Departamento de Computación
Facultad de Ciencias Exactas y Naturales
-Universidad de Buenos Aires-

Tesis de Licenciatura

"Estudio del paralelismo en programas secuenciales numéricos"

Alumnos:

Cerro, Carlos Fabián	L.U. 2581/84
Galitó, Marcelo Fabián	L.U. 1396/83
Zuccaro, Mónica Beatriz	L.U. 0446/84

Directora:

Borensztejn, Patricia Miriam

Buenos Aires, Marzo de 1996

Contenido

Abstract

Processing in parallel represents an important area to study from different approaches.

The great quantity of sequential numeric programs that are already in use, and the possibility of extracting parallelism from them, motivated the development of the present work.

The same studies the inherent parallelism of 1600 loops, of sequential numerical Fortran programs. We obtained an important quantity of statistics of parallelism contained in the innermost loops of them, considering also the execution time of the different statements. We have also executed some of these programs, belonging to the Perfect Benchmarks suite, with standard input data and we have measured the execution time of the loops in order to compare their real time results with the evaluated parallelism.

We have studied this information, we have classified and we have established the behaviour of them.

Resumen

El procesamiento paralelo representa un importante campo en donde se combinan distintos enfoques para estudiarlo.

La inmensa cantidad de programas secuenciales numéricos existentes, y la posibilidad de extraer paralelismo en sus bucles, nos motivó para la realización de este trabajo.

El mismo estudia el paralelismo implícito de más de 1600 bucles en programas secuenciales numéricos escritos en lenguaje Fortran. Obtuvimos una importante cantidad de estadísticas sobre el paralelismo contenido en los bucles más internos de estos programas, teniendo en cuenta el tiempo estimado de ejecución de las distintas instrucciones. A su vez se ejecutaron algunos de estos programas, pertenecientes a la suite Perfect Benchmarks, con entradas de datos standard y se efectuó la medición del tiempo de ejecución de esos bucles, lo que nos permitió comparar su utilización real con su grado de paralelismo.

El estudio de la información producida nos permitió clasificar los distintos tipos de bucles y establecer el comportamiento del paralelismo de los mismos.

1-INTRODUCCION

INTRODUCCIÓN

Se entiende por ejecución paralela a la ejecución **simultánea** de cálculos u operaciones iguales o diferentes, sobre varios conjuntos de datos, de igual o distinto tipo, por parte de dos o más unidades funcionales.

El estudio del paralelismo genera distintos campos de investigación que dependen básicamente del a) tipo de arquitectura, del b) nivel de procesamiento y del c) tipo programación de estos sistemas.

a) Las distintas arquitecturas de procesadores pueden dividirse en:

* Sistemas de segmentación encauzada o pipelining.
* Sistemas matriciales.
* Sistemas multiprocesadores.

Tanto Flynn (1966) como Feng (1972) y Händler (1977) han presentado distintos esquemas de clasificación de las arquitecturas que dependen de los flujos de instrucción y datos (SISD, SIMD, MISD y MIMD) -Flynn-, de la máxima cantidad de dígitos binarios procesados por unidad de tiempo -Feng , y del número de procesadores, unidades aritmético-lógicas y etapas de cauce de cada una de ellas -Händler- [HB88].

b) Los sucesos concurrentes pueden clasificarse por su nivel de procesamiento en un sistema. Es decir, si encontramos paralelismo entre programas o trabajos, entre procedimientos o tareas, entre instrucciones de un programa, o, al nivel más bajo, dentro mismo de una instrucción. También suelen llamarse a estos paralelismos como grueso, medio o fino. Estos distintos niveles se encaran de distintas maneras: en forma algorítmica a nivel de programas o frecuentemente a través del hardware en el nivel más bajo.

c) Desde el punto de vista del tipo de programación podemos expresar el paralelismo de manera explícita o implícita. El paralelismo **explícito** es aquel que está expresado y generado desde su creación en un código paralelo por los propios usuarios y está estrechamente relacionado con su aplicación y su soporte de hardware. El paralelismo **implícito** es aquel que está contenido en una aplicación creada inicialmente en código secuencial para ser ejecutada en

un computador secuencial y que mediante técnicas de paralelización pueda ser ejecutado posteriormente en una máquina paralela. Este paralelismo implícito depende de la estructura interna del problema.

La programación explícita de los sistemas que soportan paralelismo se desarrolla con la extensión de lenguajes secuenciales existentes agregando primitivas que soporten operaciones vectoriales y concurrencia en un algoritmo, como por ejemplo el CEDAR FORTRAN u otros diseñados específicamente (OCCAM).

Otro enfoque a la hora de programar estos sistemas consiste en la paralelización de programas secuenciales existentes a través de un proceso de compilación que, por medio de un análisis de las dependencias entre las distintas instrucciones, explote el paralelismo implícito de estas aplicaciones programadas para sistemas secuenciales.

Este paralelismo implícito se obtiene a través de la paralelización **automática**. Este método consiste en detectar las dependencias entre las sentencias de un programa secuencial de modo de poder identificar zonas de ejecución potencialmente paralelas y crear una nueva versión paralelizada semánticamente idéntica .

También existen herramientas de programación paralela **interactiva**. Estas herramientas requieren del usuario para guiar al compilador durante el curso de la paralelización en cada sección del programa proveyendo facilidades para analizar dependencias. En algunos casos las herramientas interactivas pueden producir una mejor respuesta pues ellas ayudan al usuario a encontrar y eliminar falsas dependencias [CP91].

La detección automática del paralelismo implícito ha producido un gran número de compiladores reestructuradores de código entre los cuales mencionamos al PARAFRASE II desarrollado en el Center for Supercomputing Research & Development (CSRD) de la universidad de Illinois en Urbana-Champaign [PGH+89], el PFC desarrollado en 1979 por K. Kennedy en la Universidad de Rice, Houston y al SUPERB desarrollado en la Universidad de Bonn por H. Zima entre 1985 y 1989 [ZCh90].

PRESENTACIÓN DEL TRABAJO

Con el objeto de obtener información para establecer el beneficio de paralelizar los bucles más internos de programas secuenciales numéricos y aprovechando que disponíamos de programas Fortran, hemos efectuado mediciones del grado de paralelismo de dichos bucles, teniendo en cuenta el tiempo de ejecución estimado de cada instrucción. Además hemos comparado estas medidas con la utilización real de estos bucles en la ejecución de los programas para llegar a mejores conclusiones.

Este trabajo está dividido en 2 partes.

La primera corresponde a la evaluación del grado de paralelismo (paralelismo implícito) de los bucles de programas secuenciales numéricos. Estos bucles pueden ser de una o más dimensiones, pero solamente tienen sentencias ejecutables en el bucle más interno. A dichos bucles los llamaremos perfectamente anidados (bpa). Para tal fin hemos implementado un programa que mide el grado de paralelismo de estos bucles. Esta evaluación también es conocida como estática e incluye la influencia del tiempo de ejecución estimado de las sentencias. Se utilizaron 12 programas secuenciales numéricos Fortran de los cuales 10 pertenecen al conjunto de programas conocidos como Perfect Benchmark [P89] más dos conjuntos de rutinas llamadas Linpack (benchmarks de punto flotante escrito por Jack Dongarra en el Argonne National Laboratory.-Librería de subrutinas de álgebra lineal-) y Blas (librerías de Álgebra Lineal - Basic Linear Álgebra Subroutines-).

Los grafos que tienen paralelismo implícito conocido poseen las siguientes características: no tienen sentencias condicionales, las distancias de dependencias son constantes y no incluyen sentencias de entrada/salida. Estos grafos, 1136, representan el 69,39% de los 1637 que hay en todos los programas estudiados.

La segunda parte corresponde a la medición del tiempo de utilización de esos bucles en el contexto de la ejecución del programa con una entrada de datos standard. Para esto se utilizó únicamente el conjunto de los 10 programas secuenciales numéricos Perfect Benchmarks.

El trabajo está organizado de la siguiente manera:

En el capítulo 2 se exponen una serie de definiciones y conceptos básicos sobre la representación de las relaciones de dependencias entre sentencias de un bucle y las diversas caracterizaciones del paralelismo.

En el capítulo 3 explicaremos las herramientas teóricas que utilizamos para determinar el paralelismo tanto en el caso de un bucle simple, como en el caso de varios bucles anidados.

En el capítulo 4 describiremos la implementación de la evaluación del paralelismo realizada sobre los bucles de un conjunto de programas Fortran sin tener en cuenta el tiempo estimado de las sentencias y teniendo en cuenta ese tiempo.

Para tener una idea de la influencia real de los bucles en cada programa analizado, hemos medido el tiempo que cada bucle insume para ejecutarse y de esta manera relativizar su paralelismo en función de su "importancia" dentro del programa. Esto es considerado en el capítulo 5.

En el capítulo 6 hacemos un análisis de los resultados y se compara el grado de paralelismo con la medición del tiempo de ejecución de los bucles.

La paralelización de programas secuenciales numéricos constituye un importante aprovechamiento de una innumerable cantidad de software de alto grado de procesamiento numérico. Obtener medidas del grado de paralelismo en tiempo de compilación y establecer el beneficio de paralelizar programas secuenciales es muy importante para decidir la conveniencia de paralelizar o no y cual es la mejor arquitectura para procesar.

2-FUNDAMENTOS BASICOS

2.1 Definiciones básicas.

2.2 Algunas caracterizaciones del paralelismo dinámico de los programas.

2.1 Definiciones básicas.

Los compiladores de reestructuración de código se basan en un análisis detallado de las dependencias existentes entre las distintas sentencias del programa. Esto, que es realizado a nivel de instrucciones, determinará qué orden de ejecución condiciona la paralelización de esa porción de código.

Dependencias entre sentencias.

Existen dos tipos básicos de dependencias: las de datos y las de control. Las dependencias de control se producen cuando la ejecución de una sentencia condiciona la ejecución de una o varias sentencias que dependen del resultado de la evaluación de dicha condición (Fig. 1).

IF (S_1)
THEN S_2
ELSE S_3

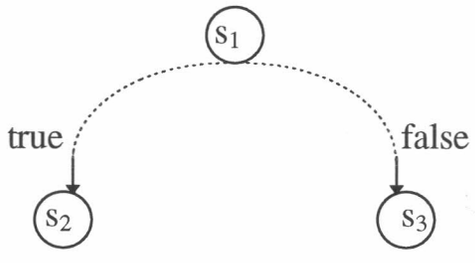


Fig. 1

Las dependencias de datos se dan cuando dadas dos sentencias Si y Sj, se tiene que [ZCh90]:

I. el orden de ejecución de Si precede a Sj

II. la intersección del conjunto de variables de escritura de Si y el conjunto de variables de lectura de Sj es no vacío.

    o

    la intersección del conjunto de variables de lectura de Si y el conjunto de variables de escritura de Sj es no vacío.

    o

    la intersección del conjunto de escritura de ambos es no vacío.

III. no hay sentencia Sk que tenga un orden de ejecución entre Si y Sj y que cumpla alguna de las condiciones del punto II.

De acuerdo a la condición que cumplan en el punto II las dependencias de datos se denominan de flujo de datos, antidependencia y dependencias de salida respectivamente.

A = B + C D = A * sqrt(C)	dependencia de flujo de datos respecto a A.
A = B + C B = D + E	antidependencia respecto a B.
A = B + C A = D + E	dependencia de salida respecto a A.

Las dos últimas pueden ser descartadas con algunas técnicas de optimización de código como renombre de variables, expansión escalar o refinamiento de nodos [KKP+81][PW86], por lo que la única dependencia propia del algoritmo es la primera. La representación de una dependencia se establece mediante un arco asociado a las dos sentencias (Fig. 2).

Fig.2

Cuando una dependencia aparece entre instanciaciones de sentencias en iteraciones de un bucle se le asigna una **distancia** que es el número de iteraciones del bucle que cubre la dependencia (Fig. 3). Por ejemplo

```
DO I= 1,3.
      A(I+2) = .....        S1
      B(I) = A(I)           S2
ENDDO
```

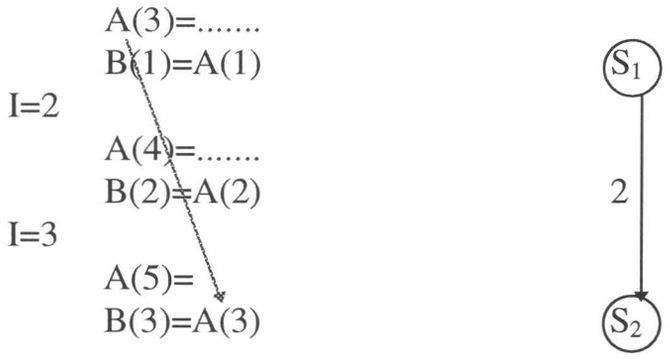


Fig. 3

En la iteración I=1 , A(I+2) de la sentencia S_1 es A(3), que es a su vez el valor que usa la S_2 en la iteración I=3.

Grafo de Dependencias

Las dependencias entre sentencias de un bucle se pueden representar mediante un **Grafo de Dependencias**. Un grafo de dependencias es un conjunto de nodos y un conjunto de arcos que expresan las relaciones de dependencias que establecen un orden parcial de ejecución de dichas sentencias en el bucle. Tenemos el conjunto $V=\{S_1,....,S_n\}$ de nodos, donde S_i es una sentencia, con el conjunto $E=\{D_{ij}/S_i,S_j \in V\}$ de arcos que representan las

dependencias. Cada arco del grafo D_{ij} lleva asociada la distancia que define la relación de dependencia que representa [AL89].

Cadenas y Recurrencias

Dada una sentencia S_i y otra S_j existe una **Cadena C_{ij}** si existen arcos distintos del grafo tal que generan una dependencia entre S_i y S_j.

$$C_{ij}=\{D_{ik},D_{kl},.....,D_{mj}\}$$

Se llama **peso $W(C_{ij})$** de la cadena a la suma de las distancias de los arcos de la cadena.

$$W(C_{ij})= \sum_{D_{kl} \in C_{ij}} D_{kl}$$

La **longitud $L(C_{ij})$** queda definida por la cantidad de nodos que intervienen en la cadena (Fig. 4).

Fig. 4

$C_{13} = \{D_{12},D_{23}\}$
$D_{12} = 2,\ D_{23} = 3$
$W(C_{13}) = 5$
$L(C_{13}) = 3$

Se denomina **recurrencia** a una cadena cerrada C_{ii} en el grafo tal que cada sentencia aparezca solo una vez. Si la recurrencia abarca todos los nodos entonces se denomina **Hamiltoniana** (Fig. 5). Si el conjunto de recurrencias es

vacío el grafo se denomina **Grafo de Dependencias Acíclico** (GDA), de lo contrario es **Grafo de Dependencias Cíclico** (GDC).

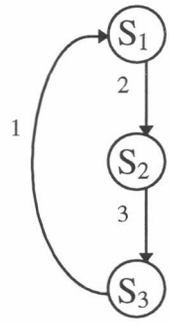


Fig. 5

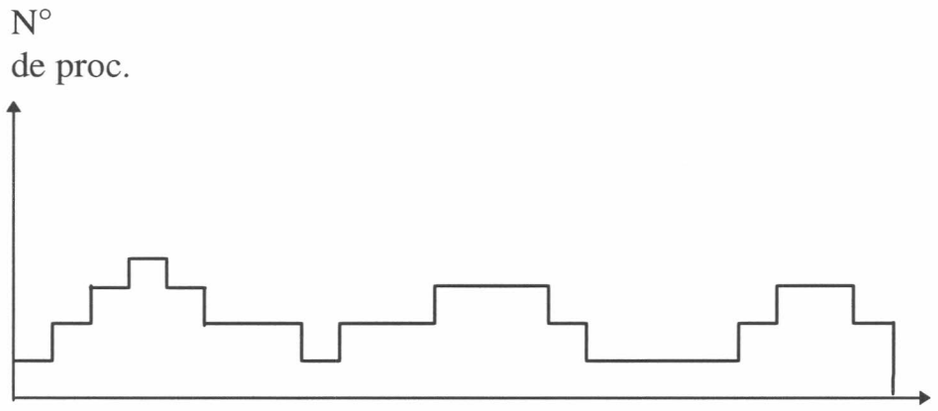
2.2 Algunas caracterizaciones del paralelismo dinámico de los programas.

