Encolhimento de ciclos por redução de dependência

Kunio Okuda

TESE DEFENDIDA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU
DE
DOUTOR EM MATEMÁTICA APLICADA

Área de Concentração: **Computação**
Orientador: **Prof. Dr. Siang Wun Song**

-São Paulo, novembro de 1996-

Encolhimento de ciclos
por redução de dependência

Este exemplar corresponde à redação
final da tese devidamente corrigida
e defendida por Kunio Okuda e aprovada
pela Comissão julgadora

São Paulo, dezembro de 1996

Banca Examinadora:

- Prof. Dr. Siang Wun Song (Orientador)-IME-USP
- Prof. Dr. Edil Severiano Tavares Fernandes
- Prof. Dr. Valmir Carneiro Barbosa
- Prof. Dr. João Carlos Setubal
- Prof. Dr. Jairo Panetta

às
L,
S,
H
e N

Agradecimentos

Ao meu orientador Prof. Dr. Siang Wun Song por tudo. Esta obra não ficaria pronta se não fosse sua incrível paciência, disposição e sabedoria.

Ao Prof. Dr. Arnaldo Mandel, meu primeiro orientador no MAC. A sua orientação academica foi de curtíssimo periodo mas hoje orientador na escolha de restaurantes de comida apimentada.

Ao Prof. Dr. Francisco Cesar Polcino, meu orientador quando eu era algebrista e hoje meu amigo de sempre.

Ao Carlinhos pela valiosa ajuda nos primeiros dias da formalização das minhas idéias.

Aos membros de GCPD/DCC/IME/USP pelo estímulo dado.

À Yo, pelas seções de chá.

Às secretárias do MAC pela paciência.

Aos membros de fofomac pelas fofocas internacionais quentes.

Aos membros de ON pelas noites agitadas no fim de semana.

Ao mestre Sassaki pelo seu treino e ensinamento.

Aos colegas animados de karate pela companhia.

Às minhas musas vestidas de preto pelo estimulo dado, em especial Nilsa, Celma, Raquel e Paula.

Sumário

No campo de computação paralela, a estrutura de laços encaixados tem uma grande importância pelo seu potencial de paralelização. Dentro desta estrutura, ciclos de dependência de fluxo apresentam restrições para paralelização. Encolhimento de ciclos é uma técnica para paralelizar laços com tais ciclos. Este trabalho propõe novos métodos de encolhimento de ciclos, com granularidade fina, para computadores paralelos com arquitetura de memória distribuída. Os novos métodos apresentam várias vantagens em relação a outros. Eles se baseiam numa transformação de grafo de dependência com diversos resultados apreciáveis. A redução de tempo total de execução é o mais importante deles. Outros resultados são a redução drástica de comunicações entre processadores, a análise mais simplificada de escalonamento e a eliminação de "gargalos" de comunicação inerentes aos algoritmos sem alterar dependências implícitas. Apresentamos primeiro a técnica *redução de dependência*. Ela procura reduzir o número de passos na execução paralela e o número de comunicações entre processadores. A seguir apresentamos *redução de dependência parcial*, que visa balancear computação e comunicação. No fim desenvolvemos uma extensão do método, *redução de dependência generalizada*, para paralelizar os algoritmos mais gerais. Estes métodos novos são descritos e ilustrados pelo conceito de *domínio explícito*, uma nova representação de dependências que ilustra as dependências de modo mais claro que as outras representações.

Abstract

In the area of parallel computing, nested loop structures are of great importance for their potential for parallelization. Among these structures, flow dependence cycles present more problems to be parallelized. Cycle shrinking is a technique to parallelize loops with such cycles. This work proposes new cycle shrinking methods of fine grain, for parallel computers with distributed memory architecture. The new methods present several advantages over other methods. They are based on a dependence graph transformation with several appreciable results. The most important is the reduction of the total execution time spent. Furthermore, we obtain a substantial reduction of communication between processors and a simpler analysis for scheduling. We also eliminate certain communication bottlenecks inherent in the algorithms without altering the implicit dependences. We first present a new technique called *dependence reduction*. It attempts to reduce the number of parallel execution steps and amount of communication between processors. We then present *partial dependence reduction* that aims balancing computation and communication. Finally we develop an extension of the method called *generalized dependence reduction*. It is capable of parallelizing more general algorithms. These new methods are described and illustrated through the *explicit domain* concept, a new dependence representation that illustrates dependence in a clearer way than other representations and gives an intuitive base for the dependence reduction method.

Conteúdo

1 Introdução **1**
1.1 Introdução 2
1.2 Estrutura da tese 5

2 Definições e nomenclatura **6**
2.1 Modelo computacional 7
2.2 Formalização e terminologia 7
2.3 Escalonamento 11
2.4 Mapeamento 12

3 Métodos conhecidos para encolhimento de ciclos **14**
3.1 Método GSS 15
3.1.1 Encolhimento de ciclo seletivo 15
3.1.2 Encolhimento de ciclo seletivo generalizado (GSS) 16
3.2 Método ISM e GSS combinado com ISM 25
3.2.1 Método ISM 25
3.2.2 Melhorando ISM 27
3.2.3 GSS combinado com ISM 28
3.3 Outros métodos 31
3.3.1 Métodos encolhimento por dependência verdadeira e encolhimento por dependência verdadeira generalizada 31
3.3.2 Método afim por comando 33

4 Novos métodos por redução de dependência **35**
4.1 Domínio explícito 36
4.2 Exemplos 37
4.2.1 Primeiro caso 37
4.2.2 Segundo caso 40
4.2.3 Terceiro caso 44
4.3 Considerações sobre conexidade 46
4.4 Redução de dependência (RD) 49
4.4.1 Grafo de dependência igual a um ciclo 49
4.4.2 Grafo de dependência com mais de um ciclo 50

4.4.3 Efeito da transformação sobre a escolha de π quando o grafo é reduzido a um vértice 55
4.4.4 Aplicabilidade 56
4.5 Redução de dependência parcial 58
4.5.1 Exemplo 58
4.5.2 Redução de dependência parcial 58
4.6 Redução de dependência generalizada 62
4.6.1 Exemplos 62
4.6.2 Definição formal de redução de dependência generalizada (RD generalizada) 68
4.6.3 Aplicabilidade 71

5 Conclusão **72**
5.1 Conclusão 73
5.2 Contribuições 74
5.3 Futuras direções 75

A Detalhes do GSS **76**
A.1 Cálculo de π_0 para o Exemplo 1 (página 21) 76
A.2 Cálculo de π_0 para o novo grafo do Exemplo 1 (página 44) 76

B Lista de símbolos **78**

Capítulo 1

Introdução

- A seção 1.1 começa com uma breve descrição do assunto desta tese. Comentamos os trabalhos relacionados e faremos algumas considerações iniciais.
- Na seção 1.2 descrevemos as estruturas gerais desta tese e o conteúdo.

1.1 Introdução

No campo de computação paralela, as estruturas de laços (*loops*) encaixados são as estruturas que oferecem ricos paralelismos implícitos. Elas são também as estruturas que têm muitas aplicações principalmente nas computações científicas. Podemos observar a sua importância nos livros clássicos sobre compiladores para supercomputadores como por exemplo os de Wolfe [Wolfe89] e de Polychronopoulos [Poly88-2].

Entre a subclasse de estrutura de laços encaixados uma estrutura bastante importante é a de laços uniformes. Darte e Robert realçam sua importância em [DarR95] onde eles citam vários artigos relacionados a este tema.

Os laços uniformes oferecem várias facilidades para sua paralelização.

Os conceitos e técnicas de escalonamento e mapeamento de algoritmos em laços uniformes surgiram originalmente no contexto de síntese de algoritmos sistólicos [Song84, Moo86, CosR86, Kung88, Okuda89, Robe90]. Dado um algoritmo em forma de laços uniformes o método procura obter um esquema de circuito VLSI mais eficiente para executar este algoritmo [MolF86, QuiR89, DarRR91]. Em seguida as idéias surgidas neste campo emigraram para contexto mais geral de paralelização de laços para execução em uma rede de processadores e tiveram novos desenvolvimentos [Darte91, DarR92, DarR94, DarR94-2, DarR95].

Dentro da classe de laços uniformes, por sua vez, o ciclo de dependências é aquele que causa mais dificuldade. No artigo de Konda e Kumer [KonK95] que trata de remover vários tipos de dependências, eles afirmam que o mais difícil de paralelizar é o ciclo formado apenas por dependências de fluxo.

Encolhimento de ciclo (*cycle shrinking*) é uma técnica para paralelizar laços que apresentam tais ciclos de dependência. Várias técnicas de transformação de laços do tipo encolhimento de ciclo para extrair paralelismo foram propostas [Poly88-2, Wolfe89].

Encolhimento de ciclo simples (*simple cycle shrinking*), *encolhimento de ciclo seletivo* (*selective cycle shrinkig*) e *encolhimento de ciclo por dependência verdadeira* (*true dependence cycle shrinking*) foram introduzidos por Polychronopoulos [Poly88]. Estes métodos transformam laços seqüenciais em laços paralelos. *Encolhimento de ciclo seletivo* procura um laço a partir do qual todos os laços internos a ele possam ser executados em paralelo (em forma de "*doall*"). Alguns casos deste método podem ser interpretados do seguinte modo: um vetor paralelo a algum dos eixos no espaço de coordenadas que representa o conjunto de índices do algoritmo é usado como vetor de escalonamento. O método apresenta certas limitações e não se aplica mesmo para algoritmos que têm paralelização óbvia. Esta limitação levou Shang, O'Keefe e Fortes a propor sua generalização: *encolhimento de ciclo seletivo generalizado* (*generalized selective cycle shrinking* (GSS)) [ShaOF91, RobS92]. A generalização permite uma maior liberdade na escolha do vetor de escalonamento, i.e., o vetor de escalonamento não precisa estar paralelo a algum dos eixos desde que sejam satisfeitas certas condições. Ele particiona o conjunto de índices em hiperplanos paralelos e todos os pontos situados no mesmo hiperplano são executados ao mesmo tempo. O método também é conhecido como método de hiperplano e muito usado na síntese de algoritmos sistólicos [QuiR89, MolF86].

Tivemos dois interessantes casos particulares do método GSS: *encolhimento de ciclo por dependência verdadeira* e *encolhimento por dependência verdadeira generalizada*. O método

de *encolhimento de ciclo por dependência verdadeira* transforma o espaço n-dimensional de um algoritmo com n laços em um espaço 1-dimensional cuja ordem total é a ordem lexicográfica do algoritmo. Ele calcula o número de iterações para cada dependência (*dependência verdadeira*) e a paralelização é feita com o mínimo destes números. Assim como encolhimento de ciclo simples, ele apresenta certas limitações. Shang, O'Keefe e Fortes propuseram a sua generalização com o método de *encolhimento por dependência verdadeira generalizada* (*generalizad true dependence shrinking*) [ShaOF91].

Por outro lado tivemos dois importantes avanços a partir do método GSS. Um deles, o método de *deslocamento de índice* (*index shift method* (ISM)) foi introduzido por Liu, Ho e Sheu [LiuHS90]. O método faz deslocamento de índices em comandos sem violar a semântica do algoritmo e torna o ciclo de dependência mais balanceado de modo a explorar mais paralelismo implícito. Ele pode ser visto como um refinamento de GSS. Porém aplicar ISM após GSS nem sempre garante resultado melhor. Robert e Song propuseram um método que combina GSS com ISM de modo mais eficiente [RobS92]. Eles deram também um interessante exemplo no qual este novo método tem fator de melhoria arbitrariamente grande em relação a GSS.

O outro é *afim por comando* (*affine by statement*) proposto por Robert e Darte [DarR92, DarR94, DarR95]. Ele é uma extensão natural de GSS. Em vez de escalonar todo corpo de laço uniforme considerando como um bloco, ele escalona separadamente cada comando que compõe o corpo a fim de obter menor número de passos na execução paralela.

Este trabalho propõe novos métodos de encolhimento de ciclo com granularidade fina para computador paralelo com arquitetura de memória distribuída. Os novos métodos apresentam várias vantagens em relação aos outros. Eles se baseiam numa transformação de grafo de dependência com diversos resultados apreciáveis. A redução de tempo total de execução é o mais importante deles. Além disso conseguimos redução drástica de comunicações entre processadores e a análise mais simplificada de escalonamento. Temos também a eliminação de certos "gargalos" de comunicação inerentes nos algoritmos sem alterar dependências implícitas.

O trabalho começa com a introdução de *domínio explícito*, uma nova representação de dependência de algoritmo. Ele ilustra as dependências de modo mais explícito que outras representações como grafo de dependência ou conjunto de índices junto com vetores de dependência. Os exemplos 2-dimensionais, apesar de simples, são ilustrativos no sentido de mostrar as dependências com maior clareza e dar uma base intuitiva do novo método denominado *redução de dependências*. Discutimos brevemente a relação entre paralelização e conexidade do grafo de dependência.

A redução de dependência é introduzida com três exemplos de modo informal através do uso de domínio explícito [Okuda95, Okuda95-2]. Este novo método para encolhimento de ciclos identifica e distingue, dentro do grafo de dependência, os vértices correspondentes aos comandos cruciais e os vértices correspondentes aos comandos não cruciais para escalonamento. O método transforma o grafo de dependência e, através da transformação correspondente ao algoritmo, define-se um macro comando para obter tempo total de execução menor. Este tempo menor é obtido com a diminuição de tempo de comunicação e aumento de tempo de computação (suposto relativamente menor em relação ao de comunicação). A sua eficiência em relação aos outros métodos é mostrada através do bem conhecido

exemplo de Peir e Cytron [PeiC89]. O método redução de dependências diminuiu para menos de metade o tempo total de execução em relação ao melhor resultado obtido até agora para este exemplo (por método afim por comando). O método reduz o número de vetores de dependência e isto resulta na análise mais simplificada de escalonamento. A seguir o método é apresentado de modo formal e analisado. Discutimos o efeito do método sobre a escolha do vetor de escalonamento quando o grafo de dependência é reduzido a um vértice. Damos uma condição suficiente para que o método seja mais eficiente em relação aos outros métodos. Uma versão reduzida deste resultado foi publicado nos Anais da Euro-Par'96 Conference, editados por Springer Verlag [Okuda96].

O trabalho inclui duas extensões de redução de dependência [Okuda96-2]. A primeira é *redução de dependência parcial.* Como redução de dependência obtém o tempo total de execução menor com diminuição de tempo de comunicação e aumento de tempo de computação, se for mal aplicado até aumenta o tempo total de execução. Damos um exemplo extremo para ilustrar este caso. A redução de dependência parcial tenta o balanceamento entre o tempo de comunicação e o tempo de computação. Mostramos quando se deve usar redução de dependência parcial em vez de redução de dependência.

A segunda extensão é *redução de dependência generalizada.* O método ataca grafos de dependência mais gerais em que não se pode aplicar redução de dependência simples. A grande vantagem do método redução de dependência generalizada é a seguinte. Ele transforma o grafo de dependência de modo que a transformação correspondente ao algoritmo elimina certos "gargalos" inerentes de comunicação no algoritmo sem violar dependência de dados original. Damos dois exemplo para ilustrar o novo método e em seguida o método é apresentado de modo formal. Novamente damos uma condição suficiente para que o método seja mais eficiente em relação aos outros métodos. Geralmente a paralelização de um algoritmo seqüencial toma como seu ponto de partida este algoritmo seqüencial e o "gargalo" inerente no algoritmo original continua presente após a paralelização. Porém o método *redução de dependência generalizada* muda o próprio algoritmo e elimina este "gargalo", o que resulta numa paralelização melhor. Resultados referentes ao método redução de dependência generalizada estão sendo preparados para serem submetidos para publicação [Okuda96-3].

1.2 Estrutura da tese

A estrutura dos capítulos subseqüentes é a seguinte:

- O capítulo 2 delimita o universo em que a tese trabalha assim como define os símbolos e a nomenclatura que serão utilizadas no resto desta tese. Esta parte de definições não é completa. Alguns novos conceitos serão introduzidos e definidos no decorrer do trabalho.
- O capítulo 3 descreve os métodos existentes para encolhimento de ciclo e correlatos.
- O capítulo 4 é a parte central deste trabalho. Introduzimos o conceito de *domínio explícito* (**DE**), uma nova representação de estrutura de dependências de algoritmo. Os exemplos vão mostrar a sua utilidade e, baseado nas observações obtidas por domínio explícito, será introduzido o método de *redução de dependência* (**RD**). A comparação deste novo método com os outros métodos é feita através do bem conhecido exemplo de Peir e Cytron [PeiC89]. Discutimos brevemente a relação entre paralelização e conexidade do grafo de dependência. Em seguida **RD** é definida formalmente e analisada. As duas últimas seções tratam de duas extensões de **RD** chamadas **RD** parcial e **RD** generalizada.
- O capítulo 5 dá a conclusão e algumas considerações finais.
- Na primeira página de cada capítulo temos uma breve descrição das seções que compõem o capítulo.
- idem no começo de cada seção se ela tiver subseções.
- O apêndice A contém detalhes de prova de alguns resultados.
- O apêndice B contém a tabela de símbolos.

Capítulo 2

Definições e nomenclatura

- Na seção 2.1 definimos o modelo computacional adotado neste trabalho.
- Na seção 2.2 definimos *laços encaixados gerais* (*general loop nest* (GLN)), domínio, dependência, GLN uniforme e *laços encaixados uniformes e regulares* (*regular uniform nest* (RUN)). Damos o exemplo de Peir e Cytron [PeiC89] que será citado freqüentemente nos capítulos subseqüentes.
- Na seção 2.3 definimos escalonamento e damos uma proposição para caracterizar escalonamento quando ele é uma função afim.
- Na seção 2.4 definimos mapeamento e trataremos também a relação com escalonamento e algumas observações relacionadas.

2.1 Modelo computacional

O modelo computacional adotado no trabalho é um computador paralelo síncrono com arquitetura de memória distribuída [Poly88-2, Wolfe89, Bert89, AmoBF88]. Neste trabalho vamos supor que o número de processadores é tanto quanto necessário. Deste modo não vamos tratar a questão de partição e seu efeito [Darte91, DarD90, MolF86].

2.2 Formalização e terminologia

Vamos considerar um algoritmo expresso como laços encaixados com a seguinte estrutura denominada *laços encaixados gerais* (*General Loop Nest* (GLN)):

GLN

for $i_1 = l_1$ to u_1 do
  for $i_2 = l_2(i_1)$ to $u_2(i_1)$ do
    .
    .
    for $i_n = l_n(i_1, \ldots, i_{n-1})$ to $u_n(i_1, \ldots, i_{n-1})$ do
      comando S_1
      .
      .
      comando S_p

onde l_1 e u_1 são constantes e $l_\alpha(i_1, \ldots, i_{\alpha-1})$ e $u_\alpha(i_1, \ldots, i_{\alpha-1})$ são os limites mínimo e máximo de laço, $1 < \alpha \leq n$, e todas as variáveis usadas em comandos S_1 a S_p têm seus índices como funções afins de índices i_1 a i_n. A parte composta pelas n primeiras linhas (as linhas que começam com "for") é chamada de *cabeçalho* e a parte composta pelas p últimas linhas (as linhas de comando) é chamada de *corpo* do algoritmo.

O conjunto de índices ou domínio para este algoritmo é definido como:

$Dom = \{I = (i_1, \ldots, i_n) \in Z^n \mid l_\alpha \leq i_\alpha \leq u_\alpha, 1 \leq \alpha \leq n\}$

Dom é o conjunto de pontos com coordenadas inteiras num poliedro convexo. Este é um modelo bastante geral e amplamente usado [Baner88, Dowl90, Wolfe90].

Para o estudo de paralelismo, o conceito de dependência é fundamental [Baner88, Poly88-2, Wolfe89, Wolfe90]. Um comando S_β *depende* de um comando S_α (denotado por $S_\alpha \delta S_\beta$) se existem $S_\alpha(I)$, a instância I de S_α, e $S_\beta(J)$, a instância J de S_β, e uma posição X da memória tais que:

1. $S_\alpha(I)$ e $S_\beta(J)$ fazem referência a X e pelo menos uma das referências é escrita.

2. $S_\alpha(I)$ é executado antes de $S_\beta(J)$ na execução seqüencial.

3. X não é escrita entre $S_\alpha(I)$ e $S_\beta(J)$.

Geralmente a dependência é dividida em três tipos:

- *dependência de fluxo* ($S_\alpha \delta^f S_\beta$): se $S_\alpha(I)$ escreve em X e $S_\beta(J)$ o lê.
- *anti-dependência* ($S_\alpha \delta^a S_\beta$): se $S_\alpha(I)$ lê X e $S_\beta(J)$ escreve em X.
- *dependência de saída* ($S_\alpha \delta^o S_\beta$): se $S_\alpha(I)$ escreve em X e $S_\beta(J)$ também.

Entre os três tipos de dependência, o mais difícil de tratar é a dependência de fluxo e as vezes ela é chamada de *dependência verdadeira* (esta nomenclatura vai ter outro significado em 3.3).

De fato, no artigo de Konda e Kumar [KonK95], eles afirmam que o mais difícil de paralelizar é o ciclo formado apenas por dependências de fluxo. Exatamente este será o tema central deste trabalho: paralelização de laços com ciclos formados apenas por dependências de fluxo.

Se $S_\beta(j_1, \ldots, j_n)$ depende de $S_\alpha(i_1, \ldots, i_n)$ então $d = (j_1 - i_1, \ldots, j_n - i_n)$ será chamado de *vetor de dependência* entre os dois comandos. Vamos adotar a notação $S_\alpha \rightarrow S_\beta$ no lugar de $S_\alpha \delta S_\beta$ e todas as dependências serão de fluxo daqui para frente. Vamos estudar casos naturais em que todos os vetores de dependência são lexicograficamente positivos, i.e., $\nexists d = (0, \ldots, 0)^t$, pois a maioria de laços encaixados tem esta estrutura. Em outras palavras, não há dependências entre comandos dentro de uma mesma iteração. Note que podemos permutar os comandos no corpo do laço sem afetar a semântica neste caso. Na seção 2.4 discutiremos brevemente os casos particulares em que existe este tipo de vetor de dependência.

Dados dois comandos S_α e S_β, podem existir vários pares de índices (I, J), $I, J \in Dom$ tais que $S_\beta(J)$ depende de $S_\alpha(I)$. Vamos restringir nossa atenção à importante subclasse de GLN chamada GLN *uniforme* na qual o vetor de dependência entre dois comandos independe dos índices da particular instância (uma outra formulação é a equação de recorrência uniforme, usada no contexto de síntese de algoritmos sistólicos [QuiR89]). A importância de GLN uniforme se deve principalmente aos seguintes dois motivos:

1. Muitos algoritmos para aplicações científicas têm esta estrutura.

2. A sua estrutura regular permite explorar bem seu paralelismo implícito.

Exemplo 1 *(Peir e Cytron [PeiC89])*

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    comando S_1: $a(i,j) = b(i, j-6) + e(i-1, j+3)$
    comando S_2: $b(i+1, j-1) = c(i+2, j+5)$
    comando S_3: $c(i+3, j-1) = a(i, j-2)$
    comando S_4: $e(i, j-1) = a(i, j-1)$

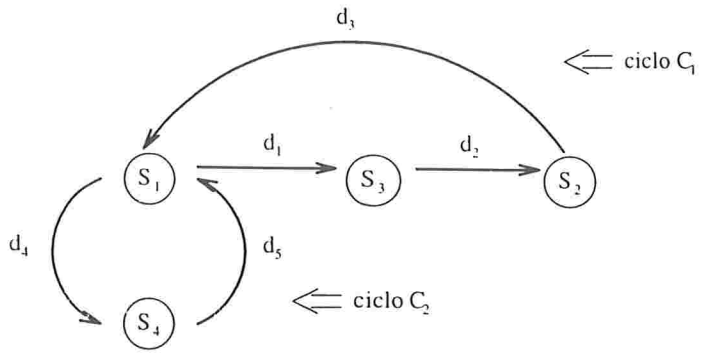

Figura 2.1: Grafo de dependência do Exemplo 1

Para este exemplo o conjunto de índices é

$Dom = \{(i,j) \in Z^2 \mid 0 \leq i,j \leq N\}$.
Temos 5 vetores de dependência:

$S_1 \rightarrow S_3 : d_1 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$

$S_3 \rightarrow S_2 : d_2 = \begin{pmatrix} 1 \\ -6 \end{pmatrix}$

$S_2 \rightarrow S_1 : d_3 = \begin{pmatrix} 1 \\ 5 \end{pmatrix}$

$S_1 \rightarrow S_4 : d_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$S_4 \rightarrow S_1 : d_5 = \begin{pmatrix} 1 \\ -4 \end{pmatrix}$

Dado GLN uniforme a matriz de dependência D é uma matriz cujas colunas são os vetores de dependência e o grafo de dependência é um grafo orientado cujos vértices são os comandos e arestas são os vetores de dependência. Um ciclo de dependência num grafo de dependência é um circuito. Para o Exemplo 1 temos como matriz de dependência:

$D = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & -6 & 5 & 1 & -4 \end{pmatrix}$

Temos dois ciclos de dependência como mostra o grafo de dependência na Figura 2.1.
Vamos fazer mais uma restrição à classe de algoritmos:

Laços encaixados uniformes e regulares (*regular uniform nest* (RUN))

for $i_1 = 0$ to N_1 do
  for $i_2 = 0$ to N_2 do

.

for $i_n = 0$ to N_n do

comando S_1

.

