativos:

1. Combinando operaciones de computación y comunicación en cada uno de los n pasos. En los primeros u pasos las únicas comunicaciones necesarias son las del operador η_j^{i-1}. Estas comunicaciones son entre PEs vecinos ($j = 0$) y el tamaño de los mensajes se dobla en cada uno de los pasos ($i = 1, \cdots, u$). En los restantes $n - u$ pasos el operador intercambio, característico del algoritmo divide–y–vencerás, debe también implementarse, lo que implica comunicaciones primero sobre las

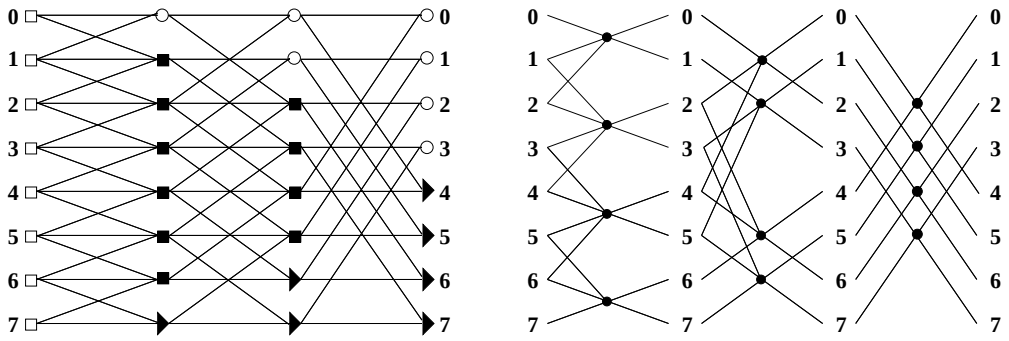


Figura 3.12: Flujo de datos de un algoritmo divide–y–vencerás extendido para un sistema de ecuaciones de $N = 8$. En el algoritmo PARA-RC los cuadrados negros representan las ecuaciones iniciales. Los cuadrados negros representan el resultado de la ecuación (3.56). Los círculos y los triángulos también representan esta computación pero identifican la ecuación identidad I como entradas superior o inferior respectivamente. La figura de la derecha es el mismo flujo de datos pero utilizando la notación de las mariposas B''.

columnas y después sobre las filas. Este caso es una generalización de 8.1 para divide–y–vencerás extendido.

2. Agrupando las operaciones de computación a un lado y las de comunicación (operador E) en el otro. A diferencia de la proposición 8.2 todavía es necesario implementar las comunicaciones generadas por el operador η_j^{i-1}. En los primeros v pasos, estas comunicaciones son entre PEs vecinos en las filas (η_0^{i-1}) y en los siguientes w pasos, entre PEs vecinos en las columnas (η_w^{i-1}). Ninguna comunicación es necesaria en los últimos u pasos. A diferencia de la ecuación (3.27), que implica u pasos, en este caso es necesario realizar v pasos del primer tipo para obtener menor número de comunicaciones (η_0^{i-1}).

3. Podemos minimizar el número de pasos de comunicaciones si comenzamos por realizar las comunicaciones que implica el operador E en los primeros $v + w$ pasos. De esta manera, en los restantes u pasos no son necesarias las comunicaciones.

Algoritmo 10 *El algoritmo divide–y–vencerás extendido se puede implementar por una de las siguientes cadenas de operadores:*

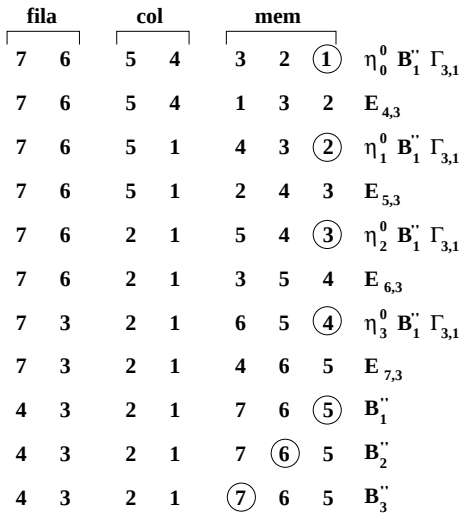


Figura 3.13: Esquema índice–dígito de la redistribución de datos del Algoritmo 10 (ecuación (3.55)) de longitud $N = r^7$. El círculo representa la mariposa.

1.

$$\prod_{i=1}^{u} \eta_0^{i-1} B_i'' \prod_{i=u+1}^{n} E_{i,u} \eta_{i-u}^{u-1} B_u''. \tag{3.53}$$

2.

$$(\prod_{i=1}^{v} \eta_0^{i-1} B_i'') E^{col,mem} (\prod_{i=1}^{w} \eta_w^{i-1} B_i'') E^{fil,mem} (\prod_{i=1}^{u} B_i''). \tag{3.54}$$

3.

$$\prod_{i=1}^{v+w} \eta_{i-1}^{0} B_1'' \Gamma_{u,1} E_{u+i,u} \prod_{i=1}^{u} B_i''. \tag{3.55}$$

La tercera cadena de operadores (ecuación (3.55), figura 3.13), es la que proporciona el mejor rendimiento, pues permite solapar comunicaciones y computaciones. Además, reduce los pasos de comunicación y la cantidad de datos que se transfieren con respecto a las dos primeras cadenas (ecuaciones (3.53) y (3.54)).

3.6.1 El método de la reducción cíclica paralela

Un algoritmo con un típico flujo de datos divide–y–vencerás extendido es el de la reducción cíclica paralela (PARA-RC) [55]. Este algoritmo incrementa el paralelismo del algoritmo RC al realizar la operación de reducción sobre todas las ecuaciones en todas las etapas [55]. El paso t-ésimo consiste en obtener las ecuaciones $E_i^{(t)}$, realizando la siguiente operación sobre las ecuaciones $E^{(t-1)}(i-2^{t-1})$, $E^{(t-1)}(i)$ y $E^{(t-1)}(i+2^{t-1})$,

$$E^{(t)}(i) = \zeta_i^{(t-1)} E^{(t-1)}(i-2^{t-1}) + E^{(t-1)}(i) + \eta_i^{(t-1)} E^{(t-1)}(i+2^{t-1}), \quad (3.56)$$

donde $\zeta_i^{(t-1)} = -a_i/b_{i-2^{t-1}}$, $\eta_i^{(t-1)} = -c_i/b_{i+2^{t-1}}$. Eliminamos los superíndices $(t-1)$ de los coeficientes para mayor claridad. Las incógnitas en $E^{(t-1)}(i-2^{t-1})$ son x_{i-2^t}, $x_{i-2^{t-1}}$ y x_i. El primer término de la derecha de (3.56) elimina $x_{i-2^{t-1}}$ en $E^{(t-1)}(i)$ e introduce la incógnita x_{i-2^t} en él. De la misma forma, el último término en la parte derecha de la ecuación (3.56) elimina $x_{i+2^{t-1}}$ e introduce x_{i+2^t} en la ecuación $E^{(t-1)}(i)$, de lo cual resulta, $E^{(t)}(i)$.

En el paso n-ésimo, el último del algoritmo, obtenemos las ecuaciones $E^{(n)}(i)$ con variables x_{i-2^n}, x_i y x_{i+2^n} que, excepto la central, son 0. Esto permite el cálculo de x_i en la ecuación $E^{(n)}(i)$.

Hay cuatro clases de nodos en la figura 3.12. Los nodos identificados como cuadrados blancos son las ecuaciones iniciales $E^{(0)}(i)$. Los nodos marcados como cuadrados negros representan el resultado de la expresión (3.56). Un círculo (triángulo negro) también computa la ecuación (3.56) pero siendo su entrada superior (inferior) la ecuación identidad, I. Identificamos como ecuación identidad la que presenta los coeficientes $\{0,1,0,0\}$.

3.7 Evaluación

Hemos implementado los algoritmos paralelos de Müller y Scheerer, RC, doblamiento sucesivo de Wang y Mou, PARA-RC y desacoplamiento recursivo de Spaletta y Evans sobre el AP1000 de Fujitsu [40].

El algoritmo de Müller y Scheerer muestra una estructura de árbol directo en su fase de eliminación y una estructura de árbol inverso en su fase de sustitución. Ambas fases han sido implementadas usando la cadena de operadores del Algoritmo 6, ecuaciones (3.10) y (3.11) respectivamente. El algoritmo RC tiene una estructura de árbol extendido durante su fase de

eliminación pero una estructura de árbol inverso en su fase de sustitución. Este algoritmo ha sido implementado usando la cadena del Algoritmo 7 para la primera fase y la ecuación (3.11) en la segunda fase.

Cuando implementamos estos algoritmos sobre el AP1000 realizamos algunas optimizaciones. En estos algoritmos debemos implementar las comunicaciones dadas por los operadores E y η (η sólo en el algoritmo RC). La implementación del primer operador la optimizamos evitando el uso de la misma conexión por varios mensajes simultáneamente. De esta manera evitamos que un mensaje esté bloqueado hasta que los otros mensajes finalicen. Este efecto de cuello de botella lo resolvimos usando una técnica bien conocida que funciona sobre muchos computadores paralelos, escala óptimamente, y requiere un número mínimo de mensajes para ser enviados y recibidos. En cada paso, un PE envía un mensaje a otro PE en su misma fila o columna dependiendo si se implementa el operador $E^{col,mem}$ o $E^{fila,mem}$. El PE que recibe concatena el mensaje recibido con su propio mensaje almacenado, doblando el tamaño del mensaje de forma que en un número logarítmico de pasos un PE acumula el vector entero (ver figura 3.14).

También hemos utilizado el método propuesto por Cox y Knisley [28]. La estrategia que sigue es particionar el sistema de ecuaciones en $V \times W$ sistemas locales de forma que la etapa de eliminación puede ser realizada en cada sistema independientemente de los demás. El coste de la independencia es una redundancia computacional. El sistema global al final de este paso se reduce a un sistema $V \times W \times q$ incógnitas, donde q es el número de incógnitas que quedan en cada PE después de finalizar la etapa local. Se pueden enviar todas las ecuaciones a un único PE, el cual resuelve el sistema y redistribuye los valores de las incógnitas calculadas. Otra opción es reducir primero el sistema por filas de forma que el primer PE de cada fila resuelva un sistema de $W \times q$ ecuaciones y, por último, el PE 0 resuelva un sistema de W ecuaciones. En el AP1000 este método no es efectivo. El tiempo computacional redundante no compensa el tiempo de comunicación que se ahorra. En este computador se ha utilizado un método híbrido. Durante las primeras etapas no se realizan comunicaciones pero, a partir de una cierta etapa (parámetro que viene determinado por el coste de tiempo de comunicación y computación) se vuelven a realizar comunicaciones entre los PEs.

Ahora consideraremos la implementación del operador η. En el algoritmo RC, la memoria de cada PE no almacena todos los datos necesarios para la computación de su primer nodo (excepto el primer PE). La aproximación más simple sería ejecutar una llamada cada vez que se necesiten datos remotos, (operadores η^{fil} y η^{col}). Esto sería inaceptablemente lento pues tiene el coste

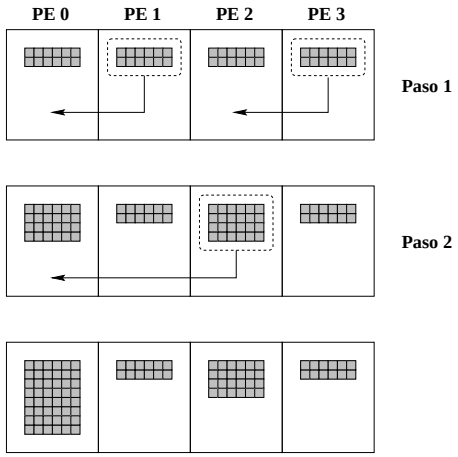


Figura 3.14: Implementación del operador $E^{fila,mem}$ de una matriz de 2 en 4 PEs. En el primer paso envía los PEs 1 y 2 a los PEs 0 y 3, respectivamente. En el segundo paso se envía del PE 2 al PE 0. Así, al final en sólo $\log_2 4$ pasos el PE 0 tiene todo el array completo que al principio estaba distribuido entre los PEs.

de un tiempo de latencia de envío por cada dato remoto accedido. Abrir los canales de comunicación es mucho más costoso temporalmente que la cantidad de datos que se pasen. Aplicamos una alternativa al algoritmo RC consistente en almacenar en una cola las peticiones para ser enviadas en el último paso, $u-1$, (después de que cada petición ha sido almacenada en la cola), y ejecutando los restantes nodos. Como se puede ver en la figura 3.15, las ecuaciones marcadas por un círculo son enviadas juntas después del paso 2 y serán ejecutadas después las ecuaciones marcadas con un cuadrado.

En las tablas 3.1 y 3.2 mostramos los resultados de la ejecución de los algoritmos Müller y Scheerer y RC sobre el AP1000.

El algoritmo de Wang y Mou tiene una estructura divide–y–vencerás y lo implementamos siguiendo la cadena de operadores del Algoritmo 8 (primera cadena). Finalmente, el algoritmo PARA-RC presenta una estructura divide–y–vencerás extendida y lo implementamos usando la cadena de operadores del Algoritmo 10 (tercera cadena). También, en ambos casos, hemos intentado optimizar las comunicaciones. Como muchos pasos requieren comunicaciones y el número de datos que se comunica es grande, intentamos solapar comunicaciones y computaciones. Para conseguirlo ejecutamos las mariposas en cada paso en dos fases diferentes. Mientras se están ejecutando la mitad de las mariposas, el resultado de la otra mitad está siendo enviada.

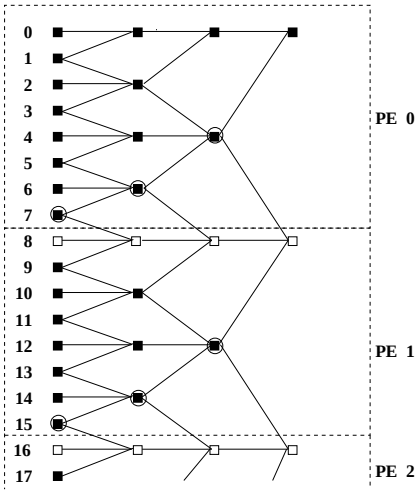


Figura 3.15: Flujo de datos del algoritmo RC con 8 datos por PE. Los nodos con círculo son enviados juntos después del segundo paso y los nodos con cuadrados blancos son ejecutados.

Tabla 3.1: Medidas del tiempo de ejecución (en milisegundos) sobre el AP1000 para el algoritmo de Müller y Scheerer.

	Número de datos					
Número de PEs	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
1	5.36	21.51	94.68	381.83	1529.24	–
2	2.73	10.75	45.96	190.74	764.47	–
4	1.40	5.42	21.66	94.84	382.04	1529.54
8	0.75	2.77	10.79	46.04	190.81	764.56
16	0.52	1.53	5.57	21.83	95.02	382.26
32	0.39	0.89	2.93	10.96	46.22	191.05
64	0.33	0.57	1.60	5.63	21.90	95.10
128	0.37	0.50	1.01	3.06	11.07	46.34
256	0.41	0.48	0.73	1.28	5.78	22.02
512	0.58	0.61	0.74	1.23	3.26	11.28

Tabla 3.2: Medidas del tiempo de ejecución (en milisegundos) sobre el AP1000 para el algoritmo RC.

	Número de datos					
Número de PEs	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
1	3.56	14.47	61.85	–	–	–
2	1.91	7.46	30.37	125.74	–	–
4	1.03	3.79	15.15	62.27	–	–
8	0.59	1.97	7.56	30.19	126.88	–
16	0.38	1.07	3.89	15.06	63.13	–
32	0.28	0.64	2.04	7.56	30.51	125.88
64	0.24	0.43	1.12	3.92	15.26	62.38
128	0.23	0.33	0.68	2.10	7.67	30.30
256	0.23	0.29	0.48	1.17	3.95	15.17
512	0.24	0.28	0.39	0.74	2.16	7.68

Esto se puede ver en las figuras 3.16.a y 3.16.b para el caso del algoritmo divide–y–vencerás. Esto se hace para evitar que un PE esté ocioso después de enviar un mensaje esperando la recepción de otro mensaje. Este solapamiento puede aplicarse a muchos de los algoritmos que hemos visto en este capítulo. La carga computacional no se incrementa y se solapa con los periodos de comunicación. Esta aproximación ofrece la posibilidad de solapar casi la totalidad de comunicaciones con computaciones, ver figura 3.16.c, como es el caso de los algoritmos PARA-RC y doblamiento sucesivo de Wang y Mou. Si el hardware soporta simultanear comunicaciones y cálculos, el código puede ser fácilmente transformado. En las tablas 3.3 y 3.4 mostramos los tiempos para los algoritmos doblamiento sucesivo de Wang y Mou y PARA-RC en el AP1000 con solapamiento de comunicaciones y computaciones.

El algoritmo de desacoplamiento recursivo de Spaletta y Evans muestra una estructura divide–y–vencerás base 2 en el paso preliminar y en los dos primeros pasos y una estructura divide–y–vencerás de base potencia de dos en el resto de las etapas del paso 3. Ha sido implementado usando la cadena de operadores de la proposición 9, ecuación 3.52. En la tabla 5 mostramos los tiempos de ejecución.

El algoritmo para ST con el mejor tiempo de los que hemos tratado ha sido el algoritmo RC. El de Müller y Scheerer (Algoritmo 6) y el RC (Algoritmo 7) son los más rápidos. Pero el algoritmo RC es casi dos veces más rápido que el de Müller y Scheerer, ver figura 3.17.a. Ambos muestran el problema

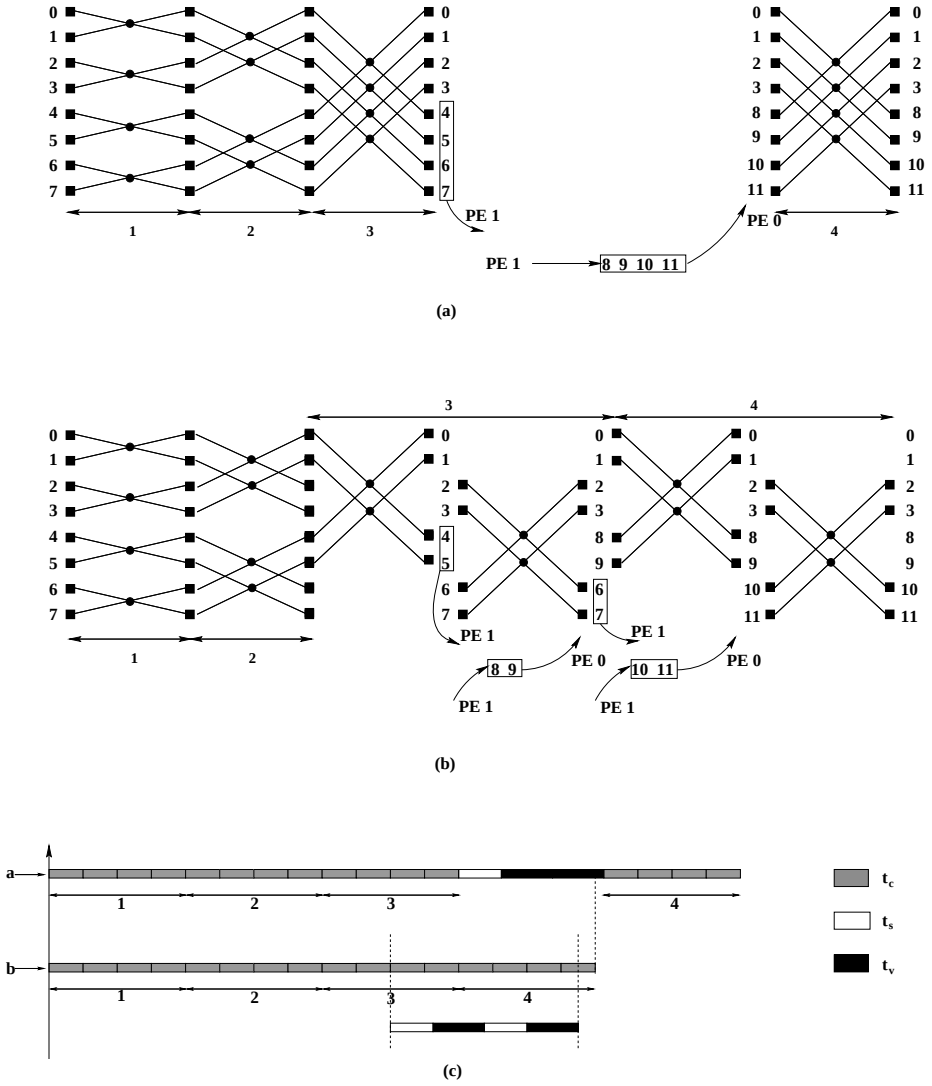


Figura 3.16: Flujo de datos en el PE 0 para el algoritmo doblamiento sucesivo de Wang y Mou de $N = 16$ datos sobre dos PEs. (a) En este caso el PE envía todos los datos en un único mensaje y espera la llegada inactivamente de los datos procedente del PE1. (b) Solapamiento de comunicaciones y computaciones por lo que el tiempo de comunicación esta enmascarado con las comunicaciones. (c) Diagrama temporal de los casos (a) y (b). t_c es el tiempo de computación de una mariposa, t_s es el tiempo de latencia y t_v el tiempo de transferencia de un elemento.

Tabla 3.3: Medidas del tiempo de la ejecución (en milisegundos) sobre el AP1000 para el algoritmo de doblamiento sucesivo de Wang y Mou.

	Número de datos					
Número de PEs	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
1	50.06	257.04	1209.91	–	–	–
2	26.26	127.94	629.24	–	–	–
4	13.88	66.06	324.01	–	–	–
8	7.39	34.13	161.42	774.43	–	–
16	4.02	17.88	82.13	393.78	1780.88	–
32	2.17	9.44	42.23	195.37	918.28	–
64	1.26	5.09	21.96	98.70	466.38	2077.20
128	0.73	2.69	11.52	50.39	229.54	1068.59
256	0.46	1.52	6.12	26.17	115.40	540.33
512	0.17	0.91	3.25	13.62	58.92	265.25

Tabla 3.4: Medidas del tiempo de la ejecución (en milisegundos) sobre el AP1000 para el algoritmo PARA-RC.

	Número de datos					
Número de PEs	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
1	32.16	190.73	906.20	4201.29	–	–
2	16.25	77.75	446.55	2074.84	–	–
4	8.31	39.08	220.71	1025.59	–	–
8	4.31	19.81	92.02	506.41	2318.58	–
16	2.22	10.13	46.14	250.34	1148.33	–
32	1.25	5.23	23.34	106.42	566.03	2566.47
64	0.73	2.73	11.99	53.42	280.61	1273.65
128	0.49	1.54	6.27	27.18	121.87	630.72
256	0.37	0.92	3.27	14.03	61.36	312.72
512	0.33	0.60	1.86	7.33	31.19	137.83

Tabla 3.5: Medidas del tiempo de la ejecución (en milisegundos) sobre el AP1000 para el algoritmo de Spaletta y Evans.

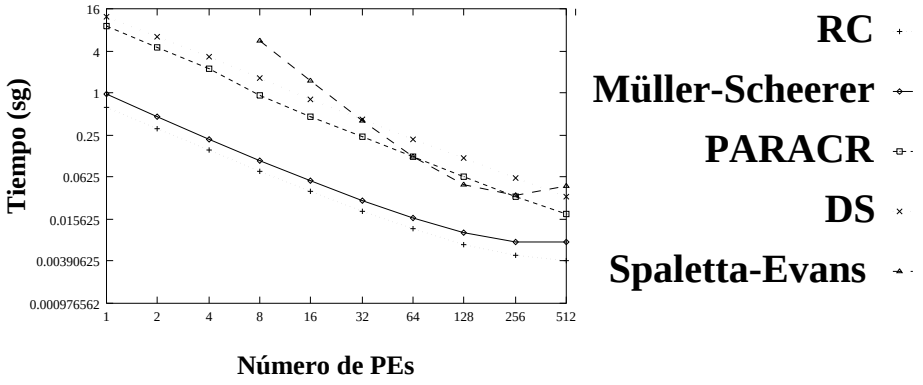
	Número de datos				
Número de PEs	2^{10}	2^{11}	2^{12}	2^{14}	2^{16}
1	195.24	766.26	–	–	–
2	37.39	143.25	557.25	–	–
4	10.40	38.18	144.74	–	–
8	3.27	10.70	38.74	561.44	–
16	1.30	3.47	11.06	146.98	–
32	0.78	1.50	3.74	40.01	566.08
64	0.78	1.08	1.82	12.11	149.48
128	1.19	1.32	1.62	4.81	41.79
256	2.14	2.21	2.33	3.47	14.17
512	4.13	4.17	4.22	4.68	8.14

de todos los algoritmos con una estructura en árbol: una mala escalabilidad, que todavía es más notable en el segundo algoritmo (ver figuras 3.17.b y 3.17.c). En ambos casos, los nodos computacionales en cada paso se reducen en un factor N/r pero el segundo algoritmo tiene una mayor complejidad computacional por nodo.

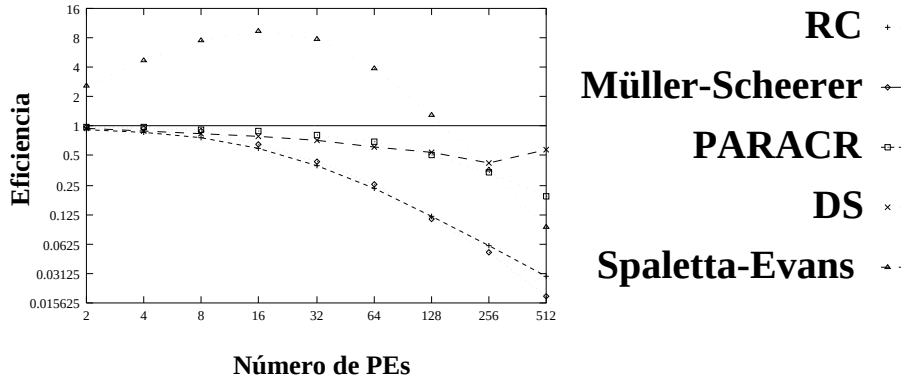
Por otra parte, los algoritmos de doblamiento sucesivo de Wang y Mou (Algoritmo 8), PARA-RC (Algoritmo 10) y el algoritmo de desacoplamiento recursivo de Spaletta y Evans (Algoritmo 9) son peores que los otros algoritmos. El algoritmo PARA-RC ha sido mejor en tiempo que el algoritmo de doblamiento sucesivo de Wang y Mou, ver figura 3.17.a. Esto es debido a que los nodos utilizan una tríada de ecuaciones para los cálculos, resultando una mayor carga computacional, y mejor escalabilidad como consecuencia del solapamiento entre comunicaciones y computaciones (ver figuras 3.17.b y 3.17.c).

Los algoritmos PARA-RC y Spaletta y Evans no pueden competir con los algoritmos RC y Müller y Scheerer en términos de tiempo de ejecución. Sin embargo, muestran un buen rendimiento en términos de aceleración y eficiencia, consiguiendo sobreaceleración, ver figura 3.17. Pero la memoria necesaria para el algoritmo de Spaletta–Evans es la peor, seguido del algoritmo de Müller–Scheerer.

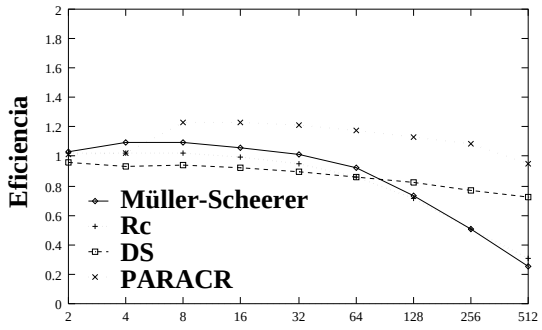
En las figuras 3.18 mostramos la aceleración relativa obtenida para algunos tamaños de la secuencia de datos de entrada para los diferentes algorit-



(a)



(b)



(c)

Figura 3.17: (a) Gráficas de tiempos para los algoritmos RC, Müller y Scheerer, PARA-RC, doblamiento sucesivo de Wang y Mou y Spaletta y Evans para $N = 2^{10}$ datos. (b) Eficiencia. (c) Eficiencia para $N = 2^{14}$ datos de los algoritmos RC, Müller y Scheerer, PARA-RC y doblamiento sucesivo de Wang y Mou.