Algunos parámetros son [S89] :

La **Fracción Secuencial,** propuesto por Amdahl, que es la suma total de los tiempos de ejecución de aquellas partes del programa que no pueden ser ejecutadas en paralelo con otras.

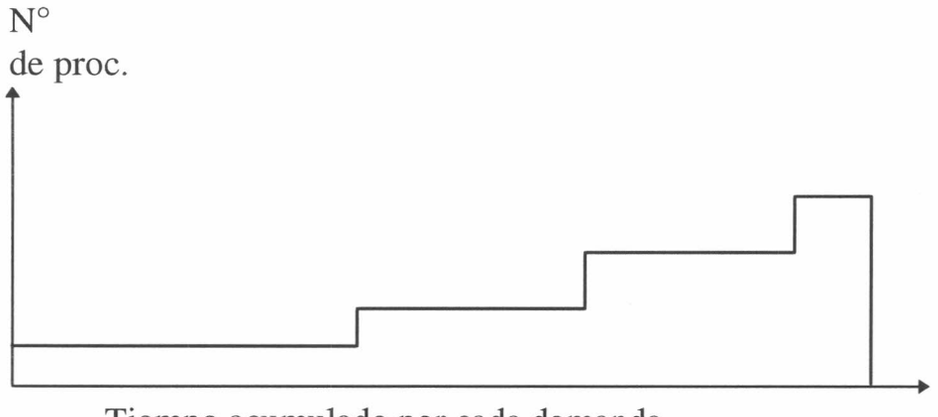
Los paralelismos **mínimo** y **máximo** que son las cantidades mínimas y máximas de procesadores activos durante la ejecución de la aplicación respectivamente.

También tenemos el **paralelismo profile** que, como un histograma en el tiempo, indica la variación de procesadores activos durante la ejecución del programa.



_ Profile

Si reordenamos el profile podemos determinar la proporción de tiempo de ejecución de los distintos grados de paralelismo de la aplicación, dándonos un gráfico ordenado por demanda de procesadores. Esto se denomina **Shape**.



_ Shape

3-HERRAMIENTAS TEORICAS DE LA TESIS

3.1 Evaluación del grado de paralelismo de bucles de programas secuenciales numéricos

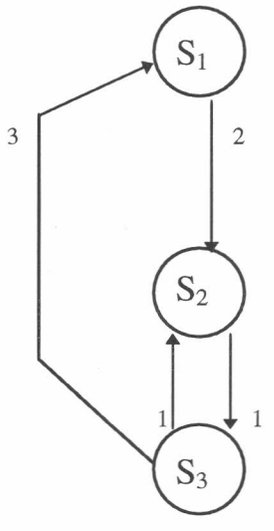
Las dependencias entre sentencias de un bucle imponen un orden de ejecución entre las mismas que limitan su ejecución en paralelo. Si estas dependencias forman un ciclo en el grafo de dependencias, ese orden de ejecución se repetirá sucesivamente. Observemos a través de un ejemplo como se manifiesta este comportamiento

Ejemplo 3.1

```
DO I=1,10

    A(I)=C(I-3)*2
    B(I)=A(I-2)+C(I-1)-2
    C(I)=B(I-1)

ENDDO
```



Este grafo posee dos recurrencias, que llamamos R_1 y R_2, que estan formadas por las dependencias d_{12},d_{23},d_{31} y d_{23},d_{32} respectivamente.

El desarrollo de la ejecución tomando la recurrencia R_1 por separado, tomando las 10 iteraciones, es

$R_1 = <d_{12},d_{23},d_{31}>$

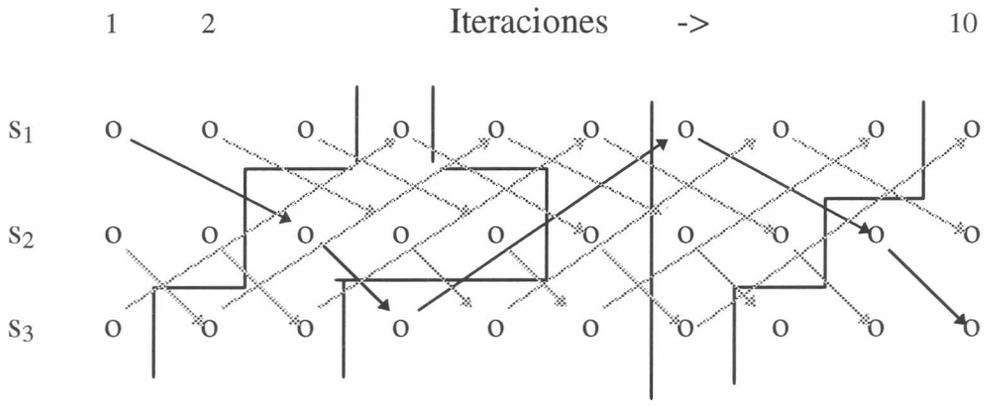


Fig. 3.1

El orden de ejecución impuesto por las dependencias permite que un grupo de sentencias (6) puedan ejecutarse en paralelo y se repita este patrón de ejecución tantas veces hasta completar el límite de iteración del bucle (fig. 3.1). Un ciclo de la recurrencia cubre 6 iteraciones del bucle ejecutándose en 3 instantes de tiempo, en lugar de las 18 ejecuciones que insumiría en forma secuencial.

$R_2=<d_{23},d_{32}>$

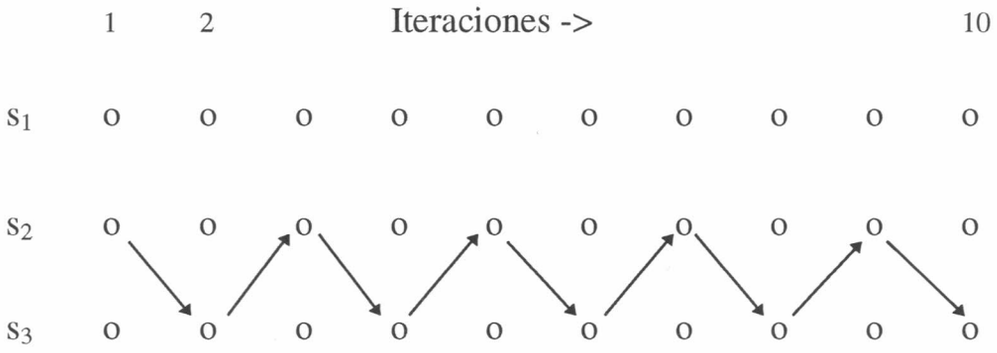


Fig. 3.2

En esta recurrencia vemos que para cubrir 2 iteraciones insume 2 instantes de tiempo (Fig. 3.2). Esto necesariamente se repetirá muchas mas veces que en el caso anterior dentro de las 10 iteraciones. Por lo tanto de ambas recurrencias la que más restringe las ejecuciones en paralelo es R_2.

Notamos que habiendo varias recurrencias, el paralelismo del grafo viene limitado por la recurrencia que más nodos debe ejecutar secuencialmente a través de todas las iteraciones del bucle, lo que implica que menos instrucciones puedan ejecutarse en paralelo.

La longitud de la cadena formada por la recurrencia R_1 en las 10 iteraciones del bucle del ejemplo 3.1 es 6, mientras que la longitud de la cadena formada por la recurrencia R_2 es 10.

De lo anterior se deriva que el paralelismo promedio puede ser expresado como el cociente entre el tiempo de ejecutar la versión secuencial del bucle y el tiempo de ejecutar la cadena de dependencias mas larga a través de todo el bucle [ALT93]. Esto lo expresamos como

$$Ts = n * N \qquad Tp = |R| * \frac{N}{w(R)}$$

donde

Ts = Tiempo de ejecución del bucle en forma secuencial.
Tp = Tiempo de ejecución de la cadena de dependencias más larga a través de todo el bucle.
|R| = Cantidad de nodos de la recurrencia
n = Cantidad de sentencias del bucle
N = número de iteraciones del bucle.
w(R) = peso de la recurrencia.
N/w(R) = Cantidad de veces que se repite la recurrencia en todo el bucle.

$$//prom.= \frac{Ts}{Tp} = \frac{n * N}{|R| * \frac{N}{w(R)}} = \frac{n * w(R)}{|R|}$$

La recurrencia que limita es la que da mayor Tp, el máx |R|/w(R) o el mín w(R)/|R| , entonces

$$//prom = n * \min_{R_i \in B} \left(\frac{w(R_i)}{|R_i|} \right)$$

B = Conjunto de recurrencias del bucle.

Como 2 casos particulares tenemos cuando el grafo sólo tiene una recurrencia hamiltoniana y cuando el grafo es acíclico. En el primer caso |R| es igual a n, entonces

$$//prom = w(R)$$

En el segundo caso no existen recurrencias , pero como vimos, la cadena de dependencias más larga es la que limita el paralelismo, y la cantidad de veces que aparece como "recurrencia" es 1, entonces

$$//prom = \frac{n * N}{l(C)}$$

Donde l(C) = longitud de la cadena de dependencias más larga. En lugar de |R| en el caso de tener recurrencias.

3.1.1 Los distintos tiempos de ejecución de las sentencias

Podemos tener sentencias que tarden mas unidades de tiempo en ejecutarse que otras. Esto es como si los nodos de esas sentencias, que tardan x unidades de tiempo, se transforman en x nodos todos encadenados con arco de distancia 0, es decir, como si se ejecutaran en forma secuencial x sentencias todas con igual peso en la misma iteración del bucle [AL89].

Los grafos con estas sentencias se tratan como si tuvieran más nodos. Supongamos un bucle con 2 sentencias, S_1 y S_2, y en el cual S_1 toma 1 unidad de tiempo en ejecutarse y S_2 3 unidades de tiempo (Fig. 3.3a). El grafo se transforma, al expresar S_2 en 3 nodos con dependencia de distancia 0 (Fig. 3.3b).

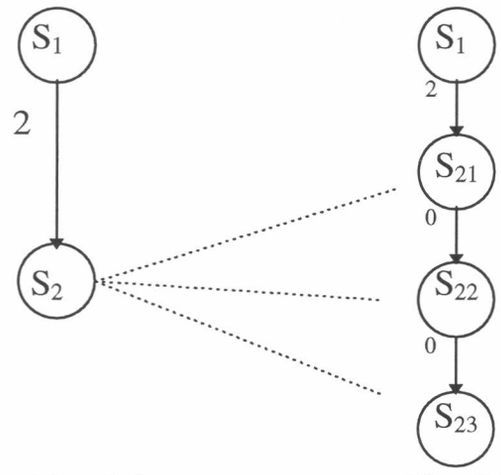


Fig. 3.3a

Fig. 3.3b

La existencia de estas sentencias provocará que

$$//prom = \sum_{i=1}^{n} u_i \; * \; Min \left(\frac{w(R^*)}{|R^*|} \right)$$

Donde

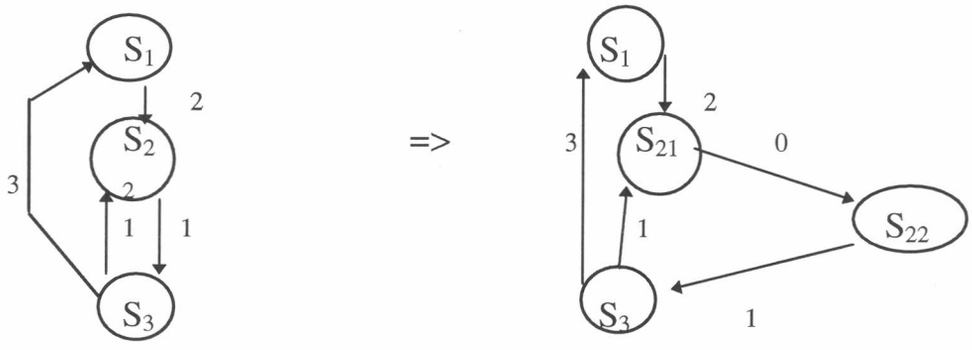
$\sum u_i$ =Sumatoria de las unidades de tiempo que tardan en ejecutarse todas las sentencias del bucle.

$w(R^*)$=Peso de la recurrencia teniendo en cuenta el peso de las sentencias. Este valor al agregarse distancias igual a 0 no se modifica respecto al anterior.

$|R^*|$=Cardinalidad de la recurrencia teniendo en cuenta el peso de las sentencias.

Si son varias las sentencias cuyo tiempo de ejecución es mayor a la unidad no se puede afirmar si su influencia aumenta o disminuye el paralelismo. En el caso en que la o todas las sentencias que tarden más en ejecutarse pertenezcan a la recurrencia que impone la cadena de dependencias más larga de todo el bucle se puede afirmar que el paralelismo puede disminuir, pues la cadena se hace aún más larga (ver Ejemplo 3.2). Lo contrario se puede afirmar en el caso en que la o todas las sentencias que tarden más en ejecutarse no pertenezcan a la recurrencia que impone la cadena de dependencias más larga de todo el bucle. Lo mismo sucede en un grafo acíclico.

Ejemplo 3.2



//= 3 * Min (2/2, 6/3) = 3 //= 4 * Min (2/3, 6/4) = 2.66

En el caso de ser un grafo con una sola recurrencia hamiltoniana el paralelismo se conserva, pues

//prom = w(R).

3.2 Paralelismo de los bucles de más de una dimensión

En esta sección explicaremos como se calcula el paralelismo promedio de bucles de más de una dimensión que están perfectamente anidados, es decir, son bucles con todas sus sentencias en el bucle más interno (Ej. 3.3).

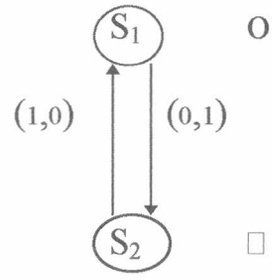
Ejemplo 3.3 (BPA)

```
DO i=1,NI
  DO j=1,NJ
    DO k=1,NK
      S1
      S2
      S3
    ENDDO
  ENDDO
ENDDO
```

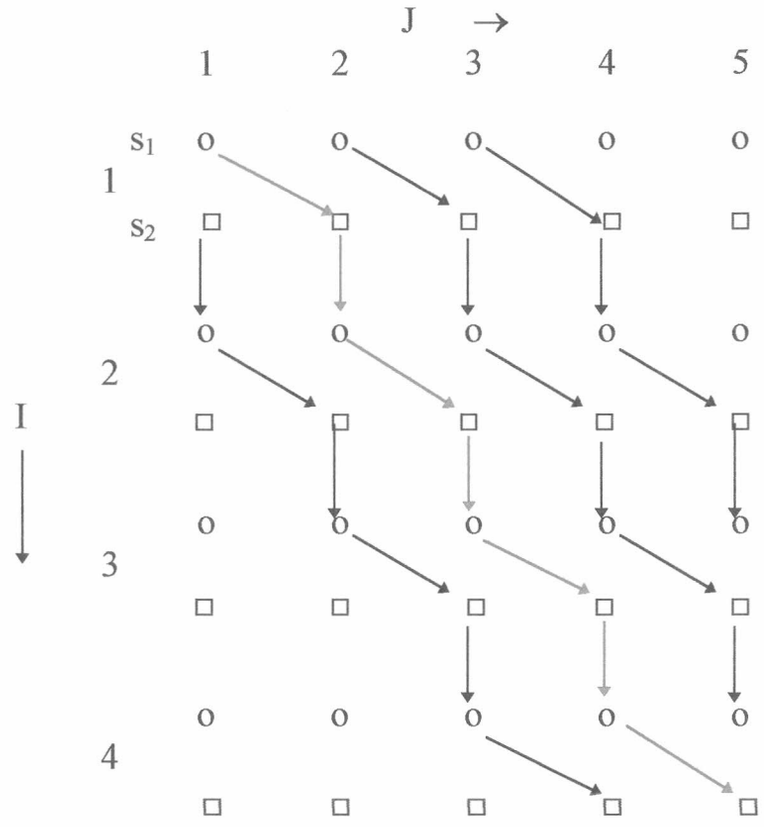
En relación a los bucles perfectamente anidados se trata también de encontrar la cadena de dependencias más larga a través de la ejecución desarrollada de los bucles anidados en todas sus iteraciones.Para ver esto necesitamos definir los espacios de sentencias por iteración, que no es otro que el espacio de ejecuciones desarrollado de las sentencias de los bucles anidados [ALT93].

