Encolhimento de Ciclo por Redução de Dependência

Kunio Okuda*
Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

4 de abril de 1995

Sumário

Apresentamos uma nova técnica chamada redução de dependência para encolhimento de ciclo. Esta técnica consiste em uma transformação do grafo de dependência que permite reduzir o número de passos e o número de comunicações entre processadores. A comparação do novo método com os outros métodos é feita através de exemplos ilustrativos.

Abstract

This paper describes a new technique called dependence reduction for cycle shrinking. This technique consists of the tranformation of the dependence graph that allows us to reduce the number of steps and number of communications between processors. The comparison between the new method and other methods is done through illustrative examples.

1 Introdução

No campo de computação paralela, as estruturas de laços (loops) encaixados são as estruturas que oferecem ricos paralelismos implícitos. Várias técnicas de transformação de laços para extrair paralelismo foram propostas [5, 11]. *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 [4]. Estes métodos transformam laços sequenciais em laços paralelos. Uma generalização de encolhimento de ciclo seletivo é o *encolhimento de ciclo seletivo generalizado* (generalized selective cycle shrinking (GSS)) [10, 8]. O método de *deslocamento de índice* (index shift method (ISM)) foi introduzido por Liu, Ho e Sheu [2]. Ele pode ser visto como um refinamento de GSS. Robert e Song propuseram um método que combina GSS com ISM [8]. *Afim por comando* (affine by statement) é proposto por Robert e Darte [6, 7].

*O autor é mestre em Matemática Aplicada, professor assistente do IME-USP e membro do projeto de pesquisa em Computação Paralela do IME/USP, que conta com apoio da FAPESP (Fundação de Ampalo à Pesquisa do Estado de São Paulo)-Proc. No. 93/0603-1, 94/4544, 95/0767 e CNPq Protem-II, e Commission of the European Communities-proj. ITDC-207.

Neste artigo propomos uma nova técnica de encolhimento de ciclo que transforma o grafo de dependência, reduz o número de comunicações entre processadores e o número de passos, identifica as dependências essenciais e permite uma análise mais simplificada para o escalonamento.

Este artigo é organizado de seguinte modo. Na seção 2 definimos algumas terminologias e discutimos rapidamente GSS. Na seção 3 explicaremos a nova técnica através de dois exemplos e comparamos o seu resultado com o resultado de outros métodos usando um exemplo bem conhecido de Peir e Cytion [3]. A conclusão é dada na seção 4.

2 Formalização, terminologia e o método GSS

Vamos considerar um algoritmo expresso como laços encaixados com a seguinte estrutura geral:

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_k

onde l_1 e u_1 são constantes, $l_j(i_1, \ldots, i_{j-1})$ e $u_j(i_1, \ldots, i_{j-1})$ são os limites mínimo e máximo de laço e todas as variáveis usadas em comandos S_1 a S_k têm seus índices como funções afins de índices i_1 a i_n.

O conjunto de índices para este algoritimo é definido como:

$Dom = \{I = (i_1, \ldots, i_n) | l_j \leq i_j \leq u_j, 1 \leq j \leq n\}$.

A seguinte definição de dependência entre instâncias de comandos $S_1 \ldots S_k$ é tirada de Robert e Song [8] que segue a definição amplamente aceita de acordo com Banerjee e Polychronopoulos [5, 1, 12].

Escrevemos $S_u \delta S_v$ se existem índices $(i_1, \ldots, i_n)$ e $(j_1, \ldots, j_n)$ em Dom tais que

1. $(i_1, \ldots, i_n) \leq (j_1, \ldots, j_n)$ na ordem lexicográfica em Z^n.
2. Comando $S_u(i_1, \ldots, i_n)$ deve ser executado antes de $S_v(j_1, \ldots, j_n)$ para preservar a semântica no laço e $\delta = (j_1 - i_1, \ldots, j_n - i_n)$ será chamado de vetor de dependência entre os dois comandos.