.

comando S_p

O motivo desta simplificação é para facilitar as descrições subseqüentes. Pelo mesmo motivo consideremos sempre $N = N_1 = \ldots = N_n$. Também é irrelevante a escolha de 0 como limite inferior. Todas essas simplificações visam apenas facilitar a descrição dos métodos. Agora faremos a última observação. A menos do Exemplo 1, os índices das variáveis do lado esquerdo do comando sempre coincidem com a instância do comando, i.e., o lado esquerdo é sempre do tipo $a(i,j)$ e nunca do tipo $a(i-1,j-3)$. Esta convenção facilita a análise de dependência. Por exemplo, o cálculo de vetores de dependência fica imediato. Porém esta convenção não é uma restrição mas sim uma transição para casos equivalentes. Por exemplo vejamos o Exemplo 1. Sejam $b'(i,j) = b(i+1,j-1)$, $c'(i,j) = c(i+3,j-1)$ e $e'(i,j) = e(i,j-1)$. Então o seguinte algoritmo é equivalente ao Exemplo 1 no que diz respeito a estrutura de dependências como veremos no capítulo 4.

Exemplo 2

for $i = 0$ *to* N *do*

for $j = 0$ *to* N *do*

comando S_1: $a(i,j) = b'(i-1,j-5) + e'(i-1,j+4)$

comando S_2: $b'(i,j) = c'(i-1,j+6)$

comando S_3: $c'(i,j) = a(i,j-2)$

comando S_4: $e'(i,j) = a(i,j-1)$

2.3 Escalonamento

Vamos definir o que é um escalonamento para execução paralela.

Dado RUN, escalonamento é uma função $\Phi_E : Z^n \rightarrow Z$ tal que a computação correspondente ao ponto $(i_1, \ldots, i_n) \in Dom$ é executada no passo $\Phi_E(i_1, \ldots, i_n)$ [DarR92]. Para que uma função $\Phi_E : Z^n \rightarrow Z$ seja um escalonamento, ela deve satisfazer à seguinte condição:

Se $S_\alpha(i_1, \ldots, i_n) \rightarrow S_\beta(j_1, \ldots, j_n)$ então $\Phi_E(i_1, \ldots, i_n) < \Phi_E(j_1, \ldots, j_n)$.

A execução paralela de RUN é a seguinte:

for $i = timemin$ to $timemax$ do

execute as computações em todos os pontos $(i_1, \ldots, i_n) \in Dom$ tais que $\Phi_E(i_1, \ldots, i_n) = i$

onde $timemin = \min\{\Phi_E(I) \mid I \in Dom\}$ e $timemax = \max\{\Phi_E(I) \mid I \in Dom\}$.

Em cada passo i os p comandos de todos os $(i_1, \ldots, i_n) \in Dom$ tais que $\Phi_E(i_1, \ldots, i_n) = i$ são executados em paralelo. O número total de passos será $timemax - timemin + 1$. O exemplo típico de escalonamento é o uso de produto escalar como no método GSS que veremos a seguir.

Seja Φ_E um escalonamento e sejam I e $J \in Dom$. Vamos supor que Φ_E é uma função afim e $J = I + d$ para algum vetor de dependência $d \in D$.
Neste caso

$$\begin{aligned} \Phi_E(J) &> \Phi_E(I) \\ \Phi_E(J - I) &> 0 \text{ logo} \\ \Phi_E(d) &> 0 \end{aligned}$$

.

Assim temos uma caracterização simples de escalonamento quando ela é uma função afim:

Proposição 1 *Sejam $\Phi_E : Z^n \rightarrow Z$ uma função afim e D a matriz de dependência para um algoritmo RUN.*
Então Φ_E é um escalonamento para este algoritmo
se e somente se
$\Phi_E(d) > 0, \forall d \in D$.

Corolário 1 *Sejam $\Phi_E : Z^n \rightarrow Z$ uma função afim e D a matriz de dependência para um algoritmo RUN.*
Seja $\overline{\Phi_E} : Z^n \rightarrow Z$ tal que $\overline{\Phi_E}(I) = \lfloor \frac{\Phi_E(I)}{\alpha} \rfloor$ para algum α inteiro positivo. Então $\overline{\Phi_E}$ é um escalonamento para este algoritmo
se e somente se
$\overline{\Phi_E}(d) > 0, \forall d \in D$.

2.4 Mapeamento

Dado RUN, mapeamento é uma função $\Psi_M : Z^n \rightarrow Z^m$ tal que a computação $(i_1, \ldots, i_n)$ é executada no ponto $\Psi_M(i_1, \ldots, i_n)$ [DarR94].

Se $\Phi_E(i_1, \ldots, i_n) = \Phi_E(j_1, \ldots, j_n)$ então $\Psi_M(i_1, \ldots, i_n) \neq \Psi_M(j_1, \ldots, j_n)$ para que as computações escalonadas no mesmo passo sejam mapeadas em pontos diferentes.

Quando Dom é mapeado a Z^m, $\Psi_M(Dom)$ pode ser interpretado como uma rede de processadores virtuais. Cada $\Psi_M(i_1, \ldots, i_n)$ é um processador virtual e os vetores de dependência mapeados representarão comunicações entre processadores virtuais nesta rede. Nesta rede de processadores virtuais $\Psi_M(Dom)$, as computações são executadas como segue:

Em cada passo, um processador $\Psi_M(I)$ recebe os dados vindos de outros processadores $\Psi_M(J)$, onde $I = J + d$ para algum $d \in D$, executa os comandos $S_1(I), S_2(I), \cdots, S_p(I)$ e envia os dados para outros processadores $\Psi_M(J')$, onde $J' = I + d$ para algum $d \in D$. Como supomos que todos os vetores de dependência são lexicograficamente positivos, não há dependência entre os comandos da mesma iteração. Deste modo para cada computação I os p comandos do corpo do laço são executados em paralelo.

Sejam

$Comm$=tempo gasto para comunicação entre processadores
$Comp$=tempo gasto para cálculo de um comando
$Tpep$=número total de passos para execução paralela

Então o tempo total gasto será geralmente $Tpep \times (Comp + Comm)$.

O exemplo típico de mapeamento é uma projeção ao longo de um vetor e neste caso $m = n - 1$. Para este tipo de mapeamento se todos os vetores de dependência forem paralelos ao vetor de projeção, então eles não representam comunicação pois os dados estão no mesmo processador. Neste caso particular o tempo total gasto será apenas $Tpep \times Comp$.

Observações

1. Quando se trata de sistema sistólico ([Kung88, QuiR89]), cada $\Psi_M(i_1, \ldots, i_n)$ é um elemento do circuito VLSI composto por p unidades de cálculo (p é o número de comandos nos laços). Porém para o caso de paralelização na rede de processadores é natural considerar cada processador como um elemento que efetua operações seqüenciais e não paralelas. Então a rigor o ponto $\Psi_M(i_1, \ldots, i_n)$ é um bloco composto por p processadores.

2. Agora tratamos o caso particular da existência do vetor de dependência nula. Considere o seguinte exemplo:

Exemplo 3

$for\ i = 0\ to\ N\ do$
$\quad for\ j = 0\ to\ N\ do$
$\quad\quad S_1\colon a(i,j) = f_1(a(i-1,j), b(i-2,j-1))$
$\quad\quad S_2\colon b(i,j) = f_2(a(i,j), c(i,j-1))$
$\quad\quad S_3\colon c(i,j) = f_3(b(i-1,j-1), c(i-1,j-2))$

Para este exemplo temos:

$$D = \begin{pmatrix} 1 & 2 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 2 \end{pmatrix}$$

Existe um vetor de dependência nula devido a dependência de $S_1(i,j)$ para $S_2(i,j)$. No caso sistólico o elemento do circuito vai ter unidades de cálculo para S_1 e S_2 em seqüência. Para rede de processadores o bloco correspondente a $\Psi_M(i_1,\ldots,i_n)$ vai ter processadores que executam em seqüência. Conseqüentemente o bloco não pode ser executado totalmente em paralelo. Para o exemplo acima, o tempo total gasto vai ser $2\ Tpep \times (Comp + Comm)$.

Capítulo 3

Métodos conhecidos para encolhimento de ciclos

Neste capítulo descrevemos os métodos existentes para paralelização de RUN com ciclo de dependência.

- Na seção 3.1 primeiro descrevemos o método *encolhimento de ciclo seletivo* (*selective cycle shrinking*). Mostramos sua limitação e definimos a sua generalização *encolhimento de ciclo seletivo generalizado* (*generalized selective cycle shrinking* (GSS)).
- Na seção 3.2 descrevemos o *método de deslocamento de índices* (*index shift method* (ISM)), em seguida uma técnica para melhorar ISM e um método que combina GSS e ISM de modo eficiente.
- Na seção 3.3 descrevemos o método *encolhimento por distância verdadeira* (*true dependence shrinking*) e sua generalização *encolhimento por distância verdadeira generalizado* (*generalized true dependence shrinking*). Descrevemos também o método *afim por comando* (*affine by statement*).

3.1 Método GSS

- Na seção 3.1.1 descrevemos a técnica de *encolhimento de ciclo seletivo* (*selective cycle shrinking*) usada em compiladores paralelizantes [Poly88].
- Na seção 3.1.2 descrevemos o método de *encolhimento de ciclo seletivo generalizado* (*generalized selective cycle shrinking* (GSS)) e discutimos a relação entre GSS e o uso de vetor racional para escalonamento.

3.1.1 Encolhimento de ciclo seletivo

Sejam A um algoritmo RUN com n laços e r vetores de dependência e $D = \{(d_\alpha^\beta) \mid 1 \leq \alpha \leq r, 1 \leq \beta \leq n\}$ sua matriz de dependência onde d_α^β denota o elemento na linha β e coluna α, i.e., β-ésimo componente do vetor de dependência d_α e $\Delta^\beta = \min\{d_\alpha^\beta \mid 1 \leq \alpha \leq r\}$ (ver [Poly88]).
Ache o menor β possível tal que $\Delta^\beta > 0$. Todos os laços internos $(\beta+1, \cdots, n)$ são executados em paralelo (do tipo *DOALL*), todos os laços externos $(1, \cdots, \beta-1)$ são executados em seqüência e o laço β é executado com incremento Δ^β.

Considere o seguinte exemplo simplificado de [Poly88, ShaOF91]:

Exemplo 4

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S_1: $a(i,j) = f_1(b(i-3, j-5))$
    S_2: $b(i,j) = f_2(a(i-2, j-4))$

Para este exemplo (ver a Figura 3.1): $D = \begin{pmatrix} 3 & 2 \\ 5 & 4 \end{pmatrix}$ e $\Delta^1 = \min\{3,2\} = 2$ e $\Delta^2 = \min\{5,4\} = 4$.

Portanto encolhimento de ciclo seletivo acha $\Delta^1 = 2$ e o algoritmo será transformado no seguinte:

Exemplo 5

for $k = 0$ *to* N *step 2 do*
  for $i = k$ *to* $k+1$ *doall*
    for $j = 0$ *to* N *doall*
      S_1: $a(i,j) = f_1(b(i-3, j-5))$
      S_2: $b(i,j) = f_2(a(i-2, j-4))$

Agora vejamos o seguinte exemplo (ver a Figura 3.2):

Exemplo 6

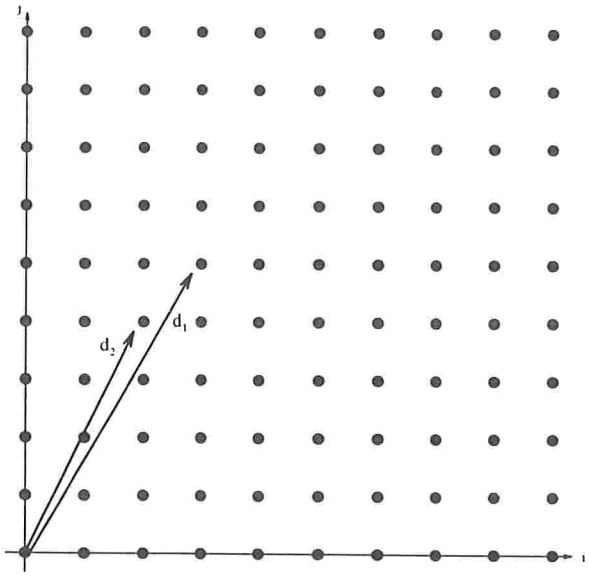

Figura 3.1: *Dom* e vetores de dependência do Exemplo 4

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S_1: $a(i,j) = f_1(b(i-1,j))$
    S_2: $b(i,j) = f_2(c(i,j-1))$
    S_3: $c(i,j) = f_3(a(i-1,j))$

Para este exemplo temos:
$D = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ e $\Delta_1 = 0$ e $\Delta_2 = 0$. Assim pelo encolhimento de ciclo seletivo é impossível paralelizar o Exemplo 6.

3.1.2 Encolhimento de ciclo seletivo generalizado (GSS)

Usaremos a generalização de encolhimento de ciclo seletivo, que é GSS [ShaOF91], para podermos paralelizar o Exemplo 6. Antes de ver esta generalização, vamos examinar outras deficiências de encolhimento de ciclo seletivo. Vejamos o que ocorre no Exemplo 4 (= Exemplo 5). A Figura 3.3 mostra como o Exemplo 4 é executado em paralelo pelo encolhimento de ciclo seletivo.

Os pontos de *Dom* localizados em duas retas paralelas ao eixo j consecutivas com mesma numeração na Figura 3.3 são executados em paralelo. Note que não existe nenhuma dependência entre estes pontos mas existe dependência entre os pontos localizados nas retas com numeração diferentes. Por outro lado considerando a uniformidade de dependência nos

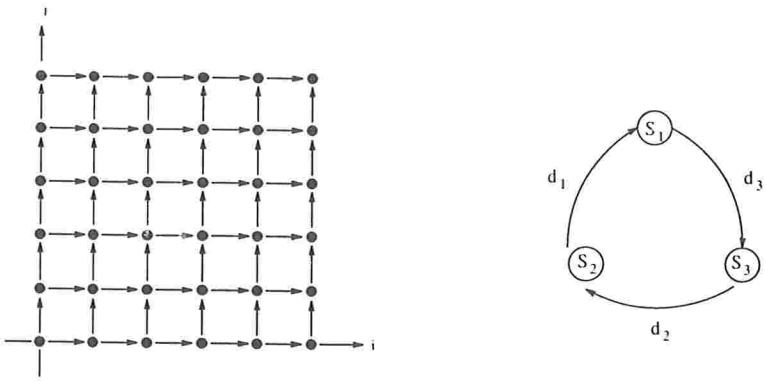

Figura 3.2: *Dom* e grafo de dependência do Exemplo 6

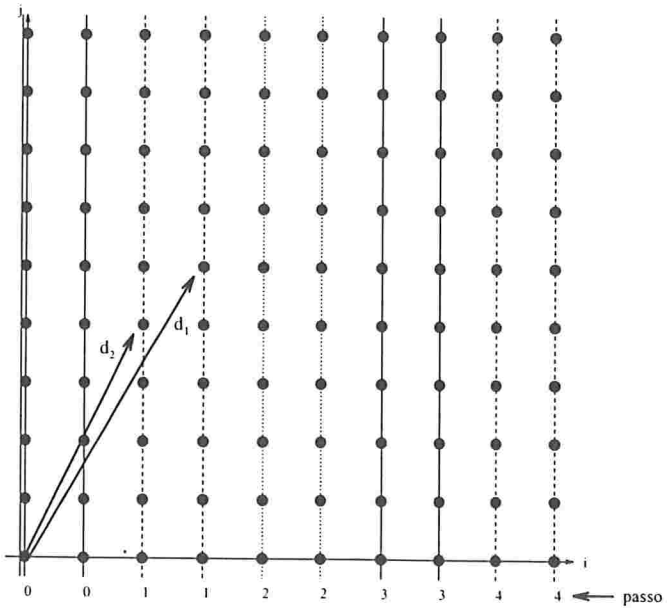

Figura 3.3: Execução paralela do Exemplo 4

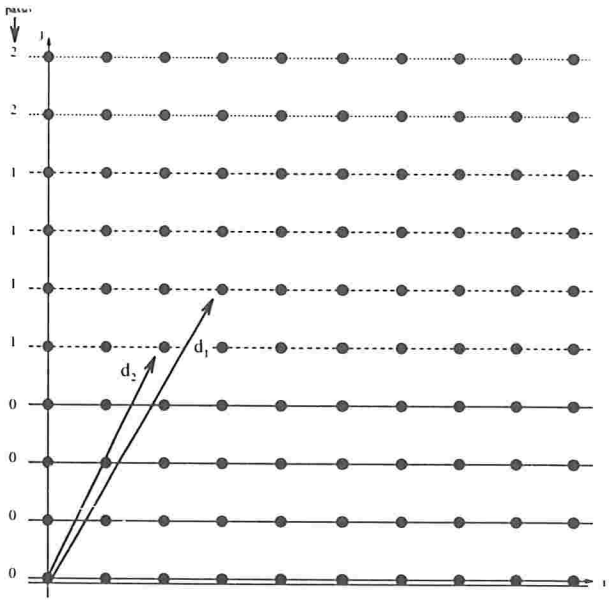

Figura 3.4: Outra execução paralela do Exemplo 4

pontos de *Dom*, podemos conceber uma paralelização alternativa como mostra a Figura 3.4.

Enquanto a execução paralela do Exemplo 4 da Figura 3.3 leva $\lceil \frac{N}{2} \rceil$ passos, a execução paralela do Exemplo 4 da Figura 3.4 leva $\lceil \frac{N}{4} \rceil$ passos. O método de encolhimento de ciclo seletivo nos dá apenas uma dessas soluções. Agora vejamos mais um outro exemplo (veja a Figura 3.5).

Exemplo 7

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S_1: $a(i,j) = f_1(b(i,j-1))$
    S_2: $b(i,j) = f_2(a(i,j-1))$

Pelo encolhimento de ciclo seletivo o Exemplo 7 tem $\Delta_1 = 0$ e $\Delta_2 = 1$, logo nenhum laço é executável em paralelo. Porém a Figura 3.6 mostra uma execução paralela trivial para o Exemplo 7.

Estas observações mostram as limitações do método. Para ir adiante procuremos como formalizar as execuções paralelas nas Figuras 3.3, 3.4 e 3.6. Vamos introduzir a seguinte notação:

Definição 1 *Dados* $\alpha \in Z$ *e* $\pi \in Z^n$, $H_\pi(\alpha) = \{I \in Dom \mid I \cdot \pi = \alpha\}$ *onde* $I \cdot \pi$ *é o produto escalar de dois vetores.*

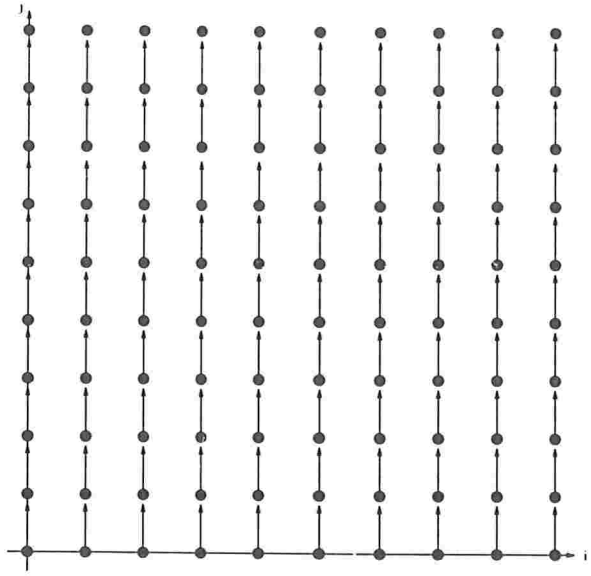

Figura 3.5: *Dom* do Exemplo 7

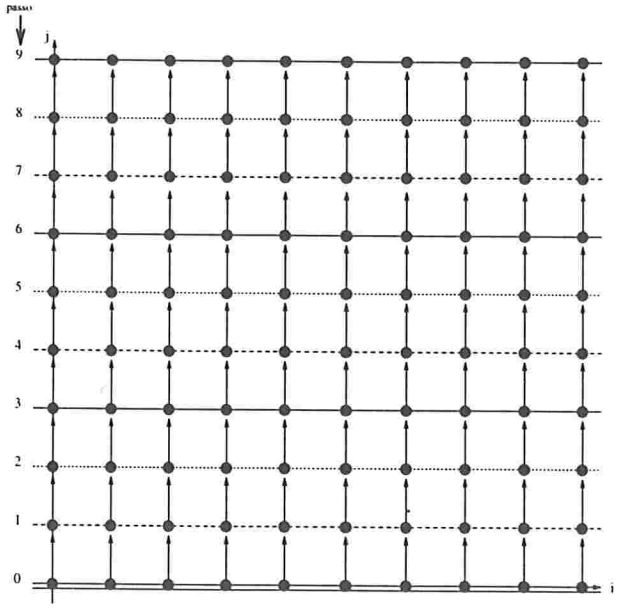

Figura 3.6: Execução paralela do Exemplo 7

Note que cada $H_\pi(\alpha)$ é um hiperplano perpendicular ao vetor π.

Então os pontos localizados em cada reta das Figuras 3.3, 3.4 e 3.6 podem ser representados por $H_\pi(\alpha)$ para algum vetor π paralelo a um dos eixos e para algum $\alpha \in Z$. O vetor π seria $(1,0)^t$, $(0,1)^t$ e $(0,1)^t$ para as Figuras 3.3, 3.4 e 3.6 respectivamente. Para o Exemplo 4 da Figura 3.3 os pontos localizados em $H_{(1,0)}(0)$ e $H_{(1,0)}(1)$ são executados no passo 0, os pontos localizados em $H_{(1,0)}(2)$ e $H_{(1,0)}(3)$ são executados no passo 1. Este escalonamento pode ser formulado como $\Phi_E : Z^n \to Z$ tal que $\Phi_E(I) = \lfloor \frac{(0,1)\cdot I}{2} \rfloor$. Este Φ_E satisfaz a Corolário 1 (página 11). Agora generalizamos o escalonamento acima. Esta generalização se faz com a maior liberdade na escolha de vetor π que nem sempre vai ser paralelo ao eixo.

Método GSS

Considere RUN com n laços e seja $D = (d_1, \ldots, d_r)$ uma matriz de dependência $n \times r$. Seja $\pi = (\pi^1, \ldots, \pi^n)^t$ um vetor tal que

1. $\pi \cdot D > 0$, i.e., $\pi \cdot D$ é um vetor com todos os seus componentes positivos.
2. $\text{mdc}\{\pi^1, \ldots, \pi^n\} = 1$

π será denominado *vetor de escalonamento* e seja o fator de redução $\text{disp}(\pi)=\min\{\pi \cdot d_\alpha \mid 1 \leq \alpha \leq r\}$.

Todos os pontos $I \in Dom$ que estejam no mesmo hiperplano $H_\pi(\alpha)$ são executados simultaneamente no passo $\lfloor \frac{\pi \cdot I}{\text{disp}(\pi)} \rfloor$ e $\text{disp}(\pi)$ hiperplanos consecutivos são executados simultaneamente. Tais hiperplanos serão denominados *hiperplanos de tempo*.