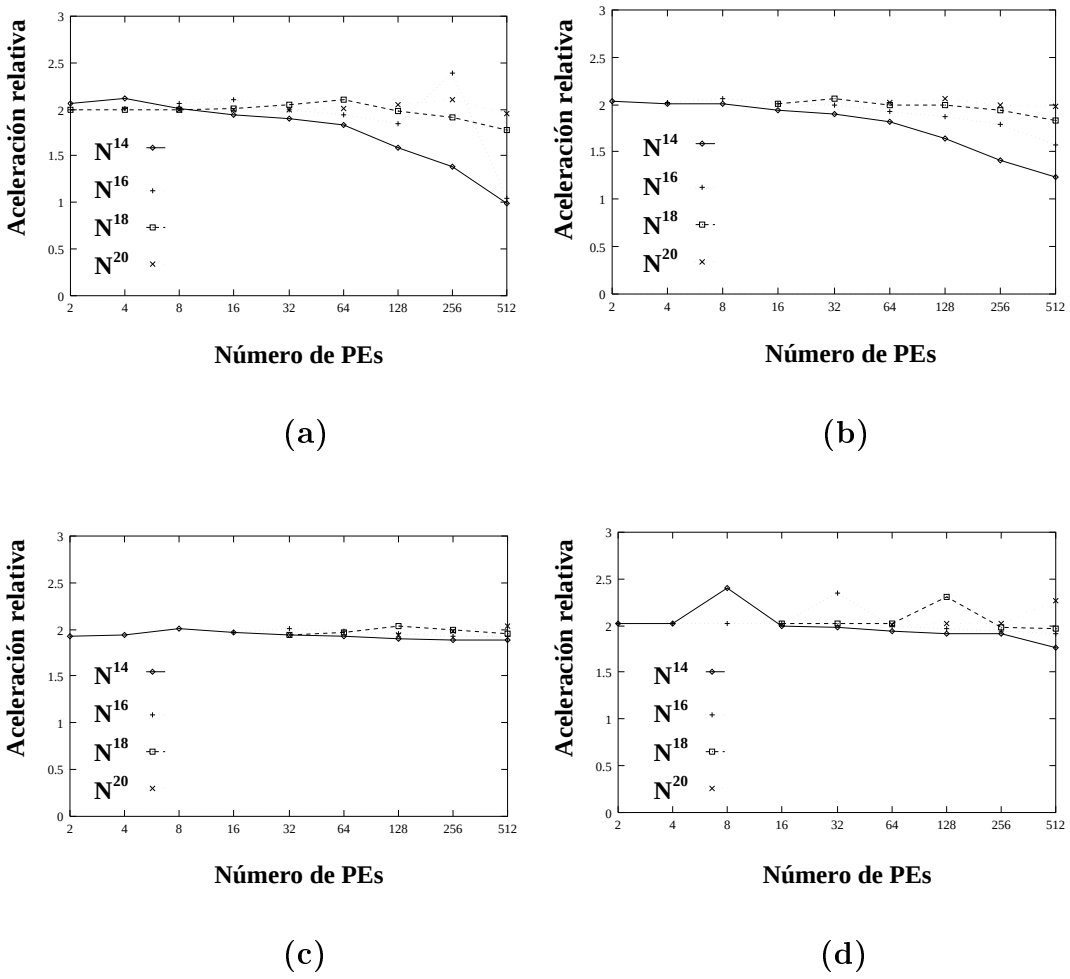


Figura 3.18: Aceleración relativa para $N = 2^{14}$, $N = 2^{16}$, $N =^{18}$ y $N = 2^{20}$ datos. (a) Algoritmo Müller y Scheerer. (b) Algoritmo RC. (c) Algoritmo doblamiento sucesivo de Wang y Mou (d) Algoritmo PARA-RC.

mos. Estos resultados muestran que las técnicas empleadas en la paralelización dan un excelente rendimiento sobre un computador con topología malla. En todos los algoritmos la aceleración relativa está próximo a dos. Hemos obtenido una sobreaceleración en estos algoritmos cuando cada PE contiene los mismos datos que el tamaño de la cache de cada PE ($2^{14} \cdot 8$bytes=128 Kbytes).

Capítulo 4

Árboles completos

4.1 Introducción

Dentro del campo general del mapeado de grafos, un problema de gran interés es el mapeado de árboles r–arios completos sobre las topologías de array lineal y malla. El interés del mapeado de árboles r-arios completos sobre arrays lineales y mallas se orienta tanto a la implementación VLSI como a la programación paralela. Los árboles aparecen en múltiples campos de las ciencias de computación donde su ámbito de aplicación se puede agrupar en los siguientes grupos:

- **En estructuras de datos**. Las estructuras de datos organizadas mediante árboles, usando listas enlazadas y punteros para su representación, se utilizan para realizar búsquedas, recuperación y actualización de la información. Los algoritmos que realizan *búsqueda hacia atrás (Bractracking)* determinan la solución óptima de un problema realizando una búsqueda sistemática en el conjunto de soluciones, que está organizado en forma de árbol.

- **En estructuras de computación**. Existen máquinas paralelas que organizan los PEs en forma de árbol, como la CM (*Connection Machine*). También son muy atractivos desde el punto de vista de la integración VLSI. Con el desarrollo de la tecnología VLSI se hace posible construir grandes bloques de silicio que contienen muchos elementos de procesamiento interconectados de muy diversas formas. Muchos patrones de interconexión han sido propuestos para resolver una gran variedad de problemas. Algunos patrones típicos son arrays lineales,

arrays rectangulares y hexagonales, árboles binarios, etc. El principal problema es que mientras un cierto patrón es bueno para una determinada aplicación lo es menos para otras. En el caso del árbol, se ha mostrado que necesita interconexiones simples y regulares lo cual reduce sensiblemente el coste.

- **Algoritmos con flujo de datos en árbol**. El árbol r–ario completo es la estructura subyacente a la aplicación del algoritmo DV para la resolución de muchos problemas. Esta estrategia es universalmente aceptada como uno de los paradigmas básicos en el diseño de algoritmos paralelos. Esta clase de algoritmos son secuenciales entre niveles, inherentemene paralelos en cada nivel e irregulares en el número de datos entre niveles. Existen algoritmos tipo árbol sencillos que realizan subtareas de otros más complejos. Ejemplos típicos son el cálculo del producto de N datos para una operación asociativa, la obtención del máximo de N elementos para determinar una relación de orden, algoritmos de ordenación y mezcla de listas, etc. Ejemplos más específicos aparecen en todas las áreas de la ciencia y la ingeniería.

En general, no es posible mapear un árbol r-ario sobre un array o malla de PEs de tal forma que se asigne un único nodo del árbol por PE y que las comunicaciones generadas por los arcos sean entre PEs vecinos (proyección de *expansión* 1 y *dilatación* 1) [46]. Esto se debe a que el número de nodos de un árbol es mayor que en un array lineal o en una malla. Así, para un multiprocesador de orden n, el número de PEs conectados en un array lineal es n y en el caso de la malla n^2, frente al número de nodos de un árbol r-ario de n niveles, que es $O(r^n)$. Este problema no existe en el mapeado de árboles binarios en multiprocesadores de topología hipercubo [117, 110, 111]. El número de PEs de un hipercubo de dimensión n es 2^n, que es del mismo orden que el número de nodos de un árbol binario de n niveles.

En la bibliografía se han propuesto distintas técnicas para mapear árboles r-arios sobre arrays o mallas que pueden encuadrarse en alguna de las tres siguientes categorías:

1. Mapeado que asigna un nodo del árbol a cada PE y con comunicaciones entre PEs no vecinos.

2. Mapeado que asigna más de un nodo del árbol a cada PE y con comunicaciones entre PEs vecinos.

3. Mapeado que asigna más de un nodo del árbol a cada PE y con comunicaciones entre PEs no vecinos.

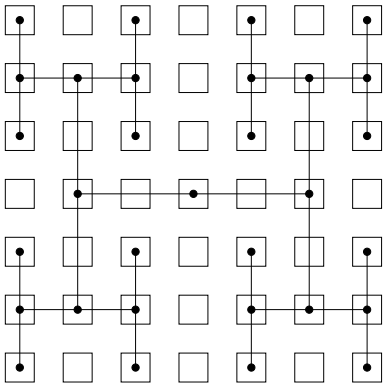


Figura 4.1: Mapeado de un árbol binario completo de cinco niveles sobre una malla 7×7 con la aproximación árbol en H.

La primera categoría de mapeado se ha utilizado extensivamente en la implementación de estructuras de tipo árbol binario en VLSI. Las primeras propuestas para proyectar (*laying out*) arquitecturas árbol sobre VLSI estaban basadas en clásica aproximación árbol en H [57], ver figura 4.1. Tales proyecciones (*layouts*) mostraban áreas ineficientes y largas interconexiones, particularmente para grandes árboles con un gran número de nodos de procesamiento. Otros diseños, [46, 118], empleaban los esquemas basados en baldosas (*tiles*) para mejorar la utilización del silicio y el retardo de propagación. El esquema de proyección del árbol propuesto usa una estrategia jerárquica tal que el árbol del tamaño requerido es construido interconectando de forma apropiada un conjunto de *tiles* básicos. Algunos de estos mapeados, si están basados en la estructura H o en *tiles*, presentan el inconveniente de que la longitud de comunicación crece indefinidamente conforme nos alejamos de la raíz del árbol.

Los mapeados pertenecientes a la segunda categoría son más adecuados para implementar tanto en VLSI como en los multiprocesadores. Las comunicaciones se restringen a PEs vecinos a costa de asignar más de un nodo del árbol a cada PE. La solución más básica perteneciente a esta categoría consiste en asignar uno de los niveles completos del árbol a cada PE [24, 26]. Esta solución es fácil de programar pero extremadamente desbalanceada, pues el número de nodos de cada nivel del árbol r-ario aumenta en un factor r. Las soluciones más eficientes pertenecientes a esta segunda categoría son las que balancean la distribución de nodos entre los PEs. A modo de ejemplo, en la figura 4.2 mostramos la distribución de un árbol binario de 32 nodos sobre una malla de 16 PEs, asignando 2 nodos a cada uno de los PEs. En

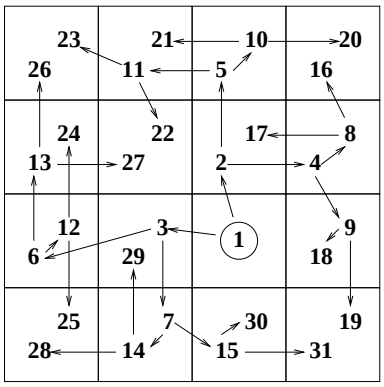


Figura 4.2: El mapeado de un árbol binario completo de 5 niveles (31 nodos) sobre una malla de 4×4 PEs. La distribución de los nodos entre los PEs es balanceada (2 nodos por PE salvo el PE que contiene el nodo raíz que sólo tiene un nodo) y todas las comunicaciones son entre PEs vecinos. La raíz del árbol se ha asignado a uno de los PEs centrales de la malla.

este ejemplo, la raíz del árbol se coloca en uno de los nodos centrales de la malla, de forma que los nodos del último nivel del árbol alcanzan los PEs de las esquinas de la malla. En el caso general, si el tamaño de la malla es pequeño, el particionamiento de los nodos del árbol entre los PEs puede realizarse directamente, explorando todos los posibles mapeados. Sin embargo, si el tamaño de la malla es grande este método es muy costoso (el problema es NP-completo). En el caso del ejemplo anterior, el árbol binario mínimo que puede mapearse sobre una malla de ese tamaño (4×4 PEs) contiene 5 niveles y 31 nodos. Sin embargo, en el caso de una malla de 16×16 PEs, el árbol binario mínimo que puede mapearse contiene 17 niveles. Esto implica distribuir 131.071 nodos entre los 256 PEs (512 nodos por PE). El problema, es por tanto, demasiado grande para abordarlo directamente.

La tercera categoría de mapeado también se emplea en los multiprocesadores. La solución básica utilizada es la que distribuye los nodos entre los PEs de tal forma que en las primeras etapas no se precisan comunicaciones. Cada PE realiza todas las computaciones posibles con los nodos que tiene almacenados. Una vez completadas estas, los PEs intercambian datos, en general, entre PEs no vecinos. El balanceo de la carga computacional entre los PEs también se sacrifica con el objeto de maximizar el número de operaciones que pueden realizarse sin necesidad de comunicaciones. Este tipo de soluciones es útil cuando las comunicaciones entre PEs vecinos son muy costosas y comparables a las que tienen lugar entre PEs no vecinos. Así, por ejemplo

en [43] se estudia el mapeado de árboles orientados sobre arrays lineales y en [109] se utiliza el ordenamiento de tipo serpiente para mapear árboles orientados sobre multiprocesadores de interconexión malla. Otro mapeado de este tipo sobre multiprocesadores de interconexión malla se propone en [44]. En [1] se utiliza la técnica *tree–contraction* como aproximación estándar en la paralelización de subproblemas de escasa granularidad. En [94] proponen una solución, llamada *distributed–tree–contraction* para subproblemas de gran granularidad basada en el método de *grain–packing* [60]. Esta solución evita el excesivo coste en las comunicaciones causado por la adaptación directa del algoritmo descrito en [1].

A continuación se definirán de forma breve y precisa los términos más usados y se analizará el ámbito de aplicación de los mismos. Luego presentaremos dos metodologías generales para realizar el particionamiento y mapeado de árboles r-arios completos sobre arrays y mallas considerando comunicaciones entre PEs vecinos. En los diseños propuestos previamente en la bibliografía, el estudio se restringe a árboles binarios completos y ninguna de las técnicas que hemos recopilado se ha generalizado a árboles completos r–arios. Como en los otros diseños existentes en la bibliografía, suponemos que el tamaño de los PEs es relativamente grande comparado con el ancho de interconexión. Por tanto, la principal preocupación de nuestro diseño es la alta utilización de los PEs; y para el rendimiento, la minimización de la distancia de interconexión de los nodos conectados en el árbol.

En la primera parte de este capítulo consideramos una distribución balanceada de la carga (categoría 2), teniendo en cuenta, sólo el problema del mapeado, sin considerar la planificación (*scheduling*) de las operaciones representadas por los nodos. Es decir, consideraremos que los árboles son no orientados.

Al final del capítulo (sección 4.5) extenderemos la metodología para considerar árboles orientados. Es decir, supondremos que una tarea representada por un nodo no podrá comenzar a ejecutarse hasta que finalicen las tareas previas. Consideraremos la distribución y temporización de las tareas para conseguir un buen baleaceamiento de la carga.

4.2 Planteamiento del problema

Consideraremos árboles r-arios completos donde cada nodo interno (un nodo no hoja), tiene r hijos.

En los procedimientos de mapeado de los árboles r-arios que vamos a

proponer permitiremos sólo comunicaciones entre PEs vecinos. Así, dos nodos unidos por un arco del árbol se asignarán al mismo PE o a PEs vecinos. Esto implica que, poniendo la raíz del árbol en uno de los PEs del array o de la malla, la distancia que pueden alcanzar los nodos del árbol se ve limitada por el propio número de niveles del árbol. En concreto, sobre arrays de V PEs y colocando la raíz en el PE de una esquina, pueden mapearse árboles tales que $n \geq V$. La igualdad, por ejemplo, puede conseguirse en el caso en que se asigne un nivel del árbol a cada PE. Sobre una malla de $W \times V$ PEs, colocando la raíz del árbol en una esquina, pueden mapearse árboles tales que $n \geq V + W - 1$. Si la raíz del árbol se coloca en uno de los PEs centrales, deberá verificarse $n > \lfloor V/2 \rfloor$ en el caso del array o $n > \lfloor V/2 \rfloor + \lfloor W/2 \rfloor$ en el caso de la malla.

Impondremos que la distribución de los nodos del árbol entre los PEs sea balanceada. En concreto, asignaremos un máximo de $U = \lceil N/W \times V \rceil$ nodos a cada PE, siendo $W \times V$ el número de PEs. A medida que consideremos arrays y mallas de mayor tamaño, el número de nodos por PE será mayor, pues los árboles crecen en mayor proporción que los arrays o mallas. Así, un array de longitud n contiene n PEs y una malla de lado n, n^2 PEs, mientras que un árbol de n niveles contiene $O(r^n)$ nodos.

Realizado el mapeado de un árbol de n niveles sobre un array o una malla, este resultado puede utilizarse para mapear un árbol de un mayor número de niveles sobre el mismo array o malla. Para ello, utilizaremos un procedimiento consistente en contraer un árbol de $n+1$ niveles sobre uno de n niveles, mapeando este último. Así, asignaremos el conjunto de nodos $\{i \cdot r + j,\ 0 \leq j < r\}$ del primer árbol al nodo i del segundo. Esta contracción presenta una singularidad en el nodo 1. Para evitar esta singularidad debemos considerar en el árbol un nodo adicional conectado a la raíz y designado como nodo 0. En la figura 4.3 mostramos un ejemplo concreto de este procedimiento.

4.3 Mapeado de árboles r-arios en arrays

En esta sección estudiaremos el mapeado de un árbol r-ario sobre un array, primero asignando la raíz del árbol al PE de una esquina del array y, a continuación, asignándola al PE central.

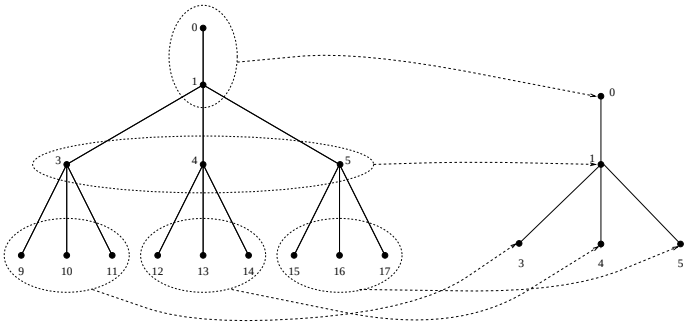


Figura 4.3: Contracción de un árbol 3-ario de 3 niveles sobre otro de 2 niveles. Así, por ejemplo, asignamos los nodos 9, 10 y 11 del primero al nodo 3 del segundo. Para que la contracción esté bien definida hemos incluido en ambos árboles un nodo adicional conectado a la raíz y designado como nodo 0.

4.3.1 Asignación de la raíz del árbol al PE de una esquina del array

En este primer caso consideraremos la distribución de los nodos de un árbol r-ario de n niveles ($N = \frac{r^n-1}{r-1}$ nodos) sobre un array de n PEs. Como hemos visto en la sección anterior esto no supone ninguna restricción, pues árboles mayores pueden mapearse contrayéndolos previamente sobre árboles de n niveles.

El mapeado de los nodos del árbol entre los PEs la haremos imponiendo las siguientes condiciones:

1. La distribución de los nodos del árbol entre los PEs será balanceada. A cada PE le asignaremos $U = \lceil N/n \rceil$ nodos, excepto al PE 1, que le asignaremos los restantes hasta completar todos los nodos del árbol.

2. Las comunicaciones serán sólo entre PEs vecinos. Los nodos que en el árbol están unidos por arcos los asignaremos al mismo PE o a PEs vecinos.

3. Las comunicaciones entre PEs serán siempre en el mismo sentido. La raíz del árbol será asignada al primer PE del array. Los hijos de cada nodo se asignarán al PE que contiene dicho nodo o al PE de índice inmediatamente superior. De esta forma, partiendo de la raíz de árbol y ascendiendo a niveles superiores, las comunicaciones entre PEs tendrán lugar siempre en el mismo sentido (de PEs de menor índice a PEs de mayor índice).

Los PEs del array los numeraremos de 1 a n y denominaremos $C(i)$ el conjunto de nodos del árbol que asignaremos al PE i. De las condiciones impuestas arriba al mapeado se deducen las restricciones dadas por el siguiente conjunto de lemas:

Lema 11 *Si $C(i)$ y $C(i+1)$ son, respectivamente, el conjunto de nodos asignados a los PEs i y $i+1$, entonces el conjunto $C(i)$ debe incluir los nodos $\{j \mid j \cdot r \in C(i+1),\ j \notin C(i+1)\}$, mientras que el conjunto $C(i+1)$ incluirá los nodos $\{j \mid j/r \in C(i),\ j \notin C(i)\}$.*

Demostración Este lema se deduce de las condiciones 2 y 3. Así conseguimos que el padre de un nodo se encuentre en el mismo PE que dicho nodo o en el PE de índice inmediatamente inferior. □

Lema 12 *Al PE i debemos asignarle nodos de niveles mayores o iguales que i.*

Demostración A partir de las condiciones 2 y 3. Puesto que hemos asignado el nodo 1 al PE 1 y las comunicaciones que pueden generar los arcos del árbol son sólo entre PEs vecinos, al llegar al PE i debemos asignarle nodos de niveles mayores o iguales que i. □

De aquí se deduce que al PE n se le debe asignar nodos del nivel n del árbol.

Lema 13 *Si m_i es el número de nodos del nivel i asignados al PE i, debe verificarse $m_{i+1} \leq r \cdot m_i,\ i = 1, \ldots, n-1$.*

Demostración A partir de las condiciones 2 y 3. Si la desigualdad no se verifica no será posible asignar comunicaciones entre PEs vecinos a los arcos que unen los padres de los niveles i con los hijos de los niveles $i+1$ del árbol. □

Lema 14 *Debe verificarse $\lceil U/r^{n-i} \rceil \leq m_i \leq \lfloor \frac{(n-i+1)U}{N_{n-i+1}} \rfloor,\ i = 1, \ldots, n$ con $N_{n-i+1} = \frac{r^{n-i+1}-1}{r-1}$.*

Demostración A partir de las condiciones 1, 2 y 3. La primera desigualdad se deduce de lo siguiente: el PE n contiene U nodos del nivel n, el PE $n-1$ debe incluir los $\lceil U/r \rceil$ padres de estos nodos, el PE $n-2$ los $\lceil U/r^2 \rceil$ padres de estos últimos, y así sucesivamente.

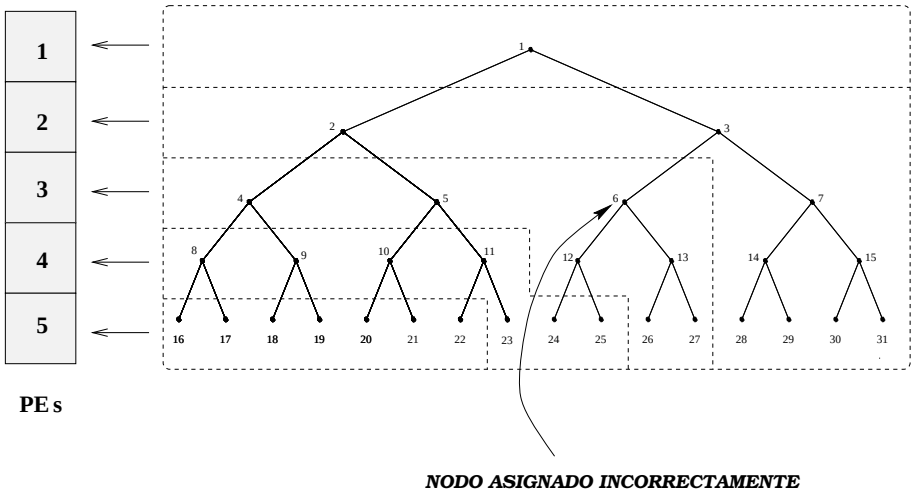


Figura 4.4: Ejemplo de mapeado que no verifica el Lema 15. Para $i = 3$, es $m_3 = 3$ (nodos 4, 5 y 6) y $\lceil m_i/r^{i-2} \rceil = 2$. Como no se verifica el lema 15, el PE 1 sólo podrá contener el nodo 1, por lo que la distribución de los nodos no puede ser balanceada.

En cuanto a la segunda desigualdad el razonamiento es como sigue. Sea Ω_i el subgrafo formado por los nodos asignados a los PEs del i al n. El número de niveles de Ω_i es $n-i+1$ y el número total de nodos es $(n-i+1)U$. El número máximo de subárboles en el nivel $n-i+1$ incluidos en Ω_i será $\left\lfloor \frac{(n-i+1)U}{N_{n-i+1}} \right\rfloor$ donde N_{n-i+1} es el número de nodos que contiene uno de estos subárboles. Como las raíces de estos subárboles deben estar en el nivel i, de aquí se deduce que $m_i \leq \left\lfloor \frac{(n-i+1)U}{N_{n-i+1}} \right\rfloor$ □

Lema 15 *Excepto en el caso en que el PE 1 contenga sólo la raíz del árbol, debe verificarse* $m_i \leq \lceil (r-1)r^{i-2} \rceil,\ i = 1, \ldots, n.$

Demostración A partir de las condiciones 1, 2 y 3. En efecto, si $m_i > \lceil (r-1)r^{i-2} \rceil$ para algún i, entonces $m_2 = \lceil m_i/r^{i-2} \rceil = r$ y $C(1)$ sólo podrá contener la raíz del árbol. □

Este lema nos va a garantizar que podamos incluir los nodos suficientes en cada uno de los conjuntos asignados a los PEs. La figura 4.4 ilustra este lema con un contraejemplo.

En los procedimientos que describimos a continuación, asignaremos al PE n los U primeros nodos del nivel n del árbol, es decir, el conjunto de

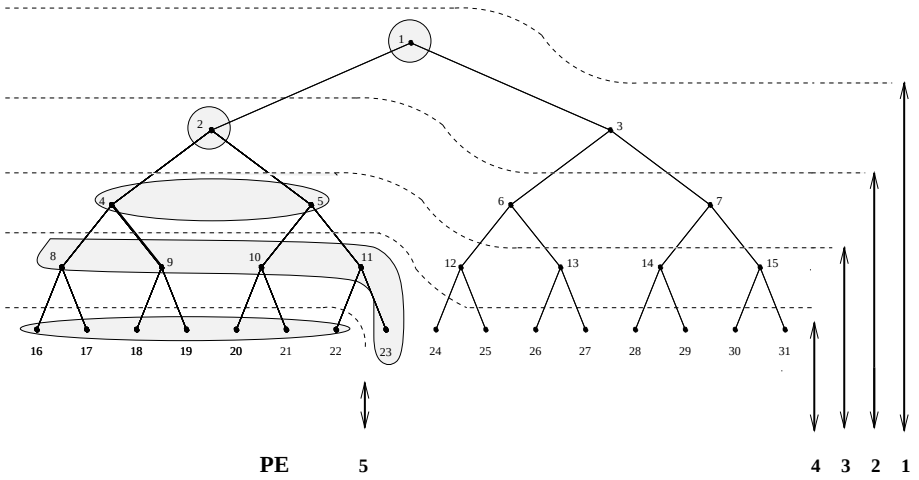


Figura 4.5: Ejemplo de las restricciones en la asignación de los nodos a los PEs. Hemos fijado la asignación de los nodos de índices 16 a 22 al PE 5. Esta primera asignación implica que los nodos de las zonas rayadas deben asignarse obligatoriamente a los correspondientes PEs. A continuación, los PEs deben completarse (hasta un total de $U = 7$ nodos) con nodos de las zonas especificadas.

nodos $C(n) = \{r^{n-1} + j,\ 0 \leq j < U\}$ (son posibles otras asignaciones). Según el lema 1, esta asignación nos fija la ubicación de los $\lceil U/r \rceil$ padres de $C(n)$, que deben asignarse al conjunto $C(n-1)$; además, debemos asignar los $\lceil U/r^2 \rceil$ padres de estos últimos al conjunto $C(n-2)$; y así sucesivamente. A continuación, debemos completar los conjuntos con nodos no asignados. Los lemas 11 a 15 nos restringen las posibilidades de elección. En la figura 4.5 mostramos estas restricciones para un ejemplo concreto.

Para completar los conjuntos, nosotros vamos a utilizar dos tipos de mapeado que, por su similitud con las técnicas de exploración de grafos, denominaremos mapeado en anchura y profundidad, respectivamente. En el primero de los casos, los conjuntos serán completados con nodos del nivel más alto posible (mapeado en anchura), mientras que en el segundo, los completaremos con nodos del nivel más bajo posible (mapeado en profundidad). La numeración de los niveles es la que se muestra en la figura 1.6.

Teorema 1 *Un árbol r-ario de n niveles puede mapearse sobre un array de n PEs empleando el siguiente procedimiento (mapeado en anchura):*

1. *Asignar al PE n el conjunto de nodos $C(n) = \{r^{n-1} + j,\ 0 \leq j < U\}$.*

2. *Asignar al PE i el conjunto de nodos* $\{j \mid j \cdot r \in C(i+1),\ j \notin C(i+1)\}$
3. *Completar el conjunto con nodos no asignados del nivel más alto posible.*
4. *Repetir los pasos 2 y 3 hasta completar los PEs.*

siendo N el número de nodos del árbol y $U = \lceil N/n \rceil$ el número de nodos asignados a cada PE, excepto al PE 1.

Demostración Debemos demostrar los dos siguientes puntos:

1. Que los hijos de un nodo son asignados al mismo PE que dicho nodo o al PE de índice inmediatamente superior.
2. Que cada PE contiene U nodos.

Comenzaremos verificando el primer punto. Del procedimiento de asignación de los nodos se deduce que, si un nodo está situado en el PE $k+1$, su padre se encontrará en el PE k (suponiendo que no se encuentre en el PE $k+1$). Debemos comprobar la situación del resto de hijos de dicho padre situado en el PE k. Comprobemos que, si sus hijos no se encuentran en el PE $k+1$, entonces se encontrarán en el PE k. Esto lo podemos asegurar si, en cada etapa, han sido asignados todos los sucesores de los nodos del nivel más bajo. Así, sea Ω_i el subgrafo formado por los nodos asignados en las $n-i+1$ primeras etapas (nodos asignados al conjunto de PEs comprendidos entre el i y el n). Ω_i incluye un total de $m_i = \lceil U/r^{n-i} \rceil$ nodos del nivel i. Puesto que el total de nodos asignados $((n-i+1)U)$ es mayor o igual que el número de sucesores de los nodos del nivel i ($m_i \cdot N_{n-i+1}$, lema 14), Ω_i contiene a todos los sucesores de los nodos del nivel i.

Verificaremos, a continuación, el segundo punto. Puesto que se verifica $m_i \leq \lceil (r-1)r^{i-2} \rceil,\ i = 1, \ldots, n$ (lema 15), no van a existir nodos aislados no alcanzables para determinados conjuntos. Por tanto, una vez incluidos los nodos obligatorios en cada conjunto (dados por el lema 11), podemos completar estos con nodos adicionales hasta el total de U nodos.□

En la figura 4.6 mostramos la aplicación de este procedimiento sobre un ejemplo. En este ejemplo mapeamos un árbol 3-ario de 4 niveles sobre un array de 4 PEs. El teorema 1 se aplica de la siguiente forma:

1. Asignamos al PE 4 el conjunto de nodos $C(4)$=$\{$27, 28, 29, 30, 31, 32, 33, 34, 35, 36$\}$.

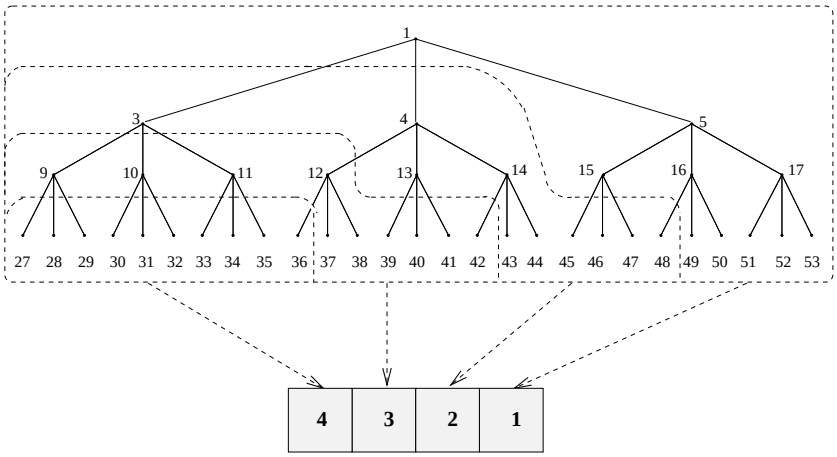


Figura 4.6: Mapeado en anchura (teorema 1) de un árbol 3-ario completo de 4 niveles (40 nodos) sobre un array lineal de 4 PEs. Cada PE contiene $U = 10$ nodos. A modo de ejemplo, observar que para el conjunto de nodos del PE 3, los nodos asignados obligatoriamente son los de índices 9, 10, 11, 12, 37 y 38. Este conjunto se completa con nodos adicionales (nodos de índices 39 a 42) del nivel más alto posible (el 4).

