DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9507

Paralelisação de Laços Uniformes por Redução de Dependência

Kunio Okuda

Julho 95

Paralelisação de laços uniformes por redução de dependência

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

Sumário

Apresentamos uma nova técnica chamada Redução de Dependência para encolhimento de ciclos. Esta técnica consiste em uma transformação do grafo de dependência que permite reduzir o número de passos e o número de comunicações entre processadores. Em seguida aplicamos o bem conhecido método de GSS (*Generalized Selective Cycle Schrinking*) mas o ciclo tranformado permite uma análise mais simplificada e resultado ainda melhor. A comparação de novo método com outros métodos é feita através de exemplos ilustrativos.

Abstract

This paper describes new thecnique called Dependence Collapsing for cycle shrinking. This thecnique consists in transformation of dependence graph that allows us to reduce number of steps and number of comunication links between processors. Next we use well known GSS (Generalized Selective Cycle Schrinking) method but transformed cycle aloows more simple analysis and better result. The comparison between the new method with other methods is done through on illustrative examples.

*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, 94/4544-2, 95/0767 e do CNPq Protem-II, e Commission of the European Communities-proj. ITDC-207.

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 do tipo encolhimento de ciclo("cycle shrinking") para extrair paralelismo foram propostas [6, 12]. *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 [5]. Estes métodos transformam laços seqüenciais em laços paralelos. Uma generalização de *encolhimento de ciclo selctivo* é *encolhimento de ciclo seletivo generalizado* (generalized selective cycle shrinking (GSS)) [11, 9]. 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 [9]. *Afim por comando* (Affine by Statement) é proposto por Robert e Darte [7, 8].

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 a terminologia e discutimos rapidamente GSS e GSS combinada com ISM. Na seção 3 explicaremos a nova técnica através de dois exemplos e comparamos o seu resultado com o resultado de outros métodos usando o exemplo bem conhecido de Peir e Cytron [4]. Na seção 4 formalizaremos a nova técnica e finalmente a conclusão é dada na seção 5.

2 Formalização, terminologia e os métodos

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

GLN(*General Loop Nest*)

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 e $l_j(i_1, \ldots, i_{j-1})$ e $u_j(i_1, \ldots, i_{j-1})$ são os limites mínimo e máximo de laço e todos 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 [9] que segue a definição amplamente aceita de acordo com Banerjee e Polychronopoulos [1, 6, 13].

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 a importante subclasse de GLN chamado GLN *uniforme* na qual o vetor de dependência entre dois comandos independe dos índices da particular instância. 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 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 dependência:

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

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

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

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

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

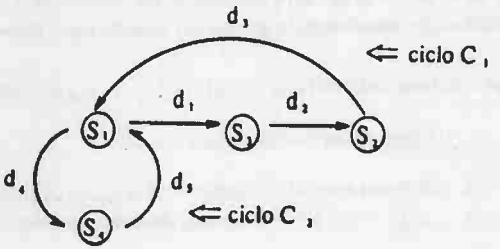

Figura 1: Grafo de dependência do Exemplo 0

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 1.

Para facilitar e simplificar as descrições subseqüentes vamos fazer mais uma pequena restrição à classe de algoritmos:

RUN(*Regular Uniform Nest*)

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_m

2.1 Escalonamento

Dado GLN uniforme, escalonamento é uma função $F : Z^n \rightarrow Z$ tal que a computação $(i_1, \ldots, i_n)$ é executado no passo $F(i_1, \ldots, i_n)$ [7]. Para que uma função $F : Z^n \rightarrow Z$ seja um escalonamento, ela deve satisfazer seguinte a condição:

Se $S_u(i_1, \ldots, i_n) \delta S_v(j_1, \ldots, j_n)$ então $F(i_1, \ldots, i_n) < F(j_1, \ldots, j_n)$.

A execução paralela de GLN uniforme é a seguinte:

for $t = timemin$ to $timemax$ do
  execute todos os $(i_1, \ldots, i_n) \in Dom$ tal que $F(i_1, \ldots, i_n) = t$

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.

2.2 Mapeamento

Dado GLN uniforme, mapeamento é uma função $G : Z^n \rightarrow Z^m$ tal que a computação $(i_1, \ldots, i_n)$ é executada no processador $G(i_1, \ldots, i_n)$ [8]. Se $F(i_1, \ldots, i_n) = F(j_1, \ldots, j_n)$ então $G(i_1, \ldots, i_n) \neq G(j_1, \ldots, j_n)$ para que as computações escalonadas no mesmo passo sejam mapeadas em processadores diferentes. O exemplo típico de mapeamento é uma projeção ao longo de um vetor e neste caso $m = n - 1$. Quando Dom é projetado a Z^{n-1}, $G(Dom)$ será uma rede de processadores e os vetores de dependência projetados representarão comunicações entre processadores na rede de processadores, exceto os vetores de dependência paralelos ao vetor de projeção. Estes últimos vetores não representam comunicação pois os dados estão no mesmo processador.