Ache π_0 que minimize $\text{GSS}(\pi) = \frac{\max\{\pi \cdot I - \pi \cdot J \mid I,J \in Dom\}}{\text{disp}(\pi)}$.

Observações

- A condição $\text{mdc}\{\pi^1, \ldots, \pi^n\} = 1$ é para evitar vetor que é produto de um vetor com um escalar não nulo. Estes vetores têm mesmo escalonamento e são equivalentes do ponto de vista do escalonamento. Seja $\pi_0 = \lambda\pi$, então $\lfloor \frac{\pi_0 \cdot I}{\text{disp}(\pi_0)} \rfloor = \lfloor \frac{\lambda\pi \cdot I}{\text{disp}(\lambda\pi)} \rfloor = \lfloor \frac{\lambda(\pi \cdot I)}{\lambda \text{disp}(\pi)} \rfloor = \lfloor \frac{\pi \cdot I}{\text{disp}(\pi)} \rfloor$.
- O método também é conhecido como método de hiperplano de Lamport [Lampo74], escalonamento linear [DarR92, DKR92, DarR94-2] e muito usado também para síntese de algoritmos sistólicos [MolF86, QuiR89].

Agora voltemos ao Exemplo 6. Aplicando GSS a este exemplo, temos que achar $\pi_0 = (x, y)$ que minimize $\text{GSS}(\pi) = \frac{\max\{\pi \cdot I - \pi \cdot J \mid I,J \in Dom\}}{\min\{\pi \cdot d_1, \pi \cdot d_2, \pi \cdot d_3\}}$ sob condições:

1. $\pi \cdot d_1 = \pi \cdot d_3 > 0$, $\pi \cdot d_2 > 0$
2. $\text{mdc}\{x, y\} = 1$

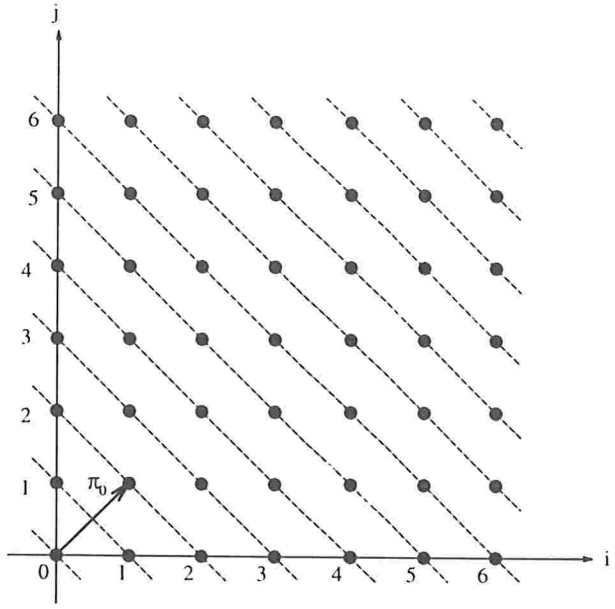

Figura 3.7: Hiperplanos de tempo e π_0 para o Exemplo 6

Das condições impostas, é fácil ver que $x > 0$ e $y > 0$ e $\mathrm{GSS}(\pi) = \frac{(x+y)N}{\min\{x,y\}}$. A solução ótima será $x = y = 1$ e $\mathrm{GSS}(\pi_0)$=$2N$ passos com $\pi_0 = (1,1)$. Os hiperplanos de tempo e π_0 estão mostrados na Figura 3.7.

Para o Exemplo 4 temos $\pi_0 = (0,1)$ e $\mathrm{GSS}(\pi_0) = \frac{N}{4}$. GSS foi capaz assim de achar a melhor das duas soluções. Para o Exemplo 1 (página 8) do começo do capítulo anterior temos $\pi_0 = (7,1)$ como a melhor solução e $\mathrm{GSS}(\pi_0) = 8N$ (ver Apêndice A.1 (página 76)) .

Shang e Fortes propuseram um método [ShaF91] bastante complexo para resolver GSS, isto é, achar π_0 que minimize $\mathrm{GSS}(\pi)$. O espaço solução é particionado em subcones convexos e em seguida resolve-se um problema para cada um desses subcones. Darte e Robert propuseram um método mais eficiente que consiste em resolver um problema de programação linear (com o uso de vetor racional, ver [DarR92, DKR92, DarR94-2]). Em [DKR92] eles provaram que escalonamento linear é quase ótimo em relação a escalonamento livre (escalonamento na qual executam-se, em cada passo, todos os comandos cujos antecessores de dependência foram executados) para os casos de laços uniformes com apenas um comando no corpo ou vários comandos considerados como um bloco.

GSS versus vetor racional

Darte e Robert têm usado vetor de escalonamento com componentes racionais em vez de inteiros [DarR92, DarR94, DarR95], devido a ferramentas poderosas de programação linear sobre corpo ordenado.

O conjunto de soluções para vetor racional é maior do que o conjunto de soluções para vetor inteiro com fator de redução (disp(π)). Porém a solução ótima de GSS coincide com a solução ótima de vetor racional. Mostremos este fato de modo informal usando exemplos simples. Nas três figuras seguintes os números indicam os passos em que cada hiperplano é executado.

- Para um algoritmo RUN, seja uma solução GSS $\pi_1 = (1, 2)$ com disp(π_1) $= 5$ (ver a Figura 3.8).
- Para o mesmo algoritmo $\pi_1^\star = (\frac{1}{5}, \frac{2}{5})$ é uma solução racional (ver a Figura 3.9).
- Vejamos a situação inversa. Se $\pi_2^\star = (\frac{1}{5}, \frac{2}{5})$ é uma solução racional de um algoritmo RUN. Neste caso $\pi_2 = (1, 2)$ com disp(π_2) $= 5$ é uma solução GSS.
- Em outras palavras, seja $\pi = (\pi^1, \cdots, \pi^n) \in Z^n$ tal que mdc$\{\pi^\alpha \mid 1 \leq \alpha \leq n\} = 1$, $\pi \cdot D > 0$ e $k =$ disp(π) $= \min\{\pi \cdot d_\alpha \mid 1 \leq \alpha \leq n\}$. Então π é um vetor de escalonamento para GSS e $\Phi_E(I) = \lfloor \frac{\pi I}{k} \rfloor$ e $\pi^\star = \frac{\pi}{k} = (\frac{\pi^1}{k}, \cdots, \frac{\pi^n}{k}) \in Q^n$ é um vetor racional de escalonamento e $\Phi_E^\star(I) = \lfloor \pi^\star \cdot I \rfloor$.
  Observamos que $\Phi_E = \Phi_E^\star$.
- Temos uma certa equivalência entre solução GSS e solução vetor racional.
- Entretanto considere $\pi_3^\star = (\frac{h\pi_3^1}{k}, \cdots, \frac{h\pi_3^n}{k}) = \frac{h}{k}\pi_3 \in Q^n$ com mdc$\{\pi_3^\alpha \mid 1 \leq \alpha \leq n\} = 1$, $\pi_3 \cdot D > 0$ e mdc$\{h, k\} = 1$ como uma solução vetor racional. Por exemplo $\pi_3^\star = (\frac{2}{5}, \frac{4}{5})$. (ver a Figura 3.10).
- É fácil ver que em GSS não existe um vetor de escalonamento que resulte nos mesmos hiperplanos de tempo.
- Porém se olharmos a Figura 3.10 levando em consideração a uniformidade de dependência de um algoritmo RUN em qualquer ponto no *Dom*, então podemos afirmar o seguinte. Se num determinado ponto de *Dom* é possível execução paralela de três hiperplanos consecutivos, então esta execução paralela deve ser possível em qualquer ponto de *Dom*. Para o exemplo da Figura 3.10, $\pi_4^\star = (\frac{1}{3}, \frac{2}{3})$ deve ser outra solução racional e $\pi_4^\star$ (melhor que a $\pi_3^\star$) admite seu equivalente em GSS:
  $\pi_4 = (1, 2)$ com disp(π_4) $= 3$.
- Concluindo, os vetores racionais de escalonamento do tipo $\pi_3^\star$ não precisa ser levado em consideração e o número de passos da solução ótima racional coincide com o número de passos da solução ótima GSS.

Apesar da complexidade da resolução de GSS na sua forma geral, ele é conveniente para manipular exemplos relativamente simples e a observação acima justifica o seu uso neste trabalho sem comprometer o escalonamento.

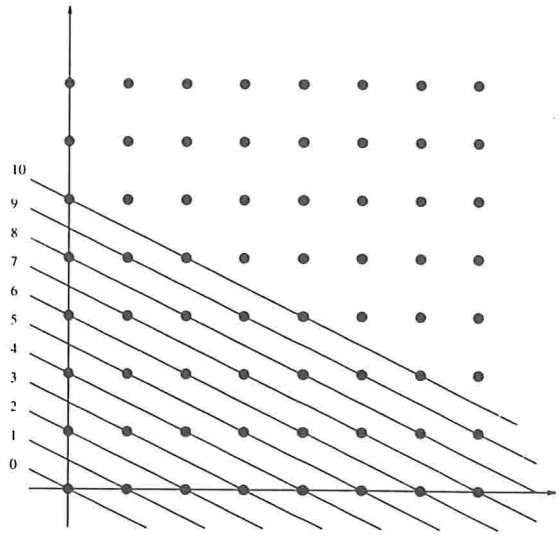

Figura 3.8: Os hiperplanos de tempo para $\pi_1 = (1, 2)$

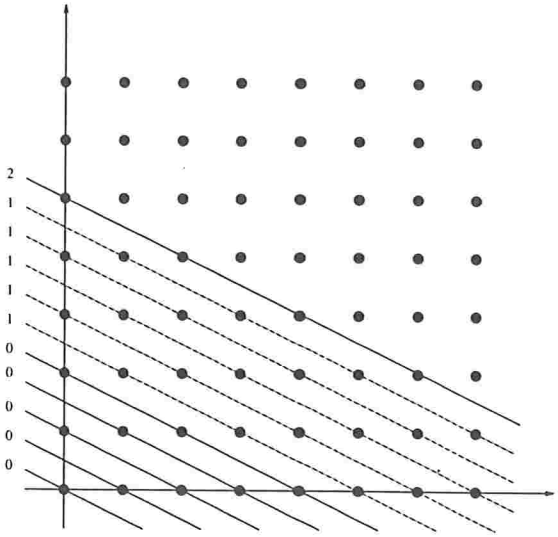

Figura 3.9: Os hiperplanos de tempo para $\pi_1^* = (1/5, 2/5)$

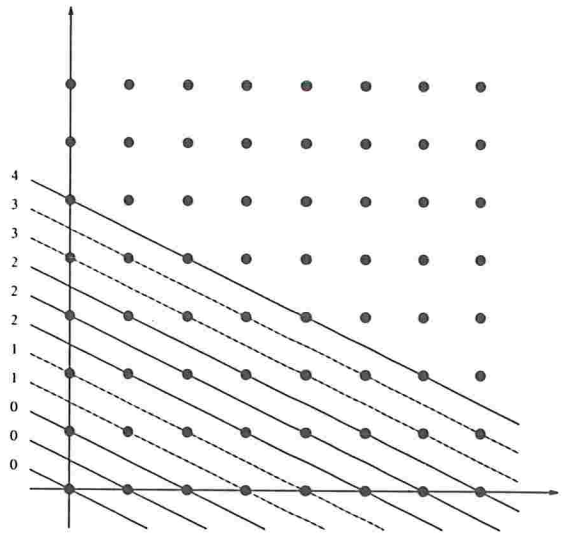

Figura 3.10: Os hiperplanos de tempo para $\pi_3^\star = (2/5, 4/5)$

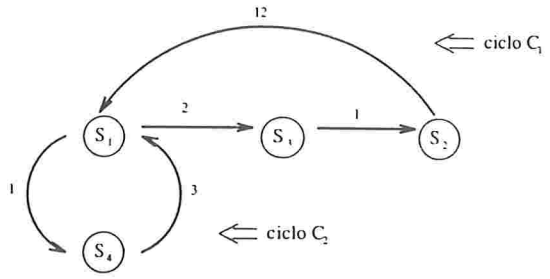

Figura 3.11: O grafo de dependência com os pesos

3.2 Método ISM e GSS combinado com ISM

- Na seção 3.2.1 descrevemos o *método de deslocamento de índice* (*index shift method* (ISM)).
- Na seção 3.2.2 descrevemos uma técnica para melhorar ISM com alguma manipulação algébrica.
- Na seção 3.2.3 após mostrar que GSS seguida por ISM não é melhor, descrevemos um método que combina GSS e ISM de modo eficiente e damos um exemplo interessante que mostra a eficiência do método.

3.2.1 Método ISM

Método de deslocamento de índices (*index shift method* (ISM)) foi introduzido por Liu, Ho e Sheu [LiuHS90]. Este método faz deslocamento de índices em comandos sem violar a semântica do algoritmo e torna o ciclo de dependência mais balanceado permitindo aumento do fator de redução (disp(π)). Esta seção segue o artigo de Robert e Song [RobS92]. Vejamos de novo o Exemplo 1 (página 8). Vimos na seção anterior que $\pi = (\bar{7}, 1)$ é a melhor solução GSS e GSS(π) $= 8N$ para este exemplo. No grafo de dependência vamos substituir os rótulos de arestas d_α por $\pi \cdot d_\alpha$. Este novo valor é o peso de aresta que representa a diferença de escalonamento entre os comandos e temos a Figura 3.11.

A idéia do método ISM é aplicar certo "retiming" ao grafo de dependência. Por exemplo considere o ciclo C_2, a aresta (S_1, S_4) tem peso 1 e a aresta (S_4, S_1) tem peso 3. Como disp(π) é o mínimo entre $\pi \cdot d_\alpha$ que são os pesos de arestas, será desejável ter um melhor balanceamento entre estes pesos. Queremos remover 1 do peso da aresta (S_4, S_1) acrescentando para a aresta (S_1, S_4). De modo análogo remover 1 do peso da aresta (S_2, S_1) para a aresta (S_3, S_2). Com este deslocamento o mínimo entre os pesos (disp(π)) passa a ser 2. A idéia é somar certo peso, que pode ser negativo, às arestas que entram num vértice e tirar o mesmo peso das arestas que saem deste vértice. Esta transformação não altera o peso total do ciclo. Como nossa meta é melhor balanceamento dos pesos num ciclo, se um ciclo tem comprimento k e peso total T, então o ideal seria associar o peso $\lfloor \frac{T}{k} \rfloor + 1$ para $(T \ mod \ k)$ arestas e o peso $\lfloor \frac{T}{k} \rfloor$ para os restantes. Liu e outros propuseram um método

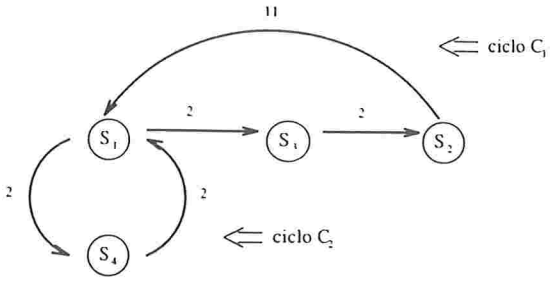

Figura 3.12: O grafo de dependência após $S_2^1 S_4^1$

simples para determinar a transformação necessária para chegar a este balanceamento em todos os ciclos. Seja S_i^k a transformação que consiste em somar peso k para todas as arestas entrando no vértice S_i e subtrair k das arestas que saem do vértice S_i. Para o Exemplo 1 aplicamos a seqüência $S_2^1 S_4^1$ e obtivemos a Figura 3.12.

O que significa aplicar S_i^k no grafo de dependência e como isto afeta os índices de laços? Aplicar S_i^k substitui a execução de dada instância $S_i(J_1)$ do comando S_i por outra instância $S_i(J_2)$ tal que a diferença de escalonamento entre as duas instâncias é igual a k, i.e., $\pi \cdot (J_1 - J_2) = k$.

Como os componentes de π são primos entre si, existe um vetor u tal que $\pi \cdot u = 1$. Tome $J_2 = J_1 - ku$ e temos $\pi \cdot (J_1 - J_2) = k\pi \cdot u = k$. Com este vetor u podemos computar um novo limite de índices de laços. Para o Exemplo 1 temos $u = (0, 1)$ e a transformação S_2^1 desloca por uma unidade o índice j de S_2. O segundo índice de laço vai de 1 para $N + 1$ para S_2. Como j vai de 0 para N no S_1, o domínio deve ser estendido para 0 a $N+1$ na direção j. Desta maneira introduzimos algumas instâncias adicionais para alguns comandos:
$j = N + 1$ para S_1 e S_3
$j = 0$ para S_2 e S_4

Nós temos o novo algoritmo:

Exemplo 8

for $i = 0$ *to* N *do*
  for $j = 0$ *to* $N + 1$ *do*
    comando S_1: $a(i, j) = b(i, j - 6) + e(i - 1, j + 3)$
    comando S_2: $b(i + 1, j - 2) = c(i + 2, j + 4)$
    comando S_3: $c(i + 3, j - 1) = a(i, j - 2)$
    comando S_4: $e(i, j - 1) = a(i, j - 1)$

Para este Exemplo 8 a matriz de dependência é a seguinte:

$$D = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 2 & -5 & 4 & 2 & -5 \end{pmatrix}$$

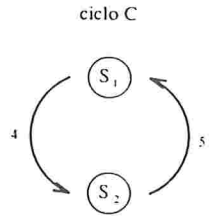

Figura 3.13 O grafo de dependência do Exemplo 4 com seus pesos

$\pi = (7, 1)$ é um vetor de escalonamento com $\text{disp}(\pi) = 2$ e temos
$\text{GSS}(\pi) = \frac{7N+(N+1)}{2} \leq 4N + 1$
O número de passos foi reduzido à metade.

Agora formalizemos o método ISM.

Método ISM

O método ISM substitui o fator de redução
$\lambda_1 = \text{disp}(\pi) = \min\{\pi \cdot d_\alpha \mid 1 \leq \alpha \leq r\}$
pelo valor
$\lambda_2 = \min\{\lfloor \frac{T(C)}{K(C)} \rfloor \mid C \in Cicl\}$

onde

π é um vetor de escalonamento tal que $\text{mdc}\{\pi^1, \cdots, \pi^n\} = 1$ e $\pi \cdot D > 0$.
$Cicl$ é o conjunto de todos os ciclos no grafo de dependência.
Cada aresta (S_α, S_β) com vetor de dependência d tem peso $\pi \cdot d$.
$T(C)$ é o peso total do ciclo C.
$K(C)$ é o comprimento do ciclo C.

O fator de melhoria é quase $\frac{\lambda_2}{\lambda_1}$. O preço desta transformação é o pequeno aumento de domínio de índices de laços visto acima. Note que este aumento depende só de π e não depende do tamanho do domínio.

3.2.2 Melhorando ISM

Considere o Exemplo 4 (página 15) da seção anterior. O método GSS determina $\pi = (0, 1)$ e $\text{GSS}(\pi) = \frac{N}{4}$. Queremos aplicar o método ISM. Temos a Figura 3.13.

Podemos notar que não há possibilidade para aplicar o método ISM, pois $\lfloor \frac{T(C)}{K(C)} \rfloor = \lfloor \frac{9}{2} \rfloor = 4$. Nesta seção mostraremos, conforme [RobS92], que a partir de π podemos chegar a π^* que dá um fator de redução 4.5 $(= \frac{T(C)}{K(C)})$.

Vamos supor que:

temos um ciclo C com k comandos S_1 a S_k com a matriz de dependência $D = (d_1, \cdots, d_k)$.
d_α é o vetor de dependência entre S_α para $S_{\alpha+1 \ mod \ k}$.
temos o vetor de escalonamento π tal que $\pi \cdot D > 0$.
$T(C) \ mod \ k \neq 0$, i.e. $\pi \cdot d_1 + \pi \cdot d_2 + \cdots + \pi \cdot d_k$ não é divisível por k.

Multiplicar os componentes de π por uma constante iria violar $\text{mdc}\{\pi^1, \pi^2, \cdots, \pi^n\} = 1$. Faremos o seguinte artifício algébrico.

Seja r um vetor com seus componentes primos entre si e tal que $\pi \cdot r = 0$.
Seja s um vetor tal que $r \cdot s = \pm 1$ e $s \cdot d_1 + s \cdot d_2 + \cdots + s \cdot d_k \geq 0$.
Note que, como r tem seus componentes primos entre si, é fácil achar s a partir de r.
Mostraremos como achar r.

Compute a forma hermitiana de π para obter $\pi = Q \cdot (1, 0, \cdots, 0)^t$ onde Q é unimodular[1] e seja r a segunda linha de Q^{-1} [Newm72]. Como Q^{-1} é unimodular também e $(1, 0, \cdots, 0)^t = Q^{-1} \cdot \pi$ então r tem seus componentes primos entre si e $r \cdot \pi = 0$ por construção. Agora seja $\pi^* = \lambda k \pi + s$. O peso $T^*(C)$ do ciclo C com respeito a π^* é:
$T^*(C) = \lambda k T(C) + s \cdot (d_1 + \cdots + d_k) \geq \lambda k T(C)$.
Logo $\lfloor \frac{T^*(C)}{k} \rfloor \geq \lambda T(C)$. Como π^* tem seus componentes primos entre si, temos $r \cdot \pi^* = \pm 1$. Para λ suficientemente grande temos $\pi^* \cdot D > 0$ e após aplicar ISM o fator de redução será arbitrariamente perto do real valor de $\frac{T(C)}{k}$.

3.2.3 GSS combinado com ISM

GSS seguido por ISM (GSS+ISM) não é o melhor possível

Aplicar ISM após obter o melhor vetor de escalonamento por GSS não garante que cheguemos a um melhor resultado. Para ilustrar esta observação vamos voltar a ver o Exemplo 1 (página 8). Este exemplo tinha $\pi = (7, 1)$ como o melhor vetor de escalonamento por GSS e temos o grafo de dependência como da Figura 3.11. O peso total para ciclo C_2 é 4 e é divisível pelo comprimento 2. Logo não temos como melhorar a aplicação de ISM como foi feita na seção anterior. Porém também não há razão para não usar um outro vetor a priori ao invés de $\pi = (7, 1)$.

Seja $\pi = (a, b)$ um vetor de escalonamento arbitrário.
De $\pi D > 0$, temos $b \geq 1$ e $a \geq 6b + 1$ (ver Apêndice A.1).
Para o ciclo C_1 temos $T(C_1) = \pi \cdot (d_1 + d_2 + d_3) = 2a + b$ e $K(C_1) = 3$.
Para o ciclo C_2 temos $T(C_2) = \pi \cdot (d_4 + d_5) = a - 3b$ e $K(C_2) = 2$.
Para $b \geq 1$ e $a \geq 6b + 1$, sempre temos $\frac{(2a+b)}{3} \geq \frac{(a-3b)}{2}$.

[1] Uma matriz é unimodular se os seus elementos são todos números inteiros e seu determinante é igual a ± 1.

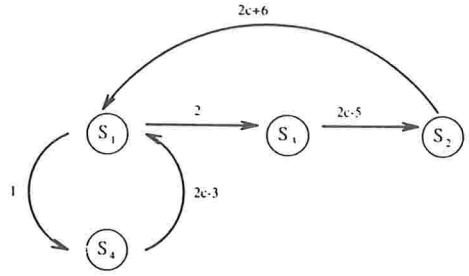

Figura 3.14: O grafo de dependência por $\pi = (2c+1, 1)$

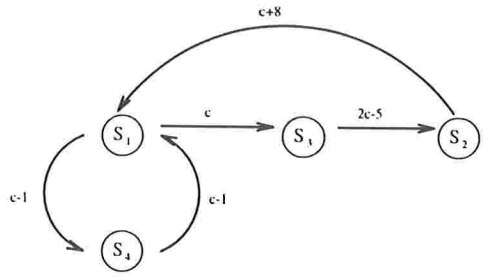

Figura 3.15: O grafo de dependência após S_1^{2-c}

Logo temos que minimizar o seguinte: $\frac{(a+b)N}{\lfloor (a-3b)/2 \rfloor}$

Podemos ter esta quantia tão perto de $2N$ quanto se queira tomando $a = 2c+1$ e $b = 1$ com c grande (necessariamente $c \geq 3$). Com $\pi = (2c+1, 1)$ temos a Figura 3.14. Aplicando a transformação S_1^{2-c} temos o grafo de dependência da Figura 3.15. Usando $u = (0, 1)$ temos:
Número de passos $\leq \frac{(2c+1)N+(N+2-c)}{c-1} \leq 2\frac{c+1}{c-1}N$. Este valor é perto de $2N$ para c grande, reduzindo para metade o valor obtido por GSS+ISM.