Dados dois comandos S_u e S_v, podem existir vários pares de índices (I, J) em Dom tais que $S_u(I)$ depende de $S_v(J)$. Vamos restringir nossa atenção à importante subclasse de laços encaixados chamados *uniformes* na qual o vetor de dependência entre dois comandos independe dos índices da particular instância. A importância desta subclasse se deve principalmente aos seguintes dois motivos:

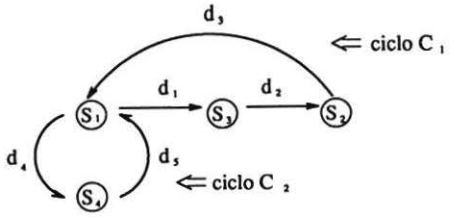

Figura 1: Grafo de dependência do Exemplo 0

1. Muitos algoritmos para aplicações científicas têm esta estrutura.
2. A sua estrutura regular permite explorar bem seu paralelismo implícito.

Exemplo 0

for $i = 0$ to N do
  for $j = 0$ to N do
    comando S_1: $a(i,j) = b(i,j-6) + d(i-1,j+3)$
    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: $d(i,j-1) = a(i,j-1)$

Para este exemplo o conjunto de índices é

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

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

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

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

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

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

Temos como a 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 1.

Para facilitar e simplificar as descrições subseqüentes vamos convencionar que l_j e u_j são constantes e l_j é sempre zero.

2.1 Método GSS

O método GSS é uma generalização de técnica *selective cycle shrinking* usada em compiladores paralizantes [5].

Método GSS

Considere laços encaixados uniformes com dimensão n e seja $D = (d_1, \ldots, d_m)$ uma matriz de dependência $n \times m$. Ache um vetor (vetor de escalonamento) $\pi = (\pi_1, \ldots, \pi_n)$ que minimiza

$GSS(\pi) = \frac{max\{\pi \cdot I - \pi \cdot J | I, J \in Dom\}}{disp(\pi)}$,

onde $disp(\pi) = min\{\pi \cdot d_j | 1 \leq j \leq m\}$
sob as seguintes condições:

1. $\pi \cdot D > 0$
2. $\text{mdc}(\pi_1, \ldots, \pi_n) = 1$ (mdc=máximo divisor comum)

Todos os pontos I e J em Dom que estejam no mesmo hiperplano perpendicular ao vetor π, i.e. $\pi \cdot I = \pi \cdot J$, serão executados simultaneamente no passo $\lfloor \frac{\pi \cdot I}{disp(\pi)} \rfloor (= \lfloor \frac{\pi \cdot J}{disp(\pi)} \rfloor)$. Tais hiperplanos serão denominados *hiperplanos de tempo.*

2.2 Domínio Explícito

A fim de explicitar as dependências no Dom de modo mais claro vamos adotar a seguinte definição:

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

A definição de vetor de dependência também vai mudar:

$S_i \rightarrow S_j : d = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}$ passa a ser $d = \begin{pmatrix} S_i - S_j \\ d_1 \\ \vdots \\ d_n \end{pmatrix}$

Assim para o exemplo 1, temos:

$DE = \{(S_h, i, j) | 1 \leq h \leq 4, 0 \leq i, j \leq N\}$

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

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

Para sua representação DE será sempre identificado como subconjunto de R^{n+1} e cada $S_h \times Dom$ será identificado como subconjunto de $\{h\} \times Dom$ e todos os pontos de DE serão ligados por vetores de dependência explicitamente.

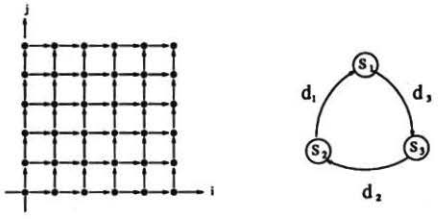

Figura 2: *Dom* e grafo de dependência do exemplo 1

A vantagem desta representação é a seguinte: Num laço uniforme 2-dimencional, podemos projetar cada $\{S_h\} \times Dom$ a R^2 com cada ponto ligeiramente deslocado em relação aos outros de modo que em R^2 todas as dependências fiquem explicitadas. Veremos tudo isso com detalhe na proxima seção.

Usaremos *Dom* e DE de acordo com a conveniência.

3 Exemplos

3.1 Primeiro caso

Primeiro vamos examinar o seguinte exemplo 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.