Ejemplo 3.4

```
      DO I=1,4
        DO J=1,5
S1        a(I,J)=b(I-1,J)*d(I)
S2        b(I,J)=a(I,J-1)
        ENDDO
      ENDDO
```



Note que la distancia asociada a las dependencias es un vector donde cada elemento representa la distancia entre sentencias en una determinada dimensión.



Las cadenas (la más larga en rojo) generadas por la recurrencia.

La cadena de dependencias del (Ej. 3.4) significa que esos dos bucles anidados no podrán ejecutarse en menos tiempo que 8 unidades de tiempo, que es la longitud de la cadena más larga. Si calculamos el paralelismo promedio como antes, tenemos

$$Ts = n * NI * NJ = 2 * 4 * 5 = 40$$

y

$$Tp = 8 \qquad => \qquad //prom. = Ts / Tp = 40 / 8 = 5$$

¿ Existe una única recurrencia que genera la cadena de dependencias más larga ? En el caso de bucles anidados no es así (Ej. 3.5). La cadena de

dependencias más larga puede ser producida por la combinación de varias recurrencias. **Se trata de encontrar la longitud de esa cadena**.

La solución al problema de encontrar la longitud de la cadena de dependencias más larga se obtiene planteando un problema de programación lineal, aunque en realidad esta es una aproximación pues se trata de un problema de programación entera [ALT93].

3.2.1 Método utilizado para resolver el problema

En principio, se pueden utilizar métodos de programación lineal standard para resolver problemas de programación lineal entera sin tener en cuenta que las variables tienen la restricción de ser enteras. En este caso, optamos por resolver el problema de encontrar la cadena de dependencias más larga usando el método simplex revisado (MSR).

El problema a resolver es de la forma

Máx. $N_1 X_1 + N_2 X_2 + \ldots\ldots + N_n X_n$

Sujeto a las siguientes restricciones

$$W_{11} X_1 + W_{21} X_2 + \ldots + W_{n1} X_n \leq I_1$$
$$W_{12} X_1 + W_{22} X_2 + \ldots + W_{n2} X_n \leq I_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$W_{1m} X_1 + W_{2m} X_2 + \ldots + W_{nm} X_n \leq I_m$$

m dimensiones
n recurrencias

$W(R_1)$

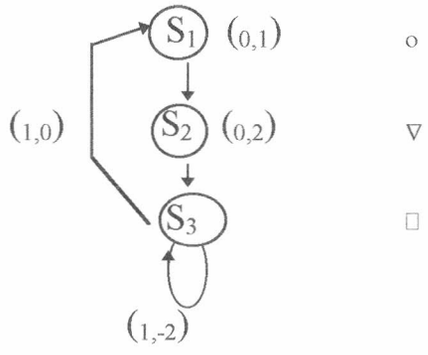
donde

X_i es la cantidad de veces que aparece la recurrencia R_i en la cadena más larga.
N_i es la cantidad de sentencias de la recurrencia R_i
W_{ij} es la sumatoria de las distancias de las dependencias que conforman la recurrencia i, en la dimensión j.
I_j es el espacio de iteración de la dimensión j
$W(R_i)$ es el vector de los pesos de la recurrencia R_i.

En el ejemplo 3.5 vemos como la cadena más larga (en rojo) utiliza ambas recurrencias para formarse.

Ejemplo 3.5

```
DO i=1,10
      DO j=1,10
            A(i,j)=2 * C(i-1,j)          o
            B(i,j)=A(i,j-1)+H(j)         ▽
            C(i,j)=C(i-1,j+2)+B(i,j-2)   □
      ENDDO
ENDDO
```



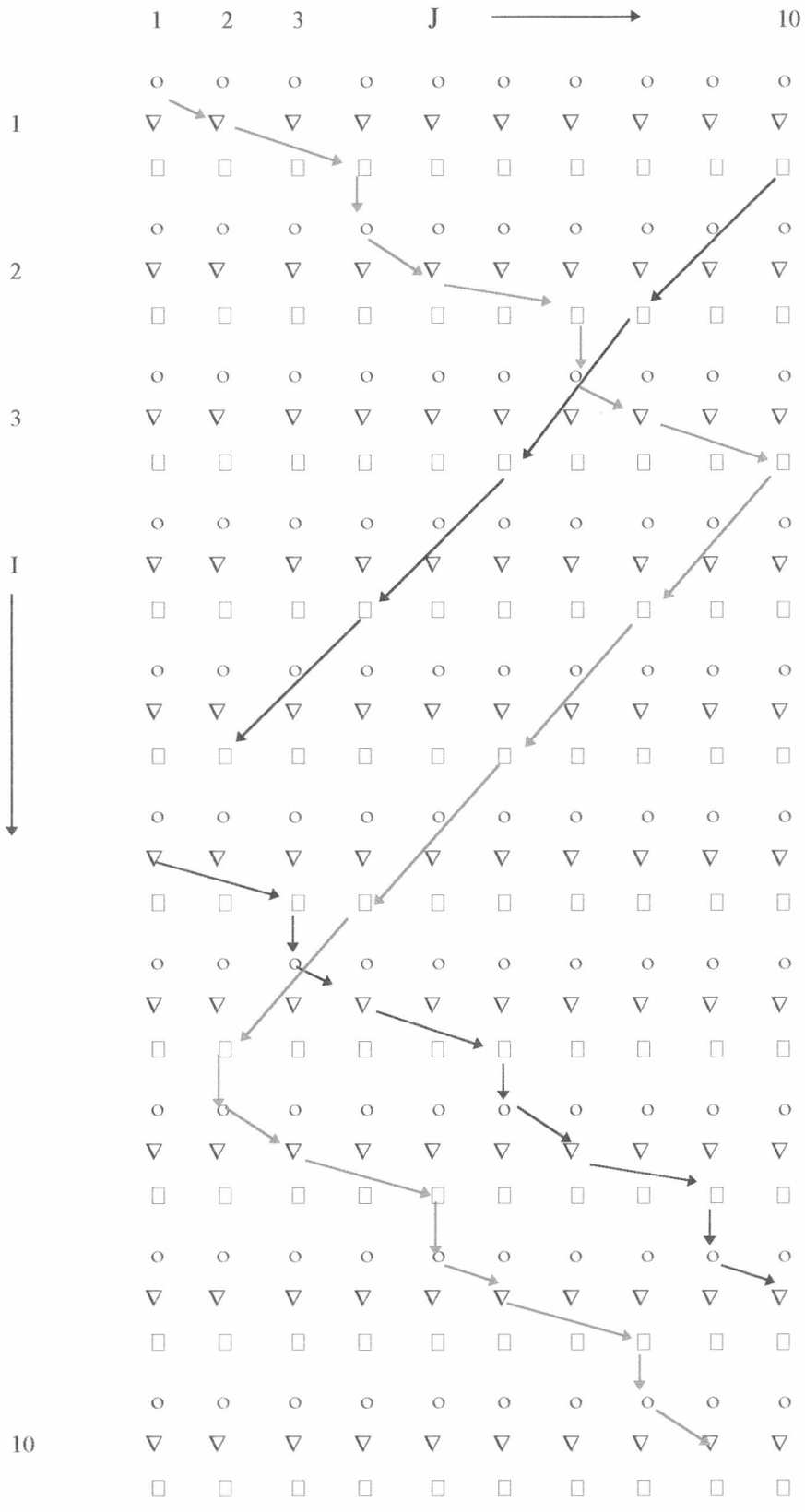
o

▽

□

$R_1 = (d_{12},d_{23},d_{31})$ $W(R_1) = W_1$

$R_2 = (d_{33})$ $W(R_2) = W_2$



longitud de la cadena l(C)=21

Las restricciones están sujetas a los límites de iteración y expresa lo que la cadena ejecuta en cada índice. El punto de comienzo de la cadena puede ser cualquier nodo del espacio de iteración, pues puede ocurrir que para algún índice, la cadena recorra el espacio de mayor a menor. En ese caso debemos verificar que la solución para la restricción en ese índice no supere el valor absoluto del límite del mismo. Con el grafo de la figura sería.

Función a Maximizar : $3\ X_1 + X_2$

Pesos de las recurrencias : $W_1=<1,3>$, $W_2=<1,-2>$

luego $W_{11}=1$, $W_{21}=1$, $W_{12}=3$, $W_{22}=-2$, $I_i=10$, $I_j=10$.

entonces las restricciones son

$$X_1 + X_2 \leq 10$$
$$3\ X_1 - 2\ X_2 \leq |\ 10\ | \qquad \text{luego}$$

$$X_1 + X_2 \leq 10$$
$$-10 \leq 3\ X_1 - 2\ X_2 \leq 10 \qquad \text{luego}$$

$$X_1 + X_2 \leq 10$$
$$3\ X_1 - 2\ X_2 \leq 10$$
$$-3\ X_1 + 2\ X_2 \leq 10$$

$X_i \geq 0$ para todo i

La Solución es $Z = (X_1=6 , X_2=4) = 3 * 6 + 4 = 22$

$T_{cad.+\ larga} = 22 \qquad \Rightarrow \quad // = 300 / 22 = 13,63$

Al resolver el problema por programación lineal encontraremos algunos errores, pero estos son más pequeños conforme aumenta el espacio de iteración [ALT93]. Resolvamos el ejemplo cambiando el espacio de iteración.

Sean I_i e I_j iguales a 100. Luego las restricciones a nuestro problema son:

$$X_1 + X_2 \leq 100$$
$$3\,X_1 - 2\,X_2 \leq 100$$
$$-3\,X_1 + 2\,X_2 \leq 100 \qquad X_i \geq 0$$

La solución es $Z = (X_1=60, X_2=40) = 3 * 60 + 40 = 220$

$T_{seq} = 30000 \qquad T_{cad.} = 220 \qquad => \qquad // = 30000 / 220 = 136,3$

Sean $I_i = 10$ e $I_j = 100$. Luego las restricciones a nuestro problema son:

$$X_1 + X_2 \leq 10$$
$$3\,X_1 - 2\,X_2 \leq 100$$
$$-3\,X_1 + 2\,X_2 \leq 100 \qquad X_i \geq 0$$

La solución es $Z = (X_1=10, X_2=0) = 3 * 10 + 0 = 30$

$T_{seq} = 3000 \qquad T_{cad.} = 30 \qquad => \qquad // = 3000 / 30 = 100$

Al aumentar el espacio de iteración, lo que en realidad estamos haciendo es aumentar el espacio de soluciones del problema, como así también aumentamos el tiempo de ejecutar secuencialmente ese bucle. Sin embargo mostramos con el ejemplo, que el aumento de uno y otro, no son proporcionales, pues el T_{seq} creció 100 veces y el $T_{cad.}$ 10 veces en el primer caso, mientras que en el segundo el T_{seq} creció 10 veces y el $T_{cad.}$ muy poco.

Teorema 3.1 : Con el método de programación lineal también resolvemos el caso de bucles de una dimensión.

Demostración :

Sin pérdida de generalidad tomemos el caso de un bucle con dos recurrencias cada una de ellas con W_1 y W_2 pesos respectivamente. El problema a resolver es

Máx. $N_1 X_1 + N_2 X_2$

Restr. $W_1 X_1 + W_2 X_2 \leq I$ I espacio de la iteración
$X_i \geq 0$

En el caso de un solo bucle solamente una recurrencia es la que nos da la cadena de dependencias a ejecutar más larga, por lo tanto tenemos dos posibles soluciones

1) $X_1 = I / W_1$, $X_2 = 0$ y 2) $X_1 = 0$, $X_2 = I / W_2$

luego una solución es $Z_1 = \frac{N_1 * I}{W_1}$ y la otra es $Z_2 = \frac{N2 * I}{W_2}$

De ambas nos quedamos con el máximo.

$$Tp = Máx\left(\frac{N_1 * I}{W_1}, \frac{N_2 * I}{W_2}\right) = I * Máx\left(\frac{N_1}{W_1}, \frac{N_2}{W_2}\right)$$

$Ts = I * N$ N = total de nodos del bucle.
I = Cantidad de iteraciones.

$$\text{luego } //prom = Ts / Tp = \frac{N}{Máx\left(\frac{N_1}{W_1}, \frac{N_2}{W_2}\right)} \quad \boxed{\alpha}$$

$$\text{ó } //prom = N * Mín\left(\frac{W1}{N1}, \frac{W2}{N2}\right)$$

$$\text{generalizando } //prom = N * Mín\left(\frac{W(R)}{|R|}\right)$$

Obsérvese que para obtener el paralelismo promedio de bucles de una dimensión no necesitamos conocer el valor del espacio de iteración. Tampoco el paralelismo promedio variará al cambiar dicho valor.

3.2.2 El tiempo de duración de las sentencias utilizando el MSR

Para facilitar el cálculo del paralelismo como en el caso de un solo bucle, el tiempo de duración de las sentencias, en caso de ser mayor que la unidad, se resuelve agregando arcos de distancia 0. Entonces la función a maximizar es

$$\sum_{i=1}^{n} U_i X_i$$

donde

U_i es la sumatoria de unidades de tiempo en que tarda en ejecutarse la recurrencia i.

Como el peso de las recurrencias no se modifican las restricciones se conservan como antes.

4-MEDIDAS DE PARALELISMO PROMEDIO DE BUCLES DE PROGRAMAS SECUENCIALES NUMERICOS

4.1 Cálculo del Paralelismo Promedio de bucles DO

Se define paralelismo de un bucle como el número medio de procesadores que ejecutan diferentes instanciaciones de las sentencias del bucle, dado un reparto de tareas óptimo y suponiendo costos de sincronización y de reparto de tareas nulo [AL89].

Para medir el paralelismo promedio de los bucles DO hemos desarrollado un programa.

Este programa tiene por objetivo proporcionarnos el paralelismo promedio de todos los bucles DO anidados, más internos, de un programa secuencial (en este caso de programas escritos en lenguaje Fortran). Este paralelismo promedio tendrá dos valores, que dependerán de si se tiene en cuenta o no el tiempo estimado de ejecución de las sentencias. Por otro lado sólo son considerados para el cálculo, bucles cuyas distancias de dependencias sean constantes y no contengan instrucciones de E/S o condicionales. En esos casos se indicará E/S , CONDICIONALES o DESCONOCIDAS

Estos datos serán clasificados estadisticamente para el estudio y las conclusiones.

La estructura de datos que contiene el grafo tiene la cantidad de sentencias que componen el bucle, cuantos índices anidados hay y sus límites

de iteración, el conjunto de dependencias y el peso (etiqueta relativa al tiempo estimado de ejecución) de cada sentencia.

4.2 Las entradas

Dimensiones máxima de los grafos

Cantidad máxima de bucles a examinar = sin límites

Cantidad máxima de sentencias de un bucle = 120

El grafo de Dependencias

La estructura de datos que contienen los grafos de dependencias es un archivo ascii conteniendo los grafos a analizar. La estructura de cada grafo es, tomándolo de un ejemplo, la siguiente:

```
3
2(1,100) (1,10)
( 1,2,[0,1]f)( 2,3,[1,2]f)
( 3,1,[3,3]f)( 3,2,[1,2]f)/
PESOS DE LAS SENTENCIAS : 1 1 1
```

La primera línea indica la cantidad de sentencias del bucle (o nodos del grafo). La segunda línea representa la cantidad de bucles anidados (2 en este caso) y los rangos de iteración de cada uno de ellos .

A partir de la tercer línea están todas las dependencias del grafo encerradas entre paréntesis. Los dos primeros números indican entre qué sentencias es la dependencia. Lo encerrado entre corchetes indica cuales son las distancias asociadas a dicha dependencia para cada índice. La última letra nos dice qué tipo de dependencia es (f=flujo, o=output,a=antidependencia).