Novo método para otimização

Baseado na observação acima Robert e Song [RobS92] formularam o seguinte novo método que combina GSS e ISM.

Método GSS combinado com ISM

Ache um vetor $\pi = (\pi^1, \ldots, \pi^n)$ que minimize
$\text{NEW}(\pi) = \frac{\max\{\pi \cdot I - \pi \cdot J \mid I, J \in Dom\}}{ciclo(\pi)}$,

onde $ciclo(\pi) = \min\{\lfloor \frac{T_\pi(C)}{K(C)} \rfloor \mid C \in Cicl\}$
$T_\pi(C) = \sum_{d \in C} \pi \cdot d$
$K(C)$=comprimento do ciclo C
$Cicl$=conjunto de ciclos

sob condições

1. $\pi \cdot D > 0$
2. $\text{mdc}\{\pi^1, \ldots, \pi^n\} = 1$

Exemplo "exótico"

Em [RobS92] Robert e Song mostraram um interessante exemplo no qual o novo método tem fator de redução arbitrariamente grande em relação a GSS. Tratemos brevemente este exemplo (veja os detalhes de demonstração em [RobS92]). Seja λ um inteiro positivo arbitrário.

Exemplo 9

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    comando S_1: $a(i,j) = f_1(b(i-1, j-\lambda), c(i-1, j+\lambda), \cdots)$
    comando S_2: $b(i,j) = f_2(a(i, j-1), \cdots)$
    comando S_3: $c(i,j) = f_3(a(i, j-1), \cdots)$

Temos $\pi = (\lambda + 1, 1)$ e $\text{GSS}(\pi) = (\lambda + 2)N$ como melhor solução para GSS.
É impossível aplicar ISM em seguida devido ao ciclo $C_2 = (S_1, S_3)$.
Entretanto aplicando o novo método temos $\pi_0 = (2\lambda + 2, 1)$ e $\text{NEW}(\pi_0) \leq 4N + 1$.

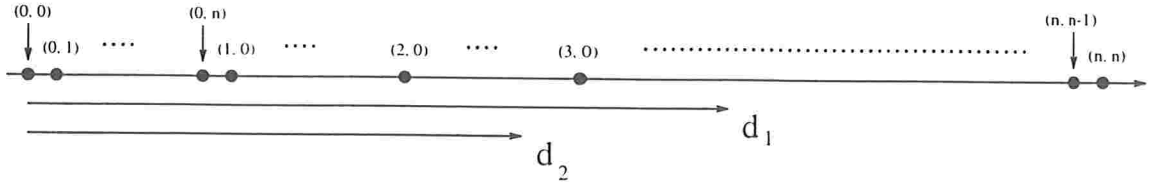

Figura 3.16: O novo espaço 1-dimensional do Exemplo 4

3.3 Outros métodos

- Na seção 3.3.1 descrevemos o método *encolhimento por dependência verdadeira* (*true dependence shrinking*). Após mostrar a sua deficiência definimos a sua generalização denominada *encolhimento por dependência verdadeira generalizada* (*generalized true dependence shrinking*).

- Na seção 3.3.2 descrevemos o método *afim por comando* (*affine by statement*).

3.3.1 Métodos encolhimento por dependência verdadeira e encolhimento por dependência verdadeira generalizada

O método *encolhimento por dependência verdadeira* (*true dependence shrinking*) consiste em computar a dependência verdadeira para cada vetor de dependência d_α definida como sendo o número de iterações (para laço 3-dimensional, como se i, j, k fossem hora, minuto e segundo de relógio e tudo é calculado em "segundo") [PeiC89, Poly88, ShaOF91]. Dado um algoritmo RUN n-dimensional de r vetores de dependência, encolhimento por dependência verdadeira transforma *Dom* n-dimensional num espaço 1-dimensional de acordo com a sua ordem seqüencial de execução (ordem lexicográfica). Cada vetor de dependência $d_\alpha = (d_\alpha^1, \cdots, d_\alpha^n)$ é transformado na distância de dependência verdadeira (vetor 1-dimensional). O vetor de dependência correspondente no novo espaço 1-dimensional é $d_\alpha^n + d_\alpha^{n-1}(N+1) + d_\alpha^{n-2}(N+1)^2 + \cdots + d_\alpha^1(N+1)^{n-1}$. Podemos interpretar isto como o uso de vetor de escalonamento $\pi_v = ((N+1)^{n-1}, (N+1)^{n-2}, \cdots, 1)$. O ponto $I \in Dom$ é mapeado no $\pi_v(I)$ num espaço 1-dimensional $\pi_v(Dom)$. O algoritmo transformado pode ser considerado um algoritmo 1-dimensional com $Dom = \pi_v(Dom)$ e a matriz de dependência é uma matriz linha $\pi_v \cdot D = (\pi_v^t \cdot d_1, \cdots, \pi_v^t \cdot d_r)$. Obviamente nenhum par de pontos I, J serão mapeados no mesmo ponto neste espaço 1-dimensional pois $\pi_v(I) \neq \pi_v(J), \forall I, J \in Dom$. O fator de redução é $\Delta = \min\{\pi_v \cdot d_\alpha \mid 1 \leq \alpha \leq r\}$. No novo algoritmo 1-dimensional Δ pontos consecutivos podem ser executados em paralelo. Para o Exemplo 4 (página 15), o encolhimento por dependência verdadeira transforma o espaço 2-dimensional da Figura 3.1 num espaço 1-dimensional da Figura 3.16 e $\pi_v = (N+1, 1)$.

Nesta figura as distâncias de dependência verdadeira de d_1 e d_2 são $3(N+1)+5 = 3N+8$ e $2(N+1)+4 = 2N+6$ respectivamente. Temos $\Delta = 2N+6$ e Δ pontos consecutivos no espaço 1-dimensional da Figura 3.16 são executados em paralelo. O ponto $I \in Dom$ é mapeado como $\pi_v(I)$ no espaço 1-dimensional. A nova matriz de dependência

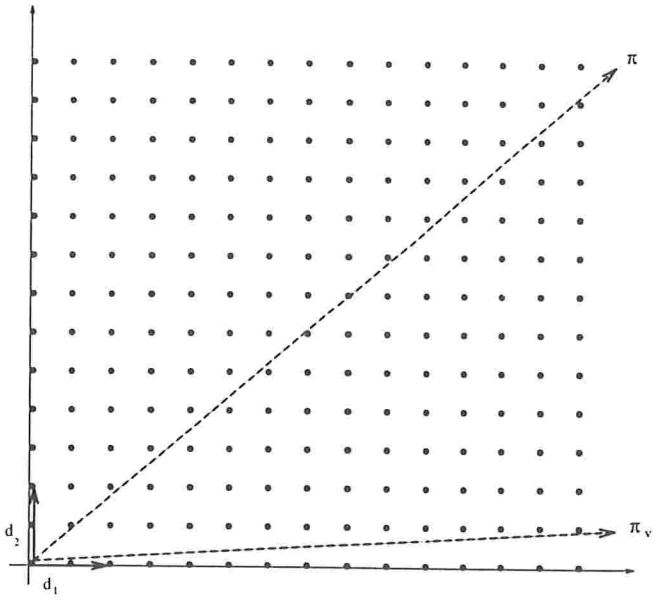

Figura 3.17: O exemplo para dependência generalizada

é $(N,1)\cdot D = (3N+8, 2N+6)$ e o fator de redução é $\Delta = \min\{3N+8, 2N+6\} = 2N+6$. Novamente temos observações interessantes. Considere o algoritmo do Exemplo 7 (página 18) com vetor de dependência $d = (0,1)^t$. Por encolhimento por dependência verdadeira a distância verdadeira correspondente é $\pi_v \cdot d = 1$ onde $\pi_v = (N+1,1)^t$. O fator de redução $\Delta = 1$. Deste modo o novo algoritmo pode ser executado apenas seqüencialmente e sem nenhuma paralelização possível. Entretanto é óbvio que se usamos outro π diferente do π_v podemos explorar o máximo do paralelismo. O número de passos é dado por $\lfloor \frac{\max\{\pi\cdot I - \pi\cdot J \mid I,J\in Dom\}}{\min\{\pi\cdot d_\alpha \mid 1\leq\alpha\leq r\}} \rfloor + 1$. No lugar do mapeamento π_v, π pode ser escolhido para que o número de passos seja mínimo. Este será ilustrado pelo exemplo da Figura 3.17 onde $Dom = \{I = (i,j) \in Z^2 \mid 0 \leq i,j \leq N\}$ e $D = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.

Usando π_v temos $\pi_v = (N+1,1)$ e o número de passos$=\lfloor \frac{(N+1,1)\cdot(N,N)^t}{2} \rfloor + 1 = \lfloor \frac{N(N+2)}{2} \rfloor + 1$. Agora considere $\pi = (N+1, N)$. $\pi\cdot D = (2(N+1), 2) > 0$ e $\pi(I) \neq \pi(J), \forall I \neq J \in Dom$. Portanto π é um escalonamento. O número de passos é $\lfloor \frac{(N+1,N)\cdot(N,N)^t}{2N} \rfloor + 1 = \lfloor \frac{N(2N+1)}{2N} \rfloor + 1$. O número de passos é menor quando $N > 1$. Temos várias escolhas para π. Em vez de projetar na n-ésima direção podemos projetar em outras direções. Estas observações levam Shang, O'Keefe e Fortes a proporem a definição de *encolhimento por dependência verdadeira generalizada* (*generalized true dependence shrinking*).

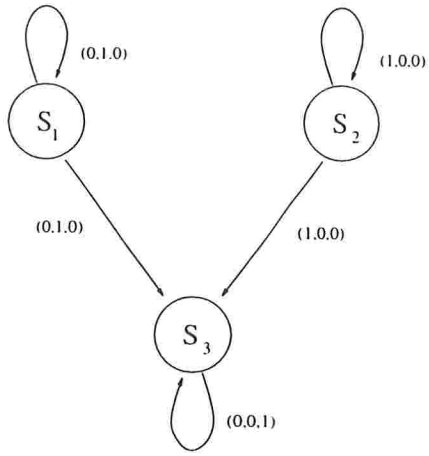

Figura 3.18: Grafo de dependência para o Exemplo 10

Encolhimento por dependência verdadeira generalizada

Ache $\pi : Z^n \rightarrow Z$ que minimiza a seguinte expressão $\frac{\max\{\pi \cdot I - \pi \cdot J \mid I,J \in Dom\}}{\min\{\pi \cdot d_\alpha \mid 1 \leq \alpha \leq r\}}$ sob condições:

- $\pi(I) \neq \pi(J), \forall I \neq J \in Dom.$
- $\pi \cdot D > 0.$
- $\text{mdc}\{\pi^1, \cdots, \pi^n\} = 1.$

Podemos notar que tanto encolhimento por dependência verdadeira como encolhimento por dependência verdadeira generalizada são casos particulares do método GSS.

3.3.2 Método afim por comando

Método *afim por comando* (*affine by statement*) [DarR92, DarR94] é uma extensão natural de escalonamento linear. Ele consiste em usar escalonamento afim para cada comando: instância I de cada comando S_α é executada no passo $\lfloor \pi_\alpha \cdot I + c_\alpha \rfloor$ ($\pi_\alpha \in Q^n$ e $c_\alpha \in Q$ onde Q é o conjunto dos números racionais). As condições que π_α devem satisfazer são as seguintes:

1. Se S_α depende de S_α por vetor de dependência d, então $\pi_\alpha \cdot d > 0$.
2. Se S_α depende de S_β por vetor de dependência d, então $\pi_\alpha \cdot I + c_\alpha > \pi_\beta \cdot (I - d) + c_\beta$ para $I \in Dom$.

Um exemplo bem simples para ilustrar este método é um algoritmo de multiplicação de matrizes quadradas. O seu grafo de dependência está na Figura 3.18.

Exemplo 10

$for\ i = 1\ to\ N\ do$
$\quad for\ j = 1\ to\ N\ do$
$\quad\quad for\ k = 1\ to\ N\ do$
$\quad\quad\quad S_1\colon a(i,j,k) = a(i,j-1,k)$
$\quad\quad\quad S_2\colon b(i,j,k) = b(i-1,j,k)$
$\quad\quad\quad S_3\colon c(i,j,k) = c(i,j,k-1) + a(i,j-1,k) * b(i-1,j,k)$

Aplicando o método GSS, é fácil ver que temos $\pi_0 = (1,1,1)$ com $\mathrm{GSS}(\pi_0) = 3N$. Pelo método afim por comando, podemos usar vetor de escalonamento π_i para cada comando S_i, $1 \leq i \leq 3$. Temos:

$S_1(I)$ executado no passo $(0,1,0) \cdot I$.
$S_2(I)$ executado no passo $(1,0,0) \cdot I$.
$S_3(I)$ executado no passo $(0,0,1) \cdot I$.

Com este escalonamento o número de passos é $2N$.

Novamente Dartes e Robert mostraram como determinar o melhor escalonamento afim por comando através de programação linear [DarR92, DarR94, DarR94-2, DarR95]. Com o uso deste método Dartes e Robert reduziram o número de passos para o Exemplo 1 (página 8) de [PeiC89] em $\frac{12}{7}N$ [DarR94].

Capítulo 4

Novos métodos por redução de dependência

Este capítulo é a parte central desta tese.

- Na seção 4.1 introduzimos o conceito de *domínio explícito* (DE), uma nova representação de dependências entre comandos em laços.
- Na seção 4.2 a utilidade de DE e o novo método *redução de dependência* (**RD**) são mostrados através de exemplos. Comparamos o novo método com os outros métodos através do bem conhecido exemplo de Peir e Cytron [PeiC89].
- Na seção 4.3 discutimos a estratégia de paralelização de acordo com a conexidade do grafo de dependência.
- Na seção 4.4 a **RD** é definida de modo formal e analisada. Damos uma condição simples e suficiente para aplicabilidade de **RD**.
- Na seção 4.5 tratamos de **RD** parcial que visa balanceamento entre o tempo de comunicação e o tempo de computação.
- A seção 4.6 trata de uma extensão de **RD** chamada **RD** generalizada. O método ataca grafos de dependência mais gerais.

4.1 Domínio explícito

Considere RUN de n laços, p comandos e r vetores de dependência. A fim de explicitar as dependências de modo mais claro vamos adotar a seguinte definição:

DE=domínio explícito=$\{S_1, \ldots, S_p\} \times Dom$

A definição de vetor de dependência também vai mudar:

$$S_\alpha \to S_\beta : d = \begin{pmatrix} d^1 \\ \vdots \\ d^n \end{pmatrix} \text{ passa a ser } d = \begin{pmatrix} S_\beta - S_\alpha \\ d^1 \\ \vdots \\ d^n \end{pmatrix}.$$

A subtração do primeiro componente é formal. Assim para o exemplo 1 (página 8), temos:

$DE = \{(S_\alpha, i, j) \mid 1 \leq \alpha \leq 4, 0 \leq i, j \leq N\}$

$$d_1 \text{ passa a ser } \begin{pmatrix} S_3 - S_1 \\ 0 \\ 2 \end{pmatrix}$$

$$\text{e } d_2 \text{ passa a ser } \begin{pmatrix} S_2 - S_3 \\ 1 \\ -6 \end{pmatrix}.$$

Para sua representação DE será sempre identificado como um subconjunto de Z^{n+1} de modo que cada $\{S_\alpha\} \times Dom$ seja identificado como $\{\alpha\} \times Dom$. Todos os pontos de DE serão ligados por vetores de dependência explicitamente.

Esta representação é similar a *augmented dependence graph* (ADG) proposto por Kyriakis-Bitzaros e Goutis [KayG92], mas é mais simples. Para laços de dimensão n com p comandos, a dimensão de ADG é $n + p + 1$ enquanto a dimensão de DE é sempre $n + 1$, independente de p, o que torna mais fácil sua representação visual.

No caso de RUN 2-dimensional, a vantagem desta representação é a seguinte: podemos projetar cada $\{S_\alpha\} \times Dom$ a Z^2 ligeiramente deslocado em relação aos outros $\{S_\beta\} \times Dom$ projetados, $\beta \neq \alpha$, evitando superposição. Deste modo, todas as dependências ficam explicitadas em Z^2. Veremos tudo isso com detalhe na próxima seção.

Usaremos Dom e DE de acordo com a conveniência e o primeiro componente adicionado a vetores de dependência fica implícito. Também não faremos distinção entre $S_\alpha \times Dom$ e $\{\alpha\} \times Dom$ que representa o anterior.

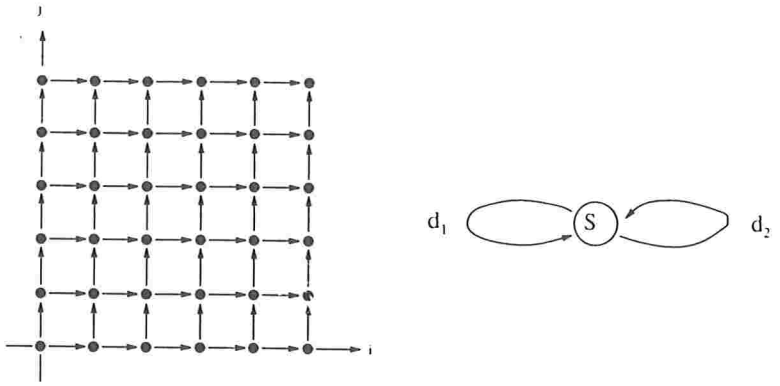

Figura 4.1: *Dom* e grafo de dependência do Exemplo 11

4.2 Exemplos

- Na seção 4.2.1 descrevemos um exemplo simples para mostrar a utilidade de *domínio explícito* (DE) e introduzimos de modo informal o método *redução de dependência* (**RD**).
- Na seção 4.2.2 descrevemos um caso mais complexo.
- Baseada na seção anterior a seção 4.2.3 faz comparação deste novo método com os outros métodos através do bem conhecido exemplo de Peir e Cytron [PeiC89].

4.2.1 Primeiro caso

Primeiro vamos examinar de novo o Exemplo 6 (página 16) no qual só existe um ciclo de dependência. O exemplo, apesar de ser bem simples, serve para mostrar a limitação do método GSS e ilustra a utilidade de DE.

Aplicando GSS ao Exemplo 6, tivemos a solução ótima GSS(π_0)=$2N$ passos com $\pi_0 = (1,1)$. É fácil observar também que NEW(π) dá o mesmo resultado. Agora considere um outro exemplo.

Exemplo 11

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S: $a(i,j) = a(i-1,j) + 2 * a(i,j-1)$

Apesar de o Exemplo 11 ser bastante diferente do Exemplo 6 (veja seu grafo de dependência na Figura 4.1 e o grafo de dependência do Exemplo 6 na Figura 3.2 (página 17)), seus *Dom* e D são idênticos aos do Exemplo 6. Conseqüentemente o método GSS dá o mesmo resultado para os dois exemplos.

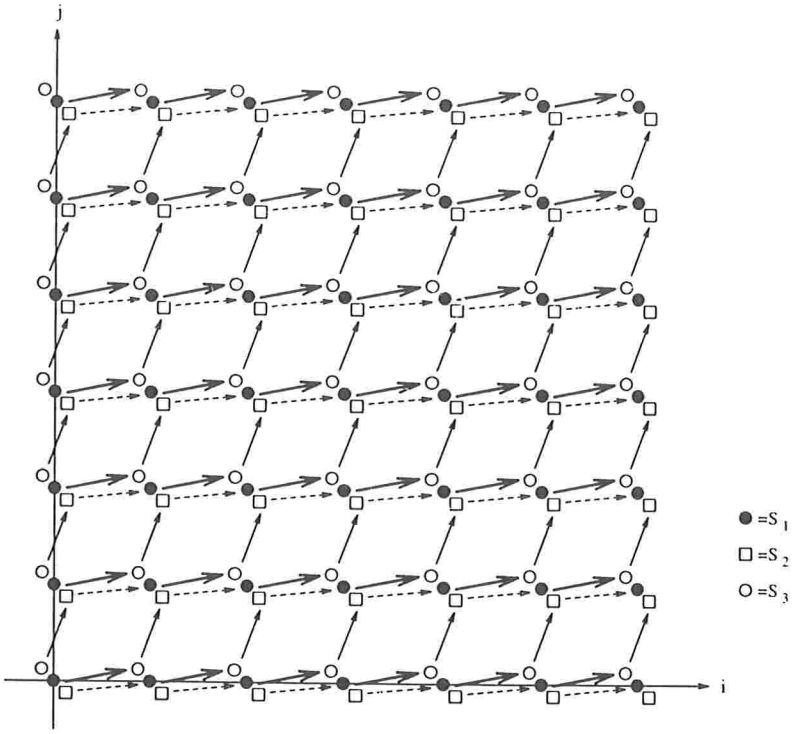

Figura 4.2: DE para o Exemplo 6

Ficamos satisfeitos com isto? Não, afirmamos que podemos melhorar bastante o escalonamento do Exemplo 6 e esta afirmação será bem ilustrada ao abandonar Dom passando a usar $DE = \{S_1, S_2, S_3\} \times Dom$. Para obter a representação 2-dimensional de DE vamos projetar cada $\{(S_\alpha, i, j) \mid \alpha \in \{1,2,3\}, (i,j) \in Dom\}$, em Z^2 de modo ligeiramente deslocado um em relação a outro evitando que $S_1(i,j)$, $S_2(i,j)$ e $S_3(i,j)$ sejam sobrepostos. O resultado com seus vetores de dependência explícitos está na Figura 4.2.

Note que para o Exemplo 11 $Dom = DE$ e, por exemplo, $S(4,4)$ depende de $S(3,4)$ que por sua vez depende de $S(2,4)$. $S(4,4)$ também depende de $S(4,3)$ que por sua vez depende de $S(4,2)$ e assim por diante (ver Figura 4.1). Entretanto a situação do Exemplo 6 é bem diferente (ver Figura 4.2): $S_1(4,4)$ depende de $S_2(3,4)$ mas não depende de nenhum $S_\alpha(i,4)$ para $\alpha \in \{1,2,3\}$ e $i < 3$, e também não depende de nenhum $S_\alpha(4,j)$ para $\alpha \in \{1,2,3\}$ e $j < 4$.

Observamos que o que temos na Figura 4.2 são vários "zig-zag's" de dependências que não interferem um com o outro, o que nos deixa maior liberdade para escalonamento do que o método GSS.

Por exemplo, considere o "zig-zag" constituído pelo ciclo $S_1(2,3) \rightarrow S_3(3,3) \rightarrow S_2(3,4) \rightarrow S_1(4,4)$. As dependências envolvem comandos diferentes. Agora deixando de lado suas localizações em DE, vamos juntar as computações $S_3(3,3)$ e $S_2(3,4)$ a $S_1(4,4)$, ou seja em $(4,4)$. Teremos um *macro* comando $\overline{S}$ que corresponde à seqüência de comandos:

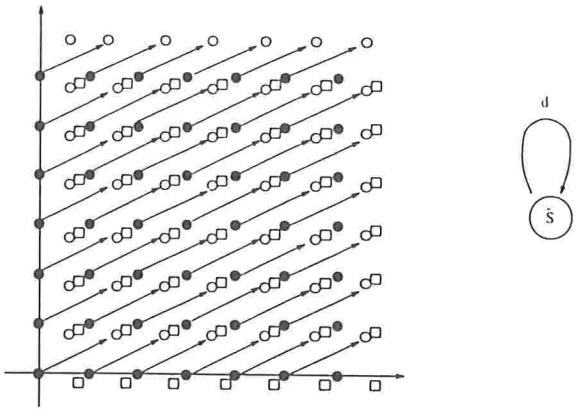

Figura 4.3: Grafo de dependência novo

$$\overline{S}\begin{cases} S_3 : c(i-1,j-1) = f_3(a(i-2,j-1)) \\ S_2 : b(i-1,j) = f_2(c(i-1,j-1)) \\ S_1 : a(i,j) = f_1(b(i-1,j)) \end{cases}$$

Agora faremos esta transformação a todos os pontos de DE e temos o seguinte.