Sejam

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

Então o tempo total gasto será geralmente $T(Comp + Comm)$. Entretanto se todos os vetores de dependência forem paralelos ao vetor de projeção então o tempo total gasto será apenas $T\ Comp$.

2.3 Método GSS

O método GSS é uma generalização de técnica *selective cycle shrinking* usada em compilador paralelizante [11].

Método GSS

Considere GLN uniforme com dimensão n e seja $D = (d_1, \ldots, d_m)$ uma matriz de dependência $n \times m$. Seja $\pi = (\pi_1, \ldots, \pi_n)$ um vetor tal que

1. $\pi \cdot D > 0$
2. $mdc(\pi^1, \ldots, \pi^n) = 1$

π será denominado *vetor de escalonamento* e seja o fator de redução $disp(\pi) = min\{\pi \cdot d_j | 1 \leq j \leq m\}$.

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*.

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

2.4 Método ISM e GSS combinado com ISM

Método ISM faz deslocamento de índices em comandos sem violar a semântica do algoritmo. Este deslocamento torna o ciclo de dependência mais balanceado permitindo aumento do fator de redução ($disp(\pi)$).

Porém GSS seguido de ISM nem sempre proporciona bom resultado e Robert e Song propuseram novo método que combina GSS e ISM [9].

Método GSS+ISM

ache um vetor $\pi = (\pi^1, \ldots, \pi^n)$ que minimize

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

onde $cycle(\pi) = min\{\lfloor \frac{T_\pi(C)}{K(C)} \rfloor | C \in cycle\}$

$T_\pi(C) = \sum_{d \in C} \pi \cdot d$

$K(C)$=comprimento do ciclo C

$Cycle$=conjunto de ciclos

sob condições

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

2.5 Método"afim por comando"

Afim por comando é estudado por Darte e Robert e este método procura escalonamento para cada comando $S_1, \ldots, S_m$ em vez de considerar o corpo de laços como um bloco só [7, 8].

2.6 Formalização

A fim de explicitar as dependências em 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 $d = \begin{pmatrix} S_j - S_i \\ d^1 \\ \vdots \\ d^n \end{pmatrix}$

Assim para o exemplo 0, 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.

Esta representação é similar a *Augmented Dependence Graph* (ADG) proposto por Kyriakis-Bitzaros e Goutis [3], mas é mais simples. Para laços de dimensão n com m comandos, a dimensão de ADG é $n + m + 1$ enquanto a dimensão de DE é sempre $n + 1$, independente de m, o que torna mais fácil sua representação visual.

A vantagem desta representação é a seguinte: No caso de GLN uniforme 2-dimensional, 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 próxima 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 $\pi_0 = (a,b)$ 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 condições:

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

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_0)=$2N$ passos com $\pi_0 = (1,1)$. Os hiperplanos de tempo e π estão mostrados na Figura 3.