2. Asignamos al PE 3 el conjunto de nodos {9, 10, 11, 12}.

3. Completamos el conjunto $C(3)$ con los nodos 37, 38, 39, 40, 41 y 42 (del nivel 4).

2. Asignamos al PE 2 el conjunto de nodos {3, 4, 13, 14}.

3. Completamos el conjunto $C(2)$ con lo nodos 43, 44, 45, 46, 47 y 48 (del nivel 4).

2. Asignamos al PE 1 el conjunto de nodos {1, 15, 16}.

3. Completamos el conjunto $C(1)$ con los nodos 49, 50, 51, 52, 53 (del nivel 4), 17 (del nivel 3) y 5 (del nivel 2).

Teorema 2 *Un árbol r-ario de n niveles puede mapearse sobre un array de n PEs empleando el siguiente procedimiento (mapeado en profundidad):*

1. *Asignar al PE n el conjunto de nodos $C(n) = \{r^{n-1} + j,\ 0 \leq j < U\}$.*

2. *Asignar al PE i el conjunto $\{j \mid j \cdot r \in C(i+1),\ j \notin C(i+1)\}$.*

3. *Completar el conjunto con nodos no asignados del nivel más bajo posible cuyos hijos ya hayan sido asignados (incluidos los hijos que se van asignando en este paso). Las restricciones son las siguientes: no asignar a $C(i)$ nodos de niveles menores que i ni más de $\lceil (r-1)r^{i-2} \rceil$ nodos del nivel i.*

4. *Repetir los pasos 2 y 3 hasta completar los PEs.*

siendo N el número de nodos del árbol y $U = \lceil N/n \rceil$ el número de nodos asignados a cada PE, excepto al PE 1.

Demostración La demostración de este teorema es similar a la del teorema 1. Sea Ω_i el subgrafo formado por los nodos asignados en las $n-i+1$ primeras etapas (nodos asignados al conjunto de PEs comprendidos entre el i y el n). Ω_i incluye un total de $m_i = \min\{\lfloor \frac{(n-i+1)U}{N_{n-i+1}} \rfloor, \lceil (r-1)r^{i-2} \rceil\}$ nodos del nivel i ($N_{n-i+1} = \frac{r^{n-i+1}-1}{r-1}$). Puesto que se verifica el lema 14, Ω_i contiene a todos los sucesores de los nodos del nivel i. Además, al verificarse el lema 15, los conjuntos pueden completarse con nodos adicionales hasta el total de U nodos. □

En la figura 4.7 mostramos la aplicación del procedimiento descrito en el teorema 2 sobre el mismo ejemplo que en el teorema 1. Distribuimos los 40 nodos del árbol 3-ario de 4 niveles sobre un array de 4 PEs de la siguiente forma:

1. Asignamos al PE 4 el conjunto de nodos $C(4)$={27, 28, 29, 30, 31, 32, 33, 34, 35, 36}.

2. Asignamos al PE 3 el conjunto de nodos {9, 10, 11, 12}.

3. Completamos el conjunto $C(3)$ con los nodos 37, 38, 39, 40, 41 (del nivel 4) y 13 (del nivel 3).

2. Asignamos al PE 2 el conjunto de nodos {3, 4}.

3. Completamos el conjunto $C(2)$ con lo nodos 42, 43, 44, 45, 46, 47 (del nivel 4), 14 y 15 (del nivel 3).

2. Asignamos al PE 1 el conjunto de nodos {1, 5}.

3. Completamos el conjunto $C(1)$ con los nodos 48, 49, 50, 51, 52, 53 (del nivel 4) y 16, 17 (del nivel 3).

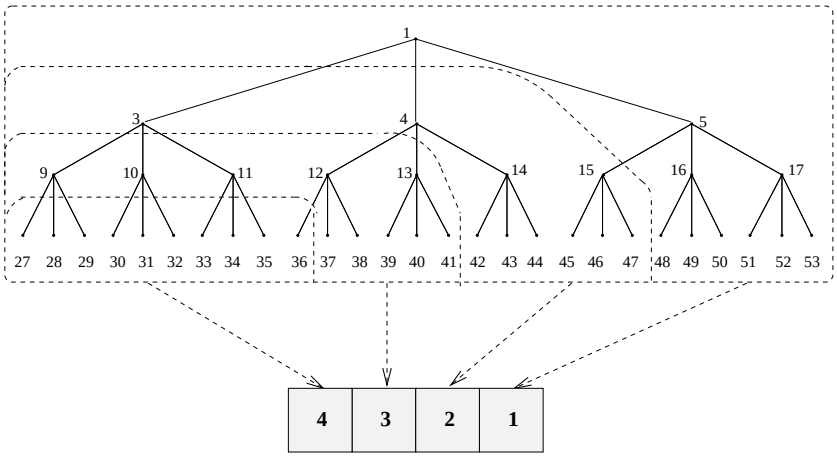


Figura 4.7: Mapeado en profundidad (teorema 2) de un árbol 3-ario completo de 4 niveles (40 nodos) sobre un array lineal de 4 PEs. Cada PE contiene $U = 10$ nodos. A modo de ejemplo, observar que para el conjunto de nodos del PE 3, los nodos asignados obligatoriamente son los de índices 9, 10, 11, 12, 37 y 38. Este conjunto se completa con nodos adicionales (nodos de índices 13, 39, 40, 41) de los niveles más bajos posibles (el 3 y el 4).

El procedimiento dado por el teorema 2 puede formularse de una forma alternativa, en la que comenzamos asignando nodos al PE 1. Esta formulación constituye el siguiente teorema.

Teorema 3 *Un árbol r-ario de n niveles puede mapearse sobre un array de n PEs empleando el siguiente procedimiento (mapeado en profundidad):*

1. *Asignar al conjunto de nodos del PE 1, $C(1)$, el nodo 1.*
2. *Completar el conjunto de la siguiente forma: si j es el nodo de índice más alto del conjunto con hijos todavía no asignados, añadir al conjunto el hijo de índice más alto del nodo j. Repetir este paso hasta completar el conjunto.*
3. *Asignar al PE i el conjunto $\{j \mid \lfloor j/r \rfloor \in C(i-1),\ j \notin C(i-1)\}$.*
4. *Repetir los pasos 2 y 3 hasta completar los conjuntos y los PEs.*

siendo N el número de nodos del árbol y $U = \lceil N/n \rceil$ el número de nodos asignados a cada PE, excepto al PE 1.

Demostración Este teorema se obtiene a partir del teorema 2, invirtiendo cada uno de los pasos de asignación de nodos a PEs. En efecto, según el teorema 2, el subgrafo Ω_i formado por los nodos asignados al conjunto de PEs del i al n puede generarse asignando, en primer lugar, los subárboles cuyas raíces son los primeros $\lceil (r-1)r^{i-2} \rceil$ nodos del nivel i y, a continuación, los subárboles cuyas raíces son los nodos del nivel $i+1$ todavía no asignados. Los subárboles se comienzan a asignar por los nodos del nivel más alto.

Según el teorema 3, el subgrafo Ψ_i formado por los nodos asignados al conjunto de PEs del 1 al $i-1$ puede generarse asignando, en primer lugar, todos los nodos de los niveles 1 a $i-1$ del árbol y los últimos r^{i-2} del nivel i y, a continuación, los subárboles que parten de los últimos nodos del nivel $i+1$. Los subárboles se comienzan a asignar por los nodos del nivel más bajo. De aquí se deduce que Ψ_i es el subgrafo complementario de Ω_i y que, por lo tanto, el teorema 3 puede obtenerse invirtiendo cada uno de los pasos del teorema 2. En la figura 4.8 mostramos un ejemplo de este proceso. □

Aplicaremos el nuevo procedimiento descrito por el teorema al ejemplo de la figura 4.7. En este caso, el teorema se aplicaría de la siguiente forma:

1. Asignamos al PE 1 el nodo 1.
2. Completamos el conjunto $C(1)$ con los nodos 5, 17, 53, 52, 51, 16, 50, 49 y 48 (en este orden).
3. Asignamos al PE 2 el conjunto de nodos {3, 4, 15}.
2. Completamos el conjunto $C(2)$ con los nodos 47, 46, 45, 14, 44, 43 y 42.
3. Asignamos al PE 3 el conjunto de nodos {9, 10, 11, 12, 13}.
2. Completamos el conjunto $C(3)$ con lo nodos 41, 40, 39, 38 y 37.
3. Asignamos al PE 4 el conjunto de nodos $C(4)$={27, 28, 29, 30, 31, 32, 33, 34, 35, 36}.

4.3.2 Asignación de la raíz del árbol a uno de los PEs centrales del array

A continuación pasamos a estudiar el caso del mapeado de árboles r-arios sobre un array de V PEs, asignando la raíz del árbol al PE central (si V es impar) o a uno de los dos PEs centrales (si V es par). Cada PE contendrá un

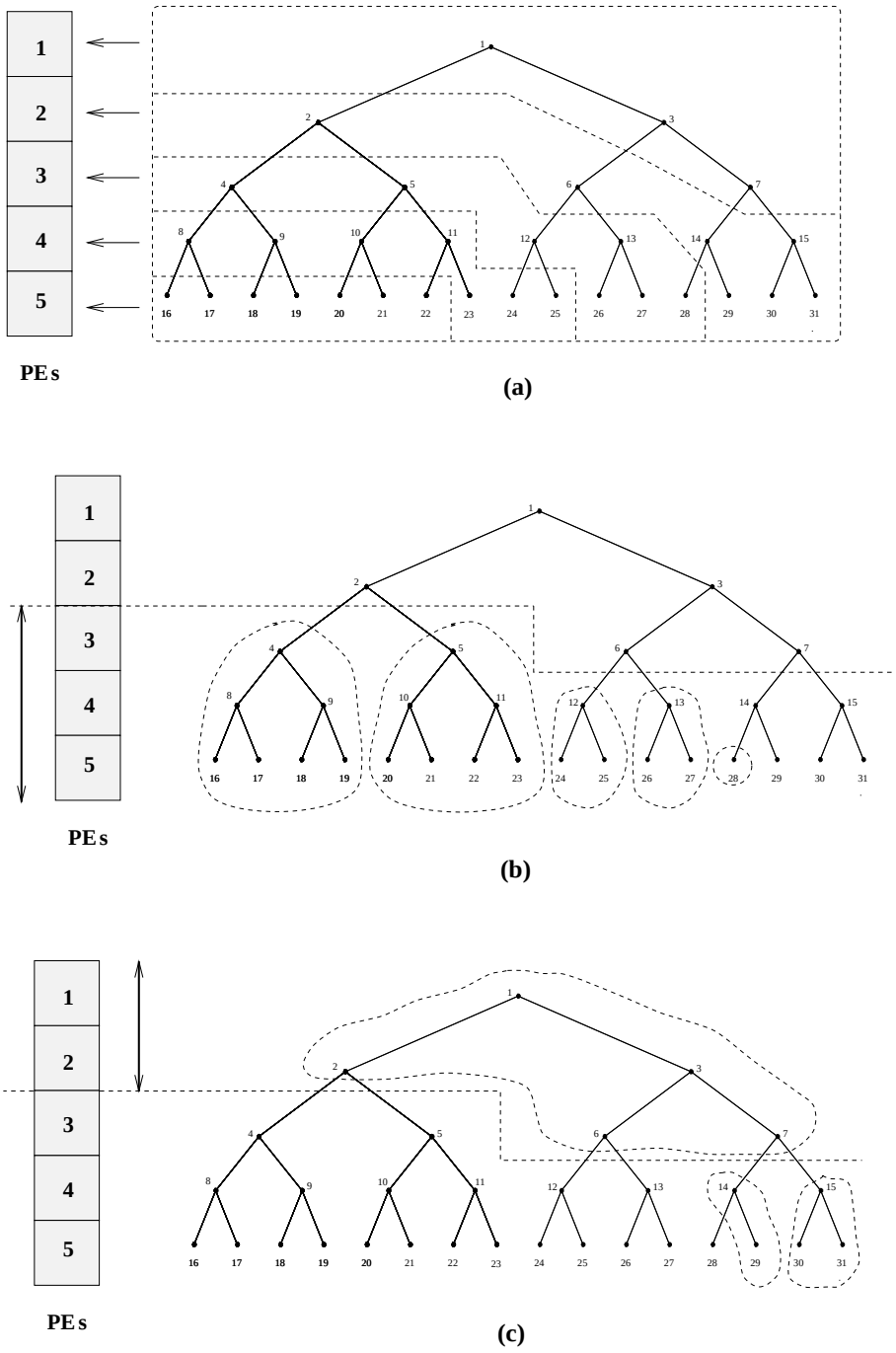


Figura 4.8: (a) Mapeado en profundidad de un árbol binario de 5 niveles sobre un array lineal de 5 PEs. (b) Obtención del subgrafo Ω_3 formado por los nodos asignados a los PEs del 3 al 5, según el teorema 2. (c) Obtención del subgrafo Ψ_3 formado por los nodos asignados a los PEs 1 y 2, según el teorema 3.

máximo de $\lceil N/V \rceil$ nodos. En este caso consideraremos árboles de un número de niveles $n = 1 + \lfloor V/2 \rfloor$. También pueden considerarse árboles mayores, contrayéndolos previamente sobre árboles de n niveles y mapeando estos últimos. El procedimiento que vamos a utilizar es el siguiente:

1. Si r es par, dividir simplemente el árbol en dos subárboles y asignar cada subárbol a una mitad del array. Si el número de PEs del array es impar, asignar la raíz al PE central. Si el número de PEs es par, asignar la raíz a cualquiera de los PEs centrales.

2. Si r es impar, dividir los nodos del árbol en dos conjuntos de $U\lfloor V/2 \rfloor$ y $N - U\lfloor V/2 \rfloor$ nodos, que asignaremos a cada una de las dos mitades del array (el PE central formará parte de la segunda mitad). Para obtener el segundo conjunto, partir de la raíz del árbol y aplicar el teorema 3 para seleccionar los $N - U\lfloor V/2 \rfloor$ nodos correspondientes. El primer conjunto contendrá los nodos restantes. A continuación, debemos subdividir cada uno de los dos conjuntos en varios subconjuntos cada uno de los cuales será asignado a un PE. En el primero de dichos subconjuntos debemos incluir aquellos nodos que están conectados por arcos a nodos pertenecientes al otro conjunto. A partir de aquí, aplicar el teorema 3 para completar la selección de los nodos.

En la figura 4.9 mostramos tres ejemplos de aplicación de este procedimiento. Así, en la figura 4.9.a mostramos el mapeado de un árbol binario de 5 niveles sobre un array de 8 PEs. Dividimos el árbol en 2 subárboles de 4 niveles y asignamos cada subárbol a una mitad del array. La raíz del árbol se la asignamos al PE 4. Por otro lado, en la figura 4.9.b mostramos el mapeado de un árbol 3-ario de 3 niveles sobre un array de 4 PEs. Dividimos el árbol en dos conjuntos de 8 y 5 nodos. El segundo conjunto se obtiene partiendo de la raíz del árbol y añadiendo sucesivamente los nodos 5, 17, 16 y 15 (en este orden). El segundo conjunto contiene los nodos restantes. Aplicamos al primer conjunto el proceso de mapeado descrito en el teorema 3, asignando al primer subconjunto los nodos conectados por arcos al segundo conjunto (nodos 3 y 4) y completado con los nodos 14 y 13 (en este orden y según el teorema 3). El segundo subconjunto contendrá los nodos 9, 10, 11 y 12. El mapeado del segundo conjunto se realiza de forma similar. El primer subconjunto del mismo debe contener los nodos conectados por arcos con nodos del primer conjunto (nodo 1), mientras que el segundo subconjunto contendrá los nodos restantes (nodos 5, 17, 16 y 15). Por último, en la figura 4.9.c mostramos el mapeado de un árbol 3-ario de 3 niveles sobre un array de 5 PEs.

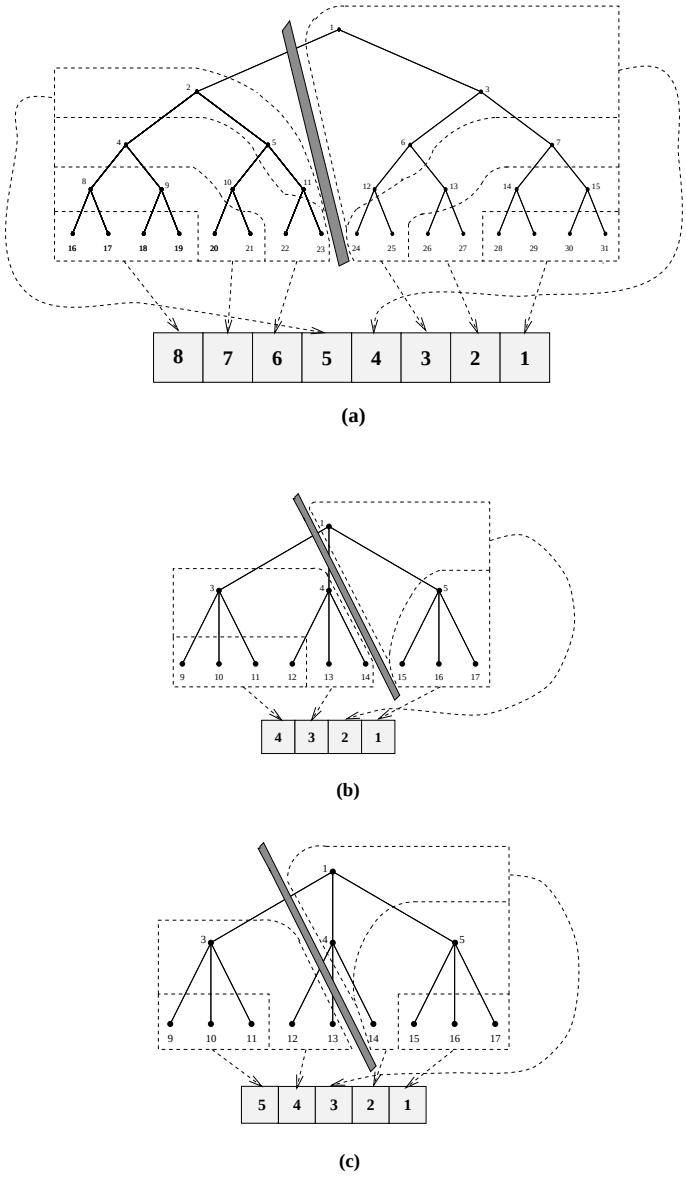


Figura 4.9: Mapeado de árboles r-arios sobre arrays lineales asignando la raíz del árbol a uno de los PEs centrales del array. (a) El array consta de 8 PEs. Puesto que r es par ($r = 2$), dividimos el árbol en dos subárboles y asignamos cada subárbol a una mitad del array (4 PEs). (b) El array consta de 4 PEs. En este caso r es impar ($r = 3$), por lo que dividimos el árbol en 2 conjuntos de 8 y 5 nodos, que asignamos a cada mitad del array (2 PEs). (c) El array consta de 5 PEs. Puesto que r es impar ($r = 3$), dividimos el árbol en dos subconjuntos de 6 y 7 PEs, que asignaremos, respectivamente, a los subarrays de 2 y 3 PEs.

4.4 Mapeado de árboles r–arios en Mallas

En esta sección describiremos un algoritmo heurístico para mapear árboles r-arios completos sobre PEs con topología malla. Este algoritmo se basa en la metodología desarrollada en la sección anterior, para mapear árboles r-arios sobre arrays lineales. Al igual que en la sección anterior, comenzaremos considerando el caso en que la raíz del árbol se asigna al PE de una esquina de la malla. Más adelante consideraremos el caso en el que la raíz del árbol se asigna a uno de los PEs centrales de la malla.

4.4.1 Asignación de la raíz del árbol al PE de una esquina de la malla

En este caso consideraremos la distribución de los nodos de un árbol r-ario de n niveles ($N = \frac{r^n-1}{r-1}$ nodos) sobre una malla de $W \times V$ PEs, tal que $n \geq V+W-1$. El PE de la esquina superior izquierda de la malla tendrá de coordenadas (1,1) y el de la esquina inferior derecha, (W,V). La distribución de los nodos del árbol entre los PEs la haremos imponiendo las siguientes condiciones:

1. La distribución de los nodos del árbol entre los PEs será balanceada. A cada PE le asignaremos un máximo de $U = \lceil N/W \times V \rceil$ nodos.
2. Las comunicaciones serán sólo entre PEs vecinos. Los nodos que en el árbol están unidos por arcos los asignaremos al mismo PE o a PEs vecinos.
3. Las comunicaciones entre PEs serán siempre en el mismo sentido. El nodo 1 del árbol será asignado al PE de coordenadas (1,1). Los hijos de cada nodo se asignarán al PE que contiene dicho nodo o al PE con una de las coordenadas igual y la otra inmediatamente superior a las del PE anterior.

El algoritmo que vamos a proponer particiona la malla en filas, divide los nodos del árbol en conjuntos asignados a las filas y, por último, distribuye estos conjuntos de nodos entre los PEs de las filas. Para ello, utilizaremos los resultados obtenidos en el caso del array lineal, en concreto, el mapeado en profundidad (teorema 3). Este algoritmo comprende los siguientes pasos,

Algoritmo 11

1. *Distribuir los nodos del árbol en W conjuntos $C(i_2)$ $(1 \leq i_2 \leq W)$, con un máximo de $\lceil N/W \rceil$ nodos por conjunto. Aplicar el teorema 3 para hacer la distribución. El conjunto $C(i_2)$ será asignado a la fila de la malla de coordenada vertical i_2.*

2. *Subdividir cada conjunto $C(i_2)$ en V subconjuntos $c(i_2, i_1)$ $(1 \leq i_1 \leq V)$, con un máximo de $\lceil N/W \times V \rceil$ nodos por subconjunto. El procedimiento para realizar la subdivisión será el siguiente:*

   (a) *Asignar al subconjunto $c(i_2, 1)$ aquellos nodos de $C(i_2)$ que están conectados por arcos a los conjuntos $C(i_2 + 1)$ y $C(i_2 - 1)$.*

   (b) *Si el nodo j se ha asignado al subconjunto $c(i_2, 1)$ y el padre de este nodo pertenece al conjunto $C(i_2)$, añadir este nodo al subconjunto. Repetir este paso si es necesario.*

   (c) *Completar el conjunto de acuerdo con el teorema 3: si j es el nodo de índice más alto del conjunto con hijos todavía no asignados, añadir al conjunto el hijo de índice más alto del nodo j. Repetir este paso hasta completar el conjunto.*

   (d) *Asignar al conjunto $c(i_2, i_1)$ los nodos $\{j \in C(i_2) \mid \lfloor j/r \rfloor \in c(i_2, i_1 - 1),\ j \notin c(i_2, i_1 - 1)\}$.*

   (e) *Repetir los pasos (c) y (d) hasta completar los nodos y los PEs.*

3. *Asignar el subconjunto de nodos $c(i_2, i_1)$ al PE de coordenadas (i_2, i_1).*

A continuación, vamos a ver como se aplica este método sobre algunos ejemplos.

- El caso más sencillo es el mapeado de un árbol binario de 3 niveles sobre una malla de tamaño 2×2. El mapeado puede realizarse tal como muestra la figura 4.10. En este caso, cada PE contiene 2 nodos, excepto el (2,1), que contiene uno. Con las condiciones impuestas al mapeado, no es posible conseguir que el PE que contenga el único nodo sea el (1,1).

- A continuación, consideraremos el mapeado de un árbol binario de 7 niveles sobre una malla de tamaño 4×4. A cada PE le corresponderá $\lceil 127/16 \rceil = 8$ nodos, excepto al (1,1) que le asignaremos 7. Comenzamos dividiendo el árbol en los cuatro conjuntos de nodos mostrados en la figura 4.11, cada uno de los cuales será asignado a una de las filas de la malla. A continuación, aplicamos el paso 2 del algoritmo

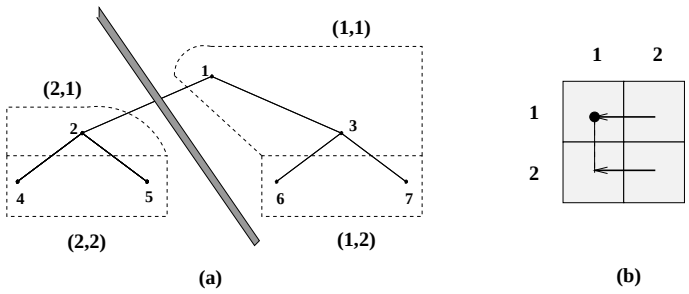


Figura 4.10: Mapeado de un árbol binario de 3 niveles sobre una malla de tamaño 2×2. (a) Particionamiento del árbol. (b) Esquema del mapeado.

para dividir cada conjunto en subconjuntos asociados a los PEs de la fila. A modo de ejemplo, en el conjunto asociado a la fila 2, $C(2)$, esta subdivisión sería realizada del siguiente modo:

(a) Asignamos a $c(2,1)$ los conjuntos de nodos {2, 48} y {6, 112, 113}.

(b) Añadimos en dos pasos los nodos 24 y 12.

(c) Completamos con el nodo 13.

(d) Asignamos a $c(2,2)$ el conjunto de nodos {49, 25, 26, 27}.

(c) Completamos con los nodos 55, 111, 110 y 54.

(d) Asignamos a $c(2,3)$ el conjunto de nodos {98, 99, 50, 51, 52, 53, 108, 109}.

(d) Asignamos a $c(2,4)$ el conjunto de nodos {100, 101, 102, 103, 104, 105, 106, 107}.

- El algoritmo puede aplicarse también a una malla cuya longitud o anchura no sea una potencia de 2. Así, en la figura 4.12 mostramos el mapeado de un árbol binario de 7 niveles sobre una malla de tamaño 3×5. Comenzamos dividiendo el árbol en 5 conjuntos de 27 nodos (excepto el primero, que contiene 19). A continuación dividimos cada conjunto en 3 subconjuntos, que asignamos a cada uno de los PEs de cada fila. A cada PE le corresponderá un máximo de $\lceil 127/15 \rceil = 9$ nodos.

- Como ejemplo de aplicación del algoritmo a un árbol 3–ario, en la figura 4.13 mostramos el mapeado de un árbol 3–ario de 5 niveles sobre una malla de tamaño 3×3. A cada PE le hemos asignado 14 nodos, excepto al de coordenadas (1,1) que le hemos asignado 9.

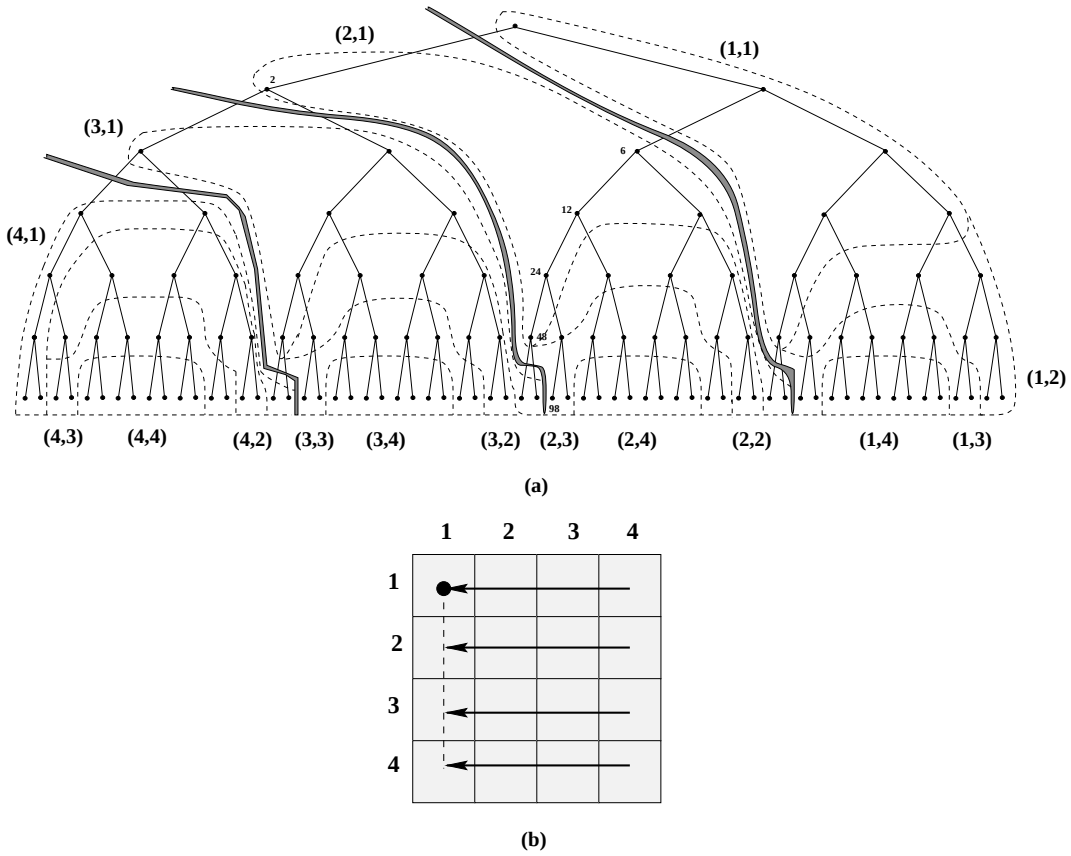


Figura 4.11: Mapeado de un árbol binario de 7 niveles sobre una malla de tamaño 4×4. (a) Particionamiento del árbol. (b) Esquema del mapeado.