1. O macro comando $\overline{S}$ será mapeado a um processador. Note que no cálculo de $\overline{S}$, S_3 calcula $c(i-1,j-1)$, que é usado em S_2 para calcular $b(i-1,j)$ que, por sua vez, é usado em S_1 para calcular $a(i,j)$. Como esses valores estão no mesmo processador, não há necessidade de comunicação.

2. O macro comando $\overline{S}$ levará maior tempo de execução por envolver a execução de três comandos de fato.

3. O vetor de dependência do macro comando $\overline{S}$ será mais simples: $d = d_1 + d_2 + d_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ (ver Figura 4.3). Em outras palavras os vetores de dependência de um ciclo são reduzidos a um só vetor, daí o nome de *redução de dependência* (**RD**).

Pela Figura 4.3 é fácil observar que o número total de passos requeridos é $\frac{N}{2}$. Como cada ponto envolve a execução de três funções (f_1, f_2, f_3), o tempo total para execução será calculado do seguinte modo:

Sejam $Comm$ e $Comp$ como na seção 2.4 (página 12) e considere nulo o tempo gasto para comunicação interna num processador. Então cada passo consiste no cálculo de três funções e uma comunicação. Portanto o tempo total gasto é $\frac{N}{2}(Comm + 3\ Comp)$ (enquanto que por GSS o tempo é $2N(Comm + Comp)$). Mais ainda, para o mapeamento (página 12) podemos projetar ao longo do vetor d e o tempo será $\frac{3N}{2}\ Comp$. Note que para o método GSS não podemos eliminar a comunicação entre processadores já que temos dois vetores de dependência não colineares.

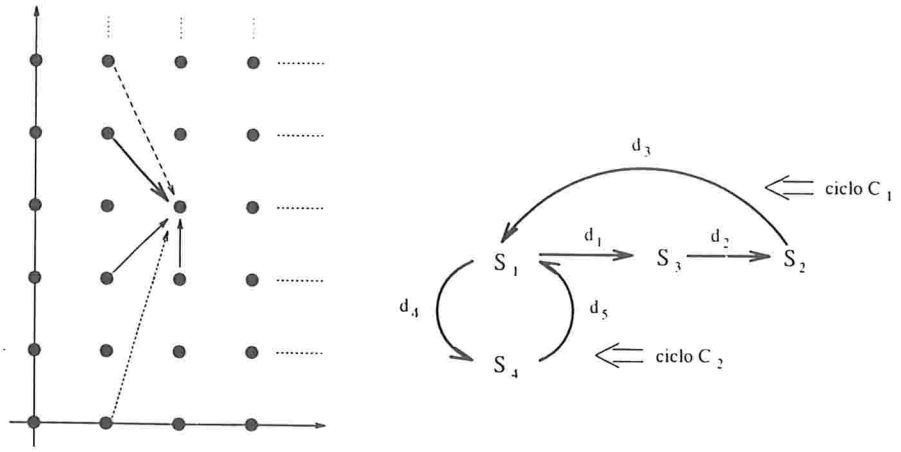

Figura 4.4: *Dom* e o grafo de dependência do Exemplo 12

4.2.2 Segundo caso

Considere agora o seguinte exemplo no qual temos dois ciclos no grafo de dependência. Neste exemplo DE vai servir para mostrar quais são as dependências essenciais e quais são as dependências secundárias, o que vai dar uma base intuitiva da transformação do grafo de dependência que faremos na seção 4.4.

Exemplo 12

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    $S_1 : a(i,j) = f_1(b(i-1, j-3) + e(i-1, j+2))$
    $S_2 : b(i,j) = f_2(c(i-1, j+1))$
    $S_3 : c(i,j) = f_3(a(i-1, j-1))$
    $S_4 : e(i,j) = f_4(a(i, j-1))$

Dom e o grafo de dependência estão na Figura 4.4.

Construímos $DE = \{S_1, \ldots, S_4\} \times Dom$ e projetamos os 4 planos a Z^2 e obtemos Figura 4.5.

A Figura 4.5 é decomposta em Figura 4.6 e Figura 4.7 que mostram as dependências dos ciclos C_1 e C_2. A Figura 4.8 mostra as dependências em relação a $S_1(i,j)$ em particular.

Nas Figuras 4.6 e 4.7 temos novamente "zig-zag's" independentes, cada um dos quais pode ser tratado como no Exemplo 6. Porém, o Exemplo 12 é mais restritivo que o Exemplo 6. O escalonamento do ciclo 1 deve ser compatível com o escalonamento do ciclo 2 por possuírem um ponto de intersecção que é S_1 (ver Figura 4.4). Pela Figura 4.8 junto com a observação acima podemos notar que as posições dos S_1's (pontos de intersecção dos dois ciclos) são essenciais para escalonamento.

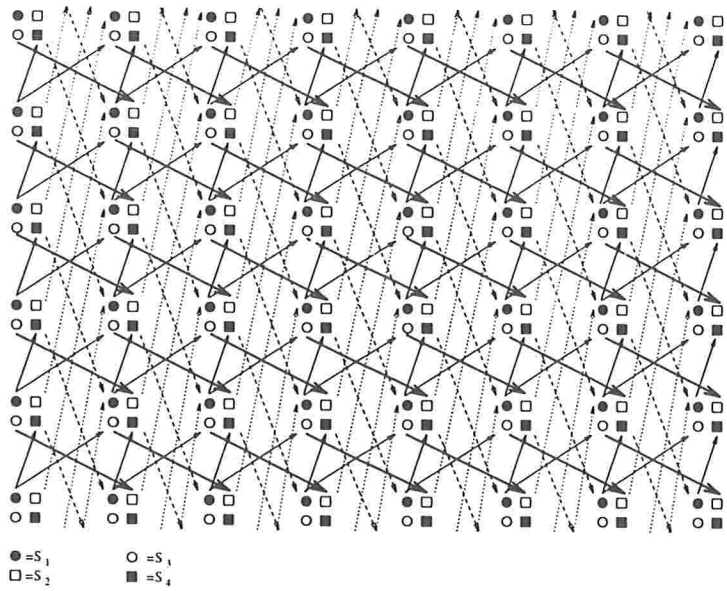

Figura 4.5: DE para o Exemplo 12

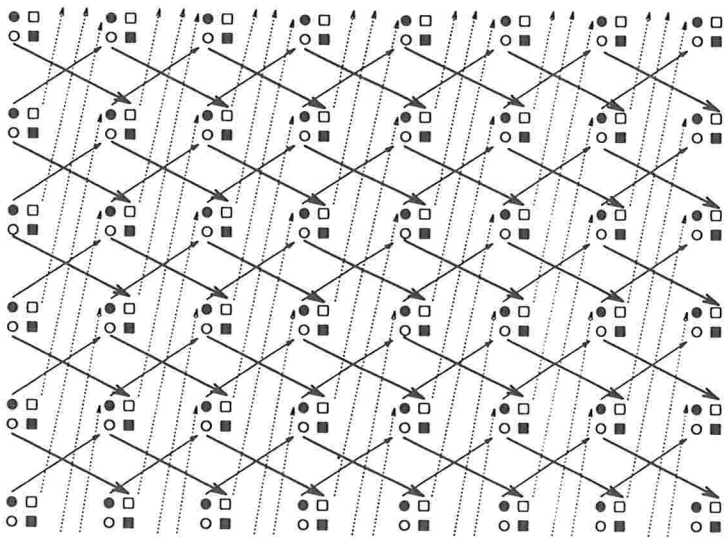

Figura 4.6: Dependência do ciclo C_1

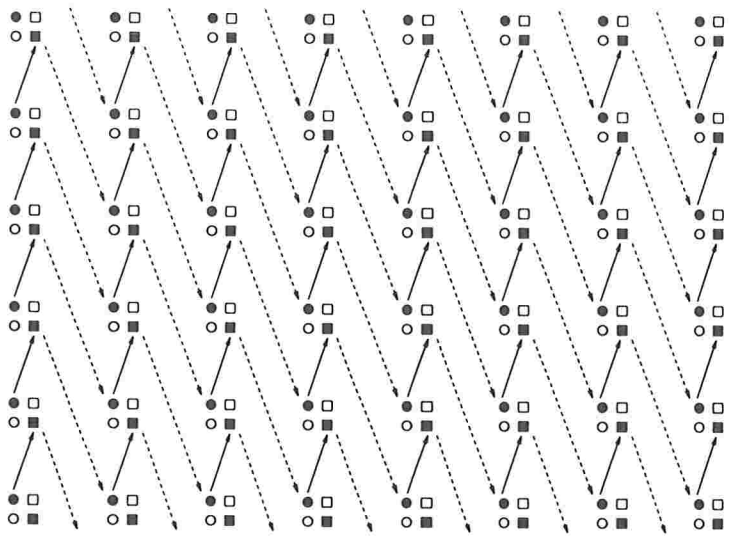

Figura 4.7: Dependência do ciclo C_2

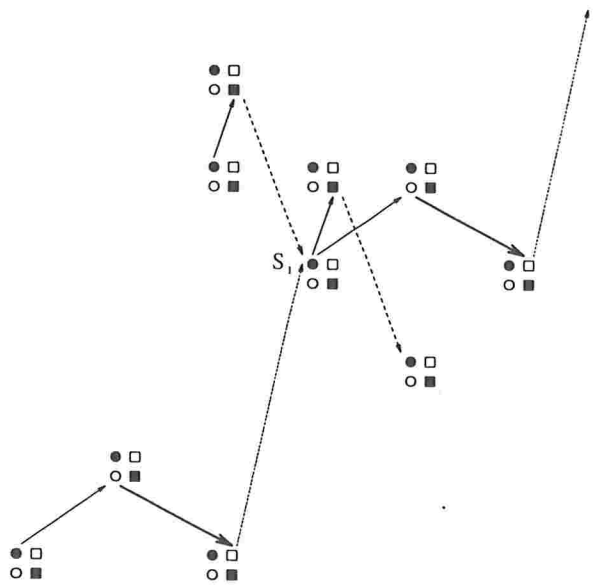

Figura 4.8: Dependência para $S_1(i, j)$

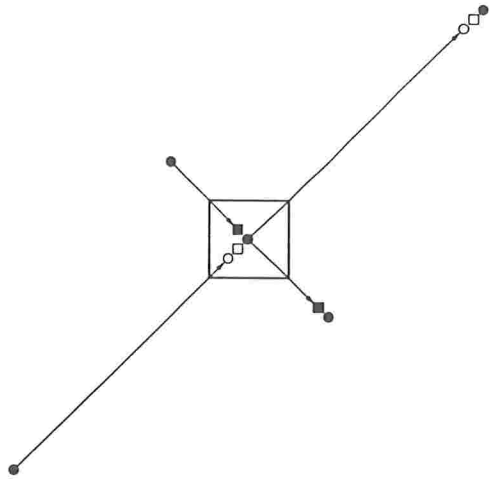

Figura 4.9: Nova dependência para macro $\overline{S}(i,j)$

Deste modo queremos analisar as dependências unicamente em função de S_1. Como foi feito no Exemplo 6, vamos juntar as computações de $S_2(i-1,j-3)$, $S_3(i-2,j-2)$ e $S_4(i-1,j+2)$ em $S_1(i,j)$ para formar macro comando $\overline{S}$.

Assim consideremos $\overline{S}$ como um macro comando para um ponto pertencente a *Dom*. Obteremos Figura 4.9 como Figura 4.8 transformada após esta consideração. O grafo de dependência com este macro é mostrado na Figura 4.10 onde os dois novos vetores de dependência são obtidos como somas dos vetores de cada ciclo.

Observações

1. Tanto no Exemplo 6 como no Exemplo 12 os macros para pontos situados na borda do domínio são incompletos (veja, por exemplo, na Figura 4.3 para o Exemplo 6 em que os macros na borda direita são compostos apenas por S_2 e S_3).

2. A criação do macro comando significa transferir algumas comunicações inter-processadores para dentro de um mesmo processador. Isto pode contribuir para a redução do tempo total gasto. Na próxima seção trataremos este aspecto detalhadamente.

3. O número de vetores de dependência é reduzido após a transformação do grafo de dependência por *redução de dependência*. Isto resulta em redução de comunicações entre processadores. Por exemplo, no Exemplo 6 em vez de ter 3 vetores de dependência para as variáveis a, b, e c passamos a ter apenas um vetor de dependência para a variável a. As comunicações para variáveis b e c ficam '*escondidas*' em processadores.

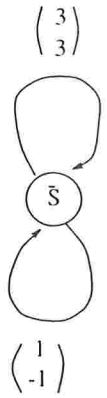

Figura 4.10: Novo grafo de dependência para o Exemplo 12

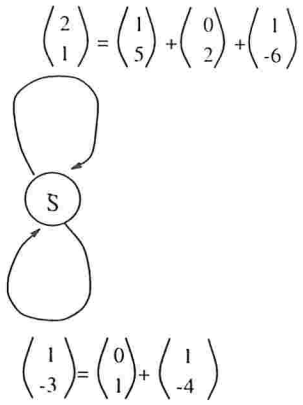

Figura 4.11: Novo grafo para o Exemplo 1

4.2.3 Terceiro caso

Vejamos agora de novo Exemplo 1 (página 8), o bem conhecido exemplo considerado em [RobS92] e outros. Note que o Exemplo 1 é praticamente igual ao Exemplo 12, exceto em alguns índices. Eles apresentam os mesmos ciclos (lado direito da Figura 4.4, com diferentes dependências). Aplicando *redução de dependência* de modo análogo ao Exemplo 12, temos o grafo da Figura 4.11.

Para este grafo $\pi_0^{\cdot} = (4, -1)$ é a solução ótima para GSS com o número de passos igual a $\frac{5}{7}N$ (ver apêndice A.2 (página 76)) . Assim o tempo gasto total será $\frac{5}{7}N(Comm+4Comp)$.

O tempo gasto por GSS é $8N(Comm + Comp)$ e o tempo gasto por GSS combinado com ISM é $2N(Comm + Comp)$ [RobS92] . Por outro lado o tempo gasto por *afim por comando* é $\frac{12}{7}N(Comm + Comp)$ [DarR94].

A tabela seguinte resume os resultados dos vários métodos para o Exemplo 1.

	Computação	Comunicação
GSS	$8.NComp$	$8.NComm$
GSS combinado com ICM	$2.NComp$	$2.NComm$
Afim por comando	$\frac{12}{7}NComp$	$\frac{12}{7}NComm$
Redução de dependência	$\frac{20}{7}NComp$	$\frac{5}{7}NComm$

Comparação entre os métodos da tabela

- GSS apresenta o maior tempo.
- Se $Comm > \frac{2}{3}Comp$, então *redução de dependência* é melhor que GSS combinado com ICM.
- Se $Comm > \frac{8}{7}Comp$, então *redução de dependência* é melhor que *afim por comando.*
- Concluindo, se $Comm > \frac{8}{7}Comp$, então *redução de dependência* apresenta o menor tempo entre os métodos da tabela.
- Seja $Comm = \gamma\ Comp$, então $\frac{\frac{12}{7}NComp+\frac{12}{7}NComm}{\frac{20}{7}NComp+\frac{5}{7}NComm} = \frac{\frac{12}{7}N+\frac{12}{7}N\gamma}{\frac{20}{7}N+\frac{5}{7}N\gamma} = \frac{12+12\gamma}{20+5\gamma}$ Assim o fator de ganho é quase $\frac{12}{5} = 2.4$ em relação ao melhor resultado obtido por *afim por comando* se γ for grande.

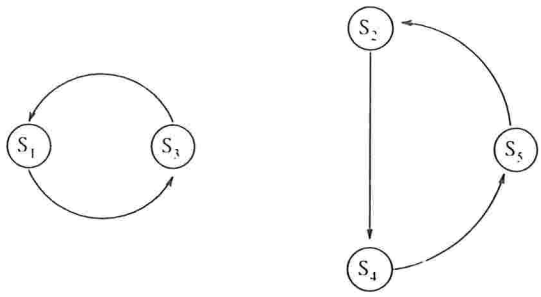

Figura 4.12: O grafo de dependência do Exemplo 13

4.3 Considerações sobre conexidade

Discutiremos brevemente sobre a conexidade do grafo de dependência.

- Se o grafo de dependência de um algoritmo RUN for desconexo, então as computações de cada componente conexo são independentes dos outros componentes. Podemos paralelizar cada componente separadamente (ver o Exemplo 13 e Figura 4.12) . Portanto vamos considerar apenas algoritmos RUN cujos grafos de dependência são conexos daqui em diante.

  Exemplo 13

  for $i = 0$ *to* N *do*
      for $j = 0$ *to* N *do*
          S_1: $a(i,j) = f_1(b(i-1,j))$
          S_2: $x(i,j) = f_2(z(i,j-1))$
          S_3: $b(i,j) = f_3(a(i-1,j+2))$
          S_4: $y(i,j) = f_4(x(i,j-3))$
          S_5: $z(i,j) = f_5(y(i-1,j-5))$

- Se o grafo de dependência for conexo mas acíclico, então DE vai ser um conjunto de várias cópias deste grafo e a paralelização será trivial neste caso. Portanto estes casos serão excluídos daqui em diante. Veja o seguinte Exemplo 14 e Figuras 4.13 e 4.14.

  Exemplo 14

  for $i = 0$ *to* N *do*
      for $j = 0$ *to* N *do*
          $S_1 : a(i,j) = f_1(i,j)$
          $S_2 : b(i,j) = f_2(a(i-1,j-1))$

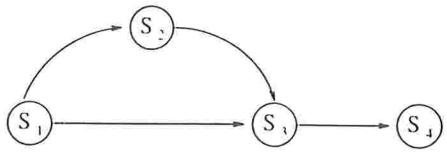

Figura 4.13: O grafo de dependência do Exemplo 14

$$S_3 : c(i,j) = f_3(a(i-2, j+1), b(i-1, j+2)$$
$$S_4 : e(i,j) = f_4(c(i-1, j-1))$$

- Como nosso alvo é encolhimento de ciclo vamos restringir nossa atenção ao grafo de dependência fortemente conexo. Logo quaisquer vértices e quaisquer arestas fazem parte de algum ciclo.

A seguir resumimos a natureza de paralelização de acordo com a conexidade do grafo de dependência G:

Resumo

$$G = \begin{cases} \text{desconexo} & \rightarrow \text{ atacar cada componente conexo} \\ \text{conexo} & \rightarrow \begin{cases} \text{acíclico} & \rightarrow \text{ trivial} \\ \text{cíclico} & \rightarrow \begin{cases} \text{fortemente conexo} & \rightarrow \text{ tema deste capítulo} \\ \text{não fortemente conexo} & \rightarrow \text{ futuro trabalho} \end{cases} \end{cases} \end{cases}$$

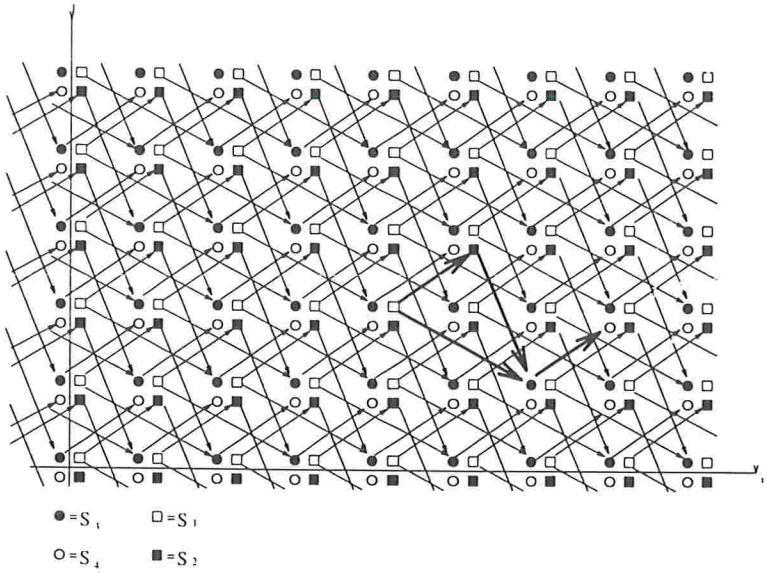

Figura 4.14: *Dom* do Exemplo 14

4.4 Redução de dependência (RD)

- Na seção 4.4.1 descrevemos o caso simples de **RD** para grafo de dependência igual a um ciclo e o macro comando resultante de **RD**.
- Na seção 4.4.2 formalizamos **RD** para grafos mais complexos e descrevemos o macro comando resultante de **RD**.
- Na seção 4.4.3 discutimos o efeito de **RD** quando o grafo é reduzido a um vértice.
- Na seção 4.4.4 descrevemos uma condição suficiente e simples para que **RD** seja vantajosa em relação a outros métodos.

4.4.1 Grafo de dependência igual a um ciclo

Este é um caso bastante simples. Como todos os vetores de dependência são lexicograficamente positivos (página 8) podemos permutar os comandos no corpo do laço sem alterar a semântica do algoritmo. Então, sem perda de generalidade, o corpo com p comandos tem a seguinte forma (ver a Figura 4.15):

S_1: $x_1(I) = f_1(x_p(I - d_p))$
S_2: $x_2(I) = f_2(x_1(I - d_1))$
$\vdots$
S_{p-1}: $x_{p-1}(I) = f_{p-1}(x_{p-2}(I - d_{p-2}))$
S_p: $x_p(I) = f_p(x_{p-1}(I - d_{p-1}))$

Tal ciclo é transformado no ciclo apresentado na Figura 4.16.

Macro comando após a transformação

Após esta transformação o algoritmo vai ter o seguinte macro comando $\overline{S}$ (exceto para as bordas do domínio como foi visto na observação 1 da seção 4.2.2 (página 43)):

$$\overline{S}\begin{cases} x_1(I - d_1 - d_2 - \cdots - d_{p-1}) = f_1(x_p(I - d_p - d_1 - d_2 - \cdots - d_{p-1})) \\ x_2(I - d_2 - d_3 - \cdots - d_{p-1}) = f_2(x_1(I - d_1 - d_2 - \cdots - d_{p-1})) \\ \vdots \\ x_{p-1}(I - d_{p-1}) = f_{p-1}(x_{p-2}(I - d_{p-2} - d_{p-1})) \\ x_p(I) = f_p(x_{p-1}(I - d_{p-1})) \end{cases}$$

O tempo gasto total será de

$$\min_{1 \le j \le n} \lfloor \frac{N}{d^j} \rfloor (Comm + p\ Comp)$$

onde d^j=j-ésimo elemento de $d = d_1 + d_2 + \cdots + d_p$.

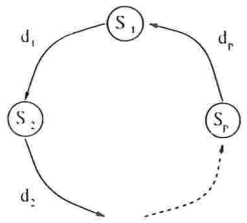

Figura 4.15: Grafo de dependência original

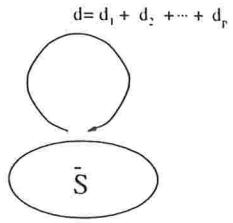

Figura 4.16: Grafo de dependência transformado

4.4.2 Grafo de dependência com mais de um ciclo

Seja A um algoritmo RUN e seja $G = (V, E)$ o grafo de dependência para A no qual V e E correspondem ao conjunto de comandos $S_1, \ldots S_p$ e ao de dependências entre comandos, respectivamente. G é suposto sempre fortemente conexo com mais de um ciclo de dependência. Cada aresta de E é rotulada pelo seu vetor de dependência. Usaremos a notação $S \xrightarrow{d} S'$ para denotar a aresta do vértice S ao vértice S' via vetor de dependência d. Esta notação não é usual mas tem a vantagem de ser uniforme tanto para dependência entre comandos como para aresta no grafo que representa o algoritmo. Usaremos S tanto para denotar um comando como para denotar um vértice correspondente no grafo de dependência.