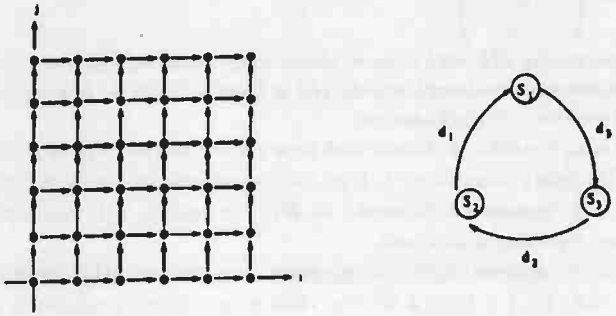

Figura 2: *Dom* e grafo de dependência do Exemplo 1

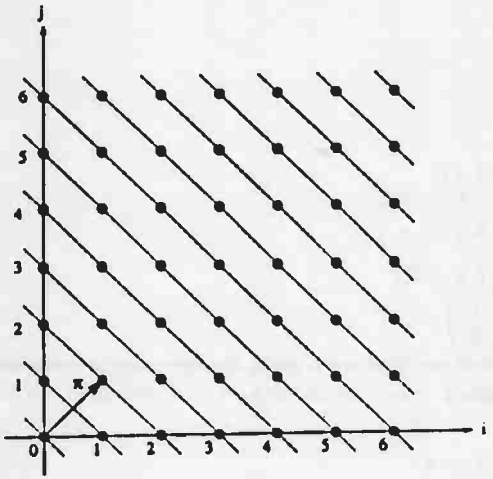

Figura 3: Hiperplanos de tempo e π_0 para o Exemplo 1

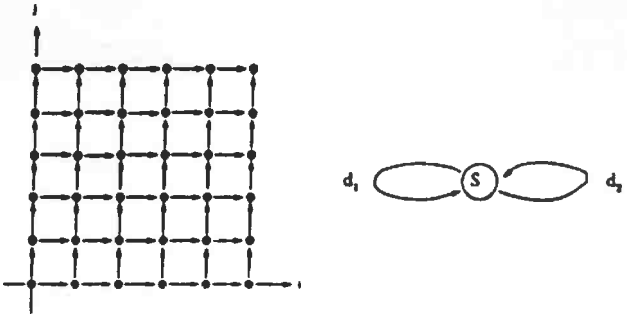

Figura 4: *Dom* e grafo de dependência do Exemplo 1b

É fácil observar também que $NEW(\pi)$ dá o mesmo resultado. Agora considere um outro exemplo.

Exemplo 1b

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

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 conseqüentemente 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-dimensional 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 $(4,4)$

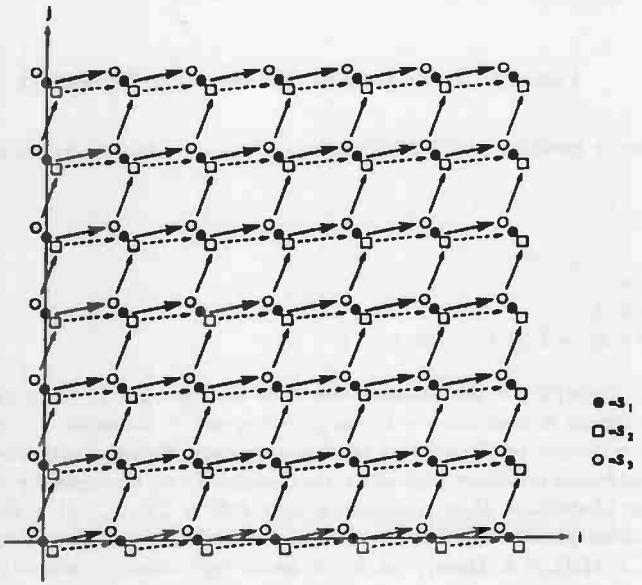

Figura 5: DE para Exemplo 1

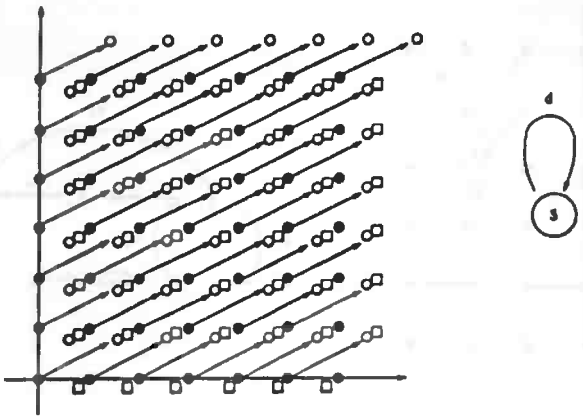

Figura 6: Grafo de dependência novo

teremos um *macro* comando S que corresponde à seqüência de comandos :
$S_3 : c(i-1, j-1) := h(a(i-2, j-1))$
$S_2 : b(i-1, j) := g(c(i-1, j-1))$
$S_1 : a(i, j) := f(b(i-1, j))$
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 comunicação entre processadores
Comp=tempo gasto para cálculo de uma função

como na seção 2 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

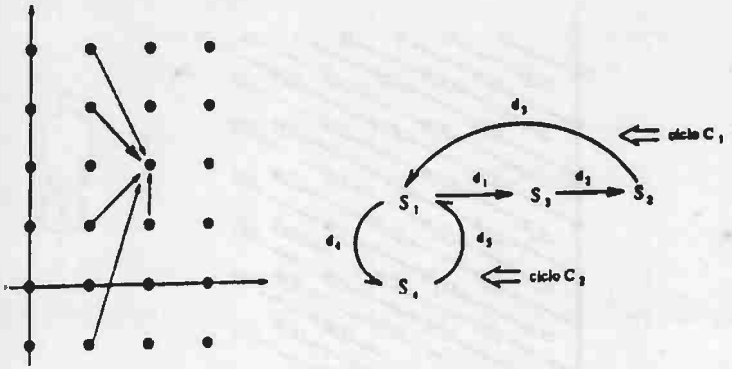

Figura 7: *Dom* e o grafo de dependência do Exemplo 2

gasto é $\frac{N}{2}(Comm+3\,Comp)$ (enquanto que por GSS o tempo é $2N(Comm+Comp)$). Mais ainda, podemos projetar ao longo do vetor d e o tempo será $\frac{3N}{2}\ Comp$. Note que para o método GSS não podemos eliminar comunicação entre processadores já que temos dois vetores de dependência não colineares.