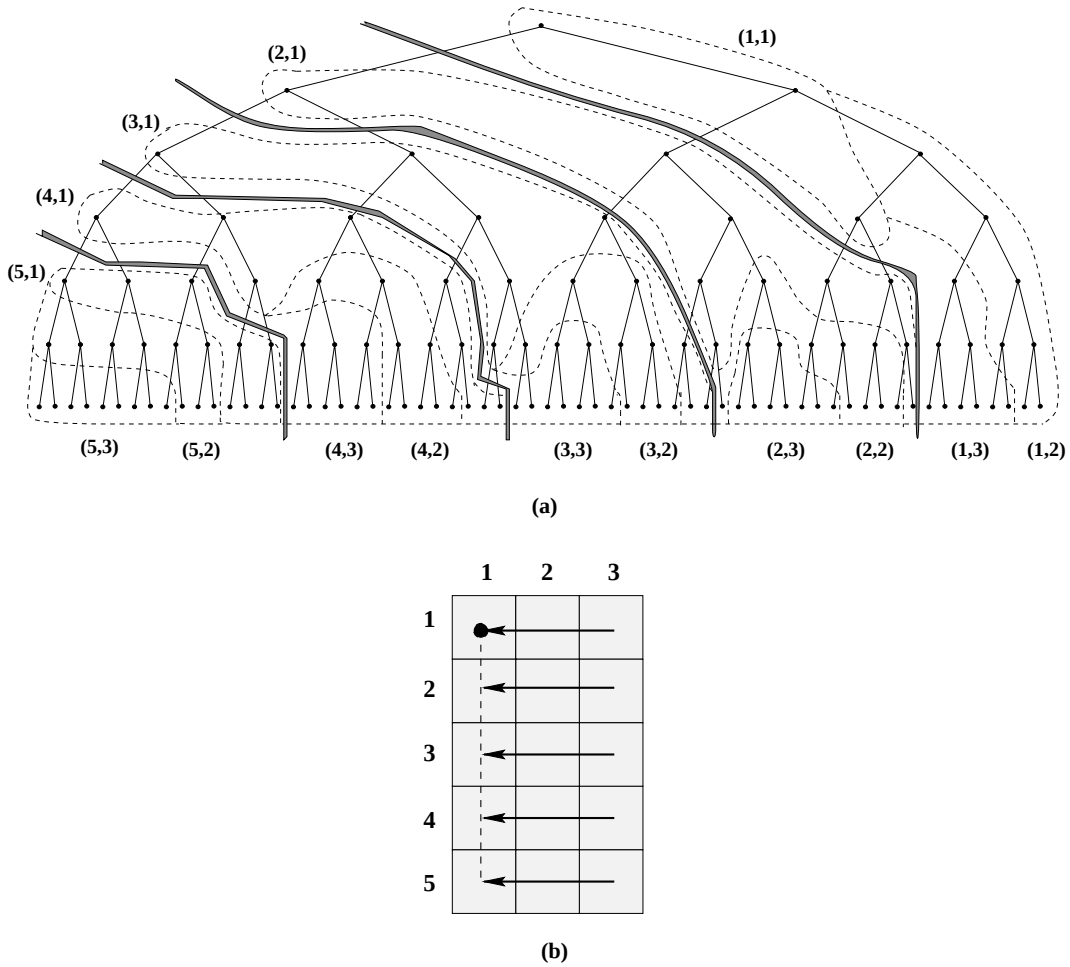


Figura 4.12: Mapeado de un árbol binario de 7 niveles sobre una malla de tamaño 3×5. (a) Particionamiento del árbol. (b) Esquema del mapeado.

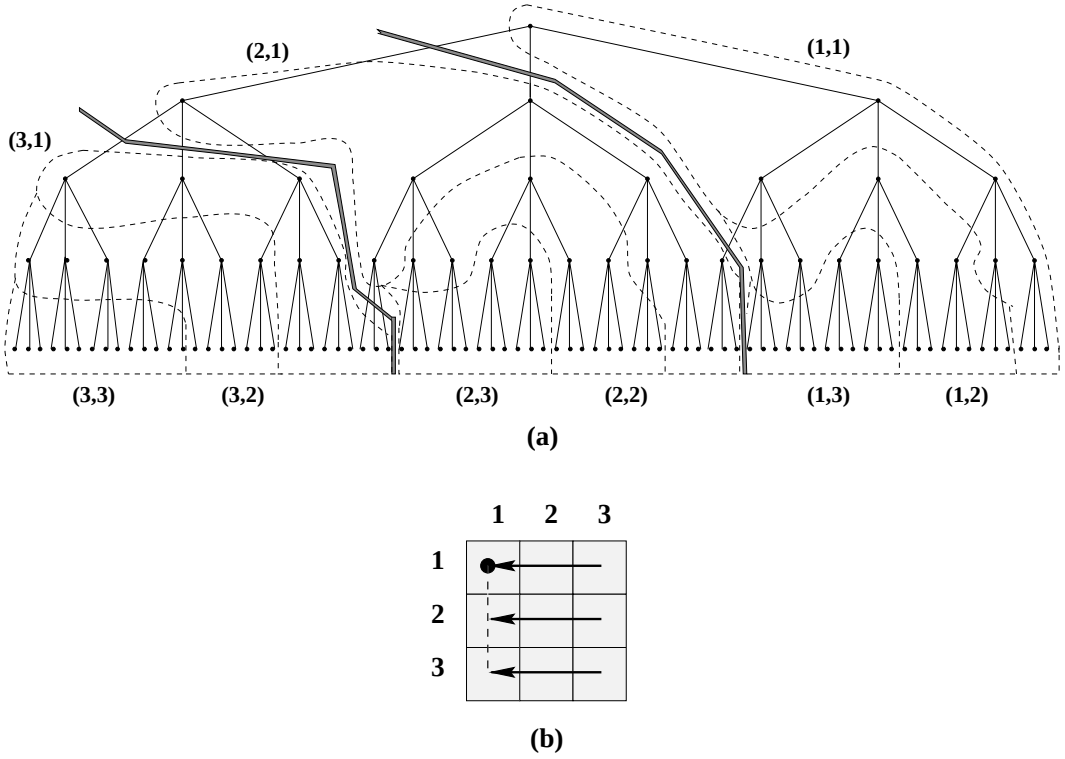


Figura 4.13: Mapeado de un árbol 3–ario de 5 niveles sobre una malla de tamaño 3×3. (a) Particionamiento del árbol. (b) Esquema del mapeado.

4.4.2 Asignación de la raíz del árbol a uno de los PEs centrales de la malla

A continuación, pasamos a considerar el caso del mapeado de árboles r-arios sobre una malla de $W \times V$ PEs, asignando la raíz del árbol a uno de los PEs centrales. En este caso debe verificarse $n > \lfloor V/2 \rfloor + \lfloor W/2 \rfloor$ para que los nodos del árbol alcancen todos los PEs de la malla.

El caso más sencillo a considerar es aquel en el cual V y W son pares y r es potencia de 2. En este caso, podemos dividir la malla en cuatro cuadrantes y el árbol en cuatro subárboles. A continuación, mapeamos cada uno de los cuatro subárboles sobre cada uno de los cuadrantes, asignando la raíz de cada subárbol a una de las esquinas de cada cuadrante. Por último, las raíces de los subárboles se conectan para generar el árbol completo, cuidando que las comunicaciones que se generan sean entre PEs vecinos.

En otros casos podremos utilizar un algoritmo similar al descrito en la sección anterior. El mapeado se realiza en dos fases: primero sobre las filas de la malla y, a continuación, sobre los PEs de cada fila. Para ello, dividimos los nodos del árbol en W conjuntos (que asignaremos a cada una de las W filas de la malla) y, a continuación, dividimos cada conjunto en V subconjuntos (que asignaremos a cada uno de los PEs de la fila). El algoritmo puede describirse de la siguiente forma:

Algoritmo 12

1. *Dividir el árbol en 2 partes de* $U \times V\lfloor W/2 \rfloor$ *y* $N - U \times V\lfloor W/2 \rfloor$ *nodos. Para obtener la segunda parte, partir de la raíz del árbol y aplicar el teorema 3 para seleccionar los nodos correspondientes: si* j *es el nodo de índice más alto ya asignado a esta parte, con hijos todavía no asignados, añadir el hijo de índice más alto del nodo* j*. La otra parte contendrá los nodos restantes. A continuación, subdividir la primera parte en* $\lfloor W/2 \rfloor$ *conjuntos y la segunda en* $\lfloor (W+1)/2 \rfloor$*. Incluir en el primer conjunto de cada parte, los nodos que estén unidos por arcos a nodos de la parte contraria. Completar los conjuntos de acuerdo con el teorema 3. Cada uno de estos* W *conjuntos de nodos será asignado a una de las* W *filas de PEs.*

2. *Siguiendo el mismo procedimiento que en el punto 1, dividir cada conjunto en 2 partes, la primera de* $U \times \lfloor V/2 \rfloor$ *nodos y la segunda, con los nodos restantes. Dividir la primera parte en* $\lfloor V/2 \rfloor$ *subconjuntos y la segunda parte en* $\lfloor (V+1)/2 \rfloor$*. Asignar cada uno de los subconjuntos a uno de los* V *PEs de la fila.*

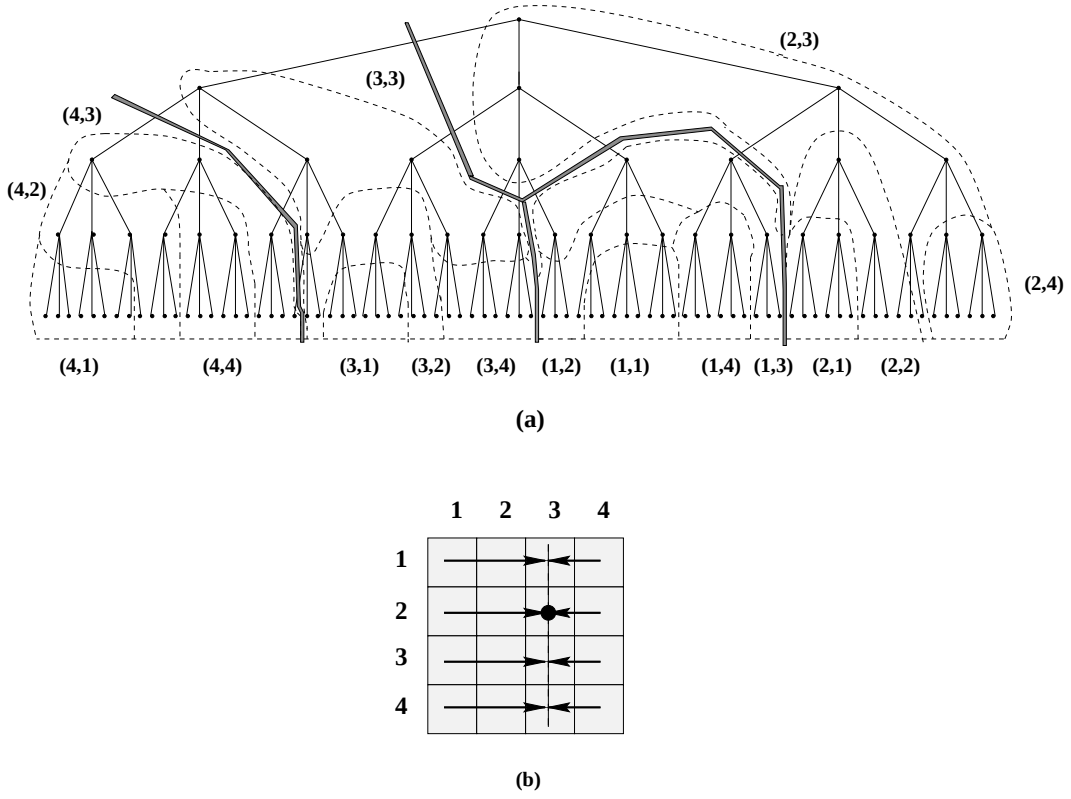


Figura 4.14: Mapeado de un árbol 3–ario de 5 niveles sobre una malla de tamaño 4×4 asignando la raíz del árbol a uno de los PEs centrales de la malla. (a) Particionamiento del árbol. (b) Esquema del mapeado.

En la figura 4.14 mostramos un ejemplo de aplicación de este algoritmo al mapeado de un árbol 3–ario de 5 niveles sobre una malla de tamaño 4×4. Cada PE contiene un máximo de $\lceil 121/16 \rceil = 8$ nodos.

4.5 Mapeado y planificación de árboles r–arios

En este apartado vamos a considerar árboles orientados. Cada nodo del árbol va a representar una tarea computacional. En general, suponemos que los arcos de los árboles se dirigen a la raíz, marcando la dependencia y las comunicaciones requeridas entre tareas. Cada tarea sólo toma como entradas las salidas de las tareas de sus hijos, y no puede comenzar su ejecución hasta que todos sus hijos completen sus tareas, ver figura 4.15. El problema inverso, el cual comienza desde la tarea representada por la raíz, y avanza hacia los

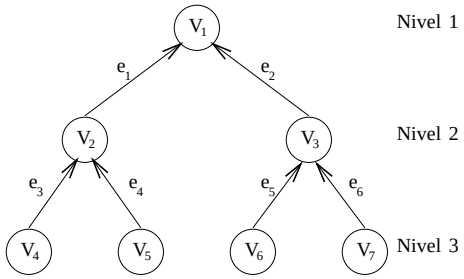


Figura 4.15: Árbol tarea binario. El arco e_1 incide a V_2 y V_1, el primer nodo, V_2 se llama nodo de salida del árbol e_1 y el nodo V_1 se llama nodo de llegada del arco. Consideramos que cada nodo representa una tarea y los arcos las dependencias entre las tareas. V_2 no puede comenzar hasta que V_4 y V_5 hayan terminado.

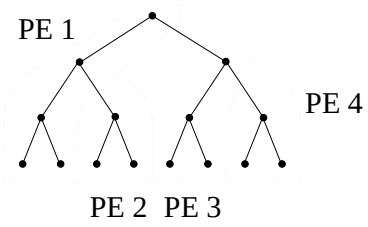


Figura 4.16: Árbol binario con cuatro niveles particionado sobre 4 PEs. Las comunicaciones generadas por los arcos no son realizadas entre PEs vecinos.

nodos hojas, puede ser fácilmente obtenido del anterior invirtiendo cada una de las operaciones. Estudiaremos la partición, proyección y planificación de un árbol tarea r–ario completo.

En el mapeado de árboles sobre mallas y arrays, el objetivo es maximizar el tiempo en que cada PE está ocupado resolviendo la carga computacional del algoritmo en cada uno de los n pasos. En una aproximación inicial, ver figura 4.16, los nodos de cada nivel se dividen entre V PEs, asumiendo $V = r^v$. En los primeros $n - v$ pasos, no se necesita realizar comunicación entre los PEs. En los últimos v pasos, los PEs deben de comunicar sus resultados a otros PEs. Observar que el número de PEs activos decrece por un factor r en cada paso. El proceso continua de esta manera, hasta que en el último paso, sólo existe un único PE activo, y los demás están inactivos. Este patrón de comunicación no tiene buena eficiencia. En los últimos pasos del algoritmo las comunicaciones son entre PEs no vecinos.

En nuestro procedimiento de mapeado de árboles r–arios que vamos a proponer sólo permitiremos comunicaciones entre PEs vecinos. Dos nodos conectados por un arco en el árbol serán asignados al mismo PE o a PEs vecinos. Esto implica que poniendo la raíz en uno de los PEs del array o de la malla, la distancia de los nodos del árbol que puede alcanzar es limitada por el número de niveles del árbol.

4.5.1 Mapeado y planificación de árboles r-arios sobre arrays

En esta sección, presentaremos algoritmos para particionar, proyectar y planificar un árbol computacional r–ario sobre un array [8]. En la primera parte de esta sección asignaremos la raíz del árbol al PE de una esquina del array mientras en la segunda parte la asignaremos al PE central.

Asignación de la raíz del árbol al PE de la esquina del array

Suponemos la distribución de un árbol r–ario de n niveles sobre un array con V PEs, siendo $V \leq n$.

El mapeado de los nodos del árbol sobre los PEs se realizarà impuestas las condiciones siguientes:

1. La tarea de un nodo no puede ser ejecutada hasta que sus hijos no hayan finalizado y sus resultados estén en posesión del PE. La relación de dependencia entre nodos debe ser respetada.

2. Sólo se realizarán comunicaciones entre PEs vecinos. Las tareas correspondientes a los hijos de un nodo se ejecutarán en el PE donde está ese nodo o en el PE con siguiente índice más alto.

3. Asumimos un mapeado estático. Cada nodo del árbol será asignado a un PE.

Además, intentaremos balancear la carga de cada PE tanto como sea posible.

Los PEs del array son numerados de 1 a V, siendo PE 1 el que contiene la raíz del árbol. Llamaremos t_i el número de nodos del árbol que computa el PE i. Suponemos que las tareas asignadas a cada uno de los nodos se realiza

en un ciclo de reloj. De esta condición de mapeado, obtenemos que el PE i contiene nodos del nivel i o más altos.

El t_i máximo lo tendrá el nodo raíz, ya que será el último en ejecutarse. El PE V contiene t_V nodos del nivel más alto del árbol y los computará en t_V ciclos del reloj. El PE $V-1$ incluirá al menos un nodo del nivel $V-1$ que deberá de computarse en el ciclo t_V+1, por lo que haremos que $t_{V-1} = t_V+1$. De igual forma el PE $V-2$ incluirá al menos un nodo que debe computarse en el ciclo $t_V + 2$, por lo que $t_{V-2} = t_V + 2$ y así sucesivamente. Por otro lado, como el número total de nodos del árbol es N deberá verificarse,

$$\sum_{i=0}^{V-1}(t_1 - i) = N \tag{4.1}$$

El número de nodos mínimo que debe de contener el PE de la raíz, se obtiene por la siguiente expresión,

$$t_1 = \left\lceil \frac{2N - V(V-1)}{2V} \right\rceil \tag{4.2}$$

esto nos va a garantizar que podamos incluir los nodos suficientes en cada uno de los conjuntos asignados a los PEs. En la figura 4.17.a se muestra un ejemplo. En esta figura, cada una de las particiones del árbol se va a computar en un PE. Los números asignados a los nodos indican el ciclo de procesamiento de cada nodo. El primer número es el ciclo de procesamiento, suponiendo que las tareas empiezan a ejecutarse desde las hojas, mientras que el número entre paréntesis supone que las tareas empiezan a ejecutarse desde la raíz. Las distintas particiones del árbol se computan en 5, 6, 7, 8 y 9 ciclos.

Al asignar los nodos a los PEs tenemos que tener en cuenta la siguiente restricción debida a la planificación: dentro de cada partición debemos incluir nodos de niveles consecutivos del árbol. Si esto no se hace así, se producirá un error en la partición. En la figura 4.17.b mostramos un ejemplo, en donde uno de los PEs contiene nodos de los niveles 2,4 y 5 (faltando nodos del nivel 3).

El procedimiento de particionamiento y planificación que proponemos es el siguiente:

Algoritmo 13

1. *Asignar al conjunto de nodos del PE* 1 *la raíz y los nodos del nivel más alto posible tal que su padre ya esté asignado, hasta completar* t_1

nodos. Entre los nodos del mismo nivel empezamos por los de más a la derecha.

2. *Asignar al PE i los nodos no asignados que estén conectados con nodos pertenecientes al PE* $i-1$.

3. *Completar el PE i con los nodos del nivel más alto posible tal que su padre este asignado, hasta completar los* t_i *nodos.*

4. *Repetir los pasos 2 y 3 hasta completar los PEs.*

5. *Asignar los ciclos de procesamiento dentro de cada PE empezando por los niveles más altos y por los nodos de más a la derecha.*

En la figura 4.18 podemos ver un ejemplo de este algoritmo para un árbol 3-ario completo de 4 niveles sobre un array de 4 PEs. Los números representan el ciclo de reloj en que se ejecuta cada nodo.

Asignación de la raíz del árbol a uno de los PEs centrales del array

A continuación pasamos a estudiar el caso del mapeado y planificación de árboles r-arios sobre un array de V PEs, asignando la raíz del árbol al PE central (si V es impar) o a uno de los dos PEs centrales (si V es par). En este caso consideraremos árboles de un número de niveles $n \geq 1 + \lfloor V/2 \rfloor$. El procedimiento que vamos a utilizar es el siguiente:

Algoritmo 14

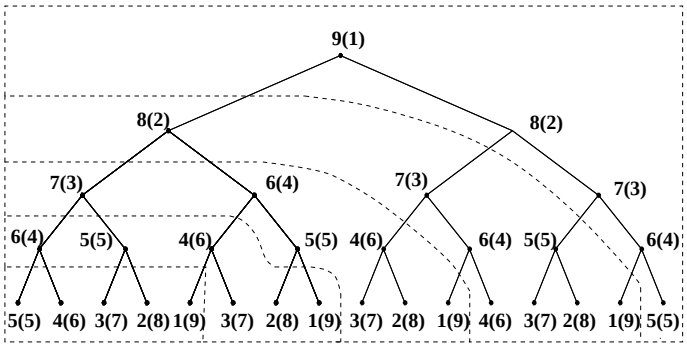
1. *Dividir el árbol en dos conjuntos usando el algoritmo 13. Estos conjuntos serán asignados respectivamente a cada una de las mitades del array. Si el número de PEs es par, hacer el número de nodos de estos conjuntos* $N_l = \left\lfloor \frac{N}{2} \right\rfloor$ *y* $N_r = \left\lfloor \frac{N}{2} \right\rfloor + 1$ *nodos con* $N_r + N_l = N$. *Si el número de PEs es impar,*

$$N_l = \sum_{i=0}^{\frac{V-1}{2}} t_V + i \tag{4.3}$$

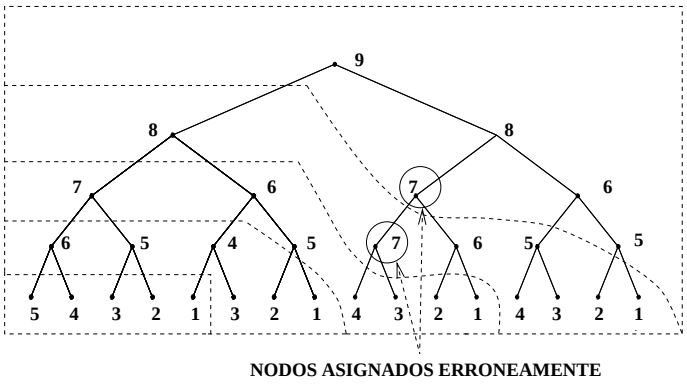
$$N_r = N - N_i \tag{4.4}$$

siendo,

$$t_1 = \left\lceil \frac{4N - (V-1)^2}{4V} \right\rceil \tag{4.5}$$



(a)



(b)

Figura 4.17: Particionamiento y planificación de un árbol binario de 5 niveles sobre un array de 5 PEs. (a) El primer número supone que las tareas empiezan a ejecutarse desde las hojas y el número entre paréntesis que las tareas empiezan a ejecutarse desde la raíz. (b) Error en el particionamiento: uno de los PEs contiene nodos de los niveles 2, 4, y 5 faltando nodos del nivel 3.

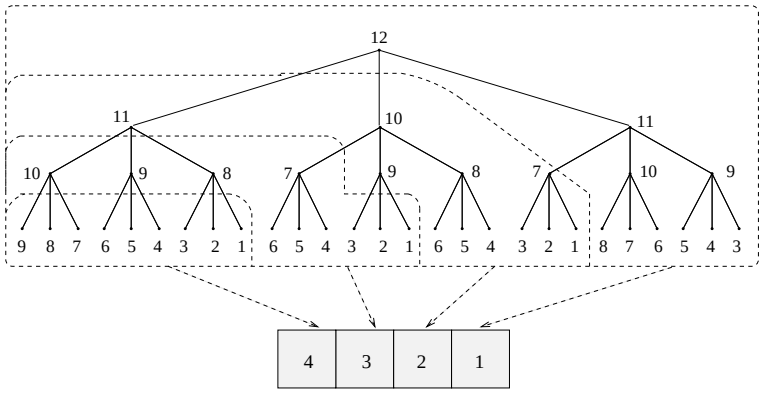


Figura 4.18: Mapeado y planificación de un árbol completo 3–ario de 4 niveles (40 nodos) sobre un array lineal de 4 PEs (Algoritmo 13). Los números indican el ciclo de procesamiento de cada nodo.

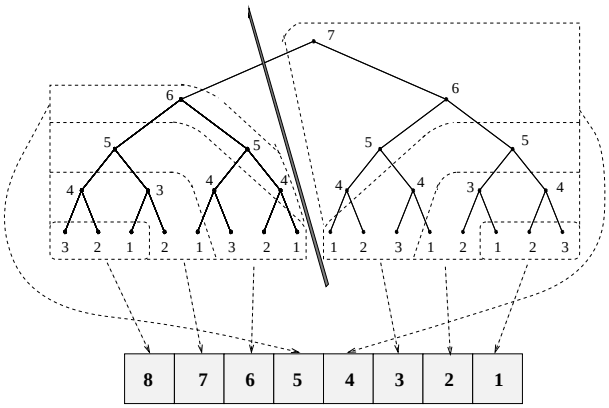
2. *A continuación, debemos subdividir cada uno de los dos conjuntos en varios subconjuntos, cada uno de los cuales será asignado a un PE. En el primero de dichos subconjuntos debemos incluir aquellos nodos que están conectados por arcos a nodos pertenecientes al otro conjunto. A partir de aquí, aplicar el algoritmo 13 para completar la selección de los nodos.*

En la figura 4.19.a mostramos el mapeado un de árbol binario de 5 niveles sobre un array de 8 PEs. Dividimos el árbol en 2 subárboles de 4 niveles y asignamos cada subárbol a una mitad del array. La raíz del árbol se la asignamos al PE 4. En la figura 4.19.b mostramos el mapeado de un árbol 3-ario de 3 niveles sobre un array de 5 PEs. En este ejemplo, $N_l = 5$ y $N_r = 8$

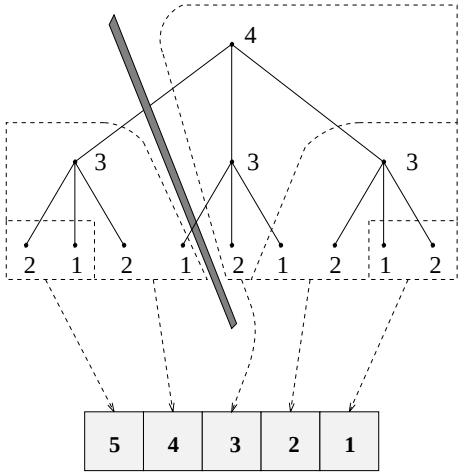
4.5.2 Mapeado y planificación de árboles r–arios en mallas

En esta sección describiremos un algoritmo para mapear y planificar árboles r–arios completos sobre computadores de topología malla [8]. Este algoritmo se basa en la metodología desarrollada en la sección anterior para mapear y planificar dichos árboles sobre arrays lineales.

En este caso, consideraremos la distribución de los nodos de un árbol r-ario de n niveles ($N = \frac{r^n-1}{r-1}$ nodos), sobre una malla de $W \times V$ PEs tal



(a)



(b)

Figura 4.19: Mapeado y planificación de árboles r–arios completos, asignando la raíz a uno de los PEs centrales (Algoritmo 14). Los números indican el ciclo de procesamiento de cada nodo. (a) Árbol binario de 5 niveles sobre un array de 8 PEs. (b) Árbol 3-ario de 3 niveles sobre un array de 5 PEs.

que $n \geq W/2 + V/2 = 1$. El PE de la esquina superior izquierda de la malla será el PE 1 y el de la esquina inferior izquierda, el $W \times V$. El PE P_i, excepto los PEs de los bordes, puede cambiar información con sus vecinos más inmediatos: P_{i-1}, P_{i+1}, P_{i+V} y P_{i-V}.

Vamos a considerar la malla dividida en 4 cuadrantes y estos a su vez divididos en filas. El algoritmo que vamos a proponer particiona, en primer lugar, el árbol en 4 conjuntos que divide entre los cuadrantes, particiona estos conjuntos en subconjuntos que divide entre las filas y, por último, distribuye estos subconjuntos de nodos entre los PEs de las filas. Los procedimientos por los cuales se hacen todas estas subdivisiones de nodos son similares a los descritos en los algoritmos 13 y 14. El algoritmo resultante puede escribirse de la siguiente forma,

Algoritmo 15

1. *Particionar el árbol en 4 conjuntos utilizando el algoritmo 13. Estos conjuntos serán asignados a cada uno de los 4 cuadrantes.*

2. *Subdividir cada conjunto en $W/2$ subconjuntos utilizando el algoritmo 13. En el primer subconjunto de cada conjunto deberán incluirse los nodos que están comunicados por arcos, a nodos del otro conjunto. Cada uno de los subconjuntos será asignado a una de las filas del cuadrante.*

3. *Subdividir cada uno de los subconjuntos en $V/2$ grupos utilizando el algoritmo 13. En el primer grupo de cada subconjunto deberán incluirse los nodos que están comunicados por arcos, a nodos del otro subconjunto. Cada uno de los grupos serán asignados a uno de los PEs del cuadrante.*

El conjunto de particiones en que se ha divido el árbol $\{T_1, T_2, \cdots, T_{V \times W}\}$ se enumera de izquierda a derecha. Asignamos al PE i, con representación índice–dígito $[i_{W \times V}, ..., i_1]$, la partición T_j con representación índice–dígito $[j_{W \times V}, ..., j_1]$ tal que

$$\begin{array}{ll} j_{W \times V} = i_V & \\ j_{W \times V - 1} = i_{W \times V} & \\ j_k = i_V \oplus i_k, & k = 1, ..., V - 1. \\ j_k = i_{W \times V} \oplus i_{k+1}, & k = V, ..., W \times V - 2. \end{array} \tag{4.6}$$

A continuación, vamos a ver como se aplica este método sobre algunos ejemplos.

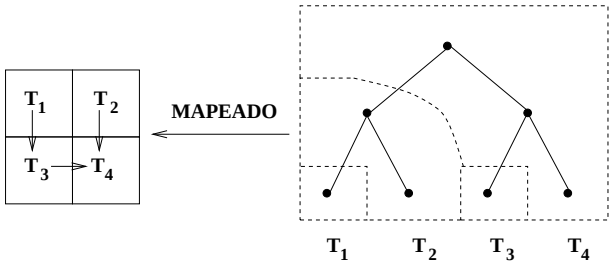


Figura 4.20: Particionamiento y mapeado de un árbol binario completo de 3 niveles sobre una malla de 2×2.

- El caso más sencillo es el mapeado de un árbol binario de 3 niveles sobre una malla de tamaño 2×2. El mapeado puede realizarse tal como muestra la figura 4.20.

- A continuación, consideraremos el mapeado de un árbol binario de 9 niveles sobre una malla de tamaño 8×8. Comenzamos dividiendo el árbol en cuatro conjuntos, cada uno de los cuales será asignado a uno de los cuadrantes de la malla. A continuación, aplicaremos los pasos 2 y 3 del algoritmo para dividir cada conjunto en subconjuntos y cada subconjunto en grupos asociados a los PEs. Este ejemplo se muestra en la figura 4.21.