Definimos a quantidade de arestas que entram em S e saem de S como segue:

Definição 2

$$g^+(S) = |\{d \mid \exists(S' \xrightarrow{d} S) \in E\}|$$
$$g^-(S) = |\{d \mid \exists(S \xrightarrow{d} S') \in E\}|$$

Excluiremos o caso trivial de $g^+(S) = g^-(S) = 0$. Também serão desconsiderados vértices com $g^+(S) = 0$ e $g^-(S) > 0$ (ou com $g^-(S) = 0$ e $g^+(S) > 0$) por ser fortemente conexo. Deste modo teremos $g^+(S) > 0$ e $g^-(S) > 0$, $\forall S \in V$.

Vamos dividir V em dois conjuntos: o conjunto dos vértices secundários (VS) e o conjunto dos vértices principais (VP).

Definição 3

Um vértice é vértice secundário *se* $g^+(S) = g^-(S) = 1$.
Um vértice é vértice principal *se não é secundário.*

Note que um vértice secundário trivial é um vértice com apenas uma aresta para ele mesmo. Todo grafo de dependência é suposto fortemente conexo. Conseqüentemente se G tem um vértice secundário trivial, então o grafo tem apenas este vértice e esta aresta. Vamos desconsiderar este caso trivial.

Definição 4

$VS = \{S \in V \mid S \text{ é secundário}\}$
$VP = \{S \in V \mid S \text{ é principal}\}$

Vejamos o Exemplo 12 (página 40). Para este exemplo, $VS = \{S_2, S_3, S_4\}$ e $VP = \{S_1\}$.

Definição 5 *Uma* rota simples *é um caminho ou um circuito no qual os vértices do começo e do fim são vértices principais e todos os vértices intermediários, quando existem, são vértices secundários.*

Uma rota simples trivial é uma aresta que liga dois vértices principais sem vértices intermediários.

O método de redução de dependência (**RD**) transforma o grafo de dependência $G = (V, E)$ com VS e VP em um novo grafo chamado *grafo transformado.*

Definição 6

$\overline{G} = (\overline{V}, \overline{E})$ *é um grafo transformado no qual*
$\overline{V} = VP$ *onde cada* $S \in VP$ *será denotado por* $\overline{S}$
$\overline{E} = \{\overline{S} \xrightarrow{d} \overline{S'} \mid S, S' \in VP$ *e* $(S \xrightarrow{d} S') \in E$ *ou em* G *existe uma rota simples não trivial entre* S *e* S' *e* d *é a soma dos vetores de dependência que compõem esta rota simples*$\}$

Note que se houver α rotas simples entre S e S' em G então $\overline{G}$ vai ter α arestas entre $\overline{S}$ e $\overline{S'}$.

Os comandos correspondentes aos vértices secundários nesta rota simples serão incorporados como um macro comando de $\overline{S'}$. Certamente se houver outra rota simples deste tipo para S' então os comandos correspondentes aos vértices secundários nesta rota simples serão incorporados também.

Após obter $\overline{G}$ por **RD**, aplicamos um método conveniente de escalonamento usando as dependências de $\overline{G}$ e seja $\overline{T_{pep}}$ o número de passos obtido com este escalonamento. Em $\overline{G}$ podem existir vértices com diferentes números de vértices secundários incorporados. Seja L o número máximo de vértices secundários que foram incorporados num vértice principal.

Então o tempo total para o algoritmo será $\overline{T_{pep}}Comm + (L+1)\overline{T_{pep}}Comp$.

Definição de macro comando correspondente a um vértice principal

Seja $S \in VP$ e suponha que S tem w rotas simples entrando nele (ver a Figura 4.17 que mostra rotas simples 1, 2 e w) . Denotamos por x uma variável calculada pela função g correspondente ao S. Supomos que cada rota simples α, $1 \leq \alpha \leq w$, tem η_α vértices secundários. Cada vértice secundário da rota simples α será denotado por $S_{\alpha\beta}$, cada variável calculada neste vértice será denotada por $y_{\alpha\beta}$, a função que calcula $y_{\alpha\beta}$ será denotada por $f_{\alpha\beta}$ e aresta que sai do vértice $S_{\alpha\beta}$ será rotulado por $d_{\alpha\beta}$, $1 \leq \beta \leq \eta_\alpha$. Cada uma destas rotas simples começa num vértice principal que pode ser o próprio S. Denotamos por S_* o vértice principal deste tipo e por x_* a variável calculada no S_*. A aresta que sai de S_* para $S_{\alpha 1}$ será rotulado por $d_{\alpha *}$, $1 \leq \alpha \leq w$.

Supomos que existem v vértices principais dos quais S depende. Note que novamente eles podem ser o próprio S. Denotamos por S_α um vértice deste tipo e x_α a variável calculada neste vértice e d_α a aresta que sai do S_α para S, $1 \leq \alpha \leq v$. Na Figura 4.17 não colocamos as arestas que saem do vértice S a fim de não sobrecarregar a figura. O vértice S calcula x em função de $y_{1\eta_1}, y_{2\eta_2}, \cdots y_{w\eta_w}$ e $x_1, \cdots x_v$. Como foi feito para grafo de dependência igual a um ciclo, podemos supor sem perda de generalidade que o corpo do algoritmo possui o seguinte bloco B.

$$
B \begin{cases}
B_1 \begin{cases} y_{11}(I) = f_{11}(x_*(I - d_{1*})) \\ y_{12}(I) = f_{12}(y_{11}(I - d_{11})) \\ \vdots \\ y_{1\eta_1}(I) = f_{1\eta_1}(y_{1\eta_1 - 1}(I - d_{1\eta_1 - 1})) \end{cases} \\
\vdots \\
\vdots \\
B_\alpha \begin{cases} y_{\alpha 1}(I) = f_{\alpha 1}(x_*(I - d_{\alpha *})) \\ y_{\alpha 2}(I) = f_{\alpha 2}(y_{\alpha 1}(I - d_{\alpha 1})) \\ \vdots \\ y_{\alpha\eta_\alpha}(I) = f_{\alpha\eta_\alpha}(y_{\alpha\eta_\alpha - 1}(I - d_{\alpha\eta_\alpha - 1})) \end{cases} \\
\vdots \\
\vdots \\
B_w \begin{cases} y_{w1}(I) = f_{w1}(x_*(I - d_{w*})) \\ y_{w2}(I) = f_{w2}(y_{w1}(I - d_{w1})) \\ \vdots \\ y_{w\eta_w}(I) = f_{w\eta_w}(y_{w\eta_w - 1}(I - d_{w\eta_w - 1})) \end{cases} \\
x(I) = g(y_{1\eta_1}(I - d_{1\eta_1}), \cdots, y_{\alpha\eta_\alpha}(I - d_{\alpha\eta_\alpha}), \cdots, y_{w\eta_w}(I - d_{w\eta_w}), x_1(I - d_1), \cdots, x_v(I - d_v))
\end{cases}
$$

B_α, $1 \leq \alpha \leq w$, é o sub-bloco de B correspondente à rota simples α. Após **RD** este bloco vai se transformar no seguinte macro comando (novamente exceto para as bordas do domínio):

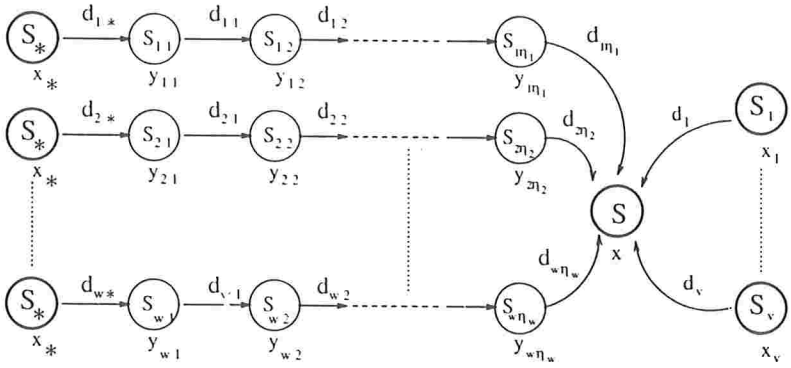

Figura 4.17: O vértice S e rotas simples entrando nela

$$
\overline{S}\begin{cases}
\overline{S_1}\begin{cases} y_{11}(I-d_{11}-d_{12}\cdots-d_{1\eta_1}) = f_{11}(x_*(I-d_{1*}-d_{11}-d_{12}\cdots-d_{1\eta_1})) \\ y_{12}(I-d_{12}-d_{13}\cdots-d_{1\eta_1}) = f_{12}(y_{11}(I-d_{11}-d_{12}\cdots-d_{1\eta_1})) \\ \vdots \\ y_{1\eta_1}(I-d_{1\eta_1}) = f_{1\eta_1}(y_{1\eta_1-1}(I-d_{1\eta_1-1}-d_{1\eta_1})) \end{cases} \\
\vdots \\
\vdots \\
\overline{S_\alpha}\begin{cases} y_{\alpha 1}(I-d_{\alpha 1}-d_{\alpha 2},\cdots,-d_{\alpha\eta_\alpha}) = f_{\alpha 1}(x_*(I-d_{\alpha *}-d_{\alpha 1}-d_{\alpha 2}\cdots-d_{\alpha\eta_\alpha})) \\ y_{\alpha 2}(I-d_{\alpha 2}-d_{\alpha 3}\cdots-d_{\alpha\eta_\alpha}) = f_{\alpha 2}(y_{\alpha 1}(I-d_{\alpha 1}-d_{\alpha 2}\cdots-d_{\alpha\eta_\alpha})) \\ \vdots \\ y_{\alpha\eta_\alpha}(I-d_{\alpha\eta_\alpha}) = f_{\alpha\eta_\alpha}(y_{\alpha\eta_\alpha-1}(I-d_{\alpha\eta_\alpha-1}-d_{\alpha\eta_\alpha})) \end{cases} \\
\vdots \\
\vdots \\
\overline{S_w}\begin{cases} y_{w1}(I-d_{w1}-d_{w2}\cdots-d_{w\eta w}) = f_{w1}(x_*(I-d_{w*}-d_{w1}-d_{w2}\cdots-d_{w\eta w})) \\ y_{w2}(I-d_{w2}-d_{w3}\cdots-d_{w\eta w}) = f_{w2}(y_{w1}(I-d_{w1}-d_{w2}\cdots-d_{w\eta w})) \\ \vdots \\ y_{w\eta_w}(I-d_{w\eta_w}) = f_{w\eta_w}(y_{w\eta_w-1}(I-d_{w\eta_w-1}-d_{w\eta_w})) \end{cases} \\
x(I) = g(y_{1\eta_1}(I-d_{1\eta_1}),\cdots,y_{\alpha\eta_\alpha}(I-d_{\alpha\eta_\alpha}),\cdots,y_{w\eta_w}(I-d_{w\eta_w}),x_1(I-d_1),\cdots,x_v(I-d_v))
\end{cases}
$$

$\overline{S_\alpha}$, $1 \leq \alpha \leq w$, é o sub-bloco do macro $\overline{S}$ correspondente a B_α.

Observações

Este processo de incorporar vértices secundários aos vértices principais reduz o número de vetores de dependência e conseqüentemente, após aplicar mapeamento, reduz as comunicações entre processadores.

A idéia básica da transformação é a seguinte: o que nos importa realmente para achar

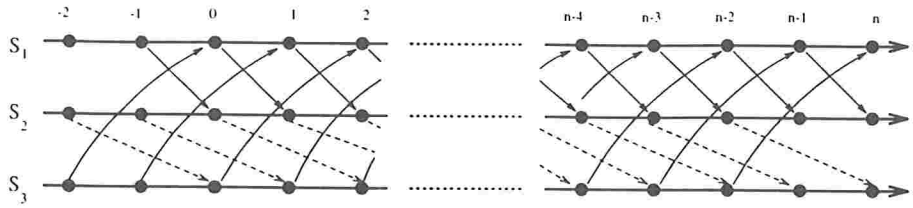

Figura 4.18: *Dom* do Exemplo 15

um bom escalonamento são as dependências entre vértices principais. As dependências entre vértices secundários, bem como entre um vértice secundário e um principal, têm pouca importância. Em outras palavras, as localizações dos pontos P de DE correspondentes a estes vértices ($P \in S \times Dom$ onde $S \in VS$) não são importantes.

A outra vantagem do novo método é a seguinte: A redução do número de vetores de dependência contribuirá para facilitar a aplicação do método GSS que em casos gerais é muito complexo [ShaF91]. Essa redução pode até tornar possível que o cálculo seja feito manualmente.

O questionamento natural ao método proposto seria o seguinte: "O método proposto visa reduzir o tempo de comunicação entre processadores, aumentando entretanto o tempo de computação. O tempo total não poderia até aumentar?" Na seção 4.5 mostraremos um novo método, *redução de dependência parcial*, visando um equilíbrio entre comunicação e computação.

Computações nas bordas do domínio

Agora veremos o que acontece nas bordas do domínio através do seguinte exemplo simples com um laço.

Exemplo 15

for $i = 0$ *to* N *do*
S_1: $a(i) = f_1(c(i-2))$
S_2: $b(i) = f_2(a(i-1))$
S_3: $c(i) = f_3(b(i-2))$

O macro para este algoritmo é o seguinte, exceto para as bordas.

$$\overline{S} \begin{cases} a(i-3) = f_1(c(i-5)) \\ b(i-2) = f_2(a(i-3)) \\ c(i) = f_3(b(i-2)) \end{cases}$$

Levando em conta as bordas podemos representar o algoritmo todo com seguinte macro.

for $i = 0$ to $N+3$ do

$$\overline{S}\begin{cases} \text{if } 3 \leq i & a(i-3) = f_1(c(i-5)) \\ \text{if } 2 \leq i \leq N+2 & b(i-2) = f_2(a(i-3)) \\ \text{if } i \leq N & c(i) = f_3(b(i-2)) \end{cases}$$

Podemos representar do seguinte modo alternativo também.

for $i = 0$ to 1 do
$c(i) = f_3(b(i-2))$

for $i = 2$ do

$$\overline{S_I}\begin{cases} b(i-2) = f_2(a(i-3)) \\ c(i) = f_3(b(i-2)) \end{cases}$$

for $i = 3$ to N do

$$\overline{S_{II}}\begin{cases} a(i-3) = f_1(c(i-5)) \\ b(i-2) = f_2(a(i-3)) \\ c(i) = f_3(b(i-2)) \end{cases}$$

for $i = N+1$ to $N+2$ do

$$\overline{S_{III}}\begin{cases} a(i-3) = f_1(c(i-5)) \\ b(i-2) = f_2(a(i-3)) \end{cases}$$

for $i = N+3$ do
$a(i-3) = f_1(c(i-5))$

4.4.3 Efeito da transformação sobre a escolha de π quando o grafo é reduzido a um vértice

Seja G o grafo de dependência original e $\overline{G}$ o grafo transformado conforme seção 4.4.2. Suporemos que $\overline{G}$ contém apenas um vértice.

Como temos apenas um vértice vamos usar GSS.
Seja π_0 vetor que minimiza $\mathrm{GSS}(\pi) = \frac{\max\{\pi \cdot I - \pi \cdot J \mid I,J \in Dom\}}{\min\{\pi \cdot d_\alpha \mid d_\alpha \in D\}}$ em G. Então será que π_0 serve como vetor de escalonamento para $\overline{G}$ também?

Como π_0 é um vetor de escalonamento para G, temos $\mathrm{mdc}\{\pi_0^1, \ldots, \pi_0^n\} = 1$ e $\pi_0 \cdot d_\alpha > 0$ para $\forall d_\alpha \in D$. Por outro lado cada $\overline{d_\beta} \in \overline{D}$ é a soma de alguns d_α's de D. Assim $\pi_0 \cdot \overline{d_\beta} > 0$, para $\forall \overline{d_\beta} \in \overline{D}$ e portanto π_0 é um vetor de escalonamento para $\overline{G}$.

Seja $\overline{\pi_0}$ o vetor que minimiza $\overline{\mathrm{GSS}}(\pi) = \frac{\max\{\pi \cdot I - \pi \cdot J \mid I,J \in Dom\}}{\min\{\pi \cdot \overline{d_\beta} \mid \overline{d_\beta} \in \overline{D}\}}$ em $\overline{G}$.

Qual é a relação entre π_0 e $\overline{\pi_0}$? Qual é o valor de $\overline{\mathrm{GSS}}(\pi_0)$?

Mostraremos que $\overline{\mathrm{GSS}}(\pi_0) \leq \mathrm{GSS}(\pi_0)$. Ou seja, o número de passos do método redução

de dependência nunca é superior ao do método GSS.

Sem perda de generalidade, seja d_1 o vetor de dependência tal que $\pi_0 \cdot d_1$ seja mínimo entre $\pi_0 \cdot d_\alpha$, para $d_\alpha \in D$. Logo $\pi_0 \cdot d_1 \leq \pi_0 \cdot d_\alpha, \forall d_\alpha \in D$ e $\pi_0 \cdot d_1 \leq \pi_0 \cdot \overline{d_\beta}$, para $\forall \overline{d_\beta} \in \overline{D}$. Temos $\min\{\pi \cdot \overline{d_\beta} \mid \overline{d_\beta} \in \overline{D}\} \geq \min\{\pi \cdot d_\alpha \mid d_\alpha \in D\}$. Deste modo temos $\overline{\text{GSS}}(\pi_0) \leq \text{GSS}(\pi_0)$ e com maior razão $\overline{\text{GSS}}(\overline{\pi_0}) \leq \text{GSS}(\pi_0)$.

Por outro lado, o caso infeliz de $\overline{\text{GSS}}(\pi_0) = \text{GSS}(\pi_0)$ só vai acontecer se $d_1 = \overline{d_1}$ onde $\overline{d_1}$ é o vetor que minimiza $\pi_0 \cdot \overline{d_\beta}$ para $\overline{d_\beta} \in \overline{D}$. Mesmo neste caso há chance de achar $\overline{\pi_0}$ tal que $\overline{\text{GSS}}(\overline{\pi_0}) < \overline{\text{GSS}}(\pi_0)$ já que temos menos restrição na busca de $\overline{\pi_0}$ do que π_0.

Em resumo, como os novos vetores de dependência são somas de vetores tais que $\pi \cdot d > 0$ e o número de vetores de dependência diminui,

> disp(π) que é o denominador da formula de GSS tende a aumentar,
> espaço de busca para π aumenta com a diminuição do número de vetores de dependência.

Assim aumenta a possibilidade de melhoria no escalonamento.

4.4.4 Aplicabilidade

Quando é vantajoso o método de redução de dependência em relação aos outros métodos? Uma condição simples e suficiente (mas não necessária) para responder esta questão é comparar um "*piso*" de tempo total gasto dos outros métodos com um "*teto*" de tempo total gasto do método de redução de dependência. Este "*piso*" é dado através do caminho mais longo no DE. Se o comprimento deste caminho for T_c, então o tempo gasto total é sempre maior ou igual a $T_c(Comm + Comp)$. Quanto ao "*teto*", o seguinte fato garante que temos boa chance de calcular um "*teto*" com facilidade.

Fato

Os vetores de dependência antes da redução são todos lexicograficamente positivos. Portanto ao efetuar redução de dependência, que consiste na soma de vetores lexicograficamente positivos, aumenta a possibilidade de ter uma direção na qual todos os vetores de dependência resultantes têm seus elementos positivos. Se tal direção existe podemos paralelizar nesta direção.

Vejamos um exemplo.

Exemplo 16

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    for $k = 0$ *to* N *do*
      comando S_1: $a(i,j,k) = f_1(e(i-1,j-1,k))$
      comando S_2: $b(i,j,k) = f_2(a(i,j-1,k-1))$
      comando S_3: $c(i,j,k) = f_3(e(i,j,k-1))$
      comando S_4: $e(i,j,k) = f_4(b(i,j,k-1), c(i,j-2,k+1))$

$$S_4 \to S_1\colon\ d_{11} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$S_1 \to S_2\colon\ d_{12} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$S_2 \to S_4\colon\ d_{13} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$S_4 \to S_3\colon\ d_{21} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$S_3 \to S_4\colon\ d_{22} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$$

O caminho mais comprido tem comprimento $\frac{3N}{2}$ que é o número de passos necessários para completar os ciclos $S_4 \to S_1 \to S_2 \to S_4$ na direção j ou na direção k. Logo um "*chão*" vai ser $\frac{3N}{2}(Comm + Comp)$.

Por outro lado efetuando a redução de dependência, temos:

$$\overline{d_1} = d_{11} + d_{12} + d_{13} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}$$

$$\overline{d_2} = d_{21} + d_{22} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}$$

Os dois vetores têm elementos positivos na direção j e o mínimo destes elementos é 2. Paralelizando nesta direção o número de passos vai ser igual a $\frac{N}{2}$. Assim um "*teto*" vai ser $\frac{N}{2}(Comm + 4Comp)$.

Comparando o "*chão*" e o "*teto*", temos

$\frac{N}{2}Comm + \frac{4N}{2}Comp < \frac{3N}{2}Comm + \frac{3N}{2}Comp$.

Podemos concluir que método de redução de dependência vai ser garantidamente vantajoso se $Comp < 2Comm$. Além disso seja $Comm = \gamma\ Comp$, então $\frac{\frac{3N}{2}Comm + \frac{3N}{2}Comp}{\frac{N}{2}Comm + \frac{4N}{2}Comp} = \frac{\frac{3N}{2}\gamma + \frac{3N}{2}}{\frac{N}{2}\gamma + \frac{4N}{2}} = \frac{3\gamma+3}{\gamma+4}$. Assim o fator de ganho é quase 3 se γ for grande.

Observação

Enfatizamos que a comparação dá apenas uma condição suficiente. O fato de a desigualdade não ser satisfeita ou não obter "*teto*" não implica em que o método de redução de dependência seja desvantajoso.

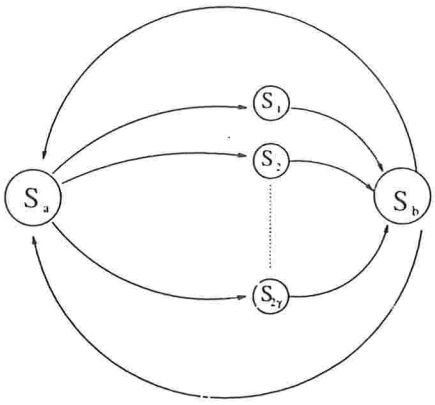

Figura 4.19: O exemplo extremo (Exemplo 17)

4.5 Redução de dependência parcial

- Na seção 4.5.1 damos um exemplo para motivar **RD** parcial.
- Na seção 4.5.2 definimos **RD** parcial e damos a condição de aplicabilidade de **RD** parcial.

4.5.1 Exemplo

Como foi visto com os exemplos na seção 4.2, o método **RD** (redução de dependência) visa diminuir o tempo total gasto com a redução de tempo de comunicação, aumentando entretanto tempo de computação.

Se não for bem aplicada, esta estratégia pode até aumentar o tempo total gasto. Daí surge a necessidade de balancear comunicação e computação. Este é o tema desta seção.

Vejamos o seguinte exemplo extremo (Figura 4.19).

Exemplo 17 *Considere um algoritmo cujo grafo de dependência é como na Figura 4.19. Supomos que* $\gamma\ Comp = 1\ Comm$, $1 \ll \gamma$.

Os S_α, $1 \leq \alpha \leq 2\gamma$, são todos vértices secundários. Podemos aplicar **RD** ao vértice S_b. Temos o grafo transformado mostrado na Figura 4.20.

Seja $P \in Dom$ correspondente ao vértice transformado. O tempo total gasto neste ponto será $1\ Comm + (2\gamma + 1)\ Comp = 2\ Comm + (\gamma + 1)\ Comp$. Porém se não aplicarmos **RD**, o tempo total gasto para o conjunto de comandos $S_1 \cdots S_{2\gamma}$ e S_b será $2\ Comm + 2\ Comp < 2\ Comm + (\gamma + 1)\ Comp$ (um passo para $S_1 \cdots S_{2\gamma}$ e um passo para S_b).

4.5.2 Redução de dependência parcial

Seja A um algoritmo RUN e G seu grafo de dependência.