3.2 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.

Exemplo 2

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

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 dos ciclos C_1 e C_2. A Figura 11 mostra as dependência em relação a $S_1(i,j)$ em particular.

Nas Figuras 9 e 10 temos novamente "zig-zag's" independentes, cada um dos quais pode ser tratado como no Exemplo 1. Porém o Exemplo 2 é mais restritivo que o Exemplo 1. O

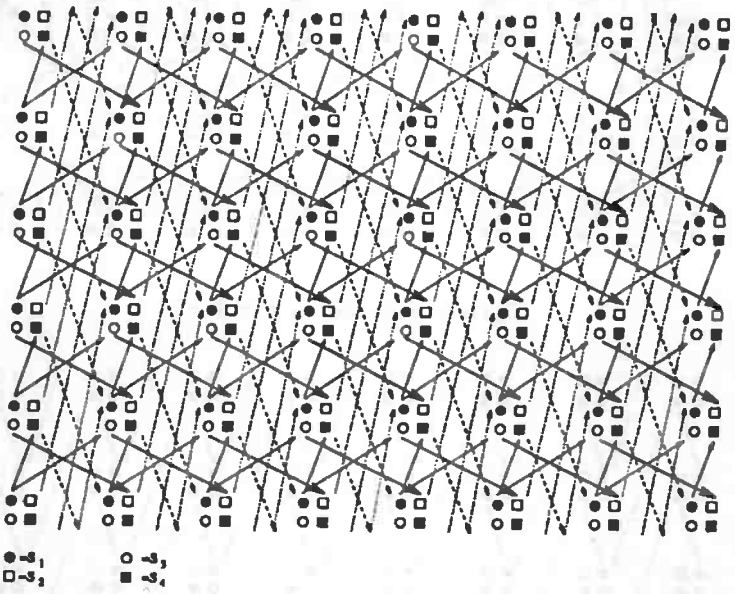

Figura 8: DE para Exemplo 2

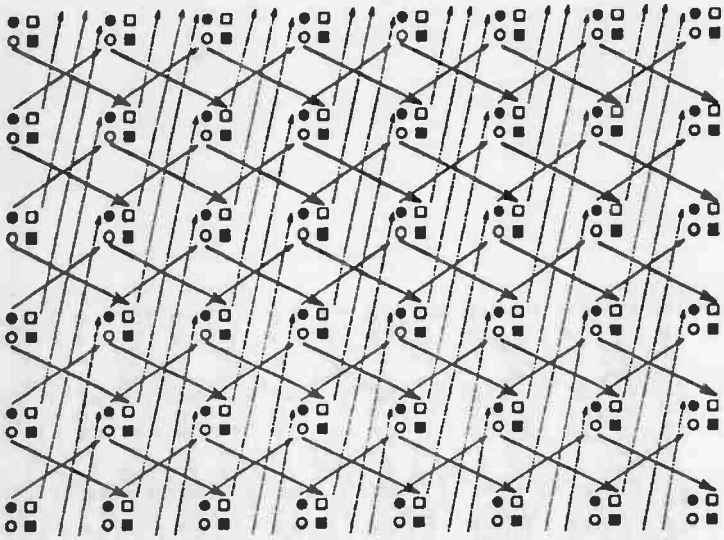

Figura 9: Dependência de ciclo C_1

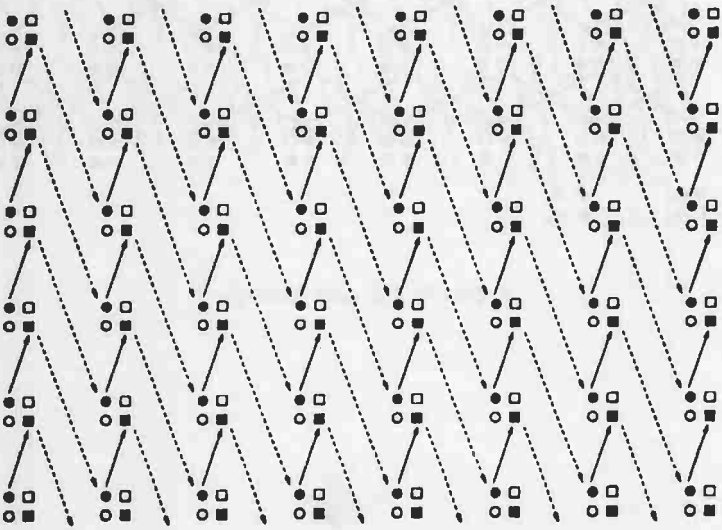

Figura 10: Dependência de ciclo C_2

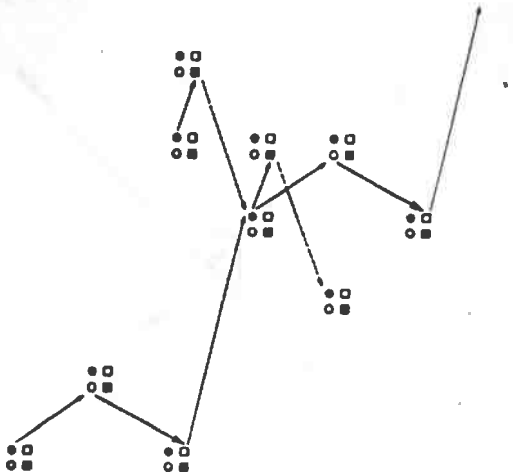

Figura 11: Dependência para $S_1(i,j)$

escalonamento do ciclo 1 deve ser compatível com o escalonamento do ciclo 2 por possuírem um ponto de intersecção. Pela Figura 11 junto com a observação acima podemos notar que as posições dos S_1's (pontos de intersecção 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 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).

2. Para rede de interconexão de processadores resultante a criação do macro comando significa tranferir algumas comunicações inter-processadores para dentro de um mesmo processador. Isto contribui para a redução do tempo total gasto. Na próxima seção trataremos este aspecto detalhadamente.

3. O número de vetores de dependência é mais reduzido após a transformação do grafo de dependência por *redução de dependência*. Isto resulta em redução do número de comunicações físicas entre processadores. Por exemplo, no Exemplo 1 em vez de ter 3 ligações para as