Por último tenemos el peso de cada sentencia expresado a partir de una etiqueta que indica el tiempo estimado de ejecución de cada una. La estructura mencionada anteriormente está representada en el grafo de la Fig. 4.1.

```
DO  i=1,100
    DO  j=1,10
      A(j) = C(i-3,j-3)

      B(i,j) = A(j) * C(i-1,j-2)

      C(i,j) = B(i-1,j-2)

      ENDDO

ENDDO
```

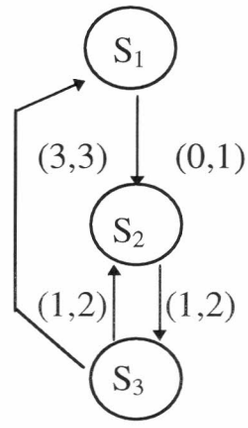
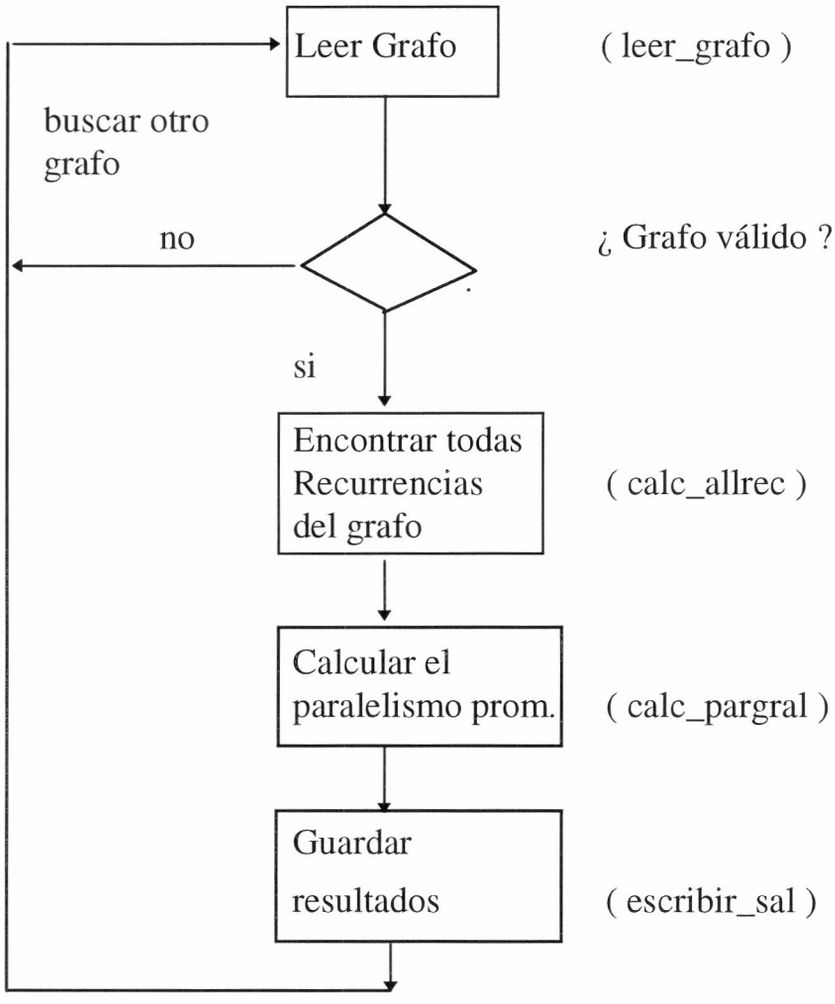


Fig. 4.1

4.3 Descripción del programa

La Lógica

La lógica del programa que calcula el paralelismo promedio puede sintetizarse como sigue:



Leer_grafo :

Rutina que lee el grafo y va cargando una estructura de datos llamada matr donde se guardan todas las dependencias del bucle, indicando el nodo fuente, el nodo destino y las distancias asociadas para cada dimensión del bucle, verificando que no existan dependencias repetidas. También se encarga de detectar la existencia de sentencias condicionales, de entrada/salida o dependencias con distancias desconocidas, en cuyo caso descarta ese grafo y pasa a leer el grafo correspondiente a otro bucle del programa. Al completar la lectura de un grafo, guarda en un vector los tiempos estimados de cada sentencia.

calc_allrec :

Rutina que se encarga de buscar todas las recurrencias del grafo a través de una función recursiva llamada **busco**. Si la cantidad de dependencias del grafo es grande, esta búsqueda puede llegar a insumir mucho tiempo. Pero no es así en la casi totalidad de los bucles.

Esta rutina tiene en cuenta si la dimensión del bucle es mayor a 1, en cuyo caso se guardan todas las recurrencias del bucle en 2 vectores (m_w y m_via). m_w(i) contiene la suma de las distancias de las dependencias que participan de la recurrencia "i" para cada dimensión del bucle. m_via(i) guarda el camino de esa recurrencia, es decir qué nodos intervienen en ella.

Si la dimensión del bucle es igual a 1 solo se va quedando con la recurrencia que le da la cadena de dependencias más larga, a medida que las va encontrando (Ej. 4.1). Esto lo va calculando con el denominador de la fómula α (teorema del cap. 3), que es el cociente entre N (cantidad de nodos de la recurrencia) y W (peso de la recurrencia) tanto para cuando no tiene en cuenta el tiempo estimado de ejecución de las sentencias y para cuando si toma en cuenta ese tiempo. Como vimos en ese teorema, no consideramos el número de iteraciones.

Ejemplo 4.1

En el grafo siguiente tenemos un bucle de 1 dimensión.

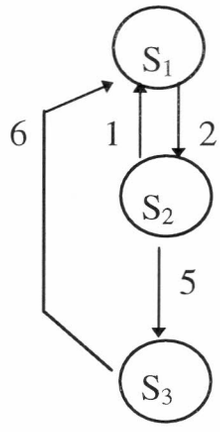
DO I = 1,máx

A(I) = B(I) * Kc + D(I-6)

B(I) = A(I-2)

D(I) = B(I-5)

ENDDO



Tenemos dos recurrencias $R_1=\{d_{12},d_{21}\}$, $R_2=\{d_{12},d_{23},d_{31}\}$

Supongamos que R_1 es la primera recurrencia encontrada, entonces calculamos m_w = 3 (W) y m_via es un número binario donde cada posición indica con el valor 1 si ese nodo interviene en la recurrencia. En este caso m_via tiene 3 dígitos para cada uno de los nodos (S_3 S_2 S_1), por lo tanto m_via = (011) para R_1. Al contar la cantidad de unos de m_via (rutina **num_unos**) obtenemos el valor N.

luego N/W = 2/3

Al encontrar R_2 tenemos m_w = 13 y m_via = (111). entonces

N/W = 3/13.

Luego comparamos este valor con el anterior y vemos que es menor, por lo cual m_w y m_via quedan con los valores correspondientes a la recurrencia R_1 que es la que nos da la cadena de dependencias más larga.

Calc_pargral :

Con esta rutina hacemos el cálculo propiamente dicho del paralelismo promedio de los bucles. Para ello calculamos el cociente entre el tiempo de ejecución secuencial (que es el producto de la cantidad de sentencias del bucle por la cantidad de iteraciones de cada dimensión), y la longitud de la cadena de dependencia más larga. En el caso de los bucles de una dimensión tomamos la cadena que ya habíamos seleccionado al buscar todas las recurrencias y la multiplicamos por la cantidad de iteraciones para poder dividir al tiempo de ejecución secuencial que ya estaba multiplicado por ese valor, de este modo obtenemos el paralelismo promedio para estos bucles. En los bucles de más de una dimensión usamos una rutina llamada **calc_paral** que arma el problema de programación lineal para que pueda ser usado por el programa **simplex** que lo resuelve. Este nos devuelve la longitud de la cadena de dependencias más larga

El tiempo de las sentencias :

Cuando queremos calcular la longitud de la cadena de dependencias más larga teniendo en cuenta los distintos tiempos de ejecución de las sentencias debemos observar que el problema planteado no es el mismo que antes. La función a maximizar pasará de

Máx $N_1 X_1 + N_2 X_2 + ... + N_n X_n$ a

Máx $\eta_1 X_1 + \eta_2 X_2 + ... + \eta_n X_n$

donde $\eta_i = \sum_k \text{tiempos}(k)$ $\forall$ k que pertenezca a la recurrencia i

El conjunto de restricciones queda igual pues no se registran modificaciones en las distancias de las dependencias al agregar arcos de distancia cero. De este modo debemos construir un nuevo problema y volver a utilizar el método simplex revisado que es el que empleamos en su resolución

Finalmente el programa graba en un archivo para el bucle analizado un número de identificación, la dimensión del bucle , el número de sentencias del bucle y, de haberse podido hacer el cálculo, los valores del paralelismo promedio con y sin el tiempo estimado de duración de las sentencias.

Simplex :

El simplex es un programa, escrito en Fortran, que lee los datos del problema de programación lineal de un archivo temporario ascii que arma la subrutina **calc_paral**, con estos datos se inicializan los parámetros de entrada del procedimiento **H01adf** , de la biblioteca N.A.G.. Esta rutina fue escrita en lenguaje Fortran por lo cual utilizamos una vinculación a través del procedimiento "*system*" de C que fue lo que se escogió. Con *system*, C puede ejecutar comandos del sistema o programas de usuario.

Restricciones :

El programa sólo resuelve problemas con las siguientes dimensiones máximas

Cantidad máxima de variables (o recurrencias del bucle) = 90

Cantidad máxima de anidación (dimensiones) del bucle = 10

4.4 Los programas medidos

Los programas utilizados son 12, de los cuales 10 pertenecen a la suite de programas Perfect Benchmarks y los otros dos son un conjunto de rutinas standard de cálculo llamadas Blas y Linpack.

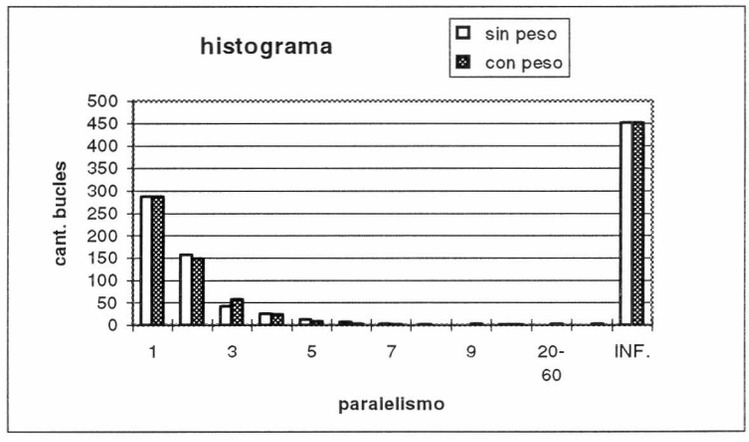
En el apéndice B están las estadísticas correspondientes a los 12 programas. La abreviación "//" significa paralelismo. Lo que se indica como paralelismo infinito son bucles cuyo paralelismo sólo está acotado por los límites de su iteración y se corresponden con bucles vectorizables. Los bucles que son indicados como otros* son bucles con grado de paralelismo desconocido.

En el apéndice D están los listados del grado de paralelismo de los bucles de cada uno de los programas. Este listado describe en cada linea, 5 columnas que indican 1) el número de grafo del programa, 2) la cantidad de bucles anidados, 3) el número de sentencias del bucle, 4) el paralelismo promedio sin tener en cuenta el tiempo estimado de cada sentencia -// sin peso- y 5) el paralelismo promedio teniendo en cuenta el tiempo estimado de cada sentencia -// con peso-

En la página siguiente están las estadísticas de los totales de todos los programas.

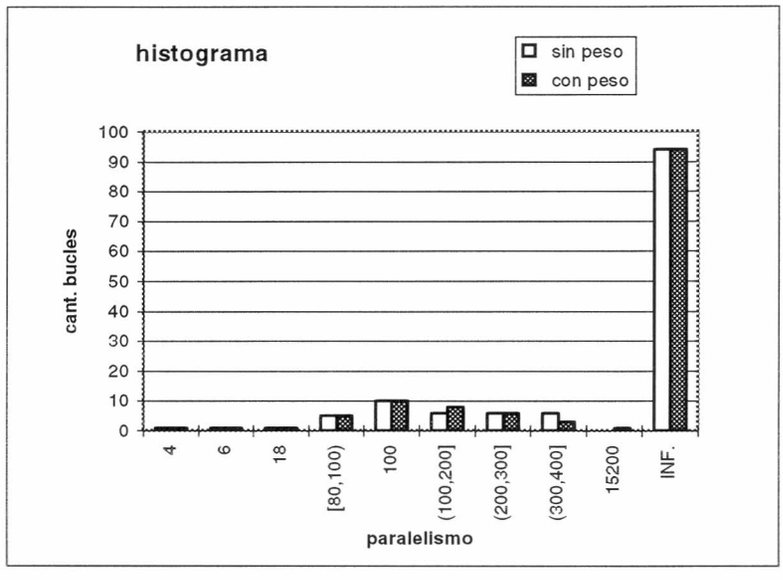
Estadísticas de todos los bucles de dimensión 1

	cantidad de bucles	
paralelismo	sin peso	con peso
1	287	287
2	158	149
3	43	57
4	25	24
5	12	9
6	7	2
7	3	1
8	1	0
9	0	3
10	1	1
20-60	0	2
90-130	0	2
INF.	451	451



Estadísticas de los bucles de dimensión 2

	cantidad de bucles	
paralelismo	sin peso	con peso
4	1	1
6	1	1
18	1	1
[80,100)	5	5
100	10	10
(100,200]	6	8
(200,300]	6	6
(300,400]	6	3
15200	0	1
INF.	94	94



5-MEDIDAS DE UTILIZACION DE LOS BUCLES DO EN TIEMPO REAL

5.1 Descripción del método empleado.
5.2 Estadísticas asociadas a las mediciones.
5.3 Profile, fracción secuencial, paralelismo mínimo y máximo.

5.1 Descripción del método empleado

Nuestro objetivo al ejecutar cada programa Fortran fue medir el tiempo total de uso de los bucles DO perfectamente anidados en una o mas dimensiones y determinar la cantidad de veces que se utilizaban dichos bucles durante la corrida del programa para tener una idea de la influencia real de los mismos y de esta manera relativizar su paralelismo en función de su "importancia" dentro del programa.

Se modificaron 10 programas Perfect Benchmarks, con los que se trabajó, para medir todos sus bucles con una rutina que nos permitió "cronometrar" el tiempo total de ejecución de todos los bucles (1249) y el tiempo total de ejecución de los programas.

Para la medicion del tiempo de duracion de los bucles DO utilizamos una rutina de la libreria de Fortran llamada **etime**. Esta rutina nos devuelve en segundos el lapso de tiempo transcurrido desde que comienza el programa. También retorna en su argumento el tiempo de cpu y el tiempo requerido en otras funciones del sistema. Tanto el valor devuelto, como los argumentos son de tipo real. Para acumular los tiempos en los cuales se ejecutan los bucles DO utilizamos un vector llamado tempo cuya dimensión depende de la cantidad de bucles a medir que tenga el programa. Para calcular la cantidad de veces que fue ejecutado cada bucle,usamos un vector al que llamamos uso. Ademas, a cada programa, le medimos el tiempo total de ejecución.

Veamos un ejemplo de lo anteriormente comentado.

Dada la siguiente porción de programa Fortran:

```
      NP=54
      DO 10 I=1,20
            VEC(I)=VEC(I)+3
            A(I)=VEC(I)*NP
10    CONTINUE
```

y suponiendo que dicho ciclo fuera el nro 20 dentro del programa, tomando el orden establecido por el programa fuente, agregandole la función **etime** y los vectores tempo y uso , obtendríamos las siguientes líneas de código:

```
      NP=54
      T0=ETIME(W0)
      DO 10 I=1,20
            VEC(I)=VEC(I)+3
            A(I)=VEC(I)*NP
10    CONTINUE
      TI=ETIME(W1)
      TEMPO(20)=TEMPO(20)+T1-T0
      USO(20)=USO(20)+1
```

Esta tarea de modificar el código fuente fué realizado en 1249 bucles de 10 programas Fortran.

Un problema que tuvimos que resolver fue que 4 programas tenían una declaración de Implicit Double Precision (A-H, O-Z) que forzaban a las variables de la función **etime** y a sus asociadas, al tener una precisión distinta de la real, independientemente de que estas estuvieran declaradas en forma explícita. Esto nos trajo inconvenientes con la salida, ya que eran resultados totalmente incompatibles con la realidad. Para resolver esto debimos declarar

explicitamente todas las variables de esos programas, teniendo en cuenta lo que ya estaba declarado de esa manera , comentariando las lineas que indicaban implicit double precision. Esto nos llevó mucho tiempo resolverlo, ademas del que nos insumió hacer las modificaciones.

Estos programas se ejecutaron con una entrada de Datos Standard [K93] en un equipo SUN SPARC Station 10, bajo unix (SunOS 4.1.3) bajo licencia del CSRD de la Universidad de Illinois - USA.