- Como ejemplo de aplicación del algoritmo a un árbol 3–ario, en la figura 4.22 mostramos el mapeado de un árbol 3–ario de 4 niveles sobre una malla de tamaño 2×4.

4.6 Evaluación del esquema propuesto orientado a la implementación VLSI

4.6.1 Criterios de evaluación

Para la evaluación de la efectividad de un esquema de proyección en el mapeado de árboles orientado a la implementación VLSI, seguiremos los tres siguientes criterios [118]:

- **Área eficiente del chip**. Este parámetro se define como el porcentaje del área activa del chip utilizada para la computación, en relación al

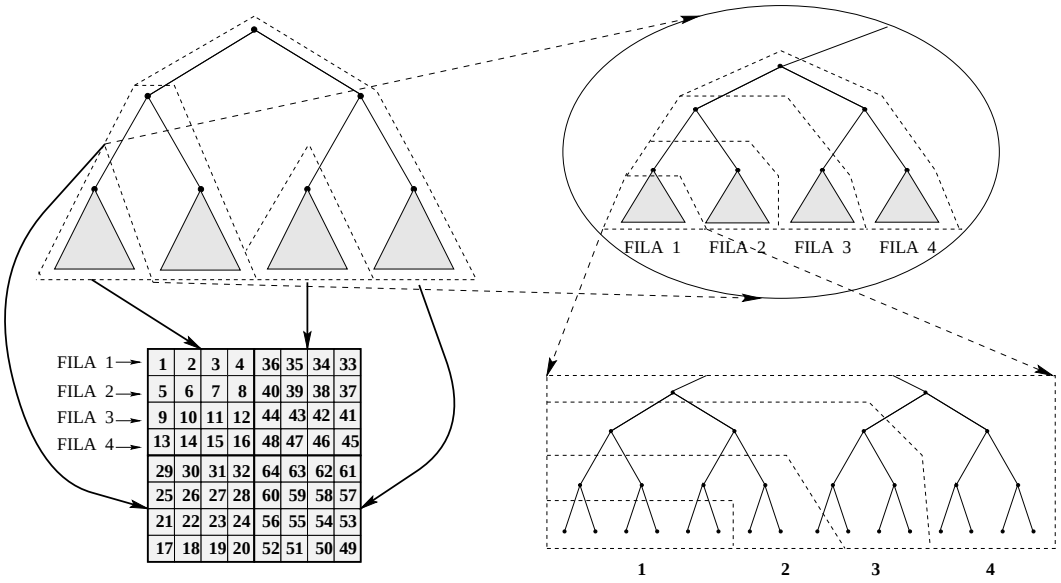


Figura 4.21: Particionamiento y mapeado de un árbol binario completo de 9 niveles sobre una malla 8×8.

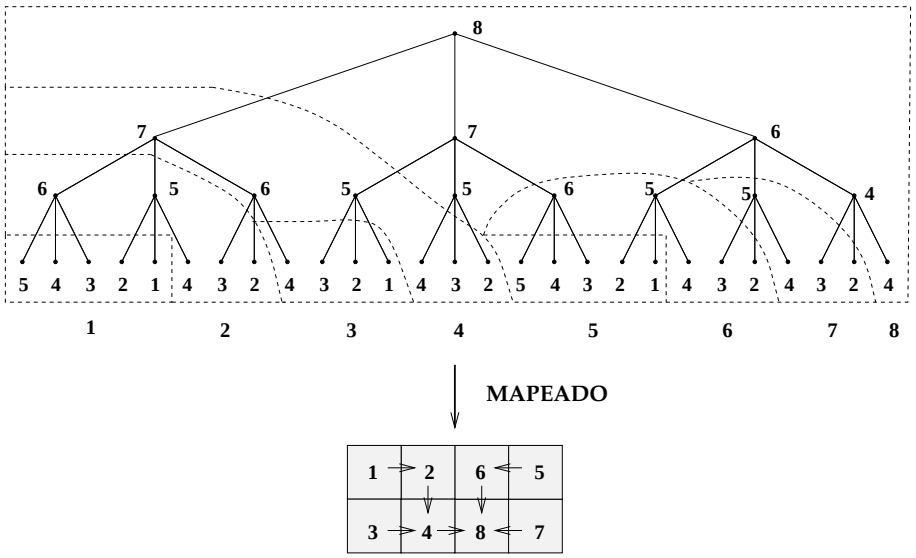


Figura 4.22: Particionamiento y mapeado de un árbol 3-ario de 4 niveles sobre una malla 2×4.

área total del chip. Cuando el PE es relativamente grande, casi todo el área del chip está dedicada a la implementación de PEs. El área eficiente es entonces el porcentaje de PEs utilizados como nodos del árbol en relación al total de PEs del chip. Ya que el coste de un circuito VLSI está muy influido por el área del chip, este factor evalúa fielmente el coste del esquema de proyección.

- **Retardo de propagación** (*Propagation delay*). Definido como el tiempo máximo de propagación de datos entre la raíz y los nodos hojas. Este factor determina el tiempo de latencia de los datos.

- **Máxima longitud de arco** (*Maximum edge length*). Definido como la máxima longitud de arco entre un par de nodos conectados directamente en el árbol. Es importante por dos razones. Si el array de PEs opera síncronamente con un reloj común, entonces la máxima frecuencia del reloj está limitada por el retardo de propagación de este arco. Este factor limita la velocidad del diseño. En segundo lugar, la densidad de comunicación entre nodos de dos niveles adyacentes del árbol se suele doblar a medida que ascendemos en el árbol, desde las hojas al nodo raíz. Por tanto, el ancho de banda de los arcos de los niveles más próximos a la raíz suele ser más elevado para una operación eficiente del sistema, para lo cual es importante que la longitud de estos arcos sea corta.

4.6.2 Área eficiente

Como hemos mencionado en el apartado anterior, el área eficiente de un esquema de proyección, tradicionalmente, se suele estimar por la relación del número de PEs utilizados, al número total de PEs, en el chip cuando la topología del árbol es proyectada. Como se puede ver en las figuras 4.6-4.22 ningún nodo se deja sin utilizar en nuestro diseño. El área eficiente en nuestro esquema es entonces del 100%. Por tanto, nuestro esquema de proyección es óptimo en cuanto a la utilización de los PEs. Notar que el área de interconexión entre módulos no se refleja en el cálculo del área eficiente, ya que hemos considerado que es pequeña en comparación con el área de los PEs.

4.6.3 Retardo de propagación

El retardo de propagación en un árbol se puede estimar por la distancia máxima entre la raíz y los nodos hojas, como en el modelo presentado en [118]. Se puede expresar en términos de la dimensión de un PE, es decir, que se considera un PE cuadrado cuyo lado tiene una distancia unidad. Todas las medidas se hacen usando esta escala. Suponemos que todos los caminos son horizontales o verticales, pero no diagonales, es decir, interconexiones de tipo Manhattan. La distancia entre dos PEs conectados se define como

$$d_{PE_1,PE_2} = |i_1 - i_2| + |j_1 - j_2| \tag{4.7}$$

donde (i_1, j_1) son las coordenadas fila, columna del primer PE y (i_2, j_2) son las respectivas coordenadas del segundo PE. La máxima distancia raíz–hoja se obtiene a partir del máximo de las distancias de todos los caminos desde el nodo raíz a los nodos hojas.

El límite inferior para el retardo de propagación que puede obtenerse para el mapeado de un árbol r–ario, sobre un array y una malla de PEs con interconexiones de tipo Manhattan lo da los siguientes lemas.

Lema 16 *El límite inferior del retardo de propagación de la proyección de un árbol de n niveles sobre un array de V PEs, siendo $n = 1 + \lfloor V/2 \rfloor$, es $\lfloor V/2 \rfloor$.*

Demostración El valor mínimo del retardo de propagación puede obtenerse sólo cuando se cumplen dos condiciones. En primer lugar, el nodo raíz debe considerarse en uno de los PEs del centro, si el número de PEs del array es par, o en el PE central, si el número de PEs del array es impar. Y en segundo lugar, si en los PEs de la esquina hay subárboles completos, es decir, o bien nodos hojas o bien nodos internos pero con todos sus descendientes (ver figura 4.23.a). Si la primera de estas condiciones no se cumple, la distancia del PE que contiene la raíz a uno de los PEs del extremo sería mayor que $\lfloor V/2 \rfloor$, mientras que si la segunda no se cumple pasará dos veces por el mismo PE y de nuevo la distancia sería mayor que $\lfloor V/2 \rfloor$. □

En cuanto a la proyección de un árbol en una malla, el siguiente lema nos dará el retardo de propagación óptimo.

Lema 17 *El límite inferior del retardo de propagación, de la proyección de un árbol de n niveles sobre un malla de $W \times V$ PEs, siendo $n \geq V + W - 1$ es $\frac{V}{2} + \frac{W}{2}$.*

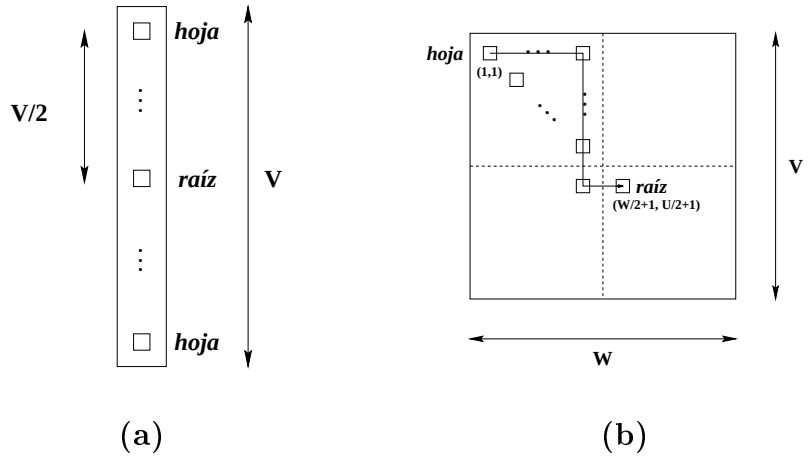


Figura 4.23: Proyección de un árbol de n niveles con óptima propagación. (a) En un array. (b) En una malla.

Demostración De nuevo, el valor mínimo del retardo de propagación sólo puede obtenerse cuando el nodo raíz se localiza en uno de los PEs del centro, y cuando el PE de la esquina más alejada contiene subárboles completos, ver figura 4.23.b. Las razones son las mismas que en el lema anterior. La distancia entre el PE central $(W/2+1, U/2+1)$ y el PE más alejado $(1,1)$ es $\frac{V}{2} + \frac{W}{2}$. □

El retardo de propagación de nuestros diseños sobre arrays y mallas es el mismo que el límite inferior. Notar que hemos derivado este valor en función de argumentos puramente geométricos. No está claro que este límite inferior pueda obtenerse en la práctica.

4.6.4 Máxima longitud de arco

En nuestro caso, la máxima longitud de arco se obtiene de la condición inicial en la cual se imponían comunicaciones sólo entre PEs vecinos. Por consiguiente, en nuestro caso toma el valor óptimo.

4.6.5 Comparación con otros esquemas

La efectividad de un esquema layout puede evaluarse efectivamente por los tres criterios (área eficiente, retardo de propagación y máxima longitud de arco) como vimos en la sección anterior. En esta sección, comparamos nuestro

método con otras dos aproximaciones, el esquema clásico de árbol en H [57] y el modelo de Youn y Singh [118].

Esquema árbol en H

El esquema árbol en H proyecta los árboles con un patrón H. Una proyección de un árbol de $n = 5$ sobre una malla de 7×7 usando este esquema se puede ver en la figura 4.1. Observar que un número significativo de PEs se usan sólo como elementos de conexión para llevar las señales a los nodos. Si suponemos $A_H(n)$ el número de PEs necesarios para proyectar un árbol de n niveles, $PD_H(n)$ el retardo de propagación y $MEL_H(n)$ la máxima longitud de arco. Entonces de [118] obtenemos:

$$
\begin{aligned}
A_H(n) &= \begin{cases} (2^{(n+2)/2} - 1) \times (2^{(n+2)/2} - 1) & \text{si } n \text{ es impar} \\ (2^{(n+2)/2} - 1) \times (2^{n/2} - 1) & \text{si } n \text{ es par} \end{cases} \\
PD_H(n) &= \begin{cases} 2^{(n+1)/2} - 2 & \text{si } n \text{ es impar} \\ 3 \times 2^{(n-2)/2} & \text{si } n \text{ es par} \end{cases} \\
MEL_H(n) &= \begin{cases} 1 & \text{si } n < 4 \\ 2^{\lfloor n/2 \rfloor - 1} & \text{si } n \geq 4 \end{cases}
\end{aligned}
\tag{4.8}
$$

El área eficiente para el esquema árbol en H es $(2^n - 1)/A_H(n)$. La figura 4.24 muestra que el área eficiente converge a 50% cuando los niveles del árbol de proyección aumentan. Las figuras 4.25 y 4.26 muestran los cambios en el PD y en el MEL para árboles de niveles 4–10.

Esquema de Youn–Singh

Youn y Singh [118] han propuesto una proyección jerárquica del árbol. El esquema usa una estrategia jerárquica, de forma que los árboles mayores de 4 niveles ($n > 4$) se proyectan por la conexión de un apropiado número y tipo de módulos básicos. Cada módulo básico es una malla cuadrada de 4×4 PEs que contiene árboles de $n = 4$. En la figura 4.27 mostramos cuatro tipos de estos módulos básicos. Si definimos $A_Y(n)$, $PD_Y(n)$ y $MEL_Y(n)$ como hicimos con el esquema árbol en H, obtenemos,

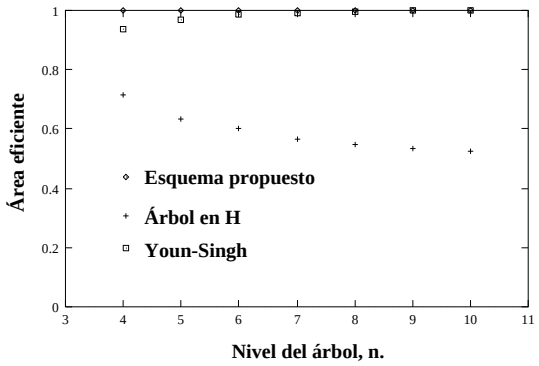


Figura 4.24: Comparación del área eficiente.

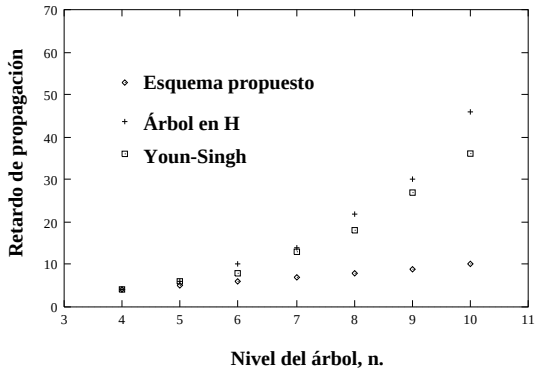


Figura 4.25: Comparación del retardo de propagación.

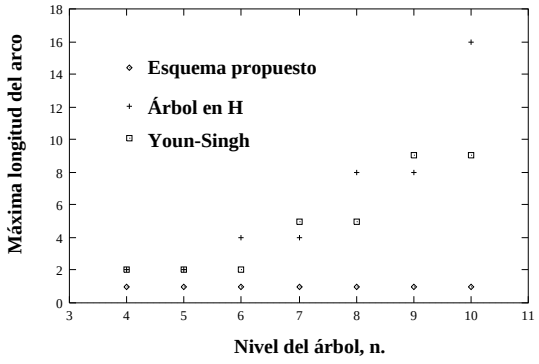


Figura 4.26: Comparación de la máxima longitud de arco.

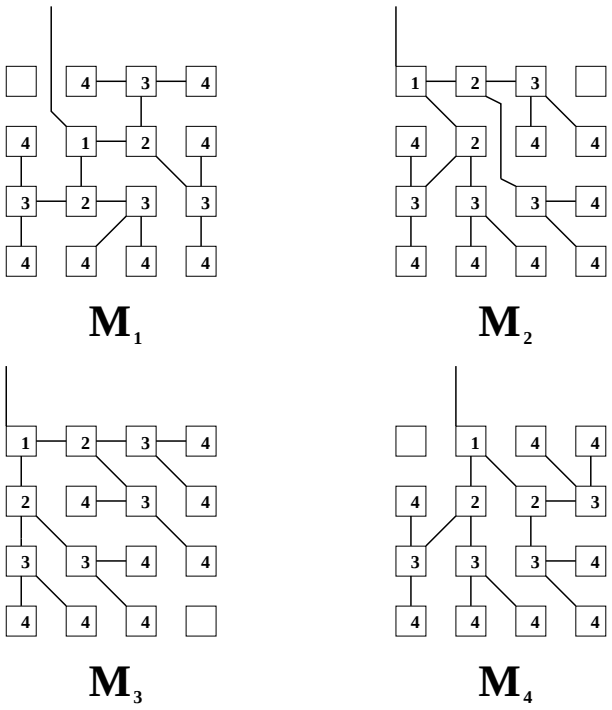


Figura 4.27: Cuatro tipos de módulos básicos del esquema jerárquicos de Youn y Singh.

$$A_Y(n) = \quad \frac{2^n-1}{2^n} \quad n \geq 4$$

$$PD_Y(n) = \begin{cases} 2n-4 & \text{si } n \leq 6 \\ 2^{n/2}+n-6 & \text{si } n > 6 \text{ y par} \\ 3 \cdot 2^{-1.5}2^{n/2}+n-6 & \text{si } n > 6 \text{ e impar} \end{cases} \tag{4.9}$$

$$MEL_Y(n) = \begin{cases} 2 & \text{si } n \leq 6 \\ 2^{\lfloor (n-7)/2 \rfloor + 2}+1 & \text{si } n \geq 6. \end{cases}$$

Como podemos ver, nuestro esquema mejora estos resultados al suponer que cada PE puede computar varios nodos del árbol.

Capítulo 5

Algoritmos jerárquicos para el problema de los N cuerpos

5.1 Introducción

El punto de comienzo de la ciencia computacional para acercarse a un fenómeno físico es un modelo matemático con ecuaciones representadas en álgebra discreta. Las ecuaciones del álgebra discreta proporcionan el *programa de simulación* que permite el estudio del fenómeno físico, vía un *experimento computacional*.

Aunque la cantidad de datos que pueden ser manejados por los computadores hoy en día es muy grande tiene, no obstante, que ser finita. El mayor esfuerzo del científico computacional es conseguir un buen modelo de simulación del sistema físico, dentro de las restricciones del computador. Los métodos de discretización usados para obtener modelos de simulación incluyen: métodos de diferencias finitas, métodos de elementos finitos, y los *métodos de los N cuerpos*.

El término genérico *modelo de los N cuerpos* se refiere a aquellos modelos de simulación, en los cuales la representación discreta del fenómeno físico implica interacción entre cuerpos. En muchas aplicaciones los cuerpos pueden identificarse con objetos físicos. Cada cuerpo tiene un conjunto de atributos, tales como masa, carga, posición, momento, etc... El estado del sistema físico se define por los atributos de un conjunto finito de cuerpos. Un hecho que hace especialmente atractivo computacionalmente el modelo de los N cuerpos es que el número de atributos de los cuerpos es constante y no necesita actualizarse mientras dura la simulación computacional.

Los fenómenos físicos que pueden simularse adecuadamente mediante el modelo de los N cuerpos son los descritos por la teoría clásica. Un sistema clásico se describe en términos de las posiciones, velocidades, y leyes de fuerza entre los cuerpos que componen el sistema. Desafortunadamente, el gran número de cuerpos de algunos sistemas hace que tales descripciones detalladas sean inútiles, para entender el sistema y para modelarlo computacionalmente. Por ejemplo, si estamos investigando el flujo de agua a través de una boquilla, la información detallada de la posición y velocidad de cada átomo carece totalmente de interés. Efectivamente, si intentamos realizar tales cálculos en el más potente de los computadores existentes, sólo obtendríamos las vibraciones y rotaciones de las moléculas en un elemento microscópico del agua.

El secreto del éxito en los experimentos computacionales es idear un modelo suficientemente detallado, para reproducir fielmente los efectos físicos de importancia y no tan detallado, para hacer los cálculos impracticables. La mejor elección del modelo depende de la escala temporal y espacial física pertinente. En el caso del agua fluyendo por una boquilla, un modelo que da las velocidades de grupos podría ser bastante adecuado, por ejemplo, un modelo de vórtices. En el otro extremo, en los experimentos de *dinámica molecular* realizados para el estudio de los líquidos, las escalas espaciales y temporales cortas son importantes en los movimientos de las moléculas.

Todos los modelos matemáticos en los cuales se aplica el método de simulación de los N cuerpos pueden formularse, mediante un conjunto de cuerpos interactuando a través de un campo. Estos cuerpos pueden representar estrellas, galaxias, átomos, moléculas o vórtices de fluidos, bajo la influencia de las fuerzas ejercidas por el resto del sistema. Los cuerpos tienen un número de atributos que se conservan (masa, carga, ...) y un número de atributos variables (posición, velocidad, ...). Los atributos variables evolucionan acorde a las ecuaciones del campo. En los casos prácticos, estas simulaciones precisan de grandes valores de N [41].

El método de simulación de N cuerpos que computa cada fuerza entre cada par de cuerpos se conoce como método directo. La complejidad de este método es proporcional a N^2, donde N es el número de cuerpos. Sin embargo, existen otros métodos con una complejidad menor. Hockney y Eastwood han revisado en [56] los métodos de simulación de N cuerpos, y su aplicación a la física del plasma, electrónica de semiconductores, astrofísica, dinámica molecular, termodinámica y a la física de superficies. En esta monografía se han clasificado los métodos para resolver los problemas de N cuerpos en tres clases: Partícula-Partícula (PP), Partícula-Malla (PM) y

Partícula-Partícula–Partícula-Malla (P^3M).

Los métodos PM [56] se han usado extensivamente, y especialmente, en la física de plasma. Desafortunadamente, la malla proporciona una resolución limitada, y en una fuente altamente no uniforme, causa una degradación significativa del rendimiento. Para mejorar la precisión del cálculo en PM, las interacciones de corto rango pueden computarse directamente [54], mientras las interacciones de largo alcance se obtienen de la malla, dando lugar al llamado P^3M. Cuando la precisión requerida es relativamente baja, y las cuerpos se distribuyen más o menos uniformemente en una región rectangular, los métodos P^3M tienen un rendimiento satifactorio. Sin embargo, cuando la precisión exigida es alta, el tiempo de CPU necesario en tales algoritmos es excesivo. Desde que se publicó la monografía de Hockney y Eastwood, aparecieron nuevos métodos de simulación de cuerpos alternativos a PP, PM y P^3M. Estos métodos son llamados "métodos jerárquicos" o "métodos árbol" por la estructura de datos que usan. Entre los algoritmos jerárquicos podemos citar el método de Appel [12], y el de Barnes y Hut (BH) [16] con una complejidad proporcional a $N \log N$, y el *Fast Multipole Method* (FMM) [49], con una complejidad tan reducida como proporcional a N.

Uno de los algoritmos más utilizados es el BH (complejidad, $N \log N$). Puesto que este algoritmo es irregular, en su implementación paralela debe ponerse especial atención a la partición del espacio físico y a la distribución de los cuerpos entre los PEs. Una de las primeras implementaciones de este algoritmo sobre un multicomputador, de tipo pase de mensaje, fue desarrollada por Warren y Salmon [114] para la simulación de una galaxia. Utilizaron la técnica de la *bisección ortogonal recursiva* (ORB) para la partición del espacio físico. Una implementación posterior de los mismos autores [115] fue realizada mediante el mapeado del espacio físico sobre una línea, llamada curva que llena el espacio (*space-filling curve*), utilizando el orden de Morton (figura 5.1.a), y particionando la línea en segmentos de $N_p = N/P$ cuerpos (P es el número de PEs). En la figura 5.1 se muestra algunos de los orndenamientos utilizados. El mapeado se almacena en la memoria de los PEs mediante una tabla. Una fuente de ineficiencia, que ya señalan Warren y Salmon en la descomposición del orden de Morton, es que los PEs pueden abarcar espacio discontinuo. La solución que proponen es el ordenamiento Peano-Hilbert para la descomposición del dominio, que no tiene discontinuidades espaciales (figura 5.1.b). En la tesis de J. P. Singh [98, 100] se propone una nueva técnica de particionamiento llamada *zonas de costo* (*costzones*), la cual utiliza el ordenamiento Peano-Hilbert. Usando esta técnica implementa los algoritmos BH y FMM. Una implementación del método jerárquico para

el problema de los N cuerpos, con un modelo de programación de paralelismo de datos, lo podemos encontrar en [59, 58]. En [69] se describe la implementación de un entorno de trabajo, para simulaciones en diferentes dominios de los N cuerpos.

En este capítulo presentamos la paralelización del algoritmo BH sobre un multicomputador con topología malla y memoria distribuida. Para este fin utilizamos una combinación de dos técnicas: llenado del espacio mediante una curva y la proyección vector para definir la distribución de los datos en el computador [4].

5.2 Métodos jerárquicos

Los algoritmos jerárquicos de los N cuerpos, se basan en que un punto en el espacio físico requiere menos información de aquellas partes del espacio que estén más alejadas. En 1687, Isaac Newton ya formuló la simplificación que utilizan los algoritmos jerárquicos de los N cuerpos: si la magnitud de la interacción entre cuerpos disminuye rápidamente con la distancia, entonces el efecto de un grupo de cuerpos suficientemente alejados del punto que está siendo evaluado, puede aproximarse por un único cuerpo equivalente (figura 5.2). En el ejemplo de una galaxia, la fuerza gravitacional de atracción entre cuerpos disminuye con el cuadrado de la distancia. La idea básica se muestra con el siguiente ejemplo. Si se quiere calcular la fuerza de la Tierra sobre una manzana, no es necesario calcular el efecto de cada átomo de la Tierra sobre los átomos de la manzana. Un buena aproximación, es considerar la Tierra y la manzana cada uno como un punto de masa localizado en su centro de masa, y analizar el sistema a través de estos dos puntos solamente.

La ley fundamental, que gobierna la dinámica de los sistemas astrofísicos, la constituyen las leyes newtonianas:

$$\frac{d^2\vec{x}_i}{dt^2} = \sum_{j\neq i}^{N} \frac{\vec{f}_{ij}}{m_i} = \sum_{j\neq i}^{N} -\frac{Gm_j\vec{d}_{ij}}{|d_{ij}^3|}, \quad \vec{d}_{ij} \equiv \vec{x}_i - \vec{x}_j, \tag{5.1}$$

donde x_i y m_i son, repectivamente, la posición y la masa del cuerpo i y $\vec{f}_{ij}$ es la fuerza que el cuerpo i ejerce sobre el cuerpo j. Las condiciones iniciales son las posiciones y las velocidades iniciales de los N cuerpos. Las incógnitas son las nuevas posiciones de los N cuerpos.

Los métodos jerárquicos aproximados utilizan una estructura de tipo

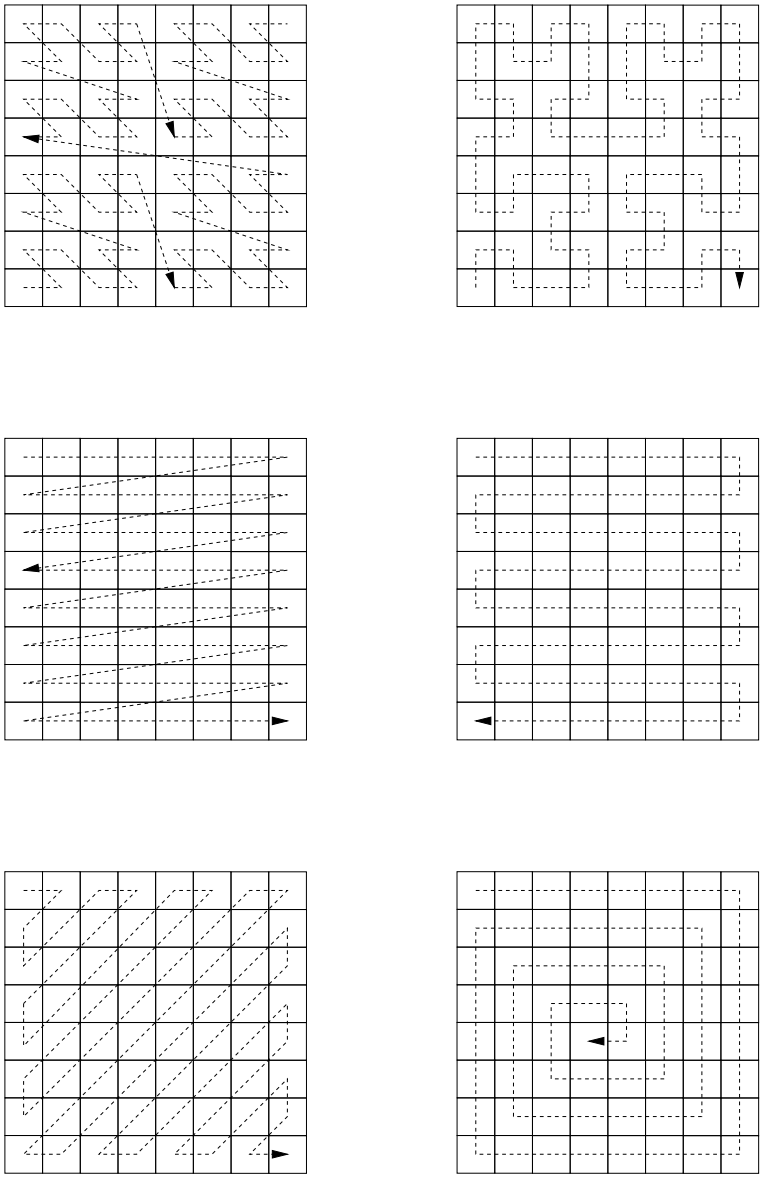


Figura 5.1: Mapeado del espacio físico por el método de la curva que llena el espacio. (a) Ordenamiento Morton. (b) Ordenamiento Peano-Hilbert. (c) Ordenamiento por filas. (d) Ordenamiento serpiente. (e) Ordenamiento Cantor–diagonal. (f) Ordenamiento espiral.