Exemplo 1

for $i = 0$ to N do
  for $j = 0$ to N do
    S_1: $a(i,j) = f(b(i-1,j))$
    S_2: $b(i,j) = g(c(i,j-1))$
    S_3: $c(i,j) = h(a(i-1,j))$

$Dom = \{(i,j) \in Z_2 / 0 \leq i,j \leq N\}$

$D = \{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}\}$

Dom com seus vetores de dependência e o grafo de dependência estão mostrados na Figura 2.

Aplicando GSS ao Exemplo 1, temos que achar $\pi = (a,b)$ que minimize $GSS(\pi) = \frac{max\{\pi \cdot I - \pi \cdot J / I,J \in Dom\}}{min\{\pi \cdot d_1, \pi \cdot d_2, \pi \cdot d_3\}}$ sob condição:

1. $\pi \cdot d_1 = \pi \cdot d_3 > 0$, $\pi \cdot d_2 > 0$
2. $\text{mdc}(a,b) = 1$

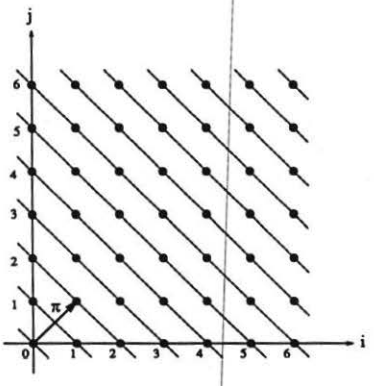

Figura 3: Os hiperplanos de tempo e π para Exemplo 1

Das condições impostas, é fácil ver que $a > 0$ e $b > 0$ e $GSS(\pi) = \frac{(a+b)N}{min\{a,b\}}$. A solução ótima será $a = b = 1$ e GSS(π)=$2N$ passos com $\pi = (1,1)$. Os hiperplanos de tempo e π estão mostrados na Figura 3.

Agora considere um outro exemplo.

Exemplo 1b

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

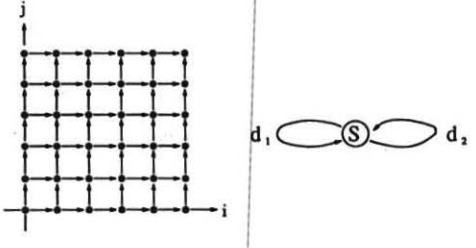

Figura 4: *Dom* e grafo de depenência do Exemplo 1b

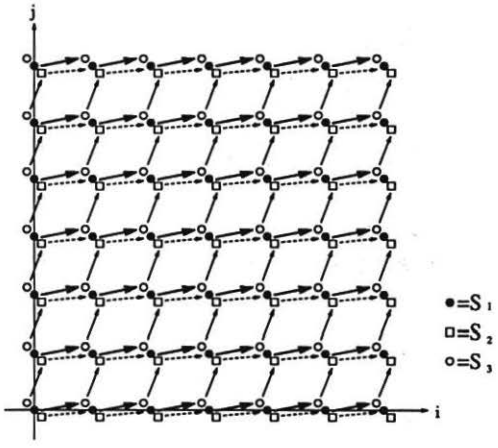

Figura 5: DE para Exemplo 1

Apesar de o Exemplo 1b ser bastante diferente do Exemplo 1, como mostra seu ciclo de dependência na Figura 4, seus *Dom* e D são idênticos aos do Exemplo 1 e consequentemente o método GSS dá o mesmo resultado para os dois exemplos. Ficamos satisfeitos com isto? Não, afirmamos que podemos melhorar bastante o escalonamento do Exemplo 1 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-dimencional de DE vamos projetar cada $\{(h,i,j)|(i,j) \in Dom\}$, que representa $\{S_h, i, j)|h \in \{1,2,3\}, (i,j) \in Dom\}$, em R^2 de modo ligeiramente deslocado um em relação a outro evitando que $a(i,j)$, $b(i,j)$ e $c(i,j)$ sejam sobrepostos. O resultado com seus vetores de dependência explícitos está na Figura 5.