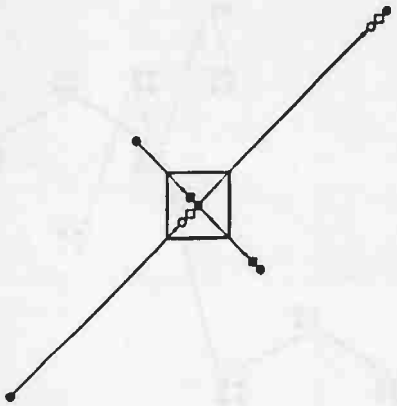

Figura 12: Nova dependência para $S_1(i,j)$

Figura 13: Novo grafo para o Exemplo 2

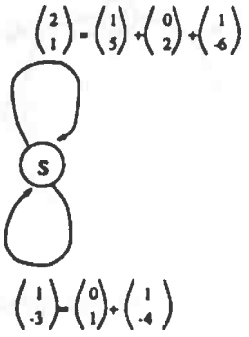

Figura 14: Novo grafo para o Exemplo 0

variáveis a, b, e c passamos a ter apenas 1 ligação para a variável a. As comunicações para variáveis b e c ficam *'escondidas'* em processadores.

3.3 Terceiro caso

Vejamos o bem conhecido exemplo considerado em [9] 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 *redução de dependência* de modo análogo ao Exemplo 2, temos o grafo da Figura 14.

Para este grafo $\pi_0 = (4, -1)$ 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 $\frac{5}{7}(NComm + 4NComp)$.

O tempo gasto por GSS é $8N(Comm + Comp)$ e o tempo gasto por GSS combinado com ISM é $2N(Comm + Comp)$ [9] . Por outro lado o tempo gasto por *affine by statement* é $\frac{12}{7}N(Comm + Comp)$ [8].

A tabela seguinte resume os resultados dos 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	$\frac{20}{7}NComp$	$\frac{5}{7}NComm$

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+ICM.
- Se $Comm > \frac{8}{7}Comp$, então *redução de dependência* é melhor que *Affine by Statement.*

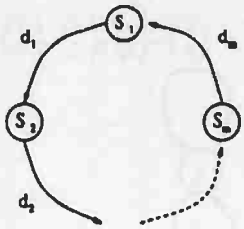

Figura 15: Grafo de dependência original

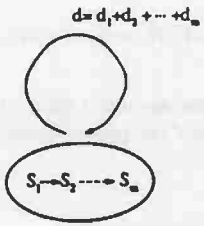

Figura 16: Grafo de dependência transformado

- Concluindo, se $Comm > \frac{8}{7}Comp$, então *redução de dependência* apresenta o menor tempo.

