Análises Estáticas para a Distribuição de Dados e Computações em Memória Distribuída

Raul Junji Nakashima [1], Gonzalo Travieso [2]
[1] Instituto de Física de São Carlos, Universidade de São Paulo
Av. do Trabalhador São Carlense, 400, São Carlos - SP, Brasil
{junji@if.sc.usp.br}
[2] Instituto de Física de São Carlos, Universidade de São Paulo
Av. do Trabalhador São Carlense, 400, São Carlos - SP, Brasil
{gonzalo@if.sc.usp.br}

Resumo—
Este trabalho descreve técnicas de análise estática de compilação baseadas na álgebra e programação linear que buscam otimizar a distribuição de *loops forall* e *array* em programas escritos na linguagem *SISAL* visando à execução em máquinas paralelas de memória distribuídas.

Na fase de alinhamento, buscamos o alinhamento de hiperplanos com o objetivo de tentar encontrar as porções dos diferentes *arrays* que devem ser distribuídas juntas.

A fase de particionamento, que tenta quebrar em partes independentes dados e computações, duas funções afins, a função de decomposição de dados e a função de decomposição de computação são usadas para isso.

A última fase, o mapeamento, distribui os elementos de computação nos elementos de processamento usando um conjunto de inequações sobre os limites.

Essas técnicas estão sendo implementadas num compilador SISAL, mas podem ser usadas sem mudanças em outras linguagens de associação simples e com a adição de análise de dependências podem ser usadas em linguagens imperativas.

Palavras-chave— **Otimização, análises estáticas de compilação, alinhamento, particionamento, mapeamento, *forall***

Abstract—
This work describes static compiler analysis techniques based on linear algebra and linear programming to optimize the distribution of *forall* loops and array elements in programs written in the SISAL programming language for distributed memory parallel machines.

In the alignment phase, that tries to find the portions of different arrays that need to be distributed together, we work with alignment hyperplanes.

In the partitioning phase, that tries to break the computations and data in independent parts, we use two afine functions: the data decomposition function and the computation decomposition function.

The last phase, the mapping, distributes the elements of computation into the processing elements through a set of inequations.

The techniques are being implemented in a SISAL compiler, but can be used without changes to other single assignment languages, and with the addition of dependecy analysis to other languages as well.

Keywords— **optimization, static compiler analysis, alignment, partitioning, mapping, *forall***

I. Introdução

A otimização para a distribuição de dados nos sistemas de memória distribuída visa à redução da comunicação de dados que não estejam na memória local dos processadores, ou seja, a minimização da quantia de dados que devem ser transmitidas pelo sistema de interconexão entre processadores. Este problema é grave nos sistemas de memória distribuída, pois a passagem de mensagem é demasiado lenta, comparada ao acesso de um dado na memória local [AND 97, BOO 97, CUL 99]. Assim, nesses sistemas o sucesso de uma aplicação vai além da existência de paralelismo, pois é fortemente dependente da localidade de dados e computações do programa.

Para minimizar esse efeito, podemos aplicar análises visando melhorar a distribuição dos dados. Basicamente, a análise para a decomposição de dados e computações de um programa pode ser dividida em três partes principais: alinhamento, particionamento e mapeamento.

A etapa do alinhamento tenta relacionar as dimensões dos diferentes *arrays* de acordo com o melhor padrão de acesso entre eles, buscando a redução da sobrecarga total envolvida em possíveis movimentações de dados entre os processadores (acessos a dados não locais).

Na etapa seguinte, o particionamento, são tomadas as decisões de quais dimensões alinhadas serão distribuídas no espaço virtual de processadores, onde se busca a maximização do paralelismo potencial e a possibilidade adicional da redução do movimento de dados.

No mapeamento, são realizadas as associações entre processadores virtuais e processadores reais segundo relações entre os custos de comunicação e custos de computação. Essa fase é dependente do sistema utilizado na execução do código paralelo e do compilador utilizado.

II. REVISÃO.

Antes de iniciarmos a discussão de nossa proposta, apresentaremos brevemente uma revisão da linguagem funcional SISAL, cuja construção *forall* foi à escolhida para a aplicação de nosso modelo.

A. Linguagem Funcional *SISAL*

O projeto da linguagem *SISAL* (*Streams and Iteraction in a Single Assignment Language*) [MGR 85], além de uma linguagem aplicativa de propósito geral, define também uma forma intermediária independente, IF1 [SKE 85a], técnicas de otimização para computação aplicativa paralela de alta performance, um ambiente de microtarefas com suporte a fluxo de dados em sistemas de computador convencionais.

Temos como importante característica de *SISAL* a semântica de associação simples, ou seja, os nomes são ligados a valores apenas uma vez e não a posições de memória, não existindo então *aliasing*. Visto que *SISAL* é uma linguagem livre de efeitos colaterais, as subexpressões podem ser avaliadas em qualquer ordem sem afetar os resultados computados.