Note que para o Exemplo 1b $Dom = DE$ 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). Entretanto a situação do Exemplo 1 é bem diferente (ver Figura 5): $S_1(4,4)$ depende de $S_2(3,4)$ mas não depende de nenhum $S_h(i,4)$ para $h \in \{1,2,3\}$ e $i < 3$, e também não depende de nenhum $S_h(4,j)$ para $h \in \{1,2,3\}$ e $j < 4$.

Observamos que o que temos na Figura 5 são vários "zig-zag's" de dependência que não se interferem um a 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 S que corresponde à sequência de comandos:
$S_3 : c(i-1, j-1) := h(a(i-2, j-1))$

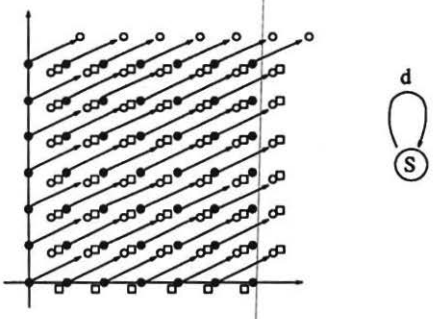

Figura 6: Grafo de dependência novo

$S_2 : b(i-1,j) := g(c(i-1,j-1))$
$S_1 : a(i,j) := f(b(i-1,j))$
Agora faremos esta transformação a todos os pontos de DE e temos o seguinte.

1. O *macro* comando S será mapeado a um processador.
2. O *macro* comando S levará maior tempo de execução por envolver a execução de três comandos de fato.
3. O vetor de dependência do *macro* comando 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 6). Em outras palavras os vetores de dependência de um ciclo são reduzidos em um só vetor, daí o nome de *redução de dependência.*

Pela Figura 6 é fácil observar que o número total de passos requeridos é $N/2$. Como cada ponto envolve a execução de três funções (f, g, h), o tempo total para execução será calculado do seguinte modo:

Sejam $Comm$ =tempo gasto para a comunicação entre processadores
$Comp$ =tempo gasto para o calculo de uma função

Sejam $Comm$ = tempo gasto para a comunicação entre processadores
$Comp$ = tempo gasto para o cálculo de uma função
e considere nulo o tempo gasto para comunicação interna num processdor. Então cada passo consiste no cálculo de três funções e uma comunicação (no metodo GSS, cada passo consiste sempre em uma comunicação e cálculo de uma função). Então o tempo total gasto é $\frac{N}{2}(Comm + 3Comp)$ (enquanto que por GSS o tempo é $2N(Comm + Comp)$).

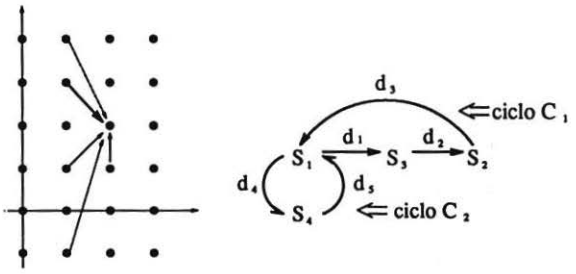

Figura 7: *Dom* e o grafo de dependência do Exemplo 2

3.2 Segundo caso

Considere agora o seguinte exemplo na qual temos dois ciclos no grafo de dependência. Neste exemplo o DE vai servir para mostrar quais são as dependências essenciais e quais são as dependências secundárias.

Exemplo 2

for $i = 0$ to N do
  for $j = 0$ to N do
    $S_1 : a(i,j) = f(b(i-1,j-3) + d(i-1,j+2))$
    $S_2 : b(i,j) = g(c(i-1,j+1))$
    $S_3 : c(i,j) = h(a(i-1,j-1))$
    $S_4 : d(i,j) = k(a(i,j-1))$

O *Dom* e o grafo de dependência estão na Figura 7.

Construímos $DE = \{S_1, \ldots, S_4\} \times Dom$ e projetamos os 4 planos a R^2 e obtemos Figura 8.

A Figura 8 é decomposta em Figura 9 e Figura 10 que mostram as dependências de ciclos C_1 e C_2. A Figura 11 mostra as dependências em relação a $S_1(i,j)$ em particular.