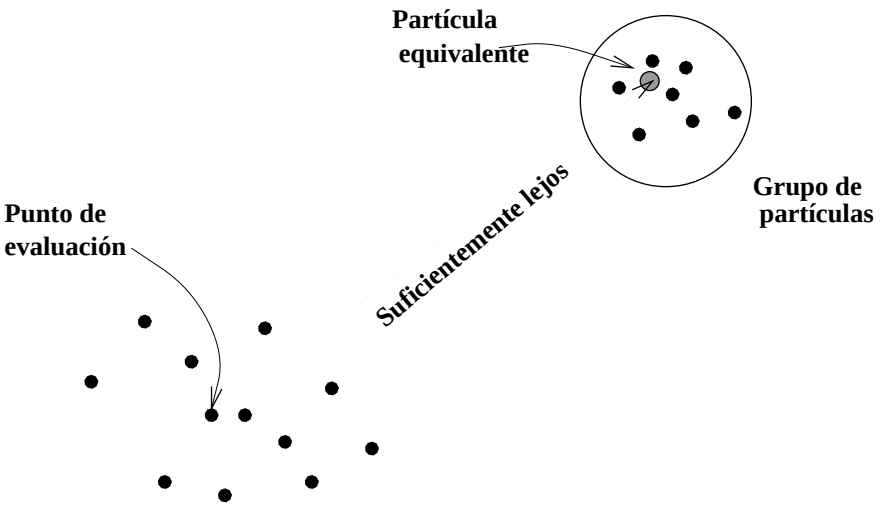


Figura 5.2: Aproximación de un grupo de cuerpos por un único cuerpo equivalente.

árbol (*treecodes*). La aproximación fundamental empleada por este método puede representarse como:

$$\sum_{j \neq i}^{N} \frac{G m_j \vec{d}_{ij}}{|d_{ij}^3|} \approx \frac{GM\vec{d}_{i,CM}}{|d_{i,CM}^3|} + ..., \tag{5.2}$$

donde $\vec{d}_{i,CM} = \vec{x}_i - \vec{x}_{CM}$ es la posición del cuerpo i relativa al centro de masa de los cuerpos que aparecen en el término izquierdo del sumatorio, y los puntos suspensivos indican los términos cuadripolar, octapolar, y términos superiores de la expansión. Generalmente, se usa la aproximación cuadripolar, la cual requiere un segundo término en el lado derecho de la ecuación (5.2).

Todos los métodos jerárquicos, primero construyen una estructura árbol, representación jerárquica del espacio físico, y entonces computan las interacciones atravesando este árbol. El primero de estos métodos fue propuesto por Appel [12]. Sin embargo, este método no es suficientemente estructurado, lo cual hace difícil su programación y análisis de precisión. Los métodos BH y FMM están mucho mejor estructurados y son más prometedores. Vamos a describir estos tres métodos brevemente.

5.2.1 El método de Appel

En 1985 Appel [12] introdujo la estructura de datos jerárquica en la simulación astrofísica de N cuerpos. En terminología de Appel, los cuerpos se incluyen en *celdas* englobadas a su vez en otras celdas, como se muestra en la figura 5.3.a. Obsérvese que las celdas representan regiones irregulares del espacio. La estructura de datos resultante es un árbol binario. La figura 5.3.b muestra el árbol binario equivalente a la jerarquía de celdas de la figura 5.3.a. Appel construye su árbol por los siguientes pasos:

1. Construir el árbol r–ario donde r es la dimensión del espacio [91], con la propiedad de que los dos hijos de un nodo dado, contengan cuerpos que en el espacio estén situados a ambos lados de un eje. En otras palabras, el hijo de la izquierda contiene todos los cuerpos a un lado del eje, y el hijo de la derecha contiene todos los cuerpos al otro lado. Las coordenadas, por ejemplo, x, y ó z se alternan en los sucesivos niveles del árbol.

2. Reorganizar el árbol, de forma que subceldas cercanas sean hijas de la misma celda (ver figura 5.4), conservando las posiciones, las velocidades, aceleraciones, y todas las demás propiedades importantes de los datos de la celda. El procedimiento puede describirse como sigue: cuando dos cuerpos se encuentran más cerca entre ellos que a su nodo padre, se produce un reordenamiento del árbol sin respetar la estructura global del árbol. La consideración de celdas *bien separadas*, se ajustan por un parámetro δ, de forma que se utiliza la aproximación del centro de masa, si el diámetro de la mayor de las celdas excede δ veces la separación entre ellas.

5.2.2 El método de Barnes Hut

El algoritmo BH realiza una división jerárquica del espacio en celdas regulares. En el caso bidimensional cada celda se divide en cuatro, generándose una estructura de árbol 4–ario (*quadtree*), mientras que en el caso tridimensional la división se realiza en ocho celdas cúbicas obteniéndose un árbol 8–ario (*octree*). La raíz del árbol representa una celda del espacio lo suficientemente grande para contener todos los cuerpos del sistema. Cada una de las 4 u 8 celdas en las que se divide el espacio se asigna a uno de los nodos del nivel 1 y, a su vez, cada celda es dividida en 4 u 8 subceldas que se asignan a los

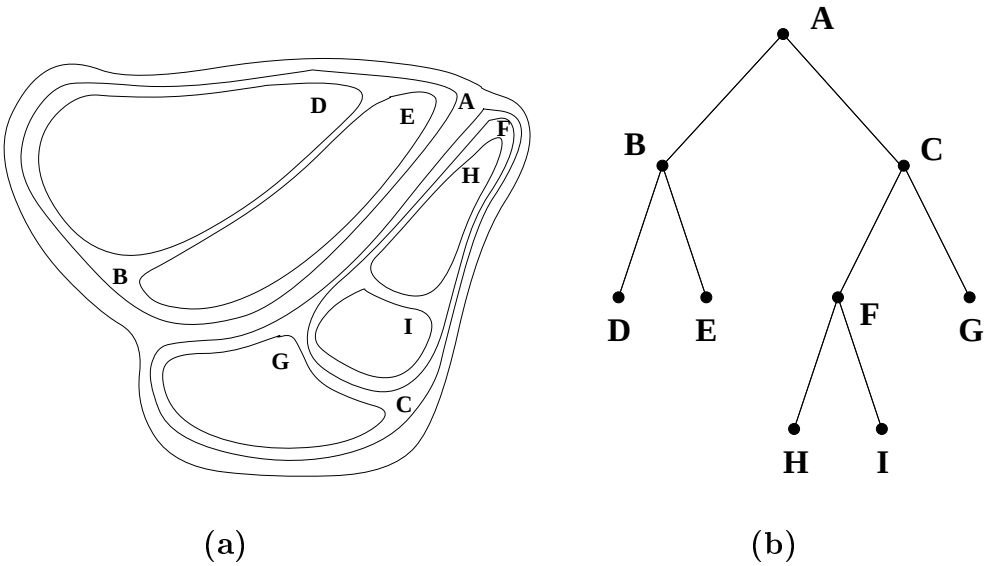


Figura 5.3: (a) Un ejemplo de la jerarquía de celdas de Appel. (b) Un árbol binario equivalente al conjunto de celdas de (a).

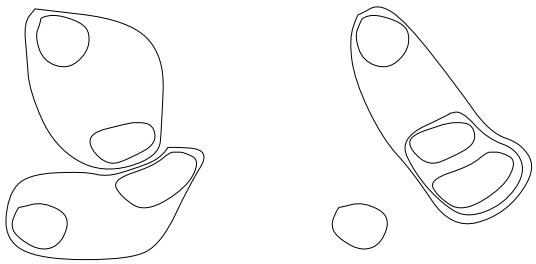


Figura 5.4: Reordenamiento de las celdas por el procedimiento de Appel.

hijos del correspondiente nodo del nivel 1, y así sucesivamente. El proceso termina cuando se alcanzan subceldas que contienen uno o ningún cuerpo. En la figura 5.5 mostramos un ejemplo bidimensional.

El algoritmo BH computa la masa y el centro de masa de cada celda interna en el árbol. De esta forma cada celda interna puede tratarse como un cuerpo con una masa y una posición. La configuración de los cuerpos puede darse con más precisión calculando además del centro de masa, el momento cuadripolar, el octopolar, etc.

En este esquema, la fuerza total que actúa sobre un cuerpo se obtiene computando la fuerza que ejerce la red de celdas sobre el cuerpo. Si el centro

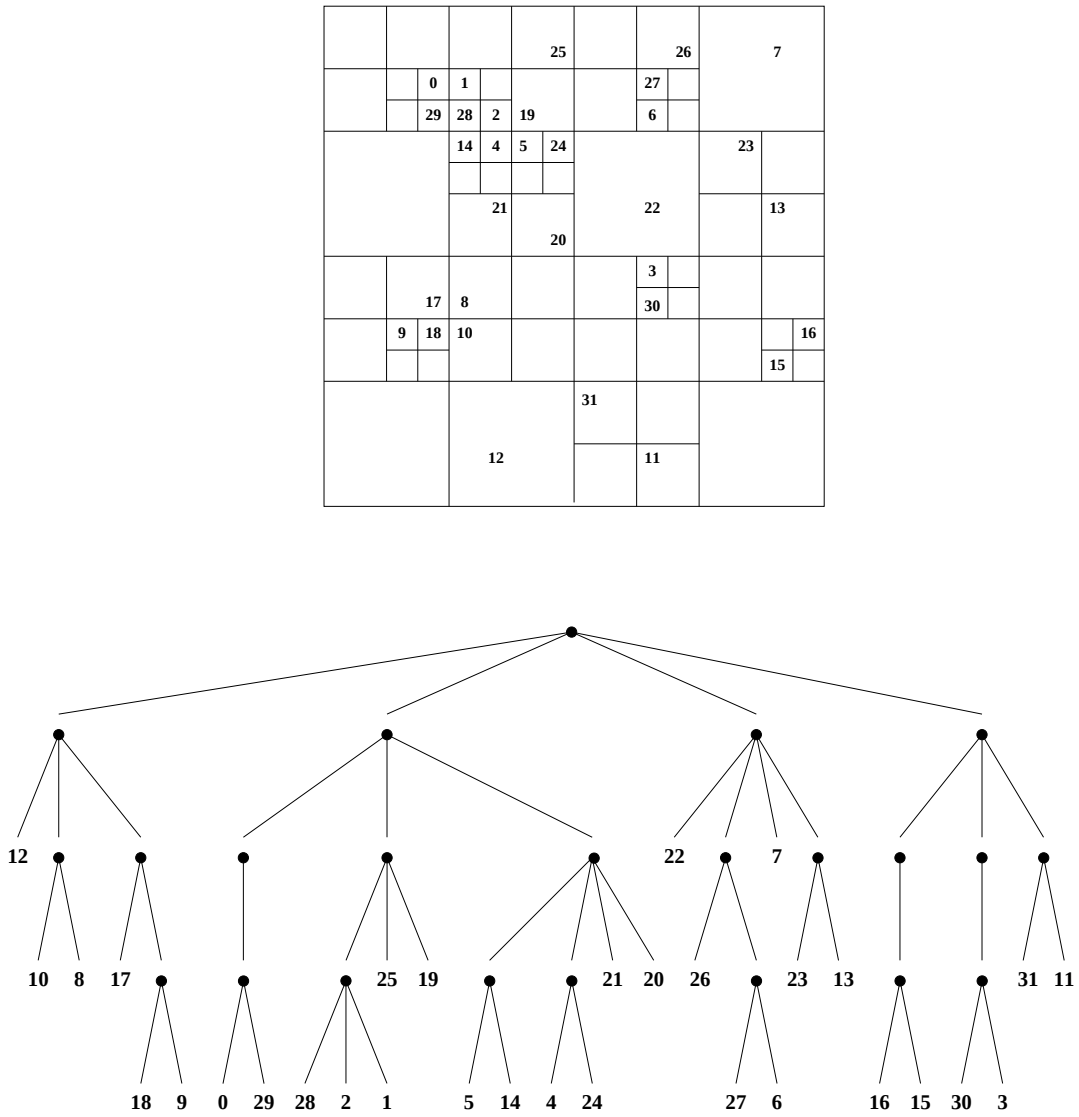


Figura 5.5: Una distribución de cuerpos en dos dimensiones y el correspondiente quadtree.

de masa de una celda está lo suficientemente lejos del cuerpo, el subárbol entero que tiene como raíz esa celda, es aproximado por un único cuerpo. En este caso se calcula la fuerza que ejerce el centro de masa de la celda sobre el cuerpo. Si en contra, el centro de masa de una celda no está lo bastante lejos del cuerpo se realizarán estos mismos pasos con cada una de las subceldas hijas de esta celda. Este test es llamado "Criterio de aceptabilidad multipolar" (*Multipole Acceptability Criterion*, MAC). El test esta basado en consideraciones geométricas sobre el tamaño de la celda, la posición del cuerpo, y quizás, el contenido de la celda. Un buen MAC es esencial para estructuras jerárquicas, pues, influye en la velocidad, precisión e implementación paralela. El MAC debe cumplir un equilibrio entre velocidad y precisión. A continuación vamos explicar tres simples MACs que pueden aplicarse a casi todos los algoritmos jerárquicos.

El MAC de Barnes y Hut

En su formulación original, Barnes y Hut [16], introdujeron un MAC parametrizado. Una celda se considera bien separada si cumple la siguiente condición:

$$\frac{l}{d} < \theta \tag{5.3}$$

donde l es la longitud del lado de la celda considerada, d es la distancia del cuerpo al centro de masa de la celda, y θ es un parámetro definido por el usuario que se utiliza como control de precisión del cálculo de la fuerza, ver figura 5.6. Obviamente, el valor de θ es crucial. Un valor muy bajo implica que un cuerpo tiene que estar muy distante de una celda, antes de que la aproximación multipolar es aceptada. El árbol es atravesado hasta los niveles más profundos, y se calculan un gran número de interacciones. Cuando $\theta \to 0$, un cuerpo computaría todas las interacciones con el resto de los cuerpos del sistema, y en este caso la complejidad del cálculo de la fuerza es proporcional a N^2. Un valor grande de θ implica mayor confianza en las aproximaciones multipolares, y muy pocas interacciones. El número de interacciones escala aproximadamente con θ^{-3} [88], con lo que es claro que el rendimiento del algoritmo es muy sensible a θ. En los trabajos astrofísicos se utiliza un rango muy pequeño de valores, $0.7 \leq \theta \leq 1.0$, y la aproximación multipolar termina usualmente con el término cuadripolar.

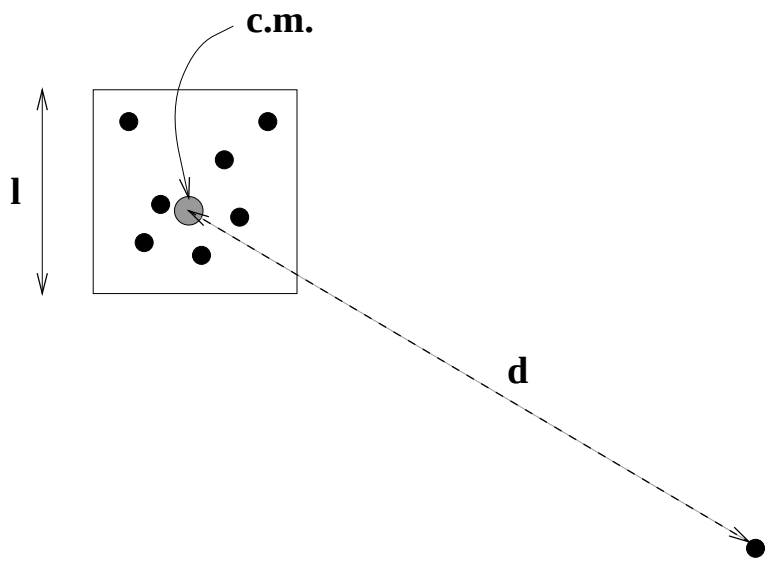


Figura 5.6: Test geométrico del criterio MAC de Barnes y Hut. La aproximación multipolo es aceptable si y sólo si $\frac{l}{d} < \theta$.

El MAC de distancia mínima

El MAC del algoritmo BH de la figura 5.7 está basado en el concepto más vago (pero ciertamente correcto) de que la precisión de una aproximación multipolar en la posición X, está determinado por la relación del tamaño de la celda, a la distancia de X a la celda. En [89] se demuestra que este criterio puede introducir grandes errores cuando un cuerpo se encuentra cerca del borde de la celda, pero lejos de su centro de masa. Esto sugiere un MAC alternativo el cual sustituye la distancia usada por el MAC de BH por la distancia mínima de un cuerpo a una celda. Nos referimos a esto como MAC de distancia mínima.

Un aspecto muy útil del MAC de distancia mínima es el hecho de que es completamente independiente del contenido de la celda. Es posible evaluar el MAC de distancia mínima sin conocer nada sobre la otra celda, salvo su tamaño y su posición. En contra, los otros MACs sólo pueden evaluarse cuando el centro de masa ha sido determinado, lo que requiere que todas las posiciones de los cuerpos de la celda sean conocidas. La posibilidad de evaluar el MAC, antes que el contenido de las celdas, hace atractivo su uso para algunas implementaciones sobre computadores de memoria distribuida [88, 114], donde se desea evaluar el MAC para celdas que residen en la memoria de otro PE.

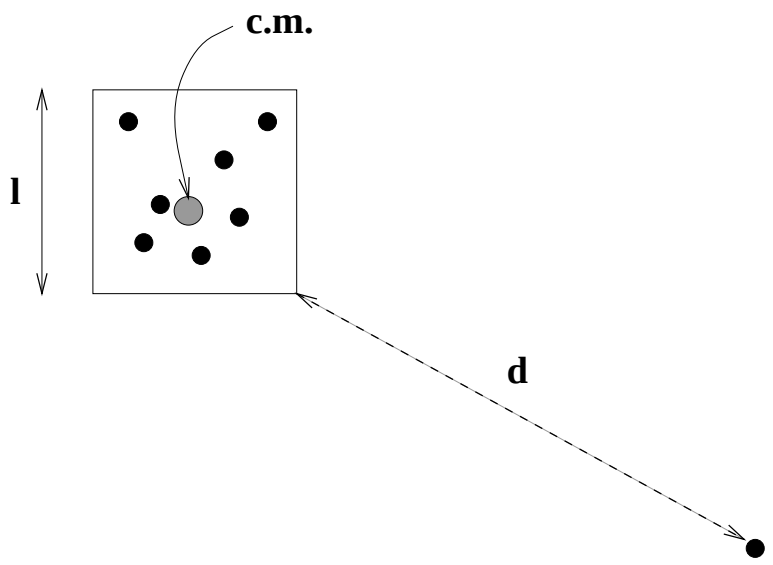


Figura 5.7: Test geométrico del criterio MAC de distancia mínima. La aproximación multipolar es aceptable si y sólo si $\frac{l}{d} < \theta$.

El MAC de distancia máxima al origen monopolar

Este MAC [89] está motivado por el hecho de que el error más grande posible para la distribución en una celda, es proporcional a la magnitud del término monopolar y se incrementa monotónicamente en función de b_{max}/d, donde b_{max} es la distancia máxima al origen monopolar, $\vec{r_0}$, usualmente el centro de masa de la celda, a algún punto de la celda,

$$b_{max} = \max_{x \in v} |\vec{x} - \vec{r_0}| \,. \tag{5.4}$$

Dado que el error es una función incremental de b_{max}/d, es natural que b_{max}/d aparezca en el MAC. El criterio permite la aproximación multipolar si $b_{max}/d < \theta$, ver figura 5.8.

El problema de los N cuerpos gravitacional

La simulación consiste en la evolución temporal del sistema. En el instante inicial se debe especificar el estado inicial del sistema en alguna región finita del espacio (*la caja computacional*). En cada una de las siguientes iteraciones, se realizan los cálculos de las nuevas posiciones y velocidades de los cuerpos.

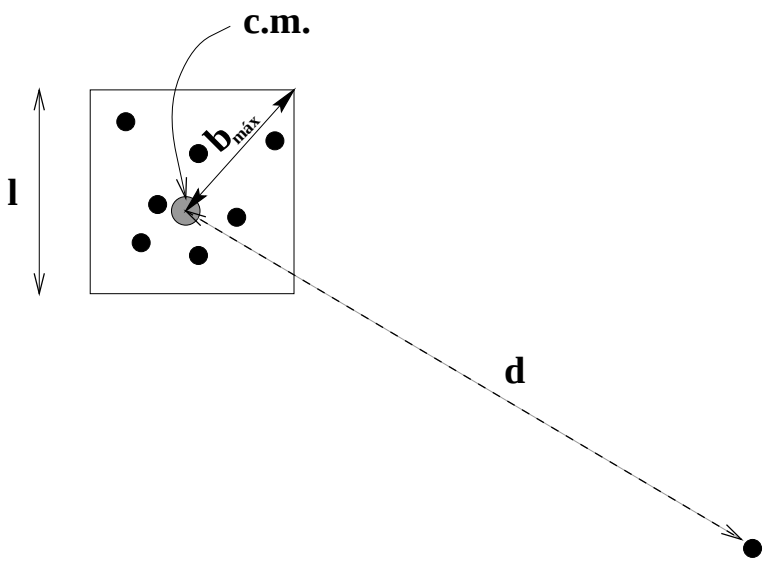


Figura 5.8: Test geométrico del criterio MAC de distancia máxima de origen monopolar. La aproximación multipolar es aceptable si y sólo si $\frac{b_{max}}{d} < \theta$.

En las iteraciones, son posibles tres clases de movimientos del cuerpo en el árbol: el cuerpo se mueve dentro de su celda (figura 5.9.a), el cuerpo emigra de una celda a otra, (figura 5.9.b) o el cuerpo sale de los límites iniciales de la caja computacional, (figura 5.9.c).

El primer caso no implica ningún problema. El segundo caso tampoco implica problemas si el paso temporal se ha elegido adecuadamente. Las celdas son recalculadas en cada ciclo, por lo que el deambulamiento de un cuerpo de una celda a otra no supone ningún problema.

El tercer caso es realmente más serio, pues si un cuerpo sale de los confines de la caja computacional, es necesario recomputar el árbol entero, que en principio puede estar distribuido entre diferentes PEs. Este problema puede eliminarse incluyendo una región de seguridad, alrededor de la región original de simulación, ver figura 5.9.d.

En el problema gravitacional, se puede implementar una mejora basada en el hecho de que los cuerpos cercanos sienten casi la misma influencia de los objetos distantes. En la analogía de nuestra manzana, vemos que dos manzanas cercanas casi sienten el mismo campo gravitacional de las múltiples partículas que constituyen la Tierra. Si se desea conocer la aceleración gravitacional de dos manzanas en el mismo árbol es suficiente evaluar la ecuación (5.1) sólo una vez, y usar el resultado para ambas. Esta analogía sólo captura

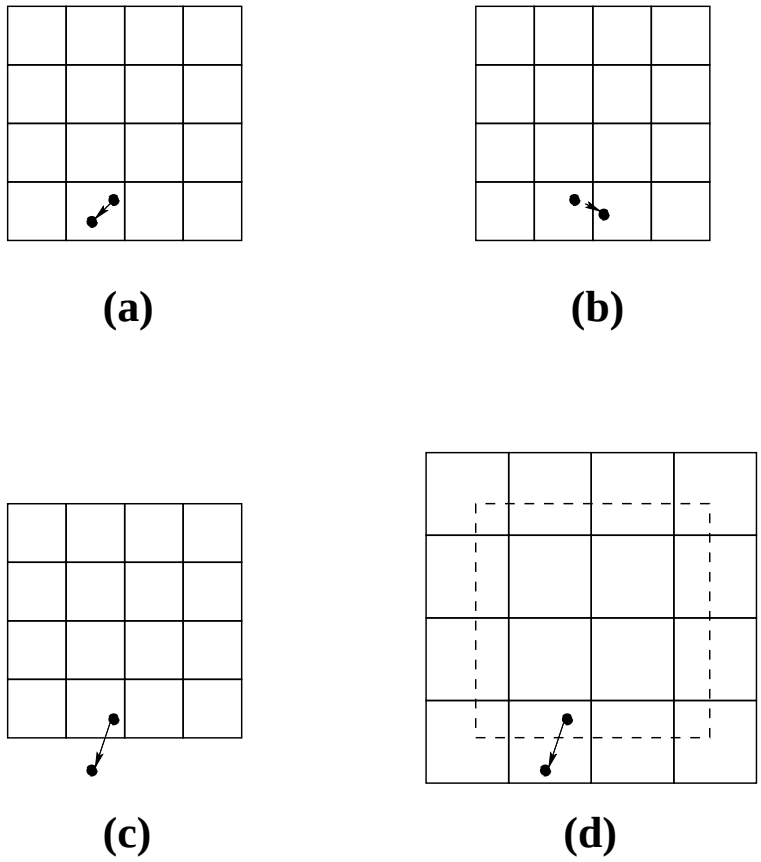


Figura 5.9: (a) Partícula deambulando dentro de su celda. (b) Partícula deambulando entre dos celdas del árbol. (c) Partícula deambulando fuera de la caja computacional, requeriendo la reconstrucción del árbol completo. (d) Región de simulación original (línea rayada) aumentada con una región de seguridad para reducir la frecuencia de partículas deambulando fuera de los límites del árbol.

el primer término (constante) en una expansión en serie. La expansión considerada es la serie de Taylor para la ecuación (5.1), alrededor del punto $\vec{x}$. Usando la serie de Taylor se reduce el número de evaluaciones de la ecuación (5.1), introduciendo una nueva clase de interacción en el problema. Ahora es necesario evaluar las interacciones entre las celdas origen y las celdas destino.

El uso de la serie de Taylor reduce la complejidad del método a $O(N)$. La evaluación de los coeficientes de la serie de Taylor puede ser mucho más costosa, temporalmente, que la evaluación discreta de la ecuación (5.1), por lo que el beneficio de calcular menos interacciones se cancela parcialmente

por el hecho de que estas interacciones son mucho más costosas.

Después de calcular la fuerza que actúa sobre cada cuerpo, se actualiza la posición y la velocidad de cada cuerpo. Resumiendo, en cada iteración se tienen que realizar las operaciones que muestra la figura 5.10.

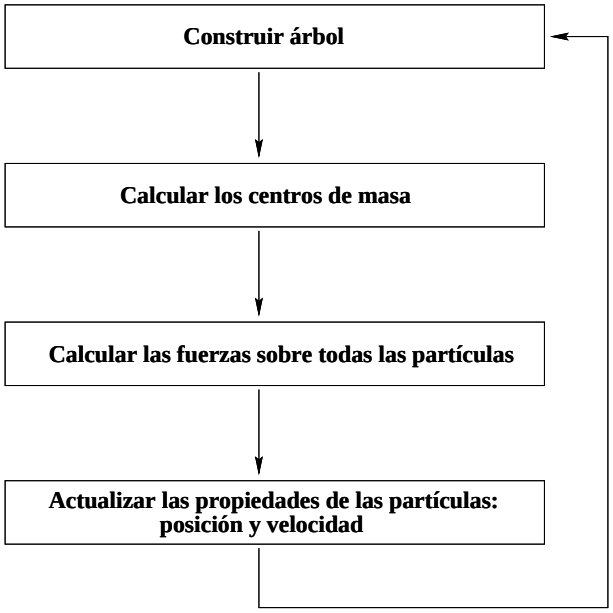


Figura 5.10: Organigrama de un paso temporal en el algoritmo BH.

5.2.3 El método de Greengard

Otro algoritmo jerárquico es el FMM de Greengard [49, 47]. Este método también usa una descomposición recursiva del espacio sobre una estructura árbol, con una estrategia de aproximación de celdas por un único cuerpo cuando está lo suficientemente lejos. En este método, una celda se considera lo suficientemente lejos, o *bien separada*, de otra celda b, si la separación de b es más grande que la longitud de b. En la figura 5.11 se muestra el criterio de celdas bien separadas. Las celdas A y C en la figura están bien separadas. La celda D está bien separada de la celda C, pero la celda C no está bien separada de la celda D.

Este método usa expansiones multipolares y conjuntos de celdas fijas de interacción, que cumplen con el criterio de "bien separado". El FMM tiene un límite bien definido de error, ϵ, que se puede conseguir cuando la expansión multipolar es del orden $p = -\log_2(\epsilon)$, siendo p el término multipolar

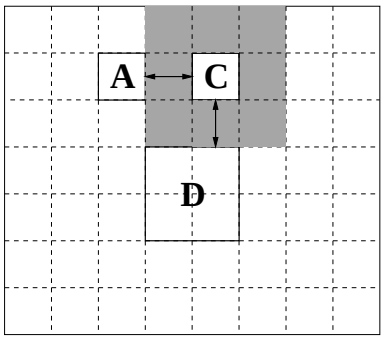


Figura 5.11: Criterio de celdas bien separadas en el algoritmo FMM.

de orden más alto. En dos dimensiones, y cuando los cuerpos no están excesivamente agrupados, el FMM es muy eficiente. Ha sido implementado en multicomputadores [48, 122]. El punto de *cruce* (el valor de N en el que el algoritmo comienza a ser más rápido que el método directo de complejidad $O(N^2)$) es tan bajo como de unos cientos de cuerpos. Por otra parte, la implementación del FMM en tres dimensiones no tiene un rendimiento tan bueno. Schmidt y Lee [92] han implementado el algoritmo para tres dimensiones, y encontraron un punto de cruce sobre 70 mil cuerpos. La razón es que el paso más intensivo del algoritmo escala como p^2 en dos dimensiones, y p^4 en tres dimensiones. Es posible obtener mejor rendimiento usando un p más pequeño [18], pero los errores pueden ser más grandes en este caso. Una de las mayores ventajas del método FMM sobre otros métodos, tales como el BH, era que el límite de error estaba más rigurosamente definido. Pero esta deficiencia ya ha sido remediada, [115].

La eficiencia del FMM se consigue permitiendo un mayor número de cuerpos por hoja del árbol que en el método BH. Greengard sugiere [47] un máximo de 40 cuerpos por celda hoja. Las fases en un paso temporal son esencialmente las mismas que en la aplicación BH:

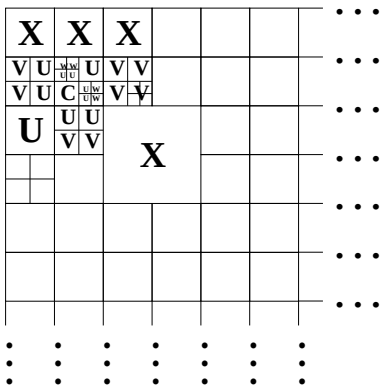
1. Construir el árbol.