4 Redução de Dependência

Nesta seção vamos formalizar as transformações efetuadas nos exemplos da seção anterior. Na seção 4.1 temos o caso de algoritmos cujo grafo de dependência é apenas um ciclo, na seção 4.2 tratemos algoritmos mais complexos e na seção 4.3 compararemos vetores de escalonamento antes e após a transformação. Considere sempre um laço RUN com n laços encaixados e m comandos no corpo do laço (ver pagina 4).

4.1 Grafo de dependência igual a um ciclo

Este é um caso bastante simples. Sem perda de generalidade, vamos supor que todos os comandos do laço RUN fazem parte do ciclo conforme a Figura 15. Tal ciclo é transformado no ciclo apresentado na Figura 16.

O tempo gasto total será de

$$(\min_j \lfloor \frac{N_j}{d^j} \rfloor) Comm + m(\min_j \lfloor \frac{N_j}{d^j} \rfloor) Comp$$

onde d^j=j-ésimo elemento de d.

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 $S_1, \ldots S_m$ e ao de dependências entre comandos, respectivamente. Cada aresta de E é rotulada pelo seu vetor de dependência. Usaremos a notação $u \rightarrow v$ para denotar a aresta do vértice u ao vértice v. Vamos dividir V em dois conjuntos: o conjunto dos vértices secundários (VS) e o conjunto dos vértices principais (VP).

VS={ $v \in V$| v tem exatamente uma aresta que entra nele e uma aresta que sai dele }
$VP = V - VS$
Vejamos o Exemplo 2 (Figura 7). Para este exemplo, $VS = \{S_2, S_3, S_4\}$ e $VP = \{S_1\}$.
Seja $\overline{G} = (\overline{V}, \overline{E})$ um grafo transformado no qual $\overline{V} = VP$ e $\overline{E}$ é definido do seguinte modo:

$\overline{E}$={ $v \rightarrow v' | v, v' \in VP$ e em G existe um caminho entre v e v' cujos vértices intermediários são todos pertencentes a VS }

O rótulo para $v \rightarrow v'$ em $\overline{G}$ será a soma dos vetores de dependência que compõem o caminho $v \rightarrow \ldots \rightarrow v'$ em G. Os comandos correspondentes aos vértices secundários neste caminho serão incorporados como um macro comando de v'. Certamente se tiver outro caminho deste tipo para v' então os comandos correspondentes aos vértices secundários neste caminho serão incorporados também. Denominamos esta transformação de *redução de dependência*.

Apos obter $\overline{G}$ por *redução de dependência*, aplicamos o método GSS para obter $\overline{\pi_0}$ que minimiza $GSS(\pi)$ usando as dependências de $\overline{G}$.

Seja $T = GSS(\overline{\pi_0})$ para $\overline{G}$, então o tempo total para o algoritmo original será $T\ Comm + (l+1)T\ Comp$ onde l é o número máximo de vértices secundários que foram incorporados num vértice principal.

Observações

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

A idéia básica da transformação é a seguinte: o que nos importa realmente para achar 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 [10]. Essa redução pode até tornar possível o cálculo ser feito manualmente.

Note que tanto para algoritmos com um ciclo como para algoritmos mais complexos, o processo de transformação é facilmente efetuado na matriz de adjacência que representa o grafo de dependência.

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 pode até aumentar." Num trabalho futuro, mostraremos um novo

método a ser chamado *redução parcial de dependência* visando um equilíbrio entre comunicação e computação.

4.3 Efeito da transformação sobre a escolha de π

Seja G o grafo de dependência original e $\overline{G}$ o grafo transformado conforme 4.2

Seja π_0 vetor que minimiza $GSS(\pi) = \frac{max\{\pi \cdot I - \pi \cdot J | I,J \in Dom\}}{min\{\pi \cdot d_i | d_i \in D\}}$ 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 $mdc(\pi_0^1, \ldots, \pi_0^n) = 1$ e $\pi_0 \cdot d_i > 0$ para $\forall d_i \in D$. Por outro lado cada $\overline{d_j}$ é a soma de alguns d_i's de D. Assim $\pi_0 \cdot \overline{d_j} > 0$, para $\forall \overline{d_j} \in \overline{D}$ e portanto π_0 é um vetor de escalonamento para $\overline{G}$.