Nas Figuras 9 e 10 temos novamente "zig-zag's" independentes, cada um dos "zig-zag's" pode ser tratado como no Exemplo 1. Porém o Exemplo 2 é mais restritivo que o Exemplo 1, o escalonamento do ciclo 1 deve ser compatível com o escalonamento do ciclo 2. Pela Figura 11 junto com a observação acima podemos notar que as posições dos S_1's (pontos de interseção dos dois ciclos) são essenciais para escalonamento.

Deste modo queremos analisar as dependências unicamente em função de S_1. Como foi feito no Exemplo 1, 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 $S_1(i,j)$.

Assim consideremos S como um *macro* comando para um ponto pertencente a *Dom*. Obteremos Figura 12 como Figura 11 transformada após esta consideração. O grafo de dependência

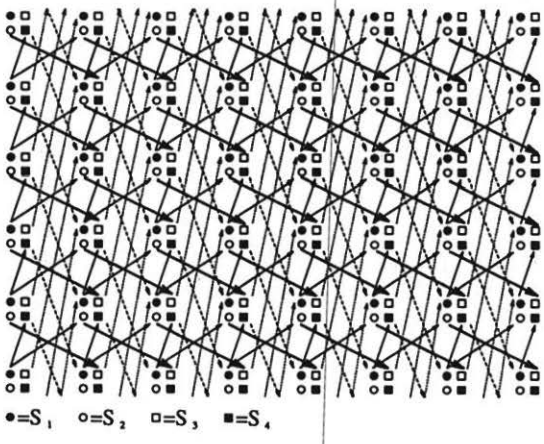

Figura 8: DE para Exemplo 2

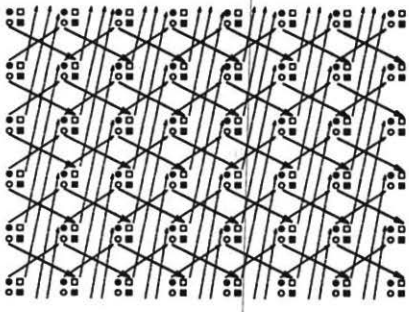

Figura 9: Dependências do ciclo C_1

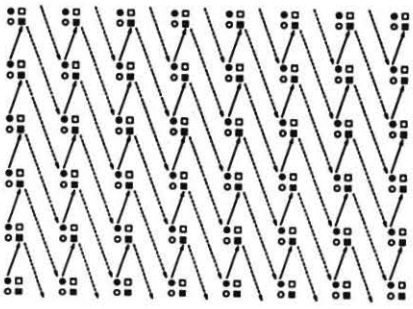

Figura 10: Dependências do ciclo C_2

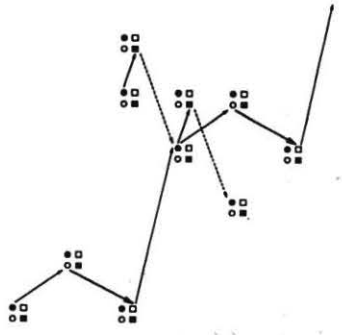

Figura 11: Dependências para $S_1(i,j)$

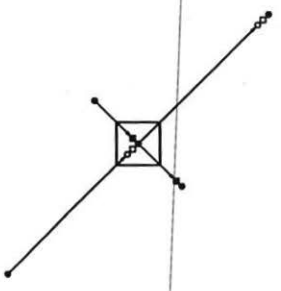

Figura 12: Nova dependência para $S_1(i,j)$

Figura 13: Novo grafo para Exemplo 2

com este macro é mostrado na Figura 13.

Observações

1. Tanto no Exemplo 1 como no Exemplo 2 os *macros* para pontos situados na borda do domínio são incompletos (veja, por exemplo, na Figura 6 para o Exemplo 1 que os *macros* na borda direita são compostos apenas por S_2 e S_3). Note também que o método ataca escalonamento e mapeamento simultaneamente.

2. O número de vetor de dependência é mais reduzido após a transformação do grafo por ***redução de dependência.*** Isto resulta em redução do número de comunicações entre processadores. Por exemplo, no Exemplo 1 em vez de ter três ligações para variáveis a, b, e c passamos a ter apenas uma ligação para variável a. As ligações para variaveis b e c ficam "***escondidas***" em processadores. Esta redução de número de vetor de dependência também simplifica bastante o cálculo do melhor escalonamento que é muito complexo em casos gerais [9].