2. Computar las expansiones multipolar de las cedas sobre su centro geométrico en pasos ascendentes a través del árbol.

3. Computar las fuerzas.

4. Actualizar las propiedades de los cuerpos.

Tres de las fases son idénticas en estructura a las correspondientes fases en BH. Vamos a ver en más detalle el cálculo de las fuerzas en el algoritmo FMM. Cada celda C divide el resto del dominio computacional en un conjunto de listas, cada lista contiene celdas que mantienen una cierta relación espacial con C. Las listas están descritas en la figura 5.12. El primer paso en la fase de cálculo de la fuerza es construir estas listas para todas las celdas. Las interacciones de cada celda son computadas con las celdas de su lista. La naturaleza jerárquica del algoritmo y la construcción de la lista asume que ninguna celda C computa interacciones con celdas que están bien separadas de su padre (las celdas en blanco para la celda C en la figura 5.12). Las interacciones con estas celdas distantes son computadas por los ascendientes de la celda C, en niveles más altos del árbol. Los detalles de los diferentes tipos de interacciones, indicados en la figura 5.12 se puede encontrar en [47].



Lista U (sólo celdas hojas): todas las celdas hojas adyacentes a C.

Estas celdas no pueden ser aproximadas de ninguna manera.

LIsta V (celdas): Hijas de los colegas del padre de C que están bien separadas de C.

Están bien separadas de C pero no del padre de C.

Lista W (sólo celdas hijas): Descedientes de los colegas de C cuyos padres son adyacentes a C, pero ellas no son adyacentes a C.

La expansión multipolar de estas celdas pueden ser evaluadas en los cuerpos de C; la expansión multipolar de sus padres no puede; y sus expansiones multipolares no pueden ser trasladadas a la expansión local sobre el centro de C.

Lista X (celdas): Dual de la lista W: celdas en que C están en su lista W.

Sus expansiones multipolares no pueden ser trasladadas al centro de C, pero el campo de sus cuerpos individuales sí.

Figura 5.12: Listas de interacción para una celda del algoritmo FMM.

Las celdas internas computan las interacciones sólo con dos listas (las

listas V y X de la figura 5.12). Los resultados de estas interacciones se almacenan como *expansiones locales* en las celdas internas. La expansión local de una celda es una expansión serie, que representa la influencia ejercida sobre ella por otras celdas que están lo bastante lejos para aproximarse por su expansión local. Las celdas hojas computan estas interacciones celda–celda, pero también computan las interacciones de los cuerpos que contienen con otros cuerpos (en la lista U) y celdas (en la lista W). Cuando se han calculado todas las listas de interacciones, las influencias de las celdas distantes sobre los cuerpos de las celdas hojas están contenidas en la lista de sus ascendientes. Esta influencia se propaga a las celdas hojas al transferir la expansión local desde los padres a los hijos, descendiendo en el árbol. Finalmente, la expansión local de las celdas hojas se evalúa sobre los cuerpos que contienen.

5.2.4 Diferencia entre los algoritmos BH y FMM

- Mientras que el método BH computa directamente solo interacciones cuerpo–cuerpo o cuerpo–celda, el método FMM también computa interacciones directamente entre celdas.
- Los algoritmos determinan de una manera diferente, si una celda está lo suficientemente lejos o bien separada de otra celda. En el FMM, el que dos celdas estén bien separadas se determina enteramente por su longitud y la distancia entre ellas, no por un parámetro definido por el usuario como el θ del método BH.
- La aproximación FMM de una celda no se hace por su centro de masa, sino por una expansión serie de las propiedades del cuerpo sobre el centro geométrico de la celda. Esta expansión serie es llamada *expansión multipolar* de la celda. En la práctica, se usa sólo un número finito de términos en la expansión multipolar, y este número determina la precisión de la representación.
- En el FMM, la precisión en el cálculo de la fuerza no es controlada por el cambio de consideración de las celdas "lejanas" (como en BH) sino por el cambio del número de términos usado en la expansión serie.
- El algoritmo BH es inherentemente adaptativo; no hace ninguna suposición sobre la distribución de los cuerpos en el dominio. El FMM, sin embargo, tiene dos vertientes distintas: una versión simple que supone una distribución uniforme de los cuerpos, y una versión más compleja que se adapta a distribuciones arbitrarias. La que hemos presentado es la FMM adaptativa.

- Mientras que las matemáticas del BH son las mismas para tres dimensiones que en dos, el FMM usa diferente formulación [47]. En tres dimensiones es más compleja la base matemática, de forma que el FMM en tres dimensiones es mucho más caro que el FMM en dos dimensiones.
- El método FMM es mucho más complicado para programar que el método BH.

Nosotros consideramos el método BH por varias razones:

1. Ha recibido, de lejos, más atención de la comunidad astrofísica y es el más usado en simulaciones "reales" científicas.
2. La distinción entre las fases del cálculo de la fuerza y la construcción del árbol hace la paralelización eficiente mucho más fácil. Los algoritmos de Appel y Greengard requieren mucha más contabilidad por la necesidad de almacenar y computar interacciones de objetos no terminales en la jerarquía.
3. La relación entre complejidades $O(N \log N)$ para los algoritmos Appel y BH, y $O(N)$ para el algoritmo FMM, no es suficiente para comparar estos algoritmos. Greengard considera $1752N$ pares de interacciones por paso temporal, mientras que las observaciones experimentales de Salmon [88] muestran que el algoritmo BH con $\theta = 0.8$, requiere aproximadamente $15 - 35N \log_2 N$ pares de interacciones por paso temporal. El regimen asintótico, en el cual se espera que el algoritmo de Greengard sea superior, es para $N \approx 10^{15}$, más allá de ninguna simulación realizada.

5.3 El algoritmo Barnes-Hut paralelo

En este apartado presentamos el particionamiento y planificación del algoritmo BH sobre un multicomputador de topología malla. En el apartado siguiente veremos la implementación y los resultados obtenidos de este algoritmo paralelo sobre un sistema concreto, el T3E de Cray.

5.3.1 Localidad física

En los algoritmos jerárquicos, cada cuerpo sólo interactúa con una parte del árbol completo: las secciones lejanas sólo interactúan en niveles bajos

del árbol, mientras que las secciones vecinas interactúan a nivel de hojas. Si unimos todos los árboles que interactúan con los cuerpos contenidos en un PE, obtenemos todos los datos que necesita ese PE para realizar las computaciones. Este árbol se denomina árbol esencial local (*locally essential tree*). En la figura 5.13, se puede ver la división en celdas que genera el árbol local esencial de un PE, cuyo dominio es la esquina inferior izquierda del espacio físico.

Después de la distribución de los datos en los PEs, cada PE controla una región del espacio, y tiene que calcular la fuerza para todos los cuerpos dentro de esa región. Por consiguiente, gran parte del árbol es completamente innecesario para los cálculos que tienen que realizarse en un PE. Esto es extraordinariamente importante, ya que si no, tendríamos que almacenar el árbol entero en cada PE. Las grandes simulaciones serían imposibles de realizar en los multicomputadores de memoria distribuida, pues no habría memoria local suficiente en los PEs.

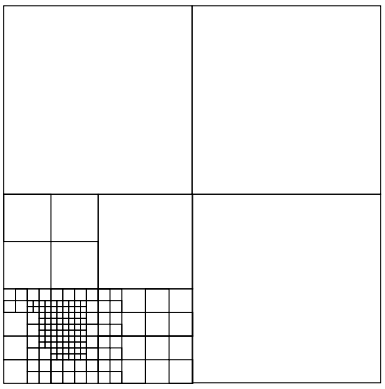


Figura 5.13: División en celdas que genera el árbol local esencial para un PE en la esquina inferior izquierda del sistema.

En la literatura se han usado básicamente dos técnicas para asegurar la localidad física de los cuerpos: bisección ortogonal recursiva (ORB) [114], y llenado del espacio mediante una curva [83]. Utilizaremos esta última técnica, ya que es más simple y consigue mejor rendimiento [98, 99, 83, 116, 115]. A continuación, vamos a explicar brevemente los dos particionamientos.

Bisección ortogonal recursiva

La ORB es una técnica que mantiene la localidad física de un problema mediante el particionamiento explícito del espacio dominio. Esta técnica fue

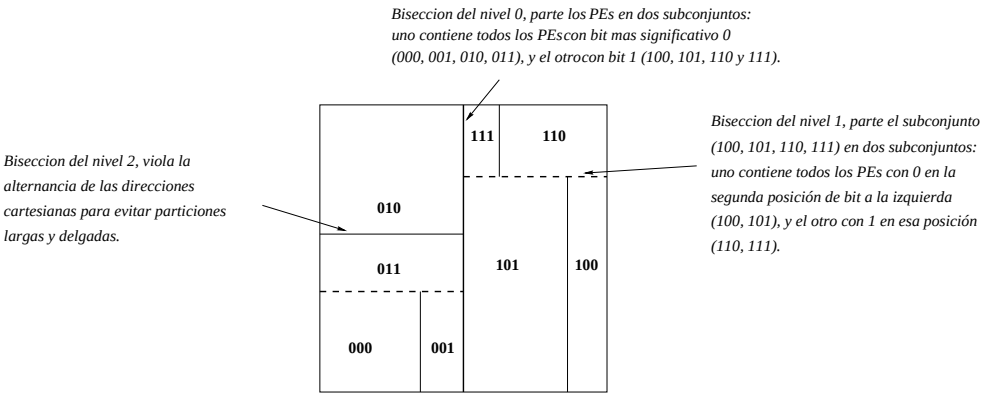


Figura 5.14: Partición ORB.

usada en la implementación por pase de mensajes del algoritmo BH por Salmon [88]. La idea de la partición ORB es la división recursiva del dominio computacional en dos subespacios con igual coste, hasta que hay un subespacio por PE, ver figura 5.14.

Para conseguir una carga balanceada, el coste de un subespacio se define, como la suma de los costes de todos los cuerpos del subespacio. Inicialmente, a todos los PEs se les asocia el dominio entero del espacio. En cada paso se divide un espacio, los PEs asociados a ese espacio se dividen igualmente en dos subconjuntos, y cada subconjunto es asignado a uno de los subespacios resultantes. La dirección cartesiana en la que se realiza la división se suele alternar en sucesivas divisiones, aunque esta regla puede violarse para evitar subespacios demasiados largos y delgados. Una descripción más detallada de la implementación de la partición ORB, para el algoritmo BH, se puede encontrar en [88].

La partición ORB introduce una nueva estructura de datos, el árbol binario ORB, distinto del árbol BH. Los nodos del árbol ORB son los subespacios subdivididos recursivamente, y las hojas son las particiones espaciales finales. La técnica ORB no es fácil de implementar ni de depurar, y puede tener una penalización temporal significativa, particularmente cuando el número de PEs se incrementa. También se restringe el número de PEs a una potencia de dos, aunque pueden desarrollarse más complejas variantes para evitar esta restricción. Por estas razones, hemos usado otra técnica que es más flexible y escala mucho mejor con el número de PEs usados.

La técnica de llenado del espacio mediante una curva

Las descomposiciones en árbol 4–ario (quadtree) son útiles como métodos de ordenamiento espacial [90]. Este mapeado de una dimensión más alta a una más baja debe mantener la localidad espacial de la distribución de los datos en el dominio. En [91] podemos encontrar diferentes métodos de ordenamiento espacial mediante una curva que llena el espacio, y que mostramos en la figura 5.1. Un método simple, para realizar la transformación del espacio multidimensional a un espacio unidimensional es el intercalado de bits (bit–interleaving) [116, 79]. Los índices de los datos se construyen a partir de las coordenadas de los cuerpos en el espacio d–dimensional, convirtiendo los números reales en el dominio $(\vec{r}_{min}, \vec{r}_{max})$ a enteros, donde los valores de estos dos vectores $\vec{r}$, definen las esquinas opuestas de la caja computacional que limita la distribución de los cuerpos. Mapeamos los bits más significativos, $(l_k - 1)/d$ de los enteros a una clave de longitud l_k bits, mediante el intercalado uno a uno de los bits, tal como se puede ver en la figura 5.15. l_k se almacena como un vector de enteros que puede variar su tamaño, usualmente se usa $l_k = 64$ bits. A modo de ejemplo, considerando un cuerpo de coordenadas $(13, 4)$ en dos dimensiones, la correspondiente representación binaria es $(1101, 0100)$, y el resultado de intercalar los bits es 10110010, por lo que $(13, 4)$ se mapea a la posición 178 de un array unidimensional. El mapeado produce una curva de llenado de orden Norton, ver figura 5.1.a. Existen otras curvas de llenado del espacio distintas [83], nosotros hemos usado la curva de orden Peano-Hilbert.

Esta curva tiene dos propiedades muy importantes y deseables para reducir las comunicaciones: dos puntos adyacentes en la curva son adyacentes en el dominio físico, y las regiones del espacio definidas por segmentos continuos de la curva de llenado son conexas. Estas propiedades contribuyen a reducir la comunicación interprocesador. La curva de Peano-Hilbert puede verse como una curva en forma de U, que visita cada uno de los cuatro cuadrantes del plano en el primer nivel. Cada subsiguiente nivel divide los cuadrantes del nivel anterior en cuatro cuadrantes más finos, los cuales son visitados por nuevas curvas en forma de U. Cada cuadrante es visitado completamente por la curva, antes que el siguiente cuadrante, por lo que la localidad espacial se conserva. El origen y dirección de la curva en forma de U depende del cuadrante y del ordenamiento que tenía en el nivel previo. Esto se muestra en la figura 5.16.

El mapeado del espacio a la línea se hace sólo una vez, y es por tanto un paso de preprocesamiento. Una vez ha sido establecida, el siguiente paso es la distribución de esta curva entre los PEs.

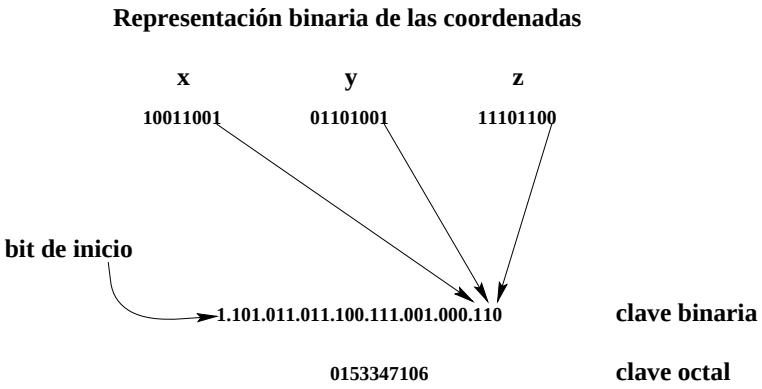


Figura 5.15: Ilustración del mapeado de una clave. Los bits de las coordenadas son intercalados, y un bit de inicio se coloca en la posición más significativa. Sin el bit de inicio, podría resultar una ambigüedad entre las claves en que todos sus bits más significativos son ceros. En este ejemplo, los 8 bits de las coordenadas "x", "y" y "z", se mapean a una clave de 25-bits.

5.3.2 Mapeado

Durante la evaluación del algoritmo sobre el multicomputador, es necesario redistribuir los datos entre los PEs para la construcción del árbol local esencial, y para mantener un buen balanceo de la carga. En general, esto implica cambios en la estructura de almacenamiento, por lo que visualizar el movimiento de los datos puede ser bastante difícil. La proyección vector simplifica la tarea de un gran rango de problemas, usando una representación compacta del mapeado de los datos sobre la memoria.

Cada PE tiene N/P datos en su memoria. Después de su conversión unidimensional indexada por una curva de llenado del espacio, el mapeado puede ser realizado por los índices de orden de almacenamiento. Denotaremos esta secuencia de datos usando la *representación índice-dígito*

$$[\underbrace{t_n \cdots t_{u+w+1}}_{fila}, \underbrace{t_{u+w} \cdots t_{u+1}}_{columna}, \underbrace{t_u \cdots t_1}_{memoria}] \tag{5.5}$$

Utilizamos una proyección vector de tipo serpiente, que tiene la ventaja de mapear sobre PEs vecinos las particiones adyacentes en el dominio físico. En la figura 5.17 podemos ver su aplicación al ejemplo de la figura 5.5.

Adicionalmente, en una tabla de tamaño $V \times W$ almacenamos el índice del primer elemento de cada PE. En el instante inicial esta tabla contiene los

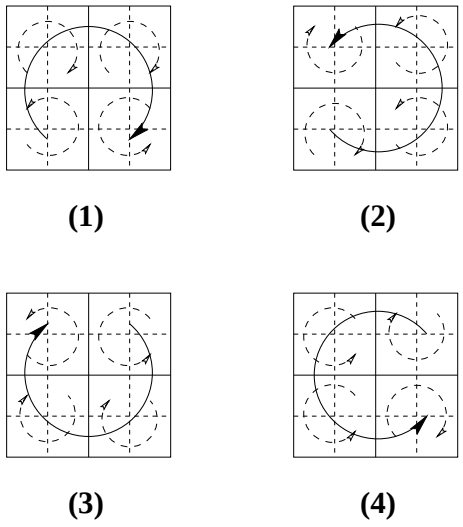


Figura 5.16: Ordenamiento de las celdas.

valores,

$$[0\cdots0, 0\cdots0\ \overset{t_{u+1}}{\boxed{1}}\ 0\cdots0, \cdots, \overset{t_n}{\boxed{1}}\cdots\ \overset{t_{u+1}}{\boxed{1}}\ 0\cdots0] \tag{5.6}$$

Este método nos permite saber, en cada momento, en qué PE está cada uno de los cuerpos y nos permite realizar las comunicaciones más eficientemente. Por ejemplo, si $N = 2^{10}$ y $V \times W = 4$ esta tabla contiene los siguientes valores,

0	256	512	768

(5.7)

5.3.3 Balanceo de la carga

No todos los cuerpos tienen asociados el mismo coste de computación. Definimos el coste de un cuerpo como el tiempo de CPU necesario para calcular la fuerza que actúa sobre él. El coste total de cada partición en el tiempo total es necesario para calcular las fuerzas sobre todos los cuerpos contenidos en esa partición. Por tanto, el hecho de que cada partición contenga el mismo número de cuerpos, no implica necesariamente que los costes de las particiones sean los mismos. Ello implica que debemos distribuir las zonas de coste entre los PEs.

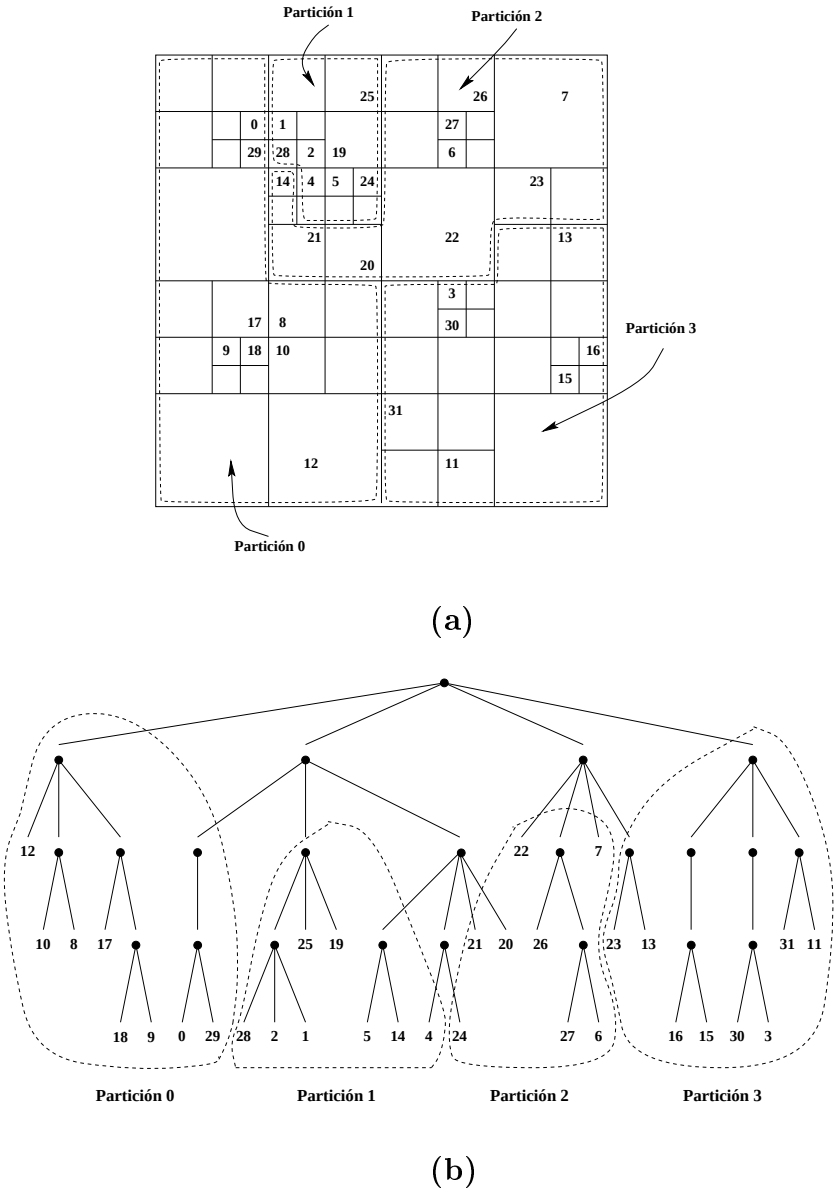


Figura 5.17: (a) El resultado de la partición si se aplica la técnica de las zonas de coste con el ordenamiento de las celdas de la figura 5.16. (b) Árbol generado.

El procedimiento que hemos usado para conseguir el balanceo de la carga es el siguiente: Al comenzar el algoritmo consideramos un coste computacional idéntico para cada uno de los cuerpos. El coste total en el dominio físico, que está en la raíz del árbol, se divide equitativamente entre los PEs. Esto generará un desbalanceo computacional que medimos y almacenamos en los PEs. Como de una iteración a la siguiente no existen muchas variaciones, se puede aproximar el coste de cada cuerpo, por el medido en la iteración previa.

Después de la primera iteración, las particiones deben ajustarse para mantener el balanceo de la carga computacional. El criterio de rebalanceo es:

$$\max_{0 \leq q < P} \mid W(q) - \mu \mid > valor\_umbral \tag{5.8}$$

donde $W(q)$ es el coste computacional de la partición asociada al PE q, y μ es el cociente entre el coste computacional total y el número de PEs. El rebalanceo se realiza si el desbalanceo supera un valor umbral.

El rebalanceo se lleva a cabo moviendo los cuerpos entre PEs vecinos. Esto es necesario para mantener la localidad de los datos y poder generar eficientemente los árboles locales esenciales.

5.3.4 Construcción del árbol paralelo

Después de la descomposición del dominio, cada PE tiene un conjunto disjunto de cuerpos. Para construir el árbol en paralelo, primero se construye el árbol utilizando los cuerpos contenidos en el propio PE. Entonces, el árbol se completa con los datos recibidos de los demás PEs. Así, todos los PEs pueden completar su árbol local esencial.

En la práctica, existen muy diversos métodos para construir la estructura árbol que precisa el algoritmo BH. El método que hemos utilizado es ir añadiendo cuerpo a cuerpo al árbol. Cada nuevo cuerpo que añadimos va recorriendo el árbol ya construido desde la celda raíz hasta la celda de nivel más alto que lo contiene, modificando el centro de masa de las celdas por las que va pasando. Si la celda final está vacía el cuerpo se añade directamente, pero si ya está ocupada, esta se divide en cuatro subceldas. Esto, tiene la ventaja de evitar una etapa adicional para calcular los centros de masa, como tienen que hacer en [114, 98]. De esta forma, nos evitamos recorrer el árbol después de creado para calcular los centros de masa de las celdas.

El método funciona como sigue. Supongamos que en una determinada

fase, la masa del CM de la celda u es $M_u = \sum_{i=1}^{U} m_i$ y que la posición del CM es $\vec{x}_u = \left(\sum_{i=1}^{U} m_i \cdot \vec{x}_i \right) / M_u$, donde U es el número de cuerpos en la celda. Si se añade un nuevo cuerpo a la celda, la nueva posición del centro de masa y su masa, se calcula por

$$\begin{aligned} M'_u &= M_u + m_{U+1} \\ \vec{x}'_u &= \frac{\vec{x}_u M_u + m_{U+1} \vec{x}_{U+1}}{M_u} \end{aligned} \tag{5.9}$$

donde m_{U+1} y $\vec{x}_{U+1}$ son la masa y la posición, respectivamente, del cuerpo incorporado a la celda.

Nosotros sólo tenemos que construir en cada PE el subconjunto del árbol necesario para evaluar la fuerza en el dominio del PE. Este subárbol lo hemos denominado árbol local esencial. La estrategia que utilizamos ha sido discutida en [88]. Esta estrategia consiste en adquirir los datos necesarios, no locales, en una fase de comunicación previa a la fase de computación. Tiene la ventaja de dejar el lazo principal de computación sin comunicación. Esta técnica requiere que sea posible determinar *a priori*, que datos pueden ser necesarios para cada PE. Con el criterio de la ecuación (5.3) se puede utilizar esta técnica.

Este paso es considerablemente rápido y fácil, cuando la descomposición del dominio está relacionada íntimamente con la topología del árbol (en contra de lo que ocurre con el método ORB usado en [114], por ejemplo). Con la recepción de todos los datos procedentes de los demás PEs, cada PE puede construir el resto de su árbol local esencial calculando la posición del centro de masa y su masa, por la ecuación (5.9). La dirección del PE origen de cada dato se pasa al PE destino, de forma que se incorpora a las celdas nuevas, creadas en el PE destino. De esta forma, el código es adaptable a una condición MAC distinta, en la cual no se puedan determinar a priori los datos locales esenciales. Una rutina que recorre el árbol puede determinar el PE que contiene las celdas hijas de una celda no local. Hemos intentado implementar la comunicación en este paso de una manera computacionalmente más inteligente, de forma que no sea necesaria una comunicación global, pero su complejidad ha tendido a eliminar su beneficio. Esto no elimina la posibilidad, de que en el futuro se encuentre un mejor método para este paso del algoritmo.

5.3.5 Formulación del algoritmo BH

Para formular los algoritmos BH sobre el multicomputador de topología malla, necesitaremos definir unos operadores adicionales a los vistos en el capítulo 1. En concreto, para definir las computaciones, definimos los siguientes operadores:

Definición 13 *El operador $T_{BH}(U)$ calcula el árbol BH de base r y el centro de masa de las celdas, por la expresión (5.9), para U cuerpos y con $r = 2^d$, donde d es la dimensión del espacio,*

$$T_{BH} = \prod_{l=0}^{U-1} \left(\prod_{i=0}^{r-1} g(z|i,l) \right) \tag{5.10}$$

y donde $g(z|i,l)$ es la función recursiva que recorre el árbol por preorden, $v|i = u$ y $|$ es el operador concatenación,

$$g(u,l) = \begin{cases} \prod_{i=0}^{r-1} g(u|i,l) & si \quad \exists\, u \quad AND \quad \vec{x_l} \in u \\ \\ T_{BH} \leftarrow T_{BH} \cup u & si \quad \nexists\, u \quad AND \quad \vec{x_l} \in u \end{cases} \tag{5.11}$$

Este operador construye el árbol utilizando los cuerpos contenidos en el propio PE, U. Cada nuevo cuerpo que añadimos va recorriendo el árbol ya construido, desde la celda raíz hasta la celda de nivel más alto, modificando el centro de masa (5.9) de las celdas por las que va pasando.

Definición 14 *El operador $T'_{BH}(X)$ calcula el árbol BH de base r para X cuerpos con $r = 2^d$, donde d es la dimensión del espacio,*

$$T'_{BH} = \prod_{l=0}^{X-1} \left(\prod_{i=0}^{r-1} h(z|i,l) \right) \tag{5.12}$$

y donde $z|i = v$ y $h(z|i,l)$ es la función recursiva que recorre el árbol por preorden,

$$h(v,l) = \begin{cases} \prod_{i=0}^{r-1} h(v|i,l) & si \quad l_v/d_{v,l} < \theta \quad AND \quad |v| \neq 1 \\ \\ T'_{BH} \leftarrow T'_{BH} \cup v & si \quad l_v/d_{v,l} \geq \theta \quad OR \quad |v| = 1 \end{cases} \tag{5.13}$$

siendo $|v|$ el número de cuerpos que pertenecen a la celda v.

El operador $T'_{BH}(X)$ construye, un árbol utilizando un conjunto de X cuerpos. Cada cuerpo va recorriendo el árbol local de cada celda desde la celda raíz. Si el centro de masa de una celda está lo suficientemente lejos del cuerpo, por el criterio MAC de Barnes y Hut, o la celda es una celda hoja, esa celda es añadida al árbol T'_{BH}. Si en contra, se realizarán estos mismos pasos con cada una de las subceldas hijas de esta celda.

Definición 15 *El operador $\vec{F}_l$ calcula la fuerza que se ejerce sobre el cuerpo l,*

$$\vec{F}_l = \sum_{i=0}^{r-1} f(z|i). \tag{5.14}$$

donde $f(i)$ es la función recursiva que recorre el árbol según

$$f(v) = \begin{cases} \sum_{i=0}^{r-1} f(v|i) & si \quad l_v/d_{v,l} < \theta \\ \\ F_{v,l} & si \quad l_v/d_{v,l} \geq \theta \ \ AND \ \ \vec{x}_u \neq \vec{x}_l \end{cases} \tag{5.15}$$

donde $F_{v,l}$ es la fuerza que la celda v ejerce sobre el cuerpo l, calculada mediante la ecuación (5.2). Finalmente, se actualiza la posición y velocidad del cuerpo l,

$$\begin{aligned} \vec{v}_l &= \vec{v}_l + \tfrac{\vec{F}_l}{m_l} \cdot dt \\ \\ \vec{x}_l &= \vec{x}_l + \vec{v}_l \cdot dt. \end{aligned} \tag{5.16}$$

Este operador calcula la fuerza que ejerce la red de celdas sobre un cuerpo. Después de calcular la fuerza que actúa sobre cada cuerpo, se actualiza la posición y la velocidad de cada cuerpo.