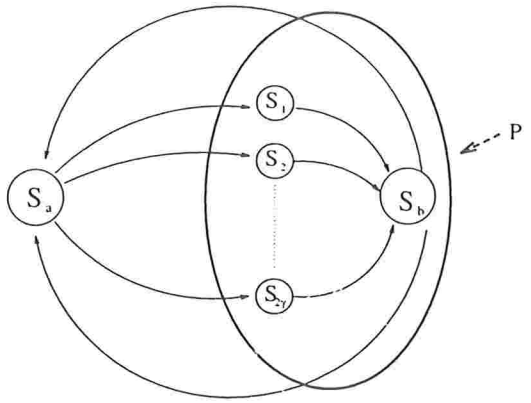

Figura 4.20: O exemplo extremo após **RD**

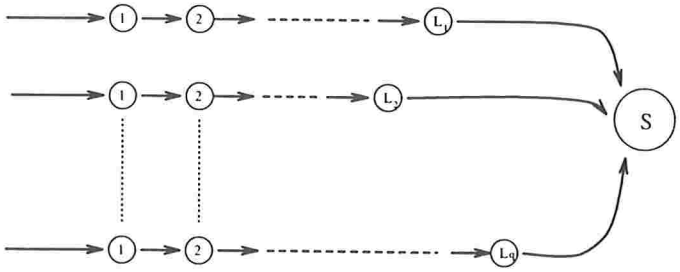

Figura 4.21: Vértice S

Seja S um vértice principal e vamos supor que existem q rotas simples entrando no vértice S (ver Figura 4.21). Cada rota simples α tem L_α vértices secundários, $1 \leq \alpha \leq q$. Rotas simples que saem de S e os outros tipos de caminhos que entram em S não interessam para a análise a seguir pois eles não têm influência no tempo gasto para comunicação e computação do ponto de Dom correspondente a este vértice. Nosso interesse neste momento é como tratar estes $(L_1 + L_2 + \cdots + L_q + 1)$ vértices. Na Figura 4.21 são mostrados apenas estes q rotas simples pelo mesmo motivo.

Se aplicarmos o método **RD** a este vértice, ponto $P \in Dom$ correspondente vai ter um macro com $(L_1 + L_2 + \cdots + L_q + 1)$ funções (ver Figura 4.22).

O tempo gasto total para este ponto será $1\ Comm + (L_1 + L_2 + \cdots + L_q + 1)\ Comp$. Como foi visto na página 51 o tempo total gasto do algoritmo é determinado por vértice principal que incorporou o maior número de vértices secundários.

Então se o coeficiente $(L_1 + L_2 + \cdots + L_q + 1)$ for muito grande, o método **RD** pode até aumentar o tempo total gasto como o exemplo extremo já mostrado. Balanceamento entre comunicação e computação vai evitar esta ocorrência indesejável. Este balanceamento será feito com possível introdução de um vértice intermediário em cada rota simples.

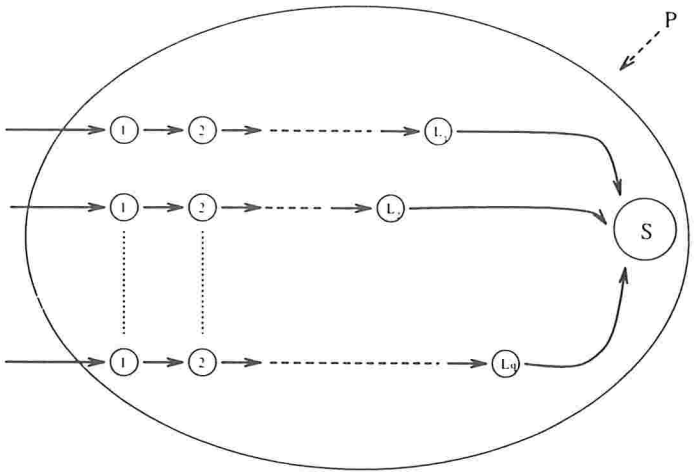

Figura 4.22: Após RD

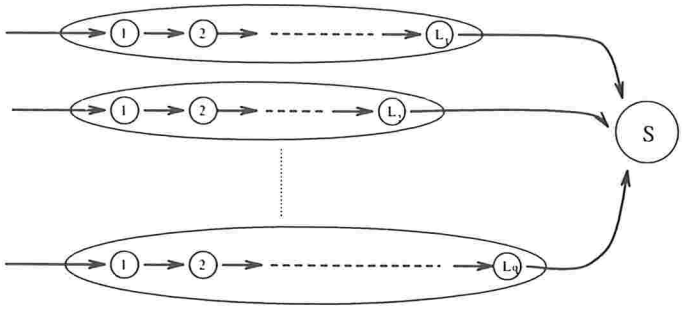

Figura 4.23: Após RD parcial

Conseqüentemente vamos ter uma comunicação a mais porém diminuímos computação.

Em cada rota simples α criamos um vértice que agrupa L_α vértices secundários, $1 \leq \alpha \leq q$ (ver Figura 4.23).

Seja $M = \max\{L_\alpha \mid 1 \leq \alpha \leq q\}$

Agora o ponto S tem q vetores de dependência todos vindo de pontos, cada qual com um macro formado por no máximo M funções.

Temos uma comunicação para estes novos pontos. O tempo gasto na execução paralela para estes pontos é $M\ Comp$. Temos ainda uma comunicação para S e a computação no S.

Assim o tempo gasto para estes pontos será

$(1\ Comm + M\ Comp) + (1\ Comm + 1\ Comp) = 2\ Comm + (M + 1)Comp$

Chamamos esta transformação de **RD** parcial.

Quando RD parcial é melhor que RD?

RD parcial é melhor que RD se e somente se:

$$\begin{aligned} 2\ Comm + (M+1)Comp &< 1\ Comm + (L_1 + L_2 + \cdots + L_q + 1)Comp \\ 1\ Comm &< (L_1 + L_2 + \cdots + L_q - M)Comp \\ \frac{Comm}{Comp} &< L_1 + L_2 + \cdots + L_q - M \end{aligned}$$

.

Se $L_1, L_2, \cdots, L_q$ e M satisfazem a inequação acima, então **RD** parcial é melhor que **RD**.

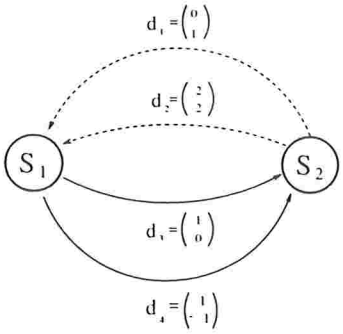

Figura 4.24: Exemplo 18

4.6 Redução de dependência generalizada

- Na seção 4.6.1 descrevemos de modo informal **RD** generalizada através de dois exemplos.
- Na seção 4.6.2 definimos formalmente **RD** generalizada e descrevemos a definição do macro.
- Na seção 4.6.3 descrevemos a condição suficiente para aplicabilidade de **RD** generalizada.

4.6.1 Exemplos

Como foi feito na seção 4.2, primeiro vamos examinar um exemplo bem simples para introduzir a idéia de *redução de dependência generalizada* (**RD** generalizada).

Exemplo 18

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S_1: $a(i,j) = f_1[b(i,j-1), b(i-2,j-2)]$
    S_2: $b(i,j) = f_2[a(i-1,j), a(i-1,j+1)]$

Temos quatro vetores de dependência (ver Figura 4.24):

$$d_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$d_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$d_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

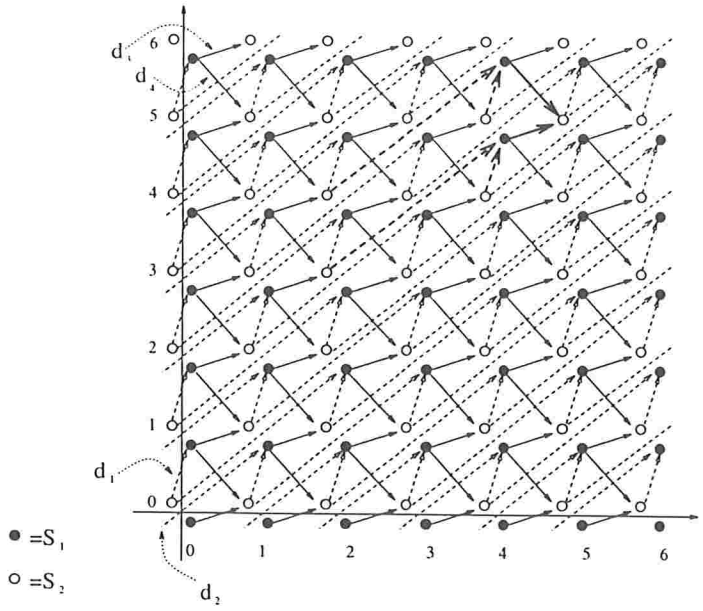

Figura 4.25: DE para o Exemplo 18

$$d_4 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

O DE do Exemplo 18 é mostrado na Figura 4.25. Qual será o tempo para executar este algoritmo? Da Figura 4.25 é fácil ver que o caminho mais longo tem comprimento igual a $2N$. Assim qualquer escalonamento deste algoritmo leva, no mínimo, $2N$ passos. O tempo total gasto será então maior ou igual a $2N$ $(Comm + Comp)$. Pela Figura 4.24 o grafo de dependência só tem 2 vértices, ambos principais. Portanto não podemos aplicar **RD**. Porém, com um pequeno truque que vamos introduzir, podemos contornar esta situação. Examinemos a Figura 4.25 como fizemos na seção 4.2.2. Focalizemos nossa atenção apenas nas dependências entre, por exemplo, os pontos (S_2, i, j). Notamos que $S_2(5,5)$ depende de $S_1(4,6)$ via vetor de dependência d_4 e de $S_1(4,5)$ via d_3. $S_1(4,6)$ por sua vez depende de $S_2(2,4)$ via d_2 e de $S_2(4,5)$ via d_1. $S_1(4,5)$ depende de $S_2(2,3)$ via d_2 e de $S_2(4,4)$ via d_1. Enfim, focalizando nossa atenção em S_2, temos $(5,5)$ dependente de $(2,4)$, $(4,5)$, $(2,3)$ e $(4,4)$.

Deixamos de lado, por enquanto, o fato de que S_1 não é vértice secundário. Aplicamos **RD** e teremos 4 novos vetores de dependência:

$$d_I = d_1 + d_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$d_{II} = d_2 + d_4 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

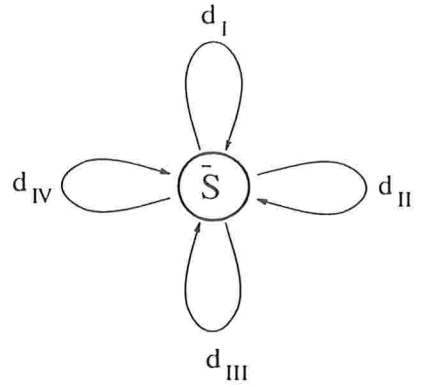

Figura 4.26: Novo grafo de dependência para o Exemplo 18

$$d_{III} = d_1 + d_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$
$$d_{IV} = d_2 + d_3 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

O novo grafo de dependência é mostrado na Figura 4.26.

O domínio *Dom* para este novo grafo de dependência está na Figura 4.27. Todos os quatro vetores de dependência d_I a d_{IV} têm elemento positivo na direção i. Podemos paralelizar no eixo i. O mínimo destes elementos é um. Logo o número de passos requeridos é igual a N. O tempo gasto para comunicação será reduzido para metade, ou seja, N *Comm*. Entretanto não sabemos ainda o tempo gasto para computação pois não sabemos como seria o macro comando $\overline{S}$. Mais ainda, nem sabemos se o processo é correto ou não.

Voltemos à Figura 4.25 para respondermos a estas dúvidas. Nesta figura temos:

- $b(i,j)$ depende de $a(i-1,j)$ e $a(i-1,j+1)$.
- $a(i-1,j)$ depende de $b(i-1,j-1)$ e $b(i-3,j-2)$.
- $a(i-1,j+1)$ depende de $b(i-1,j)$ e $b(i-3,j-1)$.

Agora faremos um truque. Alteramos o corpo dos laços encaixados para a seguinte forma:

Exemplo 19

for $i = 0$ *to* $N+1$ *do*
  for $j = -1$ *to* N *do*

$$\overline{S}\begin{cases} a(i-1,j) = f_1[b(i-1,j-1), b(i-3,j-2)] \\ a(i-1,j+1) = f_1[b(i-1,j), b(i-3,j-1)] \\ b(i,j) = f_2[a(i-1,j), a(i-1,j+1)] \end{cases}$$

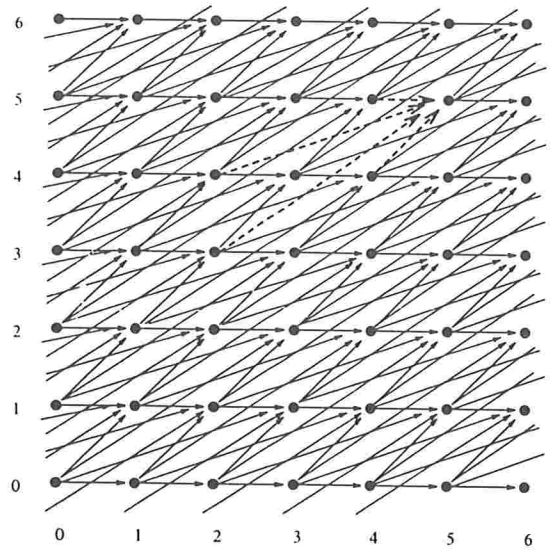

Figura 4.27: Novo Dom para o Exemplo 18

Definimos um macro $\overline{S}$ composto por estes três comandos. Para este algoritmo o tempo gasto total vai ser $N\ Comm + 3N\ Comp$ para paralelização na direção i. Com um pequeno cálculo, podemos ver que GSS também dá o mesmo resultado. O novo escalonamento leva menos tempo se e somente se

$N\ Comm + 3N\ Comp < 2N\ Comm + 2N\ Comp$

i.e. se e somente se

$Comp < Comm.$

Seja $Comm = \gamma\ Comp$. Então

$\frac{2N\ Comm + 2N\ Comp}{N\ Comm + 3N\ Comp} = \frac{2\gamma+2}{\gamma+3}$.

Assim o fator de ganho é quase 2 se γ for grande.

Observações

- Notamos que como o macro $\overline{S}$ calcula duas vezes f, cada valor de a é calculado duas vezes nos pontos diferentes de Dom. Por exemplo, para $(i,j) = (4,4)$ calculam-se $a(3,4)$ e $a(3,5)$ e para $(i,j) = (4,5)$ calculam-se $a(3,5)$ e $a(3,6)$.

- Além disso, introduzimos algumas instâncias de iteração desnecessárias para comando S_1. Por exemplo para $i = 0$, $a(-1,j) = f[b(-1,j-1), b(-3,j-2)]$ é desnecessária.

- Fora estes dois fatos, o algoritmo de exemplo 19 é semanticamente equivalente ao algoritmo do exemplo 18.

- Pela Figura 4.25, observamos que $S_2(5,5)$ e $S_2(5,4)$ dependem de $S_1(4,5)$ que por sua vez depende de $S_2(2,3)$ e $S_2(4,4)$. Pela Figura 4.27 observamos o seguinte.

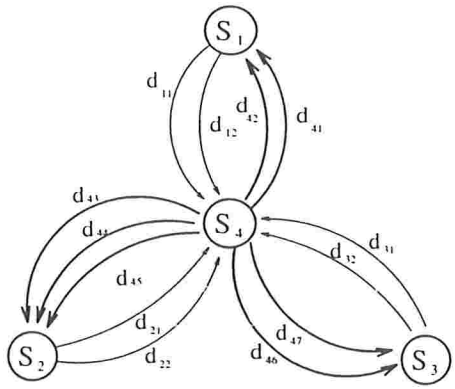

Figura 4.28: Grafo de dependência para o Exemplo 20

Ao colocarmos "duas vezes o comando S_1" no macro comando, passamos a ter as seguintes dependências:

- $\overline{S}(5,5)$ depende de $\overline{S}(2,3)$ e $\overline{S}(4,4)$
- $\overline{S}(5,4)$ depende de $\overline{S}(2,3)$ e $\overline{S}(4,4)$

A passagem por $S_1(4,5)$ na Figura 4.25 é um "gargalo" inerente do algoritmo original para comunicação e eliminamos este "gargalo" com a transformação acima.

Informalmente podemos dizer que *desamarramos* as dependências e reduzimos o número de passos.

Antes de apresentar o método de modo formal veremos mais um exemplo.

Exemplo 20

for $i = 0$ *to* N *do*
  for $j = 0$ *to* N *do*
    S_1: $a(i,j) = f_1[x(i,j-1), x(i-1,j-1)]$
    S_2: $b(i,j) = f_2[x(i-1,j+1), x(i-2,j), x(i-1,j-1)]$
    S_3: $c(i,j) = f_3[x(i-1,j-3), x(i-2,j+1)]$
    S_4: $x(i,j) = g[a(i-1,j), a(i-1,j-2), b(i,j-2), b(i-1,j-2), c(i,j-2), c(i,j-1)]$

Temos 13 vetores de dependência (ver a Figura 4.28):

- $d_{11} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $d_{12} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
- $d_{21} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$, $d_{22} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

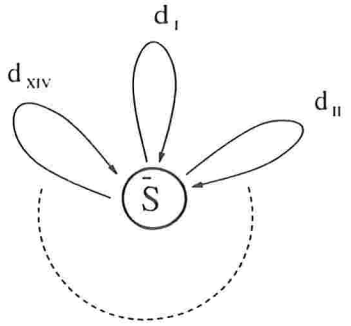

Figura 4.29: Grafo de dependência para o Exemplo 21

- $d_{31} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$, $d_{32} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- $d_{41} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $d_{42} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $d_{43} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $d_{44} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$,
  $d_{45} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $d_{46} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$, $d_{47} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$

Temos um ciclo $S_1 \xrightarrow{d_{11}} S_4 \xrightarrow{d_{41}} S_1$. É fácil ver que serão necessários $2N$ passos para completar este ciclo. Logo o tempo total gasto será no mínimo $2N$ $(Comm + Comp)$. Modificamos o algoritmo do Exemplo 20 do mesmo modo como foi feito no Exemplo 18. Temos o seguinte algoritmo.

Exemplo 21

for $i = 0$ *to* $N + 1$ *do*
  for $j = 0$ *to* $N + 2$ *do*

$$\overline{S} \begin{cases} a(i-1,j) = f_1[x(i-1,j-1), x(i-2,j-1)] \\ a(i-1,j-2) = f_1[x(i-1,j-3), x(i-2,j-3)] \\ b(i,j-2) = f_2[x(i-1,j-1), x(i-2,j-2), x(i-1,j-3)] \\ b(i-1,j-2) = f_2[x(i-2,j-1), x(i-3,j-2), x(i-2,j-3)] \\ c(i,j-2) = f_3[x(i-1,j-5), x(i-2,j-1)] \\ c(i,j-1) = f_3[x(i-1,j-4), x(i-2,j)] \\ x(i,j) = g[a(i-1,j), a(i-1,j-2), b(i,j-2), b(i-1,j-2), c(i,j-2), c(i,j-1)] \end{cases}$$

Temos 14 vetores de dependência (ver a Figura 4.29), todos com elemento positivo na direção i. O mínimo destes elementos é igual a 1. O macro $\overline{S}$ tem 7 comandos.

Logo o tempo total gasto é menor ou igual a $N\ Comm + 7N\ Comp$. O novo algoritmo gasta menos tempo se $N\ Comm + 7N\ Comp < 2N\ Comm + 2N\ Comp$, ou

$5\ Comp < Comm$.

Seja $Comm = \gamma\ Comp$, então $\frac{2N\ Comm + 2N\ Comp}{N\ Comm + 7N\ Comp} = \frac{2\gamma+2}{\gamma+7}$. Assim o fator de ganho é quase 2 se γ for grande.

Mérito de RD generalizada

Geralmente a paralelização de um algoritmo seqüencial tem como seu ponto de partida este algoritmo seqüencial. Deste modo o "*gargalo*" inerente do algoritmo original continua latente mesmo após a paralelização. Entretanto o método de **RD** generalizada, informalmente visto acima, ataca inclusive este "*gargalo*". Com isto ele alcança uma paralelização melhor mantendo a dependência implícita.

Na próxima seção veremos a formalização do método.

4.6.2 Definição formal de redução de dependência generalizada (RD generalizada)

Seja G um grafo de dependência fortemente conexo de um algoritmo RUN como na seção 4.4.

Definição 7 *Conjuntos de vértices que entram em S e saem de S:*

$$Adj^+(S) = \{S' \mid \exists\ (S' \xrightarrow{d} S) \in E\}$$
$$Adj^-(S) = \{S' \mid \exists\ (S \xrightarrow{d} S') \in E\}$$

Definição 8 *Um vértice $S \in V$ é* extremal *se*

$$|Adj^+(S)| = |Adj^-(S)| = 1$$
$$Adj^+(S) = Adj^-(S) \neq S$$

Definição 9 *Um vértice $S \in V$ é* central *se não é extremal.*

Para o Exemplo 20, S_1, S_2 e S_3 são vértices extremais e S_4 é um vértice central.

Definição 10

$$VE = \{S \in V \mid S \text{ é vértice extremal}\}$$
$$VC = \{S \in V \mid S \text{ é vértice central}\}$$

Definição 11 *Um vértice central $S_c \in VC$ é* central *a um vértice extremal $S_e \in VE$ se*

$$S_c = Adj^+(S_e)(= Adj^-(S_e)).$$

Para cada vértice extremal existe apenas um vértice central a ele.

Definição 12

$VC^+ = \{S_c \in VC \mid S_c$ *é central a algum* $S_e \in VE\}$
Para $S_c \in VC^+$, $VE(S_c) = \{S_e \in VE \mid S_c$ *é central a* $S_e\}$

Definição 13

$E^+(S) = \{d \mid \exists(S' \xrightarrow{d} S) \in E\}$
$E^-(S) = \{d \mid \exists(S \xrightarrow{d} S') \in E\}$
Para $S_c \in VC^+$, $E_{VE}(S_c) = \{(S_e \xrightarrow{d} S_c) \in E \mid S_e \in VE(S_c)\} \cup \{(S_c \xrightarrow{d} S_e) \in E \mid S_e \in VE(S_c)\}$

Definição 14 **RD** *generalizada para* $S_c \in VC^+$ *é uma transformação do grafo* $G = (V, E)$ *que resulta no grafo* $\overline{G} = (\overline{V}, \overline{E})$ *tal que:*

$$\overline{V} = V - VE(S_c)$$
$$\overline{E} = (E - E_{VE}(S_c)) \cup (\cup_{S_e \in VE(S_c)} \{(S_c \xrightarrow{d^+ + d^-} S_c) \mid d^+ \in E^+(S_e), d^- \in E^-(S_e)\})$$

Note que na definição acima S_c é central a cada S_e.

Definição 15 **RD** *generalizada para* $G = (V, E)$ *é aplicar* **RD** *generalizada para cada* $S_c \in VC^+$.

Note que para o Exemplo 18, S_1 e S_2 são ambos vértices extremais e não existe vértice central. Para casos triviais como este escolhemos um dos vértices como central arbitrariamente. No Exemplo 18 S_2 foi escolhido como central.

Definição de macro comando correspondente a um $S \in VC^+$

Seja $S \in VC^+$ e $\Omega = |VE(S)|$. Denotamos por x uma variável calculada pela função g correspondente ao S. Denotamos por y_α uma variável calculada pela função f_α correspondente ao $S_\alpha \in VE(S), 1 \leq \alpha \leq \Omega$. Para cada S_α, $|E^+(S_\alpha)| = \xi_\alpha$ e $|E^-(S_\alpha)| = \tau_\alpha$, $1 \leq \alpha \leq \Omega$ (veja a Figura 4.30).