5.2 Estadísticas asociadas a las mediciones

En el apéndice C están las estadísticas correspondientes a los 10 programas evaluados. En ellas se reflejan la cantidad de bucles que hay en un rango de tiempo de ejecución respecto del tiempo total del programa.

En el apéndice E están los listados del tiempo de ejecución de los bucles en los 10 programas evaluados. En estos listados se describe para cada bucle el tiempo total de utilización en segundos, el porcentaje que representa ese tiempo en relación al tiempo total de ejecución del programa y, cuantas veces fue usado ese bucle.

En la página que sigue está una pequeña descripción de los programas y la estadística de la totalidad de los mismos.

Nombre	Aplicación	Autor	Tipo de algoritmos usados								
			1	2	3	4	5	6	7	8	9
APSI	Polución aérea y Dinámica de fluidos	IBM			x						
BDNA	Simulación de acido nucleico y dinámica molecular	IBM						x			
DYFESM	Estructuras dinámicas e ingeniería	CSRD	x	x				x			
FLO	Flujo transónico y dinámica de fluidos en 2-D	Princeton					x	x			
LWSI	Simulación de líquidos	IBM						x			
QCD2	Cantidades cromodinámicas-matrices de medidas	Caltech							x		
SRSI	Re-acceso supercónico-Dinámica de fluídos	NASA	x			x					
TISI	Transformaciones integrales bi-electrón - Dinámica molecular	IBM								x	
TRACK	Ruteo misilístico - Procesamiento de señal	Caltech									x
WSSI	Simulaciones temporales - Dinámica de fluídos	CSRD			x	x					

Algoritmos:

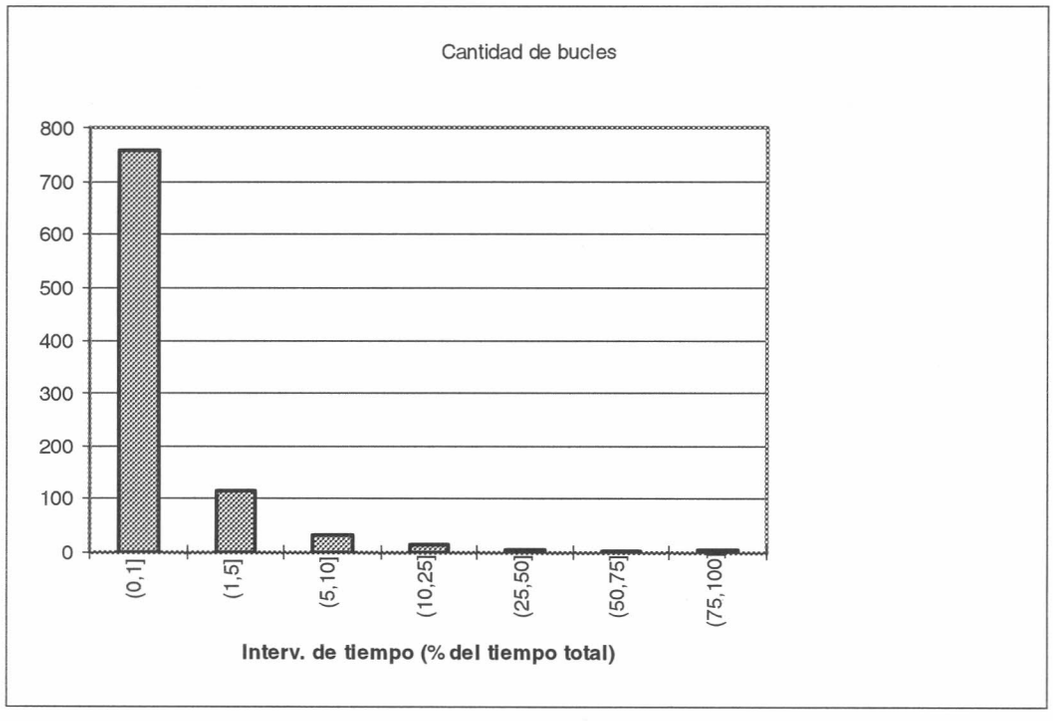
1 : Resolución de problemas lineales esparsidos.
2 : Resolución de sistemas algebraicos no lineales.
3 : Transformaciones de Fourier.
4 : Resolución de problemas elípticos.
5 : Esquemas multigrado.
6 : Resolución de ecuaciones diferenciales ordinarias.
7 : Esquemas Monte Carlo.
8 : Integrales.
9 : Convolución.

TOTALES

Cantidad total de bucles de todos los programas	1249	
Cantidad de bucles ejecutados	934	75%
Tiempo total de ejecución	10989,29 seg.	

Cantidad de bucles que se ejecutan en un intervalo del tiempo total de ejecución

% de tiempo	Cantidad de bucles
(0,1]	757
(1,5]	116
(5,10]	33
(10,25]	16
(25,50]	5
(50,75]	3
(75,100]	4



5.3 Profile , fracción secuencial , paralelismo mínimo y máximo

A continuación ilustraremos con algunos bucles de los programas analizados cómo son el paralelismo profile, mínimo y máximo y la fracción secuencial. Escogimos 4 bucles de grado de paralelismo 2 y de una dimensión, que representan a la mayoría de los bucles doacross. También tomamos a 1 bucle de 2 dimensiones para ver como se comporta. Estos grafos fueron tomados de las entradas del programa que calcula el paralelismo promedio y se corresponden en su numeración con los listados del apéndice D y E.

Hemos definido al paralelismo profile como la variación de la cantidad de procesadores activos durante la ejecución de las sentencias de un bucle. Del mismo podemos extraer la fracción secuencial y el paralelismo mínimo y máximo.

Programa ADM (APSI)
grafo n° 29 -apéndice D-
n° de iteraciones = 99
cantidad de sentencias =4

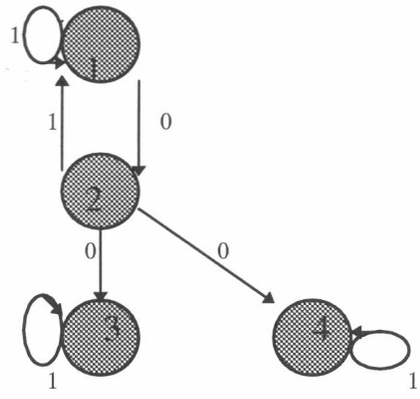
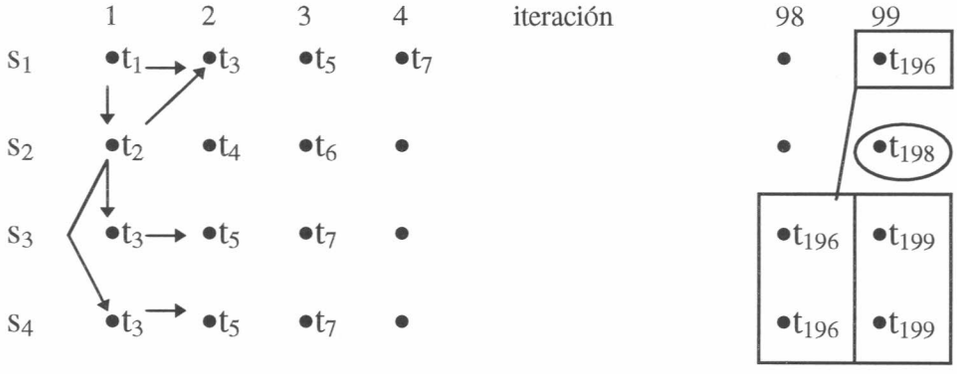
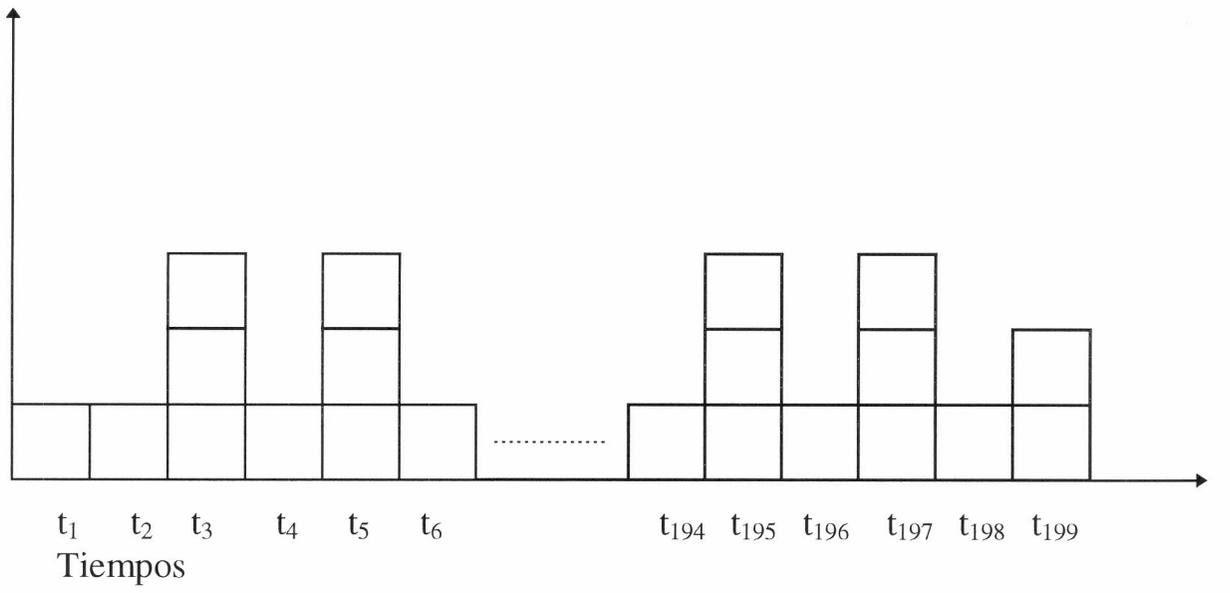


Fig. 5.1





Profile

La Fracción secuencial es la fracción de tiempo que no puede ser ejecutado en paralelo con ninguna otra parte del bucle (o generalizando, del programa). En este caso tenemos a los tiempo t_1,t_2,t_4,t_6 y el resto de los tiempos pares hasta el t_{198}, siendo esta fracción igual a 100/199 = .5

El paralelismo mínimo es el menor de los números de procesadores que hay en el profile. En este caso es 1.

El paralelismo máximo es el mayor de los números de procesadores que hay en el profile. En este caso es 3.

Con el paralelismo profile podemos calcular también el paralelismo promedio, haciendo el promedio de procesadores activos durante la ejecución del bucle. En este caso es 396/199 = 1.99.

Programa BDNA
grafo nº 4 -apéndice D-
nº de iteraciones = 100
cantidad de sentencias = 4

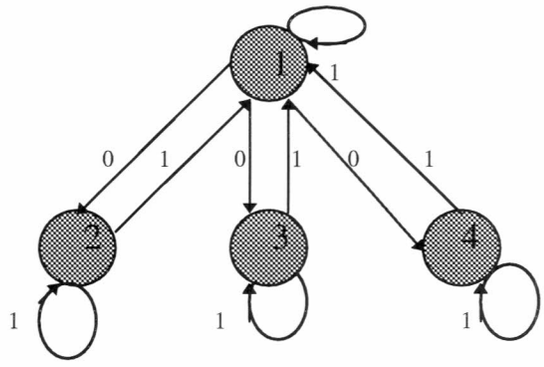
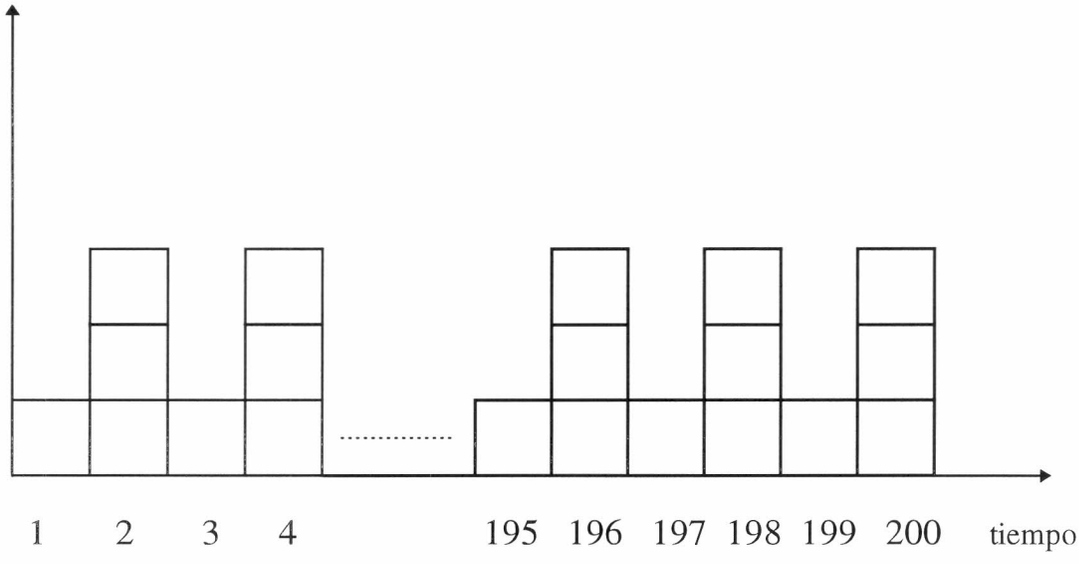


Fig. 5.2

nº proc.



Profile

Fracción secuencial = 100 / 200 = .5

// mínimo = 1

// máximo = 3

Programa MDG (LWSI)
grafo nº 13 -apéndice D-
nº de iteraciones 100
cantidad de sentencias 4

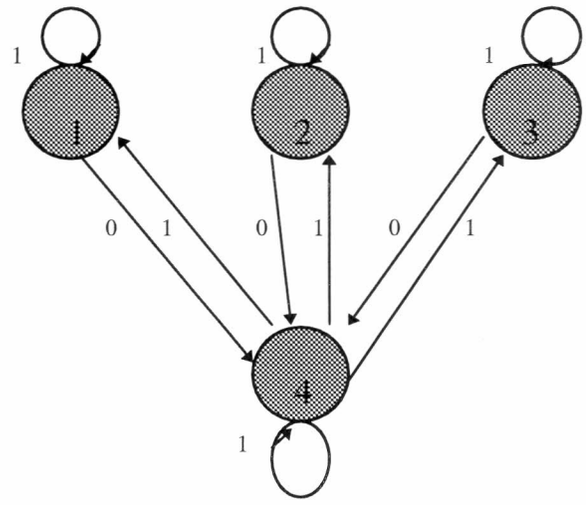
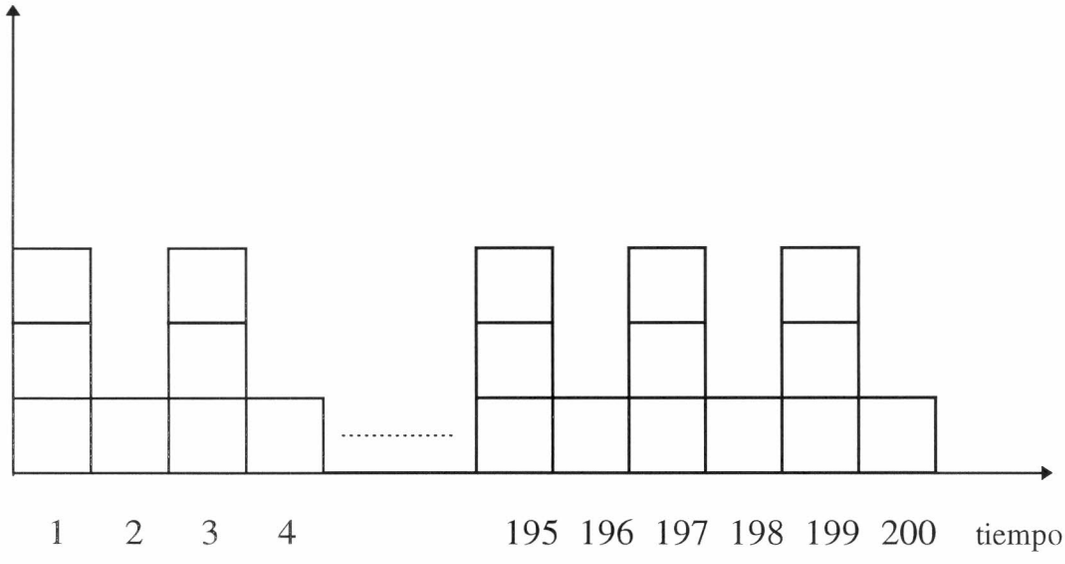


Fig. 5.3

nº proc.