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

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

Mostraremos que $\overline{GSS}(\pi_0) \leq GSS(\pi_0)$. 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 $\pi_0 \cdot d_i$, para $d_i \in D$, logo $\pi_0 \cdot d_1 \leq \pi_0 \cdot d_i, \forall d_i \in D$ e $\pi_0 \cdot d_1 \leq \pi_0 \cdot \overline{d_j}$, para $\forall \overline{d_j} \in \overline{D}$. Deste modo temos $\overline{GSS}(\pi_0) \leq GSS(\pi_0)$ e com maior razão $\overline{GSS}(\overline{\pi_0}) \leq GSS(\pi_0)$.

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

5 Conclusão

Apresentamos uma nova técnica de encolhimento de ciclo chamada redução de dependência, que consiste em identificar e distinguir 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 nos processadores 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 devido à redução do número de vetores de dependência. Uma comparação com outros métodos foi feita usando um mesmo exemplo para mostrar a sua eficiência e a simplicidade.

Apêndice

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

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

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

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

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. 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.

[3] 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, 258-264(1992).

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

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

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

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

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

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

[10] 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.

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

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

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

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1992 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

J.Z. Gonçalves, Arnaldo Mandel
COMMUTATIVITY THEOREMS FOR DIVISION RINGS AND DOMAINS
RT-MAC-9201, Janeiro 1992, 12 pp.

J. Sakarovitch
THE "LAST" DECISION PROBLEM FOR RATIONAL TRACE LANGUAGES
RT-MAC 9202, Abril 1992, 20 pp.

Valdemar W. Setzer, Fábio Henrique Carvalheiro
ALGOTRITMOS E SUA ANÁLISE (UMA INTRODUÇÃO DIDÁTICA)
RT-MAC 9203, Agosto 1992, 19 pp.

Claudio Santos Pinhanez
UM SIMULADOR DE SUBSUMPTION ARCHITECTURES
RT-MAC-9204, Outubro 1992, 18 pp.

Julio M. Stern
REGIONALIZAÇÃO DA MATRIZ PARA O ESTADO DE SÃO PAULO
RT-MAC-9205, Julho 1992, 14 pp.

Imre Simon
THE PRODUCT OF RATIONAL LANGUAGES
RT-MAC-9301, Maio 1993, 18 pp.

Flávio Soares C. da Silva
AUTOMATED REASONING WITH UNCERTAINTIES
RT-MAC-9302, Maio 1993, 25 pp.

Flávio Soares C. da Silva
ON PROOF-AND MODEL-BASED TECHNIQUES FOR REASONING WITH UNCERTAINTY
RT-MAC-9303, Maio 1993, 11 pp.

Carlos Humes Jr., Leônidas de O.Brandão, Manuel Pera Garcia
A MIXED DYNAMICS APPROACH FOR LINEAR CORRIDOR POLICIES
(A REVISITATION OF DYNAMIC SETUP SCHEDULING AND FLOW CONTROL IN MANUFACTURING SYSTEMS)
RT-MAC-9304, Junho 1993, 25 pp.

Ana Flora P.C.Humes e Carlos Humes Jr.
STABILITY OF CLEARING OPEN LOOP POLICIES IN MANUFACTURING SYSTEMS (Revised Version)
RT-MAC-9305, Julho 1993, 31 pp.

Maria Angela M.C. Gurgel e Yoshiko Wakabayashi
THE COMPLETE PRE-ORDER POLYTOPE: FACETS AND SEPARATION PROBLEM
RT-MAC-9306, Julho 1993, 29 pp.

Tito Homem de Mello e Carlos Humes Jr.
SOME STABILITY CONDITIONS FOR FLEXIBLE MANUFACTURING SYSTEMS WITH NO SET-UP TIMES
RT-MAC-9307, Julho de 1993, 26 pp.

Carlos Humes Jr. e Tito Homem de Mello
A NECESSARY AND SUFFICIENT CONDITION FOR THE EXISTENCE OF ANALYTIC CENTERS IN PATH FOLLOWING METHODS FOR LINEAR PROGRAMMING
RT-MAC-9308, Agosto de 1993

Flavio S. Corrêa da Silva
AN ALGEBRAIC VIEW OF COMBINATION RULES
RT-MAC-9401, Janeiro de 1994, 10 pp.

Flavio S. Corrêa da Silva e Junior Barrera
AUTOMATING THE GENERATION OF PROCEDURES TO ANALYSE BINARY IMAGES
RT-MAC-9402, Janeiro de 1994, 13 pp.