Como foi feita na seção 4.4.2, sem perda de generalidade supomos que o corpo do laço tem o seguinte bloco B correspondente a $S \in VC^+$:

$$B \begin{cases} y_1(I) = f_1(x(I - d'_{11}), \cdots, x(I - d'_{1\xi_1})) \\ y_2(I) = f_2(x(I - d'_{21}), \cdots, x(I - d'_{2\xi_2})) \\ \vdots \\ y_\alpha(I) = f_\alpha(x(I - d'_{\alpha 1}), \cdots x(I - d'_{\alpha\xi_\alpha})) \\ \vdots \\ y_\Omega(I) = f_\Omega(x(I - d'_{\Omega 1}), \cdots, x(I - d'_{\Omega\xi_\Omega})) \\ x(I) = g(y_1(I - d_{11}), \cdots, y_1(I - d_{1\tau_1}), \cdots, y_\Omega(I - d_{\Omega 1}), \cdots, y_\Omega(I - d_{\Omega\tau_\Omega}), \cdots\cdots) \end{cases}$$

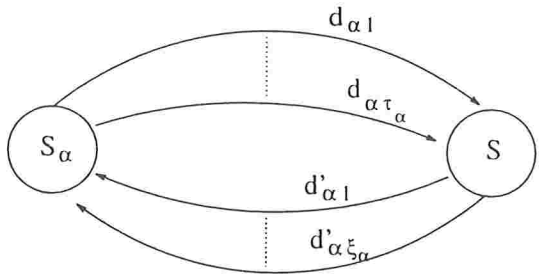

Figura 4.30: O vértice central S e o vértice extremal S_α

Na lista de parâmetros para a função f, não especificamos as variáveis calculadas nos vértices centrais.
Após **RD** generalizada para S, este bloco vai se transformar no seguinte macro comando:

$$
\overline{S}\begin{cases}
\begin{cases} y_1(I-d_{11}) = f_1(x(I-d'_{11}-d_{11}),\cdots,x(I-d'_{1\xi_1}-d_{11})) \\ \vdots \\ y_1(I-d_{1\tau_1}) = f_1(x(I-d'_{11}-d_{1\tau_1}),\cdots,x(I-d'_{1\xi_1}-d_{1\tau_1})) \end{cases} \\
\begin{cases} y_2(I-d_{21}) = f_2(x(I-d'_{21}-d_{21}),\cdots,x(I-d'_{2\xi_2}-d_{21})) \\ \vdots \\ y_2(I-d_{2\tau_2}) = f_2(x(I-d'_{21}-d_{2\tau_2}),\cdots,x(I-d'_{2\xi_2}-d_{2\tau_2})) \end{cases} \\
\vdots \\
\vdots \\
\begin{cases} y_\alpha(I-d_{\alpha 1}) = f_\alpha(x(I-d'_{\alpha 1}-d_{\alpha 1}),\cdots,x(I-d'_{\alpha\xi_\alpha}-d_{\alpha 1})) \\ \vdots \\ y_\alpha(I-d_{\alpha\tau_\alpha}) = f_\alpha(x(I-d'_{\alpha 1}-d_{\alpha\tau_\alpha}),\cdots,x(I-d'_{\alpha\xi_\alpha}-d_{\alpha\tau_\alpha})) \end{cases} \\
\vdots \\
\vdots \\
\begin{cases} y_\Omega(I-d_{\Omega 1}) = f_\Omega(x(I-d'_{\Omega 1}-d_{\Omega 1})),\cdots,x(I-d'_{\Omega\xi_\Omega}-d_{\Omega 1})) \\ \vdots \\ y_\Omega(I-d_{\Omega\tau_\Omega}) = f_\Omega(x(I-d'_{\Omega\tau_\Omega}-d_{\Omega\tau_\Omega})),\cdots,x(I-d'_{\Omega\xi_\Omega}-d_{\Omega\tau_\Omega})) \end{cases} \\
x(I) = g(y_1(I-d_{11}),\cdots,y_1(I-d_{1\tau_1}),\cdots,y_\Omega(I-d_{\Omega 1}),\cdots,y_\Omega(I-d_{\Omega\tau_\Omega}),\cdots\cdots)
\end{cases}
$$

Observação

Nos exemplos 18 e 20, os vértices centrais (S_2 e S_4, respectivamente) estão no fim do corpo dos laços. Este fato não é relevante porque os vetores de dependência são supostos lexicograficamente positivos e a ordem entre os comandos não altera a semântica do algoritmo.

4.6.3 Aplicabilidade

Assim como foi feito na seção 4.4.4, comparemos o "*piso*" de outros métodos com o "*teto*" de **RD** generalizada. Vejamos quando podemos calcular um "teto".

Definição 16

Seja $F \subset E$
α *é uma* direção viável *de* F *se* $d^\alpha > 0, \forall d \in F$
$DV(F) = \{\alpha \mid d^\alpha > 0, \forall d \in F\}$

Definição 17

Seja $S \in VE$
$dv(S) = DV(E^+(S)) \cup DV(E^-(S))$

Definição 18 $EC = \{S_c \xrightarrow{d} S'_c \mid S_c, S'_c \in VC\}$

Temos dois casos a considerar:

- **caso 1** (existe um único vértice central)
  Se $\bigcap_{S \in VE} dv(S) \neq \emptyset$,
  então podemos calcular um "*teto*" para **RD** generalizada.

- **caso 2** (existe mais de um vértice central)
  Se $(\bigcap_{S \in VE} dv(S)) \cap DV(EC) \neq \emptyset$,
  então podemos calcular um "*teto*" para **RD** generalizada.

Capítulo 5

Conclusão

- Na seção 5.1 resumimos esta tese.
- Na seção 5.2 apontamos as contribuições.
- Na seção 5.3 apontamos as possíveis futuras direções deste trabalho.

5.1 Conclusão

Apresentamos uma nova técnica de encolhimento de ciclo chamada *redução de dependência*, que consiste em identificar e distinguir, no grafo de dependência, os vértices correspondentes aos comandos cruciais e os vértices correspondentes aos comandos não cruciais a escalonamento. A nova técnica baseada nessas informações corresponde a definição eficiente de macros costituídos de um conjunto de comandos possibilitando a redução do número de passos e do número de comunicações entre processadores. A nova técnica também simplifica sensivelmente a aplicação do método GSS, ou outras técnicas de escalonamento devido à redução do número de vetores de dependência. Uma comparação com outros métodos foi feita usando um mesmo exemplo, o bem conhecido exemplo de Peir e Cytron [PeiC89], para mostrar a sua eficiência e a simplicidade. Apresentamos as formalizações deste novo método, em termos de transformação do grafo de dependência e de definição de macro comando. A condição suficiente e simples de aplicabilidade foi dada. Em seguida introduzimos a técnica chamada *redução de dependência parcial* que visa balancear o tempo gasto para comunicação e o tempo gasto para computação.
Finalmente introduzimos a técnica chamada *redução de dependência generalizada* que ataca os grafos de dependência mais gerais e altera o algoritmo sem mudar as dependências implícitas. Esta alteração de algoritmo elimina certos "gargalos" inerentes de comunicação do algoritmo original. A condição suficiente e simples de aplicabilidade foi dada.
Os métodos foram ilustrados por meio de *domínio explícito*, uma nova representação de dependência que auxilia identificar as dependências de modo mais eficiente e dá uma base intuitiva para os novos métodos.

5.2 Contribuições

Podemos resumir as contribuições da tese como se segue.

1. **domínio explícito (DE)**
   - DE é uma nova representação de dependências.
   - os exemplos 2-dimensionais , apesar de serem simples, são ilustrativos pois dão uma base intuitiva da transformação nos grafos de dependências para casos gerais.
2. **redução de dependência (RD)**
   - RD é uma nova técnica para encolhimento de ciclos.
   - Ela identifica e distingue os vértices correspondentes aos comandos cruciais e os vértices correspondentes aos comandos não cruciais a escalonamento.
   - A redução de número de vetor de dependência contribui para análise mais simplificada de escalonamento.
   - A redução de número de vetor de dependência contribui para redução de comunicações entre processadores na rede de processadores resultante.
   - Ela tem desempenho melhor em relação aos outros método para o bem conhecido exemplo de Peir e Cytron [PeiC89].
   - Ela tem uma condição simples e suficiente para sua aplicabilidade.
3. **RD** parcial
   - para grafo de dependência com muitos vértices secundários, ela balanceia o tempo gasto para comunicação e o tempo gasto para computação.
4. **RD** generalizada
   - Ela é aplicável aos grafo de dependência mais geral.
   - Em vez de tomar o algoritmo como ponto de partida fixa para paralelização, ela ataca o próprio grafo de dependência mantendo as dependências implícitas do problema. A transformação correspondente no algoritmo elimina o "gargalo" inerente de comunicação do algoritmo original.
   - Ela tem uma condição simples e suficiente para sua aplicabilidade.

5.3 Futuras direções

Como possíveis direções para a generalização do método **RD** podemos apontar o seguinte.

1. **laços não perfeitos**
   DE deve ser de utilidade para analisar este tipo de estrutura.

2. **grafo fortemente conexo mais geral**
   Tentar aplicar **RD** generalizada para grafos mais gerais possíveis do que foi tratado no Capítulo 5.

3. **grafo conexo mas não fortemente conexo**
   Como foi visto na seção 4.3, este é o tipo de grafo que não foi atacado. O desafio é como casar escalonamento de várias partes cíclicas e partes não cíclicas.

4. **laços afins**
   Quando as dependências não são uniformes, a situação se torna bastante complexa [Wong92]. O mais recente trabalho nesta área foi de Darte e Robert [DarR95]. Neste trabalho eles desenvolveram a aplicação de escalonamento afim por comando a este tipo de estrutura.

Apêndice A

Detalhes do GSS

A.1 Cálculo de π_0 para o Exemplo 1 (página 21)

Seja $\pi = (a, b)$, então temos $\text{mdc}\{a, b\} = 1$ e $\pi D > 0 \Leftrightarrow \begin{cases} b \geq 1 \\ a \geq 6b + 1 \end{cases}$

Temos a e b necessariamente positivos.

$\text{disp}(\pi) = \min\{2b, a - 6b, a + 5b, b, a - 4b\} = \min\{a - 6b, b\}$

1. se $b = 1$
   então $\text{GSS}(\pi) = (a + 1)N$. Logo tome $a = 6b + 1 = 7$.
2. se $b \geq 2$
   então temos dois casos a considerar:
   - se $6b + 1 \leq a \leq 7b - 1$
     $\text{disp}(\pi) = a - 6b$ e $\text{GSS}(\pi) = \frac{(a+b)N}{a-6b}$
     tome $a = 7b - 1$ e temos $\text{GSS}(\pi) = \frac{(8b-1)N}{b-1} \geq 8N$
   - se $7b + 1 \leq a$
     $\text{disp}(\pi) = b$ e $\text{GSS}(\pi) = \frac{(a+b)N}{b}$
     tome $a = 7b + 1$ e temos $\text{GSS}(\pi) = \frac{(8b+1)N}{b} \geq 8N$
     (observe que $a = 7b$ é excluida por $\text{mdc}\{7b, b\} = b \geq 2$)

Portanto temos $\pi_0 = (7, 1)$ com $\text{GSS}(\pi_0) = 8N$.

A.2 Cálculo de π_0 para o novo grafo do Exemplo 1 (página 44)

Seja $\pi = (a, b)$, então temos $2a + b > 0$ e $a - 3b > 0$ e $a > 0$.
Seja k tal que $b = ka$.

$\mathrm{GSS}(\pi) = \frac{(a+|b|)N}{\min\{2a+b, a-3b\}} = \frac{(a+a|k|)N}{\min\{2a+ka, a-3ka\}} = \frac{(1+|k|)N}{\min\{2+k, 1-3k\}}$
Por outro lado, como $2a + ka > 0$ e $a - 3ka > 0$, teremos $-2 < k < \frac{1}{3}$.

- se $0 < k < \frac{1}{3}$
  $\mathrm{GSS}(\pi) = \frac{(1+k)N}{(1-3k)} = \frac{1+4k}{1-3k}N > N$

- se $k = 0$
  $\mathrm{GSS}(\pi) = N$
  $\pi = (1, 0)$ e o tempo total$=N$

- se $-2 < k < 0$
  seja $t = -k$
  $\mathrm{GSS}(\pi) = \frac{(1+t)N}{\min\{2-t, 1+3t\}}$
  É fácil ver que $\mathrm{GSS}(\pi)$ é minimizado quando $2 - t = 1 + 3t$
  logo $t = \frac{1}{4}$
  $\pi_0 = (4, -1)$ e $\mathrm{GSS}(\pi_0) = \frac{5N}{7}$

Apêndice B

Lista de símbolos

símbolo	significado	página com a primeira ocorrência
A	algoritmo RUN	50
$Adj^+(S)$	$\{S' \mid \exists\ S' \xrightarrow{d} S \in E\}$	68
$Adj^-(S)$	$\{S' \mid \exists\ S \xrightarrow{d} S' \in E\}$	68
B_α	sub-bloco α de comandos no corpo de laços	52
C, C_α	ciclo, ciclo α	9
$Cicl$	conjunto de ciclos	27
$ciclo(\pi)$	$\min\{\lfloor \frac{T_\pi(C)}{K(C)} \rfloor \mid C \in Cicl\}$	29
$Comm$	tempo gasto para comunicação entre processadores	12
$Comp$	tempo gasto para cálculo de um comando	12
d, d_α	vetores de dependência	8
$\overline{d}$	d para o $\overline{G}$	55
D	matriz de dependência	9
$\overline{D}$	D para o $\overline{G}$	55
DE	domínio explícito	36
$\mathrm{disp}(\pi)$	$\min\{\pi \cdot d_\alpha \mid 1 \leq \alpha \leq r\}$	20
Dom	conjunto de índices	7
$dv(S)$	$DV(E^+(S)) \cup DV(E^-(S))$	71
$DV(F)$	$\{\alpha \mid d^\alpha > 0, \forall d \in F\}$	71
E	conjunto de arestas de G	50
$\overline{E}$	E após RD	51
$E^+(S)$	$\{d \mid \exists S' \xrightarrow{d} S \in E\}$	??
$E^-(S)$	$\{d \mid \exists S \xrightarrow{d} S' \in E\}$	??
EC	$\{S_c \xrightarrow{d} S'_c \mid S_c, S'_c \in VC\}$	71
$EVE(S)$	$\{S_e \xrightarrow{d} S_c \in E \mid S_e \in VE(S_c)\} \cup \{S_c \xrightarrow{d} S_e \in E \mid S_e \in VE(S_c)\}$	69

G	grafo de dependência	50
$\overline{G}$	G após RD	51
$g^+(S)$	$\lvert\{d \mid \exists(S' \xrightarrow{d} S) \in E\}\rvert$	50
$g^-(S)$	$\lvert\{d \mid \exists(S \xrightarrow{d} S') \in E\}\rvert$	50
GLN	laços encaixados generalizados *general loop nest*	7
GSS	encolhimento de ciclo seletivo generalizado *general selective cycle shrinking*	15
GSS(π)	$\frac{\max\{\pi\cdot I-\pi\cdot J \mid I,J\in Dom\}}{\text{disp}(\pi)}$	20
$i, j, i_1, \ldots, i_n$	índices	7
I, J	índices (vetores)	7
ISM	método de deslocamento de índices	25
$K(C)$	comprimento do ciclo C	27
L	número máximo de vértices secundários incorporados num vértice principal	51
l_1, u_1	limites inferior e superior para o laço mais externo	7
L_α	número de vértices secundários num caminho simples	59
$l_\alpha(i_1, \ldots, i_{\alpha-1}), u_\alpha(i_1, \ldots, i_{\alpha-1})$	limites inferior e superior para o laço α	7
m	dimensão do espaço mapeado	12
M	$\min\{L_\alpha \mid 1 \leq \alpha \leq q\}$	60
n	número de laços	7
N, N_α	limite superior de laço de RUN	9
$NEW(\pi)$	$\frac{\max\{\pi\cdot I-\pi\cdot J \mid I,J\in Dom\}}{ciclo(\pi)}$	29
p	o número de comandos no corpo de laços	7
q	número de caminho simples	59
r	número de vetor de dependência	15
RD	redução de dependência	39
rota simples	um caminho ou um circuito na qual os vértices do começo e do fim são vértices principais e todos os vértices intermediários, quando existem, são vértices secundários	51
RUN	laços encaixados uniformes regulares *regular uniform nest*	9
S_α	comando α	7
$S_\alpha(I)$	S_α com instância I	7
$S_\alpha \delta S_\beta$	S_β depende de S_α	7
$S_\alpha(i_1, \ldots, i_n)$	comando S_α com instância $(i_1, \ldots, i_n)$	8
$S_\alpha \rightarrow S_\beta$	S_β depende de S_α	8
$S \xrightarrow{d} S'$	aresta do vértice S ao vértice S' via vetor de dependência d	50

T_c	comprimento do caminho mais longo do Dom	56
$T(C)$	peso total do ciclo C	27
$Tpep$	número total de passos para execução paralela	12
$T_\pi(C)$	$\sum_{d \in C} \pi \cdot d$	29
V	conjunto de vértices de G	50
$\overline{V}$	V após RD	51
VC	$\{S \in V \mid S$ é vértice central$\}$	68
VC^+	$\{S_c \in VC \mid S_c$ é central a algum $S_e \in VE\}$	69
VE	$\{S \in V \mid S$ é vértice extremal$\}$	68
$VE(S_c)$	$\{S_e \in VE \mid S_c$ é central a $S_e\}$	69
VS	vértices secundários	51
VP	vértices principais	51
w	número de caminhos simples entrando no S	52
Z^n	espaço de produto cartesiano de números inteiros	7
δ^f	dependência de fluxo	8
δ^a	anti-dependência	8
δ^o	dependência de saída	8
π	vetor de escalonamento	20
π_v	veotr de escalonamento para encolhimento por dependência verdadeira	31
π^α	componente α do veotr π	20
$\overline{\pi_0}$	π_0 para o $\overline{G}$	55
γ	$Comm/Comp$	58
Φ_E	função escalonamento	11
Ψ_M	função mapeamento	12
Δ	$\min\{\pi_v \cdot d_\alpha \mid 1 \leq \alpha \leq r\}$	31
Δ^β	$\min\{d_\alpha^\beta \mid 1 \leq \alpha \leq r\}$	15
$\cdot$	produto escalar de dois vetores ou produto de um vetor com uma matriz	20

Bibliografia

[AmoBF88] Amorim, C. L., Barbosa, V. C. and Fernandes, E. S. T. *Uma introdução à computação paralela e distribuida.* VI escola de computação, UNICAMP, 1988

[Baner88] Banerjee, U. An introduction to a formal theory of dependence analysis. *J. Supercomput.* 2(1988), 133-149.

[Bert89] Bertsekas, D. P. and Tsitsiklis J. N. *Parallel and distibuted computation.* Prentice Hill, New Jersey, 1989.

[CosR86] Cosnard, M. and Robert, Y. *Parallel Algorithms and Architectures.* Elsevier Science Publishing Comp., 1986.

[Darte91] Darte, A. *Regular partitioning for synthesizing fixed-size systolic arrays.* Technical Report, Laboratoire de l'Informatique du Parallélisme-IMAG, Lyon, 1991.

[DarD90] Darte, A. and Delome, J. M. *Partitioning for array processor.* Technical Report, Laboratoire de l'Informatique du Parallélisme-IMAG, Lyon, 1991.

[DKR92] Darte, A., Khachiyan, L. and Robert, Y. Linear scheduling is close to optimality. *Application Specific Array Processors ASAP92*, J. A. B. Fortes et al. (Editors), IEEE Computer Society Press, Los Alamitos, CA. 1992, 37-46.

[DarR92] Darte, A. and Robert, Y. Scheduling uniform loop nests. Technical Report, Laboratoire de l'Informatique du Parallélisme-IMAG, Lyon, 1992.

[DarR94] Darte, A. and Robert, Y. Mapping uniform loop nests onto distributed memory architectures. *Parallel Computing* 20(1994), 679-710.

[DarR94-2] Darte, A. and Robert, Y. Constructive methods for scheduling uniform loop nests. *IEEE Trans. Parallel Distrib. System* 5(8), (Aug. 1994), 814-822.

[DarR95] Darte, A. and Robert, Y. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *Journal of Parallel and Distributed Computing* 29(1995), 43-59.

[DarRR91] Darte, A., Risset, T. and Robert, Y. *Synthesizing systolic arrays: some recent developments.* Technical Report, Laboratoire de l'Informatique du Parallélisme-IMAG, Lyon, 1991.

[Dowl90] Dowling, M. L. Optimal code parallelization using unimodular transformations. *Parallel Computing* 16(1990), 157-171.

[KayG92] Kayriakis-Bitzaros, E. D. and Goutis, C. E. An efficient decomposition technique for mapping nested loops with constant dependencies into regular processor array. *Journal of Parallel and Distributed Computing* 16(1992), 258-264.

[KonK95] Konda, V. and Kumar A. A systematic framework for the dependence cycle removal in practical loops. *Journal of Parallel and Distributed Computing* 27(1995), 157-171.

[Kung88] Kung, S. Y. *VLSI array processors*. Prentice Hill, New Jersey, 1988.

[Lampo74] Lamport, L. The parallel execution of do loops. *Communications of The ACM* 17(2):1974, 82-93.

[LiuHS90] Liu, L. S., Ho, C. W. and Sheu, J. P. On the parallelism of nested for-loops using index shift method. *Proc. Internat. Conf. on Parallel Processing* (Aug. 1990), II-119-II-123.

[MolF86] Moldovan, D. and Fortes, J. A. B. Partitioning and mapping algorithms into fixed size systolic array. *IEEE Transactions on Computers*, vol. c-35, No. 1 January, 1986.

[Moo86] Moore, V. *Systolic Arrays*. Adam Hilger, 1986.

[Newm72] Newman, M. *Integral matrices*. Academic Press, New York, 1972.

[Okuda89] Okuda, K. *Desenvolvimento Formal de Algoritmos Paralelos Sistólicos*. Dissertação de mestrado, IME/USP, 1989.

[Okuda95] Okuda, K. Encolhimento de Ciclo por Redução de Dependência. *Anais de VII Simpósio Brasileiro de Arquitetura de Computadores-Processamento de Alto Desempenho, SBC*, Canela, 1995, 553-567.

[Okuda95-2] Okuda, K. *Paralelização de Laços Uniformes por Redução de Dependência*. Relatório Técnico RT-MAC-9507, IME/USP, São Paulo, 1995.

[Okuda96] Okuda, K. Cycle shrinking by dependence reduction. *Euro-Par'96 Parallel Processing, Lecture Notes in Computer Science Series 1123*, Springer-Verlag, 1996, 398-401.

[Okuda96-2] Okuda, K. *Redução de dependência parcial e redução de dependência generalizada*. Relatório técnico RT-MAC-9602, IME/USP, São Paulo, 1996.

[Okuda96-3] Okuda, K. Cycle shrinking by generalized dependence reduction. (Em preparo.)

[PeiC89] Peir, J. K. and Cytron, R., Minimum distance: a method for partitioning recurrence for multiprocessors. *IEEE Trans. Comput.* 38(8), (Aug. 1989), 1203-1211.

[Poly88] Polychronopoulos, C. D. Compiler optimization for enhancing parallelism and their impact on architecture design. *IEEE Trans. Comput.* 37(8), (Aug. 1988), 991-1004.

[Poly88-2] Polychronopoulos, C. D. *Parallel Programming and Compilers.* Kluwer Academic Publishers, 1988.

[QuiR89] Quinton, P. and Robert, Y. *Systolic Algorithms and Architectures.* Prentice Hall, 1989.

[Robe90] Robert, Y. *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm.* Halsted Press, 1990.

[RobS92] Robert, Y. and Song, S.W. Revisiting cycle shrinking. *Parallel Computing*, 18(1992), 481-496.

[ShaF91] Shang, W. and Fortes, J. A. B. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Trans. Comput.* 40(6), (Jun. 1991), 723-742.

[ShaOF91] Shang, W., O'Keefe, M. T. and Fortes, J. A. B. Generalized cycle shrinking. *Parallel Algorithms and VLSI Architecture II.* P. Quinton and Y. Robert (editors), North Holland, 1991.

[Song84] Song, S. W. *Algoritmos Paralelos e Arquitetura VLSI.* IV escola de Computação, IME/USP, 1984.

[Wolfe89] Wolfe, M. *Optimizing Supercompilers for Supercomputers.* MIT Press, Cambridge, MA, 1989.

[Wolfe90] Wolfe, M. Data dependence and program restructuring. *J. Supercomput.* 4(1990), 321-344.

[Wong92] Wong, Y. Transformation of Broadcasts into Propagations in Systolic Algorithms. *Journal of Parallel and Distributed Computing* 14(1992), 121-145.