Profile

Fracción secuencial = 100 / 200 = .5

// mínimo = 1

// máximo = 3

Programa ARC2D (SRSI)
grafo nº 70 -apéndice D
nº de iteraciones = 100
cantidad de sentencias = 6

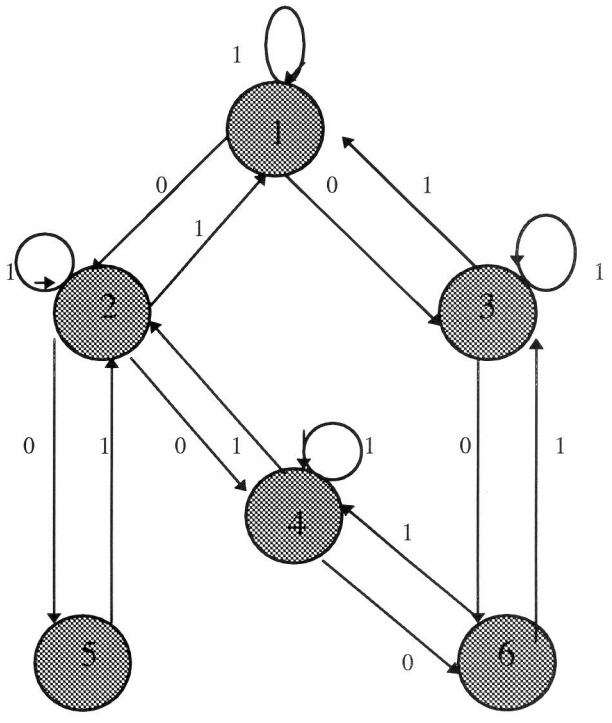
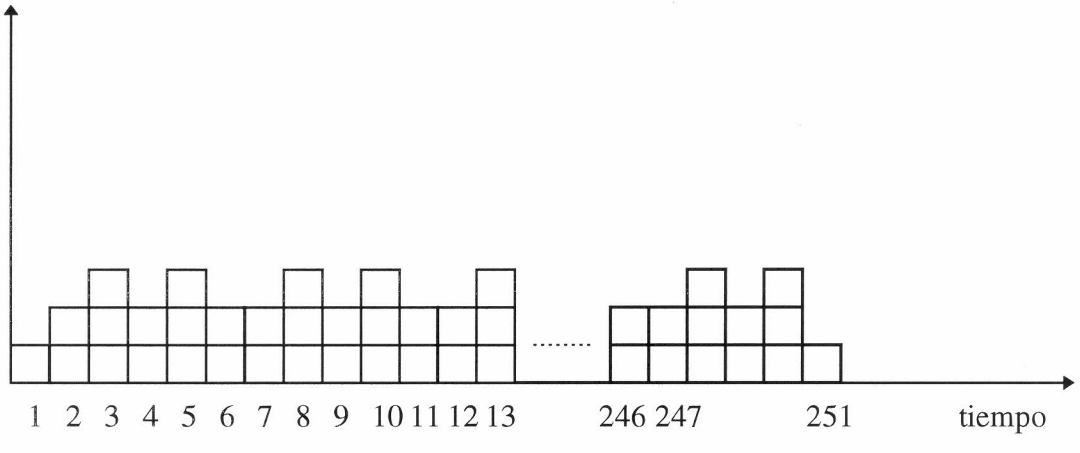


Fig. 5.4

nº proc.



Profile

Fracción secuencial = 2 / 251 = .008

// mínimo = 1

// máximo = 3

Programa SPEC77 (WSSI)
grafo nº 228 -apéndice D-
nº de iteraciones = 10000 (100 x 100)
cantidad de sentencias = 3

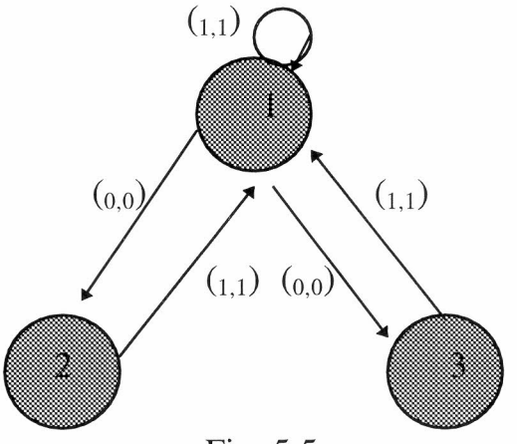
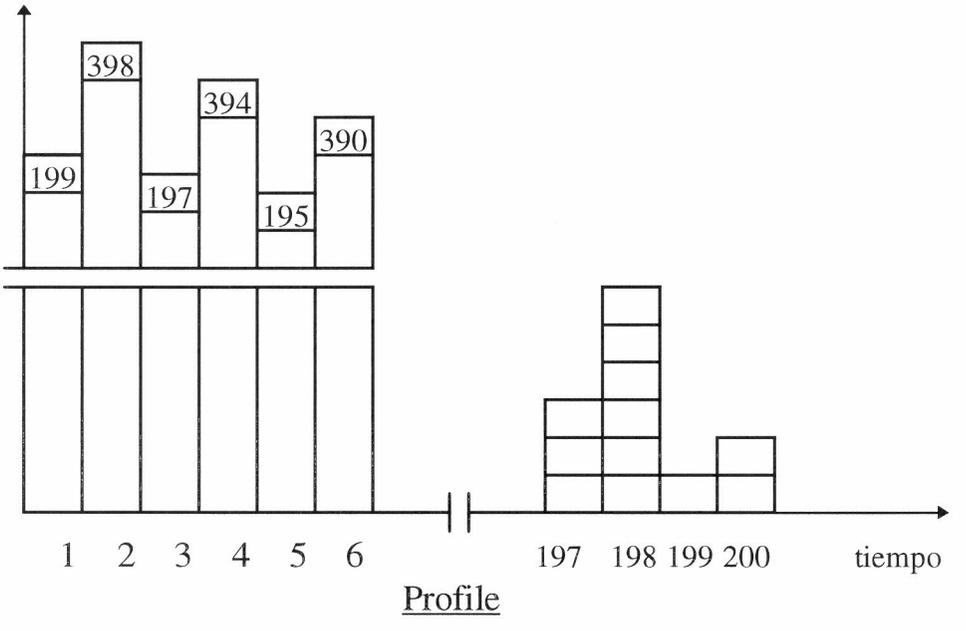


Fig. 5.5



Por razones de espacio se han realizado partes del profile en distintas escalas. Para comprender mejor el mismo veamos el orden de ejecución de las sentencias en el espacio de iteración de la Fig. 5.6.

Fracción secuencial = 1 / 200 = .005
// mínimo = 1 ; // máximo = 398 ; // prom. = 30000/200 = 150

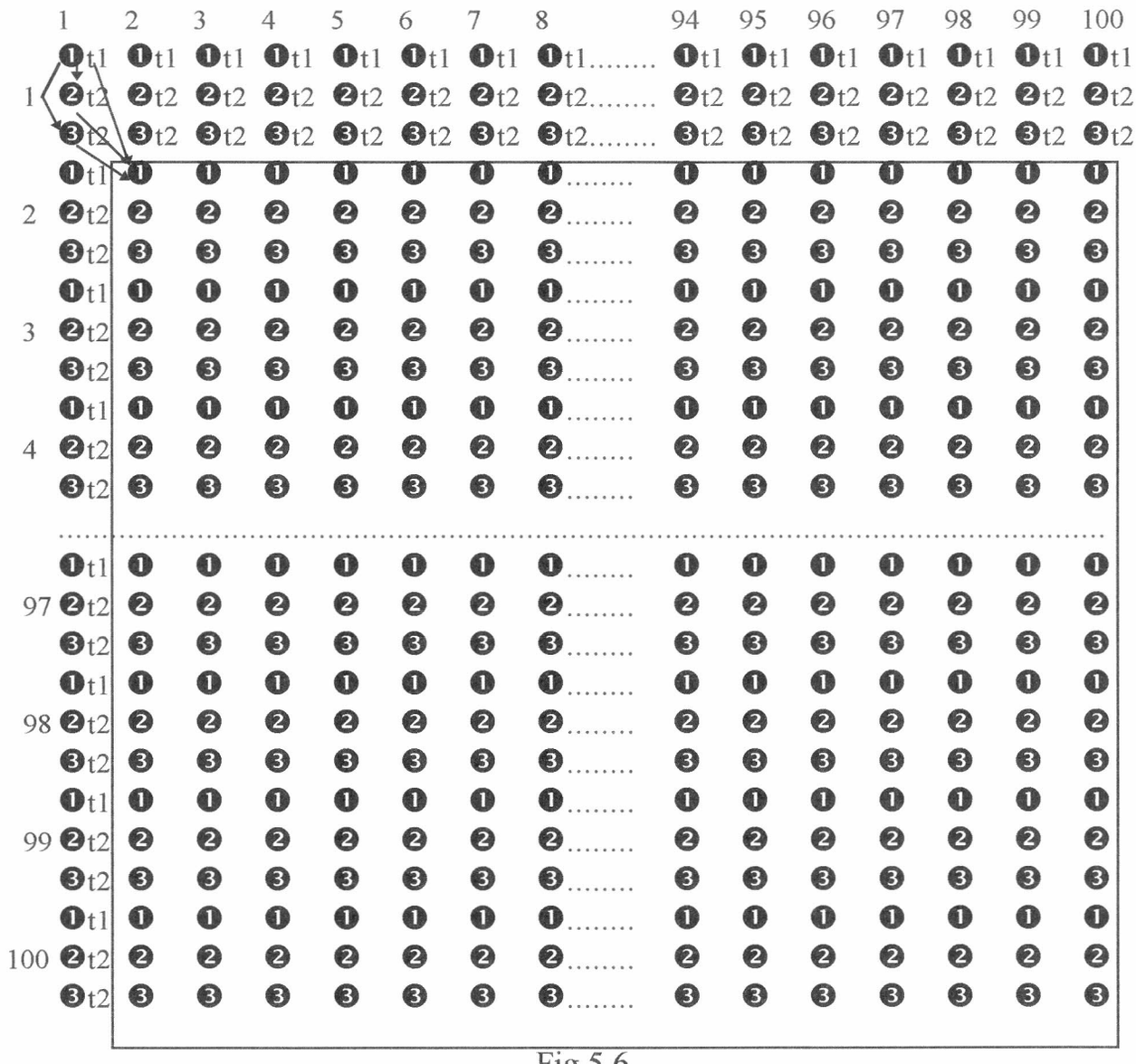


Fig 5.6

En t1 se ejecutan todas las sentencias libres de dependencias, en este caso todas las sentencias S_1 de las iteraciones para (filas) J=1 e I (columnas) de 1 a 100 y de I=1 y J de 2 a 100; un total de199. Las sentencias S_2 y S_3 dependen de la ejecución de la sentencia S_1, por lo tanto se ejecutan todas estas sentencias, en t2, que se liberan de la ejecución de la sentencia S_1 en t1. Este subtotal es 398. Al ejecutarse t1 y t2 nos queda un espacio de iteración de 99x99. Procedemos igual que antes y obtenemos los distintos tiempos sucesivos que se muestran en el profile.

6-INTERPRETACION DE LOS RESULTADOS Y APORTES

6.1 Resultados de las medidas de paralelismo promedio

Al analizar el grado de paralelismo de los programas hemos extraído los siguientes totales discriminados por cantidad de bucles anidados y por tipo de resultado encontrado.[1]

	// Finito	// Infinito	Otros*	
1 Dim.	537	451	456	1444
2 Dim.	36	94	37	167
3 Dim.	4	14	8	26
Total	577	559	501	1637

* se refiere a bucles con distancias de dependencias desconocidas, sentencias de entrada/salida o condicionales. Estos bucles no fueron tomados en cuenta para el análisis en este capítulo. Al resto de los bucles se los considera con grado de paralelismo conocido.

---

[1] Una gran cantidad de bucles tenían valores no constantes en sus límites de iteración. En estos casos le tuvimos que cargar un valor constante y elegimos que fueran bucles que iteran de 1 a 100. [Zch90] en página 117 define a los bucles cuyo límite inferior es igual a 1 como bucles do **normalizados**

6.1.1 Bucles de 1 dimensión

De un total de 988 bucles con grado de paralelismo conocido hemos encontrado que un 45.6% (451) se corresponden con bucles vectorizables (doall), de los cuales, vemos que el 75.4% (340) son bucles de una sentencia.

Los bucles que son secuenciales, es decir que tienen grado de paralelismo igual a 1 son el 29% (287). De estos, 51.2% (146) son bucles de una sola sentencia y el 41.5% (119) son bucles de 2 sentencias.

De aquellos bucles con grado de paralelismo conocido, distintos de paralelismo infinito y que no son secuenciales (grado de paralelismo mayor que 1) hemos encontramos 250, lo que representa un 25.3%. De ellos, la gran mayoría (158 , 60%) son bucles con un paralelismo igual a 2, excediendo el grado 5 en muy pocos casos.

En el gráfico de la Fig. 6.1 se reflejan los porcentajes de cada uno de los tipos de bucles encontrados. Llamamos bucles *secuenciales* (sec.) a los bucles de grado de paralelismo 1, bucles *doacross* a los de grado de paralelismo mayor a 1 y que no son infinitos y bucles *doall* a aquellos con paralelismo infinito. En los dos primeros casos los bucles incluyen recurrencias. En los bucles *doall* no hay recurrencias.

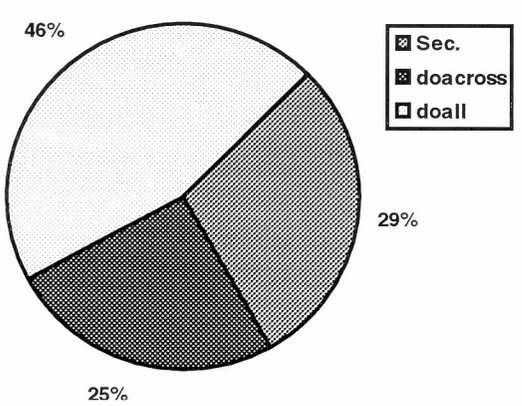


Fig. 6.1 - bucles con grado de paralelismo conocido.

6.1.2 Bucles de 2 dimensiones

De un total de 130 bucles de 2 dimensiones con grado de paralelismo conocido, 94 (el 72.3%) se corresponden con bucles *doall*. De éstos, al igual que en los bucles de 1 dimensión, una gran parte (el 77.6%) son de bucles de una sentencia.

El resto de los bucles (36) han dado valores de paralelismo, no incluyendo bucles secuenciales. De estos, 18 han dado valores entre 80 y 150 de paralelismo promedio, 15 entre 200 y 400 y, por último, 3 entre 4 y 18 (un bucle perteneciente al QCD2, el n°10, y dos al TRACK, 39 y 56). Estos valores más altos nos indican que estos bucles tienen un grado de paralelismo mayor que los bucles de una dimensión. Sin embargo, notamos que los valores de paralelismo promedio de estos bucles dependen mucho del valor de los límites de iteración. En los últimos casos los espacios de iteración eran de 4x4 en dos de ellos, y de 6x18 en el restante. Luego hemos incrementado estos valores a 100x100 en todos ellos, dándonos un paralelismo de 150 en un caso y 100 en dos. Haremos una interpretación más detallada de estos resultados en el punto 6.2.

6.1.3 Bucles de 3 dimensiones

Solamente hemos encontrado 18 bucles de 3 dimensiones con valores conocidos, de los cuales 14 son de valor infinito y 4 de grado de paralelismo (10000). En estos últimos se confirma lo que dijimos anteriormente en el sentido de que ese valor depende de los límites de iteración de las dimensiones del bucle.

6.2 Análisis del paralelismo promedio en bucles de más de una dimensión

Vamos a explicar los resultados de paralelismo promedio obtenidos en los bucles de más de una dimensión cuyos valores son conocidos y distintos de infinito. Para ello vamos a clasificar los bucles correspondientes a estos valores de paralelismo de acuerdo al comportamiento de la cadena de dependencias más larga. Hemos visto que estos valores son altos (entre 80 y 400) y que dependen del espacio de iteraciones del bucle.