Junior Barrera, Gerald Jean Francis Banon e Roberto de Alencar Lotufo
A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9403, Janeiro de 1994, 28 pp.

Flavio S. Corrêa da Silva
ON THE RELATIONS BETWEEN INCIDENCE CALCULUS AND FAGIN-HALPERN STRUCTURES
RT-MAC-9404, abril de 1994, 11 pp.

Junior Barrera; Flávio Soares Corrêa da Silva e Gerald Jean Francis Banon
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES
RT-MAC-9405, abril de 1994, 15 pp.

Valdemar W. Setzer; Cristina G. Fernandes; Wania Gomes Pedrosa e Flavio Hirata
UM GERADOR DE ANALISADORES SINTÁTICOS PARA GRAFOS SINTÁTICOS SIMPLES
RT-MAC-9406, abril de 1994, 16 pp.

Siang W. Song
TOWARDS A SIMPLE CONSTRUCTION METHOD FOR HAMILTONIAN DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9407, maio de 1994, 13 pp.

Julio M. Stern
MODELOS MATEMATICOS PARA FORMAÇÃO DE PORTFÓLIOS
RT-MAC-9408, maio de 1994, 50 pp.

Imre Simon
STRING MATCHING ALGORITHMS AND AUTOMATA
RT-MAC-9409, maio de 1994, 14 pp.

Valdemar W. Setzer e Andrea Zisman
*CONCURRENCY CONTROL FOR ACCESSING AND COMPACTING B-TREES**
RT-MAC-9410, junho de 1994, 21 pp.

Renata Wassermann e Flávio S. Corrêa da Silva
TOWARDS EFFICIENT MODELLING OF DISTRIBUTED KNOWLEDGE USING EQUATIONAL AND ORDER-SORTED LOGIC
RT-MAC-9411, junho de 1994, 15 pp.

Jair M. Abe, Flávio S. Corrêa da Silva e Marcio Rillo
PARACONSISTENT LOGICS IN ARTIFICIAL INTELLIGENCE AND ROBOTICS.
RT-MAC-9412, junho de 1994, 14 pp.

Flávio S. Corrêa da Silva, Daniela V. Carbogim
A SYSTEM FOR REASONING WITH FUZZY PREDICATES
RT-MAC-9413, junho de 1994, 22 pp.

Flávio S. Corrêa da Silva, Jair M. Abe, Marcio Rillo
MODELING PARACONSISTENT KNOWLEDGE IN DISTRIBUTED SYSTEMS
RT-MAC-9414, julho de 1994, 12 pp.

Nami Kobayashi
THE CLOSURE UNDER DIVISION AND A CHARACTERIZATION OF THE RECOGNIZABLE Z-SUBSETS
RT-MAC-9415, julho de 1994, 29pp.

Flávio K. Miyazawa e Yoshiko Wakabayashi
AN ALGORITHM FOR THE THREE-DIMENSIONAL PACKING PROBLEM WITH ASYMPTOTIC PERFORMANCE ANALYSIS
RT-MAC-9416, novembro de 1994, 30 pp.

Thomaz I. Seidman e Carlos Humes Jr.
SOME KANBAN-CONTROLLED MANUFACTURING SYSTEMS: A FIRST STABILITY ANALYSIS
RT-MAC-9501, janeiro de 1995, 19 pp.

C.Humes Jr. and A.F.P.C. Humes
STABILIZATION IN FMS BY QUASI- PERIODIC POLICIES
RT-MAC-9502, março de 1995, 31 pp.

Fabio Kon e Arnaldo Mandel
SODA: A LEASE-BASED CONSISTENT DISTRIBUTED FILE SYSTEM
RT-MAC-9503, março de 1995, 18 pp.

Junior Barrera, Nina Sumiko Tomita, Flávio Soares C. Silva, Routo Terada
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY PAC LEARNING
RT-MAC-9504, abril de 1995,16 pp.

Flávio S. Corrêa da Silva e Fabio Kon
CATEGORIAL GRAMMAR AND HARMONIC ANALYSIS
RT-MAC-9505, junho de 1995,17 pp.

Henrique Mongelli e Routo Terada
ALGORITMOS PARALELOS PARA SOLUÇÃO DE SISTEMAS LINEARES
RT-MAC-9506, junho de 1995, 158 pp.

Kunio Okuda
PARALELISAÇÃO DE LAÇOS UNIFORMES POR REDUÇÃO DE DEPENDÊNCIA
RT-MAC-9507, julho de 1995, 27 pp.