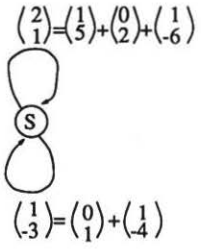

Figura 14: Novo grafo para Exemplo 0

3.3 Terceiro caso

Vejamos o bem conhecido exemplo considerado em [8] e outros que foi Exemplo 0. Note que o Exemplo 0 é praticamente igual ao Exemplo 2, exceto alguns índices. Eles apresentam os mesmos ciclos (lado direito da Figura 7, com diferentes dependências). Aplicando o método de *redução de dependência* de modo análogo ao Exemplo 2, temos o grafo da Figura 14.

Para este grafo $\pi = (4, -1)$ é a solução ótima para GSS com o número de passos igual a $\frac{5}{7}N$ (ver apêndice). Assim o tempo gasto total será $\frac{5}{7}(NComm + 3NComp)$.

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

A tabela seguinte resume os resultados de vários métodos para o Exemplo 0.

	Computação	Comunicação
GSS	$8NComp$	$8NComm$
GSS+ICM	$2NComp$	$2NComm$
Affine by Statement	$\frac{12}{7}NComp$	$\frac{12}{7}NComm$
redução de dependência	$\frac{15}{7}NComp$	$\frac{5}{7}NComm$

Comparação entre os métodos da tabela

- GSS apresenta o maior tempo.
- Se $Comm > \frac{1}{9}Comp$, então o método de *redução de dependência* é melhor que GSS+ICM.
- Se $Comm > \frac{3}{7}Comp$, então o método de *redução de dependência* é melhor que *Afim por comando.*
- Concluindo, se $Comm > \frac{3}{7}Comp$, então o método de *redução de dependência* apresenta o menor tempo.

4 Conclusão

Apresentamos uma nova técnica de encolhimento de ciclo, *redução de dependência*, que consiste em identificar e distinguir os comandos cruciais e os comandos não cruciais a escalonamento. A nova técnica baseada nessas informações corresponde a definição eficiente de macros nos processadores possibilitando a redução de número de passos e número de comunicação entre processadores. Uma comparação com outros métodos foi dada usando um mesmo exemplo para mostrar a sua eficiência e a simplicidade.

5 Apêndice

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

$GSS(\pi) = (a + |b|)N/min\{2a + b, a - 3b\} = (a + a|k|)N/min\{2a + ka, a - 3ka\} = (1 + |k|)N/min\{2 + k, 1 - 3k\}$

Por outro lado como $2a + ka > 0$ e $a - 3ka > 0$, teremos $-2 < k < 1/3$.

- se $0 < k < 1/3$
  $GSS(\pi) = (1 + k)N/(1 - 3k) = (1 + 4k/(1 - 3k))N > N$
- se $k = 0$
  $GSS(\pi) = N$
  $\pi = (1, 0)$ e o tempo total=N
- se $-2 < k < 0$
  seja $t = -k$
  $GSS(\pi) = (1 + t)N/min\{2 - t, 1 + 3t\}$
  é fácil ver que $GSS(\pi)$ é minimizado quando $2 - t = 1 + 3t$
  logo $t = 1/4$
  $\pi = (4, -1)$ e $GSS(\pi) = 5N/7$

6 Agradecimento

O autor agradece ao Prof. Dr. Siang Wun Song pelas sugestões ao Prof. Dr. Carlos Eduardo Ferreira pela ajuda e aos membros de GCPD/DCC/IME/USP pelo estímulo dado.

Referências

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

[2] Liu, L.S., Ho, C.W., 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.

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

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

[5] Polychronopoulos, C.D. *Parallel programing and compilers.* Kluwer Academic Publishers, 1988.

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

[7] Robert, Y., Darte, A. Mapping Uniform Loop Nests onto Distributed Memory Architectures. *Parallel Computing* 20(1994) 679-710.

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

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

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

[11] Wolfe, M., *Optimizing supercompilers for supercopmuters.* MIT Press, Cambridge, MA, 1989.

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