Esto se da porque en la mayoría de ellos se forman cadenas (la más larga) que avanzan en una de las dimensiones y la distancia para esa dimensión es igual a 1. Estas cadenas se forman por la participación de solo una recurrencia, entonces W(R) es (0,1) ó (1,0) (para bucles de 2 dimensiones) dependiendo de en qué dimensión avanza (Fig. 6.2).

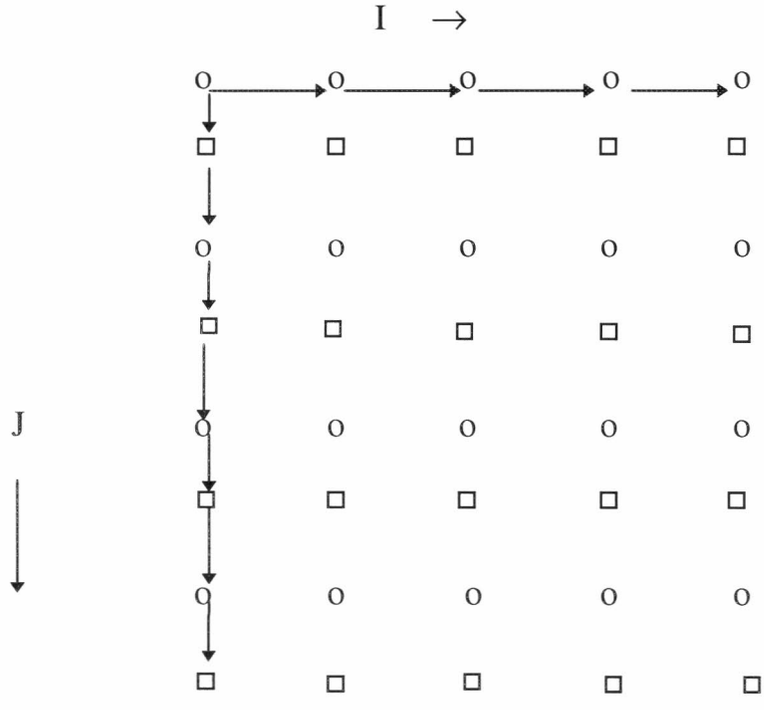


Fig. 6.2 - dos cadenas que avanzan en una dimensión.

6.2.1 Cadenas de dependencias que avanzan en una dimensión

Notación : W_i (R) es el peso de la recurrencia R en la dimensión i. Por lo tanto W(R) = (W_1 (R),W_2 (R),...,W_i (R),...,W_n (R)) para un espacio de iteración de n dimensiones.

Definición : Dada una recurrencia R de un grafo de dependencias de un bucle de n dimensiones (n>1) y x sentencias, diremos que R es una *recurrencia base* si R es una recurrencia hamiltoniana y existe un único i, tal que W_i (R)=1 $\wedge$ W_j(R) = 0 $\forall$ j $\neq$ i, 1$\leq$i,j$\leq$n.

Ejemplo 6.1

Sea n=4, dimensiones de un espacio de iteración, de 4 sentencias. Son *recurrencias base* las siguientes:

$W(R_1) = (0,0,0,1)$	y	$\lvert R_1 \rvert = 4$
$W(R_2) = (0,0,1,0)$	y	$\lvert R_2 \rvert = 4$
$W(R_3) = (1,0,0,0)$	y	$\lvert R_3 \rvert = 4$
$W(R_4) = (0,1,0,0)$	y	$\lvert R_4 \rvert = 4$

Teorema 6.1 : La longitud de la cadena de dependencias más larga formada por una *recurrencia base* en el espacio de iteración para un bucle de n sentencias, es $\mathbf{n*I_i}$, donde I_i es el espacio de iteración de la dimensión de avance de la cadena.

Demostración : Calculemos la longitud de la cadena de dependencias formada por la *recurrencia base* en el espacio de iteración por el método simplex.

Máx. $n * X$

S.a. $X \leq I_i \qquad X \geq 0$

donde I_i es el espacio de iteración de avance de la cadena.

Entonces el máximo valor que puede tener X es I_i , por lo tanto la longitud de la cadena es **$n * I_i$.**

Definición : Llamamos *Cadena base* a la cadena formada por una recurrencia base.

Teorema 6.2 : Dado un bucle de dimensión mayor a 1, cuya cadena de dependencias más larga en su espacio de iteraciones es una *cadena base* que avanza en la dimensión i, entonces el paralelismo promedio del bucle es el producto de todos los espacios de iteración correspondientes a las dimensiones donde **no** avanza la cadena.

$$//prom. = \prod_{j=1}^{n} I_j \qquad \forall\, j \neq i$$

Demostración :

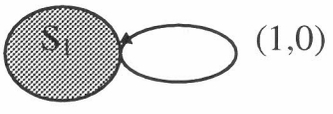
$$// \text{ Prom.} = \frac{Tseq}{T_{cad.base}} = \frac{n * I_1 * I_2 * \ldots * I_n}{n * I_i} = \prod_{j=1}^{n} I_j \qquad \forall\, j \neq i$$

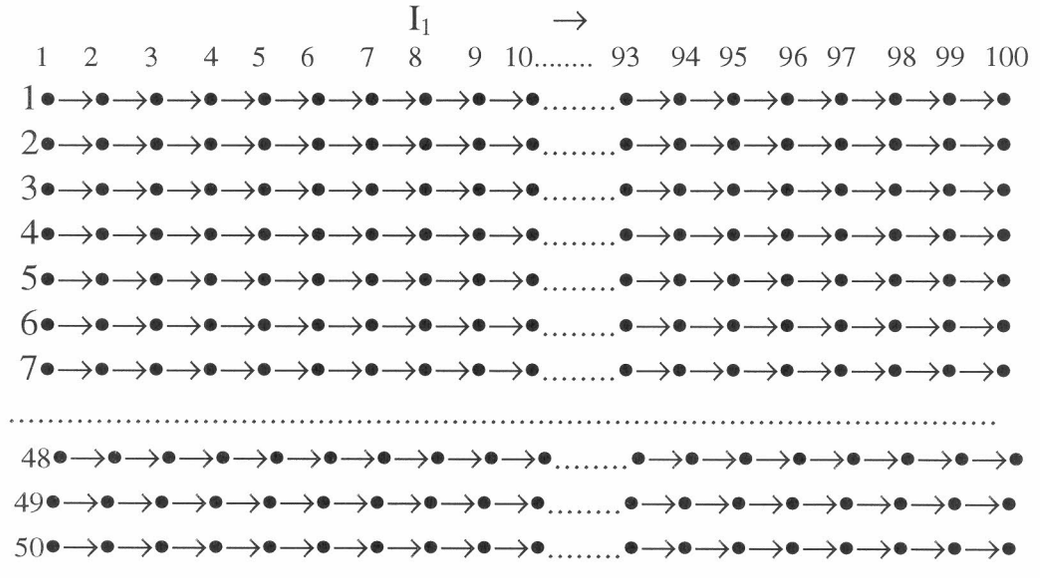
Si las sentencias de la cadena más larga que avanza en una sola dimensión **no** son todas las del bucle (no es cadena base), el n del denominador sería menor y por lo tanto el paralelismo más grande. Si el peso de la recurrencia para esa dimensión es mayor a 1, en lugar de I_i tenemos que poner I_i / $W_i(R)$. Es por esto que los resultados de paralelismo promedio de esos bucles oscilan alrededor de 100, que es el espacio de iteración en la mayoría de ellos.

Si la cadena de dependencias más larga avanza en una sola dimensión entonces el bucle tiene dependencias de datos en una sola dimensión y **no** en las otras. A veces es posible aplicar técnicas de intercambio de bucles que ayudan a explotar el paralelismo mediante la generación de construcciones DOALL en los bucles más externos, donde no hay dependencias. Cuanto más grande sea el espacio de iteración de los índices donde no hay dependencias, mayor será el paralelismo.

Ejemplo 6.2

```
DO I1=1,100
      DO I2=1,50
         A(I1,I2)=A(I1-1,I2)+B(I1)
      ENDDO
ENDDO
```





Por lo tanto $// \text{Prom.} = \dfrac{T_{seq}}{T_{cad.mas\ larga}} = \dfrac{n * I_1 * I_2}{n * I_1} = \dfrac{1*100*50}{1*100} = 50$

Si las iteraciones de las dimensiones de un bucle, que determinan el espacio de iteración, son distintos, podemos decir:

Teorema 6.3 : Para los grafos de los bucles que sólo tienen una recurrencia con dependencias en sólo una de las dimensiones del bucle entonces la cota inferior del paralelismo para esos bucles es el generado por la cadena base en la dimensión de mayor iteración.

Demostración : Tenemos que ver que la longitud de la cadena base que avanza en la dimensión de mayor iteración es mayor que la longitud de cualquier cadena que avance en solo una de cualquiera de las dimensiones del bucle.

Por el teorema 6.1 tenemos que la longitud de una cadena base es $n*I_i$ donde I_i es el número de iteraciones por donde avanza la cadena y n la cantidad de sentencias del bucle. Como es el mayor, llamemos I_M a ese número de iteraciones. Luego la longitud es $n*I_M$.

La longitud de otra cadena que avanza en solo una dimensión del bucle viene dado por la siguiente generalización

Máx. $\eta*X$ S.a. $w*X \leq I$

donde η es la cantidad de sentencias de la recurrencia y $\eta \leq n$.

w es el peso de la recurrencia en la dimensión del espacio de iteración I

La longitud de esta cadena es $\eta*I/w$.

Como $w \geq 1$ y $\eta \leq n$ e $I \leq I_M$ $\Rightarrow$ $n*I_M \geq \eta*I/w$.

Ejemplo 6.3

Sea un bucle de tres dimensiones y tres sentencias, con dependencias en sólo una de sus dimensiones que forman una recurrencia, y los espacios de iteración son I=15, J=50 y K=100.

entonces, el paralelismo mínimo esperado para este bucle es

$$//prom.= \frac{n*I*J*K}{n*K} = I*J = 15*50 = 750$$

6.2.2 Cadenas de dependencias que avanzan en más de una dimensión

Ahora vamos a tratar a las cadenas de dependencias más larga que avanzan en más de una dimension, es decir, que la o las recurrencias que intervienen en la cadena tengan $W_i(R)>0$ para más de una de las dimensiones y $W_i(R)=0$ para el resto, como en el caso de la Fig. 6.3.

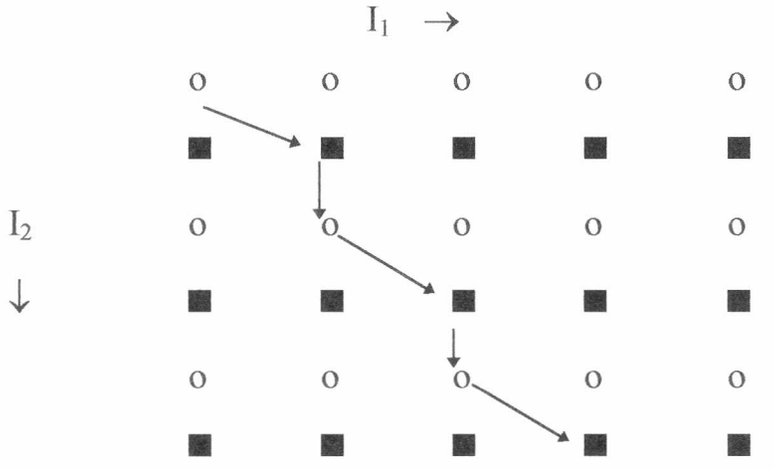


Fig. 6.3 - cadena que avanza en dos dimensiones.

En este caso podemos decir que el paralelismo promedio del bucle tiene una cota límite inferior para ese tipo de cadenas.

Teorema 6.4 : Para los grafos de los bucles que tienen recurrencias que forman la cadena de dependencias más larga en el espacio de iteraciones y que avanza en más de una dimensión, la cota inferior del paralelismo es el generado por la cadena base en la dimensión de mayor iteración.

$$//prom.mín = \prod_{j=1}^{n} I_j \qquad \forall\, j \neq i$$, siendo i la dimesión del mayor número de iteraciones del bucle.

Demostración: (para dos dimensiones)

Solo necesitamos demostrar que cualquier cadena de dependencias formada por una o más recurrencias donde sus pesos son mayores que 0 es menor que la cadena base que avanza en la dimensión de mayor iteración del bucle.

Si llamamos I_M al valor de mayor iteración que exista en el bucle para alguna dimensión, la longitud de la cadena base que avanza en esa dimensión es $n*I_M$. En el caso de 2 dimensiones, $I_M = Máx.(I_1,I_2)$.

Ahora planteemos el problema para el caso donde existan recurrencias con $W(R_i) > 0$ para más de una dimensión. Entonces el planteo para hallar la cadena de dependencias más larga, con n recurrencias, es

Máx $\quad n_1 X_1 + n_2 X_2 + \ldots\ldots + n_n X_n \quad (\mathbf{1}) \qquad\qquad n_i \leq n$

S.a. $\quad w_{11} X_1 + w_{21} X_2 + \ldots\ldots + w_{n1} X_n \leq I$

$\qquad w_{12} X_1 + w_{22} X_2 + \ldots\ldots + w_{n2} X_n \leq I_M$

$X_i, w_{ij} \geq 0 \qquad i=1,n \; ; \; j=1,2$

Quiero ver que se cumple (**1**) $\leq n*I_M$

Luego, razonando por el absurdo, supongamos que (**1**) $> n*I_M$

tomando a cada n_i de (**1**) igual a n (en este caso obtendríamos el "mayor" máximo de (**1**)), tenemos que

$$(\mathbf{1}) = n*(\sum X_i) > n*I_M \quad \Rightarrow \quad \sum X_i > I_M$$

Por una de las restricciones tenemos

$w_{12} X_1 + w_{22} X_2 + \ldots + w_{n2} X_n \leq I_M$

En particular si $w_{i2} = 1 \qquad \forall\, i = 1,n$

cumple que $\quad X_1 + X_2 + \ldots + X_n \leq I_M$

$\Rightarrow \qquad \sum X_i \leq I_M$

entonces tenemos que $\sum X_i \leq I_M$ y $\sum X_i > I_M$ absurdo.

Por lo tanto, $\boxed{n\, I_M \geq (\mathbf{1})}$

$$//\text{prom.mín} = \frac{Tseq}{Tcad.} = \frac{n*I*I_M}{n*I_M} = I$$

Ejemplo 6.4

Dado un bucle de dos dimensiones con 5 recurrencias en su grafo, $W(R_k)=(w_i,w_j)$ con $w_i,w_j > 0$ para todo R_k con k=1,5 ,y los límites de iteración $I_1=15$ e $I_2=100$.

Entonces, el paralelismo promedio mínimo esperado, aplicando el teorema 6.4 para este bucle, es 15.

Estos tipos de cadenas son los que en los bucles de 2 y 3 dimensiones explican el alto grado de paralelismo que hay cuando este es distinto de infinito. Por otro lado, depende de los espacios de iteración de esos bucles, que en casi todos los casos son de 100.

6.3 La influencia del tiempo estimado de ejecución

Podemos decir que, teniendo en cuenta el tiempo estimado de ejecución[2] de las sentencias no se producen grandes modificaciones de conjunto en la muestra de acuerdo a las mediciones realizadas (ver apéndice D).

Bucles de 1 dimensión

En los bucles donde sí encontramos diferencias, confirmamos que si la sentencia de mayor peso forma parte de la cadena de dependencias más larga, entonces el grado de paralelismo disminuye. Como ejemplo mostraremos el grafo n° 32 del SRSI de 3 sentencias (S_1,S_2 y S_3) (apéndice D) y un espacio de iteración de 2 a 100 con un peso de 3,1 y 15 respectivamente.

```
      DO 5 L = 2,LMAX
        K = K + 1
        GN = (1+EPS/SQRT(FLOAT(K+4)))**4
        R(L) = R(L-1) + DYM * GN
5     CONTINUE
```

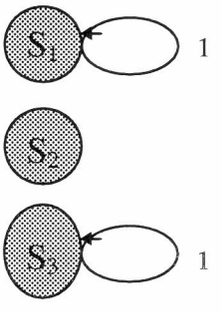


Fig. 6.4

[2] La estimación de tiempo es 1 = asignaciones, 2 = sumas y restas, 4 = productos, 100 = call´s.