Entre los operadores que representan comunicaciones, definimos los siguientes,

Definición 16 *El operador $\mathcal{P}^N_{V\times W}$ reordena los N datos en un array unidimensional con el orden de Peano–Hilbert, y los proyecta sobre la malla de tamaño $V \times W$.*

Definición 17 *El operador $\mathcal{Y}^X_i$ envía un conjunto de X datos del PE k al PE $k + i \ mod \ P$, y recibe un conjunto de Xdatos del PE $k - i \ mod \ P$.*

El PE k calcula el árbol T'_{BH}. Después lo envía al PE $k - i \ mod \ P$ y recibe del PE $k + i \ mod \ P$ el árbol T''_{BH}, que lo incorpora a su árbol local esencial, $T_{BH} \leftarrow T_{BH} \cup T''_{BH}$.

El algoritmo BH puede expresarse de la siguiente forma,

Algoritmo 16 *El algoritmo BH base r de N cuerpos puede implementarse en paralelo por la siguiente cadena de operadores*

$$\mathcal{P}^N_{V\times W} \prod_{t=0}^{t_{final}} \left[T_{BH}(U) \left(\prod_{i=1}^{P-1} \mathcal{Y}_i^X T'_{BH}(X) \right) \left(\prod_{l=1}^{U} \vec{F}_l \right) \right] \tag{5.17}$$

Primero se proyectan los N cuerpos, $\mathcal{P}_{V\times W}$, sobre la malla utilizando la curva de llenando de espacio Peano-Hilbert y la proyección vector de tipo serpiente. Después de la descomposición del dominio, cada PE tiene un conjunto disjunto de cuerpos. En cada interacción se tiene que construir el árbol en paralelo, primero se construye el árbol utilizando los cuerpos contenidos por el PE, $T_{BH}(U)$. Entonces, el árbol se completa con los datos recibidos de los demás PEs, $T'_{BH}(X)$, después de enviar un conjunto de prueba de cuerpos del PE al resto de los PEs, $\mathcal{Y}_i^X$. Y por último, $\vec{F}_l$ calcula la fuerza sobre los cuerpos.

5.4 Implementación y evaluación

Trataremos de la implementación del algoritmo paralelo BH sobre el multicomputador T3E de Cray [93], usando el modelo de programación de pase de mensajes. Utilizamos el entorno de programación MPI [50]. El MPI trata de recopilar las mejores características de muchos sistemas de paso de mensajes, mejorándolos donde sea apropiado, y estandarizándolos. Su mayor ventaja es la portabilidad. En cuanto a la eficiencia, todo depende de la implementación de las librerías MPI que se haya realizado sobre cada computador paralelo particular. Y con respecto a la claridad, MPI ha sido diseñado para dar una definición completa y conveniente del modelo de paso de mensajes.

El desbalanceo de la carga que medimos por la ecuación (5.8) es la que determina en que iteración se tiene que realizar el rebalanceo. El PE que inicia el proceso de rebalanceo es el que tiene el coste más bajo, es decir, el destino. El PE fuente es el PE vecino del destino que está más cerca del PE con coste más alto. El PE que recibe la petición para transferir, tiene que

enviar la información más adecuada posible. El criterio que sigue es pasar aquel número de cuerpos j, que verifique la siguiente relación,

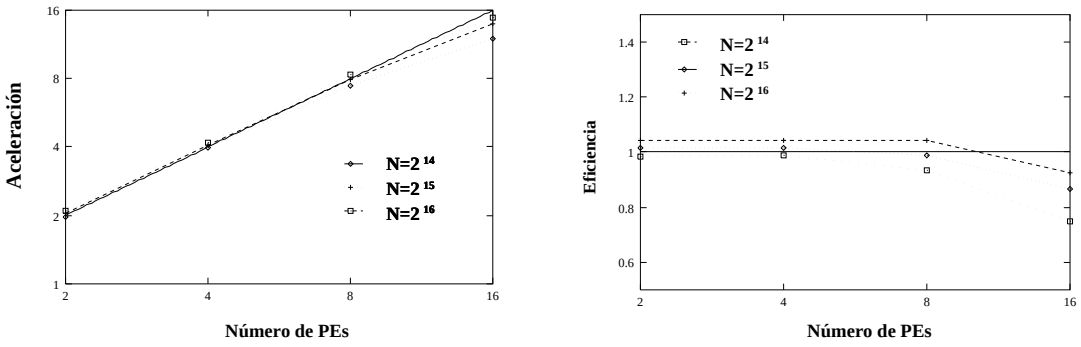
$$\sum_{i=0}^{j-1} w^t(i) > \frac{\max(W(q) - \mu)}{2} > \sum_{i=0}^{j-2} w^t(i),\ 0 \leq\ q\ <\ P \tag{5.18}$$

y que estén más próximas al PE destino siguiendo la curva de llenado del espacio. En este proceso de rebalanceo no influye que el número de PEs sea elevado, ya que al inicio del proceso el PE destino elige al PE fuente y luego se comunicarán independientemente del resto de PEs. Además, la evaluación de la carga de los PEs del sistema lo realizan concurrentemente todos los PEs. Cada vez que se cambia la localización de un cuerpo, todos los PEs tendrán que actualizar la función de mapeado, que en nuestro caso es equivalente a modificar el vector de la ecuación (5.6).

En un principio hemos implementado el algoritmo siguiendo la cadena de operadores con el que se representa (algoritmo 16). Posteriormente hemos realizado transformaciones del programa con el fin de explotar mejor el paralelismo inherente. Hemos cambiado el orden de las computaciones y aplicado una serie de técnicas de paralelismo, manteniendo la equivalencia del programa.

Abrir las vías de comunicación interprocesador es generalmente bastante costoso en tiempo. En nuestro caso el algoritmo presenta una comunicación densa: cada PE comunica en cada paso con el resto de PEs y con una cantidad variable de datos. Se ha aplicado la alternativa de almacenar en una cola las peticiones para ser enviada posteriormente (después que todas las peticiones a un PE han sido recopiladas). Mientras tanto, se determina qué otros datos son los que necesitan otros PEs. Esto reduce en gran medida el número total de mensajes enviados, y por tanto la paralelización asociada a las comunicaciones.

En la figura tabla 5.1, mostramos algunos resultados obtenidos en la simulación newtoniana de N cuerpos sobre el Cray T3E. El tiempo mostrado es el tiempo total sobre 100 iteraciones, e incluye la primera iteración que presenta un tiempo de ejecución anómalo, debido al desbalanceo inicial de la carga y que es corregido en interacciones posteriores, después de que se conozca el coste asociado a cada cuerpo. Hemos usado una aproximación monopolar para el cálculo de la fuerza por la expresión (5.1), y una serie de Taylor de primer orden para la traslación, que nos garantiza las ventajas de la expresión (5.9). Para mayor simplificación hemos considerado que N y P son potencias de 2. La figura 5.18.a, muestra la aceleración medida para



(a) (b)

Figura 5.18: Resultados de la implementación del algoritmo BH sobre el Cray T3E con $N = 2^{14}$, $N = 2^{15}$ y $N = 2^{16}$ datos. (a) Aceleración. (b) Eficiencia.

Tabla 5.1: Tiempo de ejecución (en segundos) medidos sobre el Cray.

	Número de cuerpos					
Número de PEs	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
1	25.83	41.12	91.44	201.13	442.39	968.31
2	10.23	22.19	47.52	102.34	217.34	463.62
4	5.29	11.34	23.87	50.90	108.75	232.58
8	3.41	6.47	12.77	26.87	55.98	115.92
16	4.16	5.76	9.58	16.80	31.89	65.54

simulaciones con $N = 2^{14}$, $N = 2^{15}$ y $N = 2^{16}$ cuerpos.

La figura 5.18.b muestra la eficiencia obtenida para los distintos casos. La eficiencia se acerca al valor ideal para sistemas de hasta 8 PEs. Para sistemas con un mayor número de PEs la eficiencia es menor, siendo la principal limitación, la etapa de construcción de los árboles locales esenciales en los cuales se precisa realizar una comunicación casi global. Por ejemplo, para un sistema de 16 PEs se consigue una aceleración de 14.78.

Conclusiones y principales aportaciones

En esta memoria hemos propuesto una metodología general para particionar y proyectar algoritmos *divide y vencerás* sobre computadores paralelos de topología malla y memoria distribuida. La metodología se ha aplicado a la paralelización de las versiones más utilizadas de la transformada rápida de Fourier, a los más significativos algoritmos de resolución de sistemas tridiagonales, y a algoritmos irregulares, como el algoritmo Barnes–Hut, del problema de los N cuerpos.

Las principales aportaciones de esta memoria son las siguientes:

- Hemos presentado una metodología para la paralelización de algoritmos divide–y–vencerás regulares, basada en una combinación de dos técnicas: la proyección vector y las permutaciones índice–dígito. El método empleado proporciona una descripción clara y precisa de los reordenamientos de datos que, en muchos casos, ayuda a simplificar los algoritmos. Cada una de las permutaciones índice–dígito definidas tiene una implementación directa sobre el multiprocesador.

- Hemos presentado la formulación unificada de un conjunto de algoritmos representativos de la estrategia divide–y–vencerás sobre un multicomputador de topología malla. En concreto, hemos estudiado diversas versiones de la transformada rápida de Fourier y varios algoritmos de resolución de sistemas tridiagonales. Los flujos de datos que se presentan son muy diversos: normal, extendido, salida dígito–inverso, árbol y árbol extendido. Por consiguiente, la metodología empleada es general, y podrá extenderse para formular otros algoritmos obtenidos empleando la estrategia divide–y–vencerás sobre computadores paralelos de topología malla.

- Hemos estudiado un caso especial del mapeado de grafos, el mapeado

de árboles r–arios completos sobre array lineal y malla, orientado a la implementación VLSI. En concreto, establecemos un algoritmo que particiona la malla en rutas de procesadores y distribuye el árbol entre estas rutas. En todos los casos, la distribución de los nodos del árbol entre los procesadores es balanceada y las comunicaciones son sólo entre procesadores vecinos. Esta metodología puede ser una alternativa a las más usuales, basadas en árbol en "H" o en baldosas.

- Por último, hemos abordado la partición y proyección de algoritmos divide y vencerás irregulares sobre multicomputadores de topología malla, tomando como ejemplo el algoritmo Barnes–Hut para el problema de los N cuerpos. El algoritmo paralelo requiere en cada iteración la construcción de un árbol local esencial en cada procesador, con numerosas comunicaciones, por lo que es necesario un especial cuidado en la distribución de los cuerpos sobre los procesadores. Además de las técnicas citadas, hemos empleado el *llenado del espacio a través de una curva* (*space–filling curve*), que nos permite mantener la localidad de los datos y reducir las comunicaciones. Además, hemos simplificado la etapa de cálculo de los centros de masa, que se realiza en paralelo a la construcción del árbol.

Como trabajo futuro nos planteamos adaptar las técnicas presentadas para que puedan ser utilizadas desde HPF. En concreto, pretendemos diseñar una librería de subrutinas basadas en las permutaciones índice–dígito. Esto nos permitirá implementar eficientemente los algoritmos divide–y–vencerás en HPF. Otra tarea que nos planteamos es la generalización de la metodología presentada a multicomputadores con nodos biprocesadores, como ejemplo el Origin2000.

Bibliografía

[1] K. Abramhson, N. Dadoun, and T. Przytycka. A simple parallel tree contraction algorithm. *J. Algorithms*, 10(2):287–302, 1989.

[2] J. C. Agüí and J. Jiménez. A binary tree implementation of a parallel distributed tridiagonal solver. *Parallel Computing*, 21:233–241, 1995.

[3] P. Amodio and L. Brugnano. The parallel QR factorization algorithm for tridiagonal linear systems. *Parallel Computing*, 21:1097–1110, 1995.

[4] M. Amor, F. Argüello, and J. López. A parallel N–body treecode on the AP1000. In *VII Parallel Computing Workshop, PCW97, Canberra, Australia*, 1997.

[5] M. Amor, F. Argüello, M. Martín, and D. H. Heras. Vectorización de algoritmos divide–y–vencerás. *Informática y Automática*, 29(3):28–37, 1996.

[6] M. Amor, D. Blanco, M. Martín, O. G. Plata, F. F. Rivera, and F. Argüello. Vectorization of the radix r self–sorting FFT. *Lecture Notes in Computer Science, Springer–Verlag*, 854:208–217, 1994.

[7] M. Amor, J. López, F. Argüello, and E. L. Zapata. Mapping tridiagonal systems algorithms onto mesh connected computers. *International Journal of High Speed Computing*, 1997. Pendiente de publicación.

[8] M. Amor, J. López, D. H. Heras, and F. Argüello. Mapping and scheduling of r–arys trees onto arrays and meshes. In *Parallel Computing ParCo'97, Bonn, Alemania, 16–19 Sept.*, 1997.

[9] M. Amor, J. López, M. Martín, and F. Argüello. Parallelization of cyclic reduction algorithms on multiprocessor system with mesh topology. In *Parallel Computing ParCo'97, Bonn, Alemania, 16–19 Sept.*, 1997.

[10] M. Amor, J. López, S. Romero, F. Argüello, and E. L. Zapata. Métodos de reducción cíclica en mallas de computadores. In *VII Jornadas de Paralelismo, Santiago*, :39–55 1996.

[11] M. Amor, M. Martín, D. B. Heras, V. Blanco, J. C. Cabaleiro, T. F. Pena, F. Argüello, and F. F. Rivera. A self–sorting FFT on the AP1000. In *PCW'95 London, England*, :161–180 1995.

[12] A. W. Appel. An efficient program for many–body simulation. *SIAM J. Sci. Stat. Comput.*, 6(1):85–103, 1985.

[13] F. Argüello. *Diseño de arquitecturas paralelas–segmentadas para las transformadas ortogonales.* PhD thesis, Universidad de Santiago de Compostela, 1992.

[14] F. Argüello, M. Amor, and E. L. Zapata. FFTs on mesh connected computers. *Parallel Computing*, 22:19–38, 1996.

[15] M. Ashworth and A. G. Lyne. A segmented FFT algorithm for vector computer. *Parallel Computing*, 6:217–224, 1988.

[16] J. Barnes and P. Hut. A hierachical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[17] M. J. Berger and S. H. Bokhari. A partitioning strategy for non–uniform problems across multiprocessors. *IEEE Transactions on Computers*, C–36(5):570–580, 1987.

[18] J. A. Board, Z. S. Hakura, W. D. Elliott, D. C. Gray, W. J. Blanke, and J. F. Leathrum. Scalable implementation fo multipole–accelerated algorithms for molecular dynamics. *SHPCC'94*, pages 87–94, 1994.

[19] S. H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, C-30(3):207–214, Marzo 1981.

[20] S.H. Bokhari. *Assignment Problems in Parallel and Distributed Computing.* Kluwer Academic Publisers, 1987.

[21] G. Brassard and P. Bratley. *Algorítmica. Concepción y análisis.* Masson, s.a., 1990.

[22] J. C. Cabaleiro, F. F. Rivera, O. G. Plata, and E. L. Zapata. A parallel algorithm for householder's tridiagonalization of a symmetric matrix. *Cybernetics and Systems*, 23:345–357, 1992.

[23] C. Calvin. Implementation of parallel FFT algorithms on distributed memory machines with a minimum overhead of communication. *Parallel Computing*, 22:1255–1279, 1996.

[24] M.J. Carey and C.D. Thomson. An efficient implementation of search trees on $\lceil \log N + 1 \rceil$ processors. *IEEE Transactions on Computers*, C-33(11):1038–1041, Nov. 1984.

[25] D. A. Carlson. Ultrahigh–performance FFTs for the CRAY–2 and CRAY Y–MP supercomputers. *The Journal of Supercomputing*, 6:107–116, 1992.

[26] A. Colbrook, E. A. Brewer, C.Ñ. Dellarocas, and W. E. Weihl. Algorithms for search trees on message–passing architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(2):97–108, 1996.

[27] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297–301, 1965.

[28] C. L. Cox and J. A. Knisley. A tridiagonal system solver for distributed memory parallel processor with vector nodes. *Journal of Parallel and Distributed Computing*, 13:325–331, 1991.

[29] P. P. N. De Groen. Base–p–cyclic reduction for tridiagonal system of equations. *Applied Numerical Mathematics*, 8:117–125, 1991.

[30] D. Dodson and S. Levin. A tricyclic tridiagonal equation solver. *SIAM J. Matrix Anal. Appl*, 13(4):1246–1254, Oct. 1992.

[31] A. Dubey, M. Zubair, and C. E. Grosch. A general purpose subroutine for fast fourier transform on a distribuited memory parallel machine. *Parallel Computing*, 20:1697–1710, 1994.

[32] P. Duhamel and H. Hollman. Split–radix FFT algorithm. *Electronic Letters*, 20(1):14–16, 1984.

[33] D.J. Evans. Parallel algorithms in numerical linear algebra. Report interno, Loughborough University of Technology, Loughborough, Leicestershire, UK, 1992.

[34] Ö. Eğecioğlu, Ç. K. Koç, and A.J. Laub. A recursive doubling algorithm for solution of tridiagonal system on hypercube multiprocessor. *J. of Computational and Applied Mathematics*, 27:95–108, 1985.

[35] Z. Fei, V. M. Pérez-García, and L. Vázquez. Numerical simulation of nonlinear schrödinger systems: A new conservative scheme. *Appl. Math. Comput.*, 80:1–13, 1995.

[36] P.M. Flanders. A unified approach to a class of data movements on an array processor. *IEEE Trans. Comput.*, C–31:809–819, 1982.

[37] M. J. Flynn. Some computers organizations and their effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.

[38] G. Fox, M. Johnson, G. Lyzenga, S Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Computers.* Englewood Cliffs, NJ. Prentice Hall, 1988.

[39] D. Fraser. Array permutation by index-digit permutation. *J. on the ACM*, 23(2):298–309, Abril 1976.

[40] Fujitsu Laboratories Ltd. *AP1000 Users Guide*, release 1.4 edition, Marzo 1994.

[41] D. P Fullagar, P. J. Quinn, C. J. Grillmair, J. K. Salmon, and M. S. Warren. N-body methods on MIMD supercomputers: Astrophysics on the intel touchstone delta. In *Fifth Australian Supercomputing Conference*, 1992.

[42] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness.* Freeman, San Francisco, 1979.

[43] D. Ghosal, A. Mukherjee, R. Thurimella, and Y. Yesha. Mapping task trees onto a linear array. *Proc. Int. Conf. on Parallel Processing*, I.629–I.633 1991.

[44] A. Gibbons and R. Ziani. The balanced binary tree technique on mesh-connected computers. *Information Processing letters*, 37(10):101–109, 1991.

[45] G. H Golub and Ch. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

[46] D. Gordon. Efficient embeddings of binary trees in VLSI arrays. *IEEE Transactions on Computers*, C-36(9):1009–1018, Sept. 1987.

[47] L. Greengard. *The Rapid Evaluation of Potential Fields in Particles Systems.* PhD thesis, Yale University, 1987.

[48] L. Greengard and W. D. Gropp. A parallel version of the fast multipole method. *Computers Math. Applic.*, 20(7):63–71, 1990.

[49] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *Journal Comput. Phys.*, 73:325–348, 1987.

[50] W. Gropp, E. Lusk, and A. Skjellum. *USING MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, 1994.

[51] S. W. Hammond. *Mapping Unstructured Grid Computations to Massively Parallel Computers*. PhD thesis, Rensselaer Polytechnic Institute, Agosto 1992.

[52] N. Hartsfield and G. Ringel. *Pearls in Graph Theory: a Comprehensive Approach*. Academic Press, 1994.

[53] M. Hegland. An implementation of multiple and multi-variate Fourier transforms on vector processors. *SIAM J. Scie. Comp.*, 16(2):271–288, 1995.

[54] B. Hendrickson and S. Plimpton. Parallel many-body simulation without all-to-all communication. *Journal of Parallel and Distributed Computing*, 27:15–25, 1995.

[55] R. W. Hockney and C. R. Jesshope. *Parallel Computers*. Adam Hilger, 1988.

[56] S. J. Hockney and J. W. Eastwood. *Computer Simulation using Particles*. Hilger, New York, 1988.

[57] E. Horowitz and A. Zorat. The binary tree as an interconnection network: Applications to multiprocessor systems and VLSI. *IEEE Transactions on Computers*, C-30(4):247–253, Abril 1981.

[58] Y. Hu. *Efficient Data Parallel Implementations of Hierarchical N–body Algorithms*. PhD thesis, Harvard University, 1997.

[59] Y. Ch. Hu, S. L. Johnsson, and S. H. Teng. High performance fortran for highly problems. *Proceedings of the 1997 International Conference on Parallel Processing*, pages 137–144, Ago. 11–15 1997. IEEE Computer Society.

[60] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw Hill, 1993.

[61] K. Hwang and F. A. Briggs. *Arquitectura de computadores y procesamiento paralelo.* McGraw–Hill, 1987.

[62] A. Jorba, J. L. Larriba-Pey, and J. J. Navarro. A proof for the accuracy of OPM. Technical Report No. RR–93-09, CEPBA, Sept. 1992.

[63] A. Krechel, H. J. Plum, and K. Stüben. Parallelization and vectorization aspects of the solution of tridiagonal linear systems. *Parallel Computing*, 14:31–49, 1990.

[64] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introdution to Parallel Computing. Design and Analysis of algorithms.* The Benjamin/Cummings Publishining Company, INC., 1994.

[65] J. L. Larriba-Pey, A. Jorba, and J. L. Navarro. OPM: A parallel method to solve banded systems of equations. Technical Report No. RR–93-09, CEPBA, Sept. 1992.

[66] F. C. Lin and K. L. Chung. A cost–optimal parallel tridiagonal solver. *Parallel Computing*, 15:189–199, 1990.

[67] W.-Y. Lin and Chen Ch.-L. A parallel algorithm for solving tridiagonal linear systems on distributed–memory multiprocessors. *International Journal of High Speed Computing*, 6(3):375–386, 1994.

[68] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[69] P. Liu and J. J. Wu. A framework for parallel tree–based scientific simulations. *Proceedings of the 1997 International Conference on Parallel Processing*, pages 137–144, Ago. 11–15 1997. IEEE Computer Society.

[70] J. López, O. Plata, F. Argüello, and E. L. Zapata. Unified framework for the parallelization of divide and conquer based tridiagonal systems. *Parallel Computing*, 23(6):667–686, Junio 1997.

[71] J. López and E. L. Zapata. Unified architecture for divide and conquer based tridiagonal system solvers. *IEEE Transactions on Computers*, 43:1413–1425, Dic. 1994.

[72] I. Martín. *Simulación de ecuaciones de onda no lineales sobre arquitecturas paralelas.* PhD thesis, Departamento de Informática y automática. Universidad Complutense de Madrid, 1995.

[73] I. Martín and F. Tirado. Simulation of some nonlinear schrödinger systems on a distributed memory parallel computer. In *IV Jornadas de Paralelismo*, 27–29 Sept. 1993.

[74] N. Mattor, T. J. Williams, and D. W. Hewett. Algorithm for solving tridiagonal matrix problems in parallel. *Parallel Computing*, 21:1769–1782, 1995.

[75] D. I. Moldovan. *Parallel Processing. From Aplications to Systems.* Morgan Kaufman Publishers. San Mateo, California, 1993.

[76] S. M. Müller and D. Scheerer. A method to parallelize tridiagonal solvers. *Parallel Computing*, 17:181–188, 1991.

[77] D. M. Nicol and J. H. Saltz. An analysis of scatter decomposition. *IEEE Transactions on Computers*, 39(11):1337–1345, 1990.

[78] A. Norton and A. J. Silberger. Parallelization and performance analysis of the Cooley–Tukey FFT algorithm for shared–memory architectures. *IEEE Transactions on Computers*, C–36(5):581–591, Mayo 1987.

[79] C. Ou, S. Ranka, and G. Fox. Fast and parallel mapping algorithms for irregular problems. *Journal of Supercomputing*, 10:119–140, 1996.

[80] M. C. Pease. An adaptation of the fast fourier transform for parallel processing. *J. Ass. Comput. Mach.*, 15:252–264, 1968.

[81] T. F. Pena. *Simulación 3D de dispositvos semiconductores en sistemas multiprocesador.* PhD thesis, Dept. de Electrónica y Computación. Universidad de Santiago de Compostela, Mayo 1994.

[82] T. F. Pena, E. L. Zapata, and D. J. Evans. Finite element simulation of semiconductor devices on multiprocessor computers. *Parallel Computing*, 1994.

[83] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non–uniform structured workloads with spacefilling curves. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):288–299, Marzo 1996.

[84] L.R. Rabiner and B. Gold. *Theory and Aplication of Digital Signal Processing.* Prentice Hall, Englewood Cliffs, NJ, 1975.

[85] F. F. Rivera. *Partición y Proyección de Algoritmos en Computadores Hipercubo: Reconocimiento de Formas.* PhD thesis, Departamento de Electrónica y Computación, Universidad de Santiago de Compostela, Dic. 1989.

[86] L. F. Romero and E. L. Zapata. Data distribution for sparse matrix vector multiplication. *Parallel Computing*, 21:583–605, 1995.

[87] P. Sadayappan and F. Ercal. Nearest–neighbor mapping of finite element graphs onto processor meshes. *IEEE Trans. on Computers*, C–36(12):1408–1442, 1987.

[88] J. K. Salmon. *Parallel Hierarchical N–Body Methods.* PhD thesis, California Institute of Technology, Pasadena, California, 1991.

[89] J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. *Journal Comp. Phys.*, 111(1):136–155, 1994.

[90] H. Samet. The quadtree and related hierarchical data structures. *Computing surveys*, 16(2):187–260, 1984.

[91] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison–Wesley Publishing Company, 1989.

[92] K. E. Schmidt and M. A. Lee. Implementing the fast multipole method in tree dimensions. *J. Stat. Phys.*, 63(5/6):1223–1235, 1991.

[93] S. L. Scott. Synchronization and communication in the T3E multiprocessor. In *7th Conference on Architectural Support for Programming languages and Operating Systems ASPLOS VII*, Cambridge, Massachusetts, :26–27 1996.

[94] Ch. N. Sekharan, V. Goel, and R. Sridhar. Load balacing methods for ray tracing and binary tree computing using PVM. *Parallel Computing*, 21:1963–1978, 1995.

[95] R. Shankar and S. Ranka. Hypercube algorithms for quadtree operations. *Journal of Pattern Recognition*, 25(7):741–747, Sept. 1992.

[96] R. Shankar and S. Ranka. Computer vision algorithms for sparse images. *Journal of Pattern Recognition*, 26:1511–1519, Oct. 1993.

[97] E. Shimon. *Graph Algorithms.* Computer Science Press, Ellis Horowitz, 1979.

[98] J. P. Singh. *Parallel Hierarchical N-body Methods and their implications for multiprocessor.* PhD thesis, Stanford University, 1993.

[99] J. P. Singh, J. L. Hennessy, and A. Gupta. Implications of hierarchical N-body methods for multiprocessor architectures. *ACM Transactions on Computer Systems*, 13(2):141–202, 1995.

[100] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. L. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27:118–141, 1995.

[101] G. Spaletta and D. J. Evans. The parallel recursive decoupling algorithm for solving tridiagonal linear systems. *Parallel Computing*, 19:563–576, 1993.

[102] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE–3(1):85–93, Enero 1977.

[103] H. S. Stone. Program assignment in three processor systems and tricutset partitioning of graphs. Technical Report ECE-CS-77-7, Departament of Electrical and Computer Engineering, University of Massachusetts, 1977.

[104] H. Sun, X. S. Zhang and L. M. Ni. Efficient tridiagonal solvers on multicomputers. *IEEE Transactions on Computers*, 41(3):286–296, 1992.

[105] P. N. Swarztrauber. FFT algorithms for vector computers. *Parallel Computing*, 1:45–63, 1984.

[106] T. R. Taha and M. J. Ablowitz. Analytical and numerical aspects of certain nonlinear evolution equations. *Journal of Computational Physics*, 55:203–240, 1984.

[107] C. Thompson and H. Kung. Sorting on a mesh–connected parallel computer. *Communicatios of the ACM*, 20:263–271, 1977.

[108] C. Tong and P. N. Swarztrauber. Ordered fast fourier transforms on a massively parallel hypercube multiprocessor. *Journal of Parallel and Distributed Computing*, 12:50–59, 1991.

[109] J. J. Tsay. Mapping tree-structured computations onto mesh-connected arrays of processors. In *Proceedings of the fourth Symposium on Parallel and Distributed. Proceedings'92 DEC 1-4, Arlington, Texas*, :77–84 1992.

[110] A. S. Wagner. Embedding arbitrary binary trees in a hypercube. *Journal of Parallel and Distributed Computing*, 7:503–520, 1989.

[111] A. S. Wagner. Embedding the complete tree in the hypercube. *Journal of Parallel and Distributed Computing*, 20:241–247, 1994.

[112] H. H Wang. A parallel method for tridiagonal equations. *ACM Trans. on Math. Software*, 7:170–183, 1981.

[113] X. Wang and Z. G. Mou. A divide and conquer method of solving tridiagonal system on hypercube massively parallel computers. *Proceedings of the third IEEE Symposium of Parallel and Distributed Processing*, pages 810–817, 1991.

[114] M. S. Warren and J. K. Salmon. Astrophysical N–body simulations using hierarchical tree data structure. In IEEE Computer Society, editor, *Proceedings of Supercomputing'92*, Minneapolis, :570–576 1992.

[115] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In IEEE Computer Society, editor, *Proceedings of Supercomputing'93*, :12–21 1993.

[116] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 67:266–290, 1995.

[117] A. Y. Wu. Embedding of tree networks into hypercubes. *Journal of Parallel and Distributed Computing*, 2:238–249, 1985.

[118] H. Y. Youn and A. D Singh. On implementing large binary tree architectures in VLSI and WSI. *IEEE Transactions on Computers*, C-38(4):526–537, Abril 1989.

[119] E. L. Zapata and F. Argüello. Application specific architecture for fast transforms based on the successive doubling method. *IEEE Trans. Signal Processing*, 41:1476–1481, 1993.

[120] E. L. Zapata, F. Argüello, and F. F. Rivera. Multidimensional fast hartley transform into SIMD hypercubes. *Journal on Microprocessing and Microprograming*, 29(2):121–134, 1990.

[121] E. L. Zapata, J. A. Lamas, F. F. Rivera, and O. G. Plata. Modified Gram–Schmidt QR factorization on hypercube SIMD computers. *Parallel and Distributed Computing*, 12(1):60–69, 1991.

[122] F. Zhao and S. L. Johnsson. The parallel multipole method on the connection machine. *SIAM J. Sci. Stat. Comp.*, 12:1420–1437, Nov. 1991.