Si tomamos a las 3 sentencias como idénticas en el tiempo de su ejecución, el paralelismo promedio es Tseq/Tp = 99*3/99 = 3

En cambio, si consideramos el tiempo estimado de ejecución expresado por las etiquetas 3,1 y 15 (t_1, t_2 y t_3), la cadena de dependencias más larga es la formada por la recurrencia R_{33} que surge de la dependencia d_{33} (S_3 --> S_3) de distancia 1, pues debido al peso de S_3 esta cadena tiene una longitud de 15*99 en todo el espacio de iteración.

En el caso de la Fig. 6.4 era indistinto formar la cadena más larga tanto con la recurrencia R_{11} como con la R_{33}, sin embargo si tomamos en cuenta ese tiempo esperado de ejecución que simbolizamos a través de las etiquetas, vemos que eso ya no es así.

Ahora el nuevo paralelismo promedio viene dado por el cociente entre $Tseq = \sum t_i * Ei$ y $Tp = t_3 * E_i$, es decir 19 / 15 = 1.27.

El otro caso que confirmamos es aquel en que la sentencia de mayor tiempo de ejecución no forma parte de ninguna recurrencia y por lo tanto no forma parte de la cadena de dependencias más larga. En este caso el tiempo de ejecutar esa cadena es, en proporción al caso en que no tomamos en cuenta estos pesos, mucho menor y el cociente entre el tiempo de ejecución secuencial y el tiempo de ejecución de la cadena más larga es mayor. Como en el grafo n°125 del programa BDNA que tiene 4 sentencias y un espacio de iteración de 100. Este grafo solo tiene una recurrencia formada por la dependencia d_{11} de distancia 1 (Fig. 6.5). La cadena de dependencias más larga siempre será formada por esta única recurrencia.

En el caso donde no tomamos el peso, tenemos que Tseq =4 * 100 = 400 y Tp = 1 * 100 , luego el paralelismo promedio es 4.

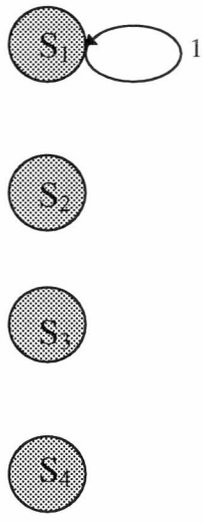


Fig. 6.5

En el caso contrario tomamos los pesos 3,111,111,111 para S_1,S_2,S_3 y S_4 respectivamente. Luego tenemos que Tseq = 336*100 = 33600 y Tp = 3*100 = =300 con lo que el paralelismo promedio da 33600 / 300 = 112.

Bucles de 2 dimensiones

Tampoco se han observado muchas modificaciones para los bucles de 2 dimensiones, sin embargo, de haberlas el valor puede cambiar considerablemente. Esto se debe a que se incrementa el espacio de iteración al ser bucles de más de una dimensión y el hecho de que la sentencia con más peso sea o no parte de la cadena de dependencias más larga resulta más importante. Por ejemplo el grafo n° 12 del LWSI (apéndice D) pasa de 350 a 15200. Ello se debe a que solo existen dependencias entre 4 de las 7 sentencias del bucle, teniendo estas 4 un peso de 1 cada una. Las otras 3 que no tienen dependencias con ninguna otra tienen un peso de 100 cada una. Estas 3 sentencias llaman a funciones de biblioteca. Las otras 4 son de operaciones simples.

6.4 Resultados de los tiempos de ejecución de los bucles.

En 10 programas de la suite Perfect Benchmarks ejecutados se encuentran un total de 1249 bucles, de los cuales 934 han sido usados por los programas con las entradas que disponíamos.

De los bucles usados se ha contabilizado que 757, un 81%, han sido ejecutados en hasta un 1% del tiempo total de ejecución del programa en que intervenían esos bucles,es decir que el porcentaje de tiempo de uso de esos bucles en su programa de ejecución varía en un rango entre el 0 y 1%.

De lo anterior concluimos que la mayor parte de los bucles que hemos estudiado consumen un tiempo ínfimo en relación con el tiempo total de ejecución del programa en el que intervienen.

Sin embargo existen algunos bucles que pueden ser de interés dado que el tiempo de ejecución de los mismos es superior al 50% del tiempo total de ejecución del programa al que pertenecen. Estos bucles son en general bucles del cuerpo principal del programa y tienen sentencias que llaman a su vez a otras subrutinas que contienen también varios de los bucles estudiados. La excepción es el bucle 159 del programa BDNA que pertenece a una subrutina invocada 2 veces en el programa y contiene 310 sentencias. Además, este bucle es usado 7458 veces dentro de otro bucle que lo incluye entre otras sentencias.

En el próximo punto efectuaremos comparaciones entre el tiempo de ejecución de los bucles y el grado de paralelismo de los mismos, en aquellos casos en que fueron evaluados.

6.5 Medidas del grado de paralelismo comparadas con las medidas del tiempo de utilización de los bucles.

Al analizar las medidas de utilización de los bucles en los distintos programas hemos visto que la mayoría de ellos son bucles que solo consumen entre el 0 y el 1% del tiempo total de ejecución del programa. Sin embargo también vimos que existen algunos bucles *en algunos programas* que consumen mucho tiempo de ejecución del tiempo total, como por ejemplo el **bdna, dyfesm, lwsi, qcd2, srsi** y el **tisi** que tienen bucles de más del 25% del tiempo total de ejecución de su programa.

Lamentablemente, en algunos de esos bucles no pudimos calcular su grado de paralelismo, pues no se correspondían con las características de los bucles analizados (**bdna**, **lwsi**, **srsi, tisi**) porque son bucles con distancias de dependencias desconocidas, sentencias condicionales y de entrada/salida.

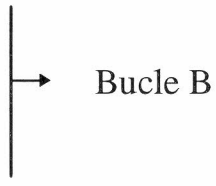
Definición : Dados dos bucles A y B de un programa secuencial, decimos que B es *independiente* de A, si en la ejecución del bucle A no está contenida la ejecución del bucle B. De lo contrario decimos que B es *dependiente* de A (Ej. 6.5)

Ejemplo 6.5

```
PROGRAM
.....
DO I=1,100
     A=A+1
     CALL SUB1
ENDDO
.....
.....
.....
END
.....
```

→ Bucle A

```
SUBROUTINE SUB1
.....
DO K=1,12
      B(K+1)=VAR(K)
      C(K+1)=B(K)
ENDDO
.....
END
```



En un conjunto de programas, como el **apsi**, **flo, srsi**, **lwsi** o el **wssi,** existe un conjunto de bucles que consumen entre el 1 y el 5% o entre el 5 y el 25%. Estos, de ser paralelizables e *independientes* se convierten en bucles con una importancia mucho mayor en el programa. En particular, en el **flo** y en el **srsi**, muchos de ellos son de 2 dimensiones. El único programa en el cual los bucles no juegan prácticamente ningún rol es en el **track** que solo tiene 2 bucles entre el 5 y 7%, ambos secuenciales.

Veamos, en números, algunos de estos diferentes casos.

DYFESM

En este programa encontramos un bucle (n° 60 - ver apéndice E) que ejecuta el 99,86% (588,37 seg.) del tiempo total ejecución del mismo. Este bucle tiene un paralelismo promedio igual a 4 (apéndice D). Si suponemos costos de sincronización y reparto de tareas despreciables, este bucle podría ejecutarse, con 4 procesadores, en el 24,96% del tiempo total de ejecución del programa (147,09 seg.).

Tiempos de ejecución

	Secuencial	Paralelo
del bucle	588,37 seg.	147,09 seg.
del Pgm.	589,20 seg.	147,92 seg.

Por otro lado existen otros bucles como el n°74 o el n°84 (ver en el apéndice E) que son también importantes y forman parte del tiempo de ejecución del bucle 60 y que pueden ayudar a disminuir el tiempo de ejecución de este. En particular el bucle 74, que es doall.

QCD2

En este programa encontramos 3 bucles con un tiempo de utilización mayor al 25% del tiempo total del programa (ver apéndice E). No obstante, 2 de ellos son bucles secuenciales, y el otro tiene un valor de paralelismo desconocido. Esto nos impide saber si, a través, del último bucle podemos disminuir el tiempo de ejecución del programa. Existen también 3 bucles que utilizan cada uno, el 19% del tiempo total de ejecución del programa, pero 2 son bucles secuenciales y el tercero de valor de paralelismo desconocido . Solo hay un bucle (n°12) que es doall (apéndice D) y representa el 12% del tiempo de ejecución total (apéndice E).

LWSI

Este programa tiene un bucle, en su cuerpo principal, que ocupa el 99,08% del tiempo total de ejecución del programa (ver Apéndice E), pero su valor de paralelismo es desconocido. Sin embargo existen otros bucles con un tiempo de ejecución que sumados representan un porcentaje importante. Por ejemplo, en la rutina **Interf** de ese programa existen 8 bucles que consumen, entre todos, 36,83 % del tiempo total (esta rutina es llamada por el bucle arriba nombrado, entendiéndose que este 36,83 % forma parte del 99,08%). Cinco de

ellos son doall y otro de paralelismo promedio igual a 4 (los otros 2 son de valor desconocido) (apéndice D). Si contamos con estos 6 bucles de paralelismo conocido, el tiempo de ejecución del programa puede verse reducido en un 22,81%. En segundos sería pasar de 3571,56 (tiempo total) a 2756,67.

Tiempos de ejecución

	Secuencial	Paralelo
bucle nº 26 (//=4)	303,16 seg	75,79 seg
5 bucles de // inf.	587,52 seg	valor despreciable.
de esos 6 bucles	890,68 seg	75,79 seg
del programa	3571,56 seg.	2756,67 seg

FLO

Aquí encontramos 6 bucles (nº 32,35,53,55,75,79 del apéndice E) que consumen entre el 1% y 3 % del tiempo total (paralelismo promedio entre ellos de 182 -apéndice D-) y 2 que se ejecutan en un 13% cada uno y de paralelismo promedio de 223 (nº 37,40 -apéndice D-). Todos estos bucles son de dos dimensiones. Si ellos fuesen *independientes* en sus ejecuciones (los de 13% lo son) entonces. el programa se reduce en un 35,93% (584,02 seg a 374,18 seg.). Este es un caso (como también en el **srsi**) donde los bucles de 2 dimensiones juegan un rol destacado.

Tiempos de ejecución (en porcentajes)

	Secuencial	Paralelo	
bucle nº 37 y 40 (//=223)	26,60%	0,12%	(26,6/223)
6 bucles de (//=182, en prom.)	9,50%	0,05%	(9,5/182)
	36,10%	0,17%	
del programa	100,00%	64,07%	
Reducción del 35,93%			

Si lo hacemos para 4 procesadores tenemos

	Tiempos de ejecución (en porcentajes)		
	Secuencial	Paralelo -4 proc.	
bucle nº 37 y 40	26,60%	6,65%	(26,6/4)
6 bucles	9,50%	2,38%	(9,5/4)
	36,10%	9,03%	
del programa	100,00%	72,93%	
Reducción del 27,07%			

En los programas se encuentran entre 1 y 3 bucles que son muy importantes por el tiempo que ocupan en relación al programa, por ello nosotros enfocamos nuestro análisis en ese conjunto. También hemos visto que hay programas que tienen varios bucles de una importancia relativa, pero que, sumados pueden, de ser *independientes*, consumir mucho tiempo de su programa. En este último aspecto cabe mencionar a aquellos que son perfectamente anidados de 2 dimensiones.

6.6 Temas abiertos

Este trabajo deja abierta algunas propuestas para un futuro desarrollo:

a) Estudio del paralelismo de programas secuenciales no numéricos.

b) Estudio del paralelismo de bucles multidimensionales no perfectamente anidados.

c) Mediciones del paralelismo de los bucles por medio del profile.

d) Realización de medidas de paralelismo utilizando una máquina virtual paralela.

7-ASPECTOS DE IMPLEMENTACION

7.1 Requerimientos de hardware.

Nuestro trabajo fue desarrollado en la red de Docentes de la Facultad de Ciencias Exactas de la Universidad de Buenos Aires, ingresando a la misma a través del nodo ZORZAL. El sistema operativo utilizado fue: SunOS 4.1.3 (Unix) y la características técnicas del nodo son las siguientes: equipo Sun SPARC Station 10, 32 Mb RAM.

7.2 Lenguajes utilizados - Compiladores.

Utilizamos el lenguaje C para el desarrollo del programa que calcula el paralelismo promedio de los programas, usando el compilador de C provisto por Sun que se encuentra disponible para cualquier usuario de la red en los directorios /usr/5bin y /usr/xpg2bin .

Los programas de la Suite Perfect Benchmarks y el programa Simplex fueron escritos en lenguaje Fortran y compilados con el Compilador Fortran de Sun que se encuentra en el directorio /appl/fortran de Zorzal.

Los archivos de grafos de los programas fuentes fueron generados por el compilador PARAFRASE II.

Los archivos de datos del Perfect Benchmarks, asi como los fuentes Fortran fueron suministrados por el Center for Supercomputing Research and Development - University of Illinois at Urbana Champaign - EEUU.

8-BIBLIOGRAFIA

BIBLIOGRAFIA

[AL89] Eduard Ayguadé i Parra, Jesús Labarta Mancho. *"Paralelización Automática de recurrencias en programas secuenciales numéricos"* Depto. de Arquitectura de computadoras, Universidad Politécnica de Cataluña. Octubre 1993.

[ALT93] Eduard Ayguadé, Jesús Labarta, Jordi torres. *"Compile-time parallelism evaluation in a loop-parallelizing enviroment"*.Depto. de arquitectura de computadores, Universidad Politécnica de Cataluña.Barcelona 1993

[BLB93] Patricia Borensztejn,Jeús Labarta, Cristina barrado.*"Measures of parallelism at compile time"*.Depto.de Arquitectura de computadores, Universidad Politécnica de Cataluña.Barcelona 1993

[BLB94] Patricia Borensztejn,Jeús Labarta, Cristina barrado.*"Measures of parallelism (II)"*.Depto de Computación - Fac.Cs.Exactas y Naturales. Universidad Nacional de Buenos Aires (Arg.).Depto.de Arquirectura de Computadores de la Universidad Politécnica de Cataluña.(Esp).Buenos Aires 1994.

[CP91] Doreen Y. Cheng, Douglas M.Pase. *"An evaluation of Automatic and Interactive parallel Programing Tools"*.Julio 1991

[CY91] Ding-Kai Chen, Pen Chung Yew.*"An empirical study on Do Across loops"*.Center for supercomputing Research and Development. Universidad de Illinois at Urbana-champaign.1991.

[HB88] Kai Hwang, Fayé A. Briggs. *"Arquitectura de computadores y procesamiento en paralelo"*.McGraw Hill. Mexico 1988.

[KKP+81] D.J. Kuck, R.H. Kuhn, D.A. Padua, B. Leasure and M. Wolfe, *"Dependence Graphs and Compiler Optimizations"*.Proc. of the 8th ACM Symposium on Principles of Programing Languages (POPL'81), Williamsburg,January 1981, pp.207-218.

[K93] Lyle Kipp,*"Perfect Benchmarks Documentation-suite 1"*.Center for Supercomputing Research and Development .University of Illinois at Urbana Champaign.Octubre 1993.

[P89] L.Pointer,*"Perfect Report:1"*, CSRD Report N 896, Center of Supercoputing Research & development, Universidad de Illinois at Urbana Champaign, JL, july 1989.

[PGH+89] C.D. Polychronopoulos, M. Girkar, M.R. Hadhighat, C.L. Lee, B.Leung and D.Schouten. *"Parafrase 2: An Envirenment for Parallelizing, Partitioning, Synchronizing and Schduling Programs on Multiprocessors"*.Proc. of the International Conference on Parallel Processing, vol II.. pp 39-48,1989.

[PW86] D.A. Padua and M.Wolfe,*"Advanced Compiler Optimizations for Supercomputers"*, Communications of the ACM,December 1986,vol.29, N0.12,pp.1184-1201.

[S89] Sevcik K. C.. *"Characterizations of parallelism in Aplications and their use in scheduling"*.ACM.Performance Evaluation Review. pp 171-181. (1989).

[ZCh90] Hans Zima, Barbara Chapman. "*Supercompilers for parallel and vector computers"*. 1990.