Também as iterações de *loops forall* são elementos independentes, assim os valores definidos dentro da iteração não podem ser acessados fora dela. Todo *forall* é composto de uma parte geradora de série, corpo e cláusula de retorno. No Sisal 1.2 duas formas da construção *forall* são definidas, uma permite que sejam especificados produtos (cartesianos) índices de *array* e *streams* internos ou externos (forma produto) e a outra que não (forma "não-produto") [MGR 85].

B. Trabalhos Relacionados

Existe na literatura um grande número de trabalhos que tratam do problema de distribuição de dados, mas comentaremos apenas os dois trabalhos que estão mais diretamente relacionados a nossa proposta.

B.1 Anderson e Lam

O trabalho apresentado por Anderson e Lam [AND 97], formulado como um sistema de equações sobre condições que devem ser satisfeitas nas decomposições de computações e dados, busca o mapeamento de dados e computações eficiente sobre um espaço virtual de processadores.

As restrições sobre decomposições de dados e computação são expressas através de requerimentos para o núcleo das funções lineares que representam os mapeamentos de dados e computações. O cálculo do núcleo é baseado num método iterativo, que compara restrições mútuas sobre os mapeamentos de dados e computação, retorna decomposições livres de comunicação.

O algoritmo de Anderson e Lam também trata da distribuição dinâmica de dados, utilizando para isso um grafo não direcional com valores de custo (peso) para vértices e arcos. No grafo os vértices correspondem a aninhamentos de *loops* e os arcos representam possíveis remapeamentos de dados entre os aninhamentos de *loops*. Os pesos dos arcos são obtidos pela combinação do pior caso de custo de comunicação com a probabilidade de ocorrência dos remapeamentos.

B.2 O'Boyle e Gurd

A proposta de particionamento automático de dados, para sistemas de memória distribuída de O'Boyle e Gurd [OBO 93] trata de fatores como o balanceamento de carga, alinhamento e particionamento de *arrays* e iterações de programas *SISAL* com sintaxe reduzida.

A análise visando o balanceamento de carga é expressa matematicamente como uma condição de invariância do espaço de iteração. Também são apresentadas transformações auxiliares para reordenação e intercalação dos iteradores.

O alinhamento relativo de *arrays* é descrito em termos das ocorrências dos *arrays* (matrizes expressando os subscritos dos *arrays*) e regiões factíveis das computações.

A distribuição dos dados é elaborada em função do número de processadores, do domínio dos índices dos *arrays*, do número de ciclos na distribuição do *array* e da quantia de elementos de dados contínuos, permitindo distribuições em colunas, linhas, blocos e cíclicas. O método para a partição do espaço de iteração é formulado em função do número de processadores, espaço poliedral das iterações e as matrizes de ocorrência do *array*.

As técnicas apresentadas no trabalho foram tratadas separadamente e entre elas existem conflitos das funcionalidades, e visando saná-los, O'Boyle e Gurd, desenvolveram uma solução para a utilização de seus métodos baseada em heurística.

III. PROPOSTA PARA DISTRIBUIÇÃO DE DADOS E COMPUTAÇÃO

Os métodos para as análises estáticas apresentadas a seguir são baseados na álgebra linear e programação linear. Partimos do pressuposto que o código analisado encontra-se na forma canônica [AND 97, WOL 92] com *loops* inteiramente permutáveis.

A.1 Alinhamento

Para nosso trabalho, o *forall* cercando um segmento de comandos qualquer será representado como o vetor de iteradores $\vec{i} = (i_1, i_2, \cdots, i_n)$ onde n é o número de *loops* aninhados definindo um espaço de iteração I e um *array* de

m dimensões define um espaço de *array* A e cada elemento é acessado por um vetor de índices $\vec{a} = (a_1, a_2, \cdots, a_m)$.

Iniciamos a discussão do alinhamento de dados apresentamos uma função afim $f_l(\vec{i}) = L\,\vec{i} + \vec{l}$ que representará o acesso ao *array* **x** (com m dimensões) num aninhamento de loops j e $f_r(\vec{i}) = R\,\vec{i} + \vec{r}$ representando o acesso a um *array* **y** (com m dimensões) no mesmo aninhamento de loops j. Ainda com relação às funções de acesso, L é a matriz de acesso ao *array* **x** e R é a matriz de acesso ao *array* **y**.

Podemos então formular o alinhamento de dados como o seguinte sistema onde buscamos uma transformação linear que iguale as duas funções de acesso a *array*, l e r:

$$L\,\vec{i} + \vec{l} = T.\left[R\,\vec{i} + \vec{r}\right] + \vec{t}$$

ou seja procuramos o par $(T, \vec{t})$. Para facilitar a observação do sistema podemos reescrever o sistema acima como $T.R = L$ e $T.\vec{r} + \vec{t} = \vec{l}$ sendo a solução da última equação facilmente obtida por operações de soma de vetores, razão pela qual analisaremos o primeiro sistema.

Desenvolvendo o sistema $T.R = L$, supondo m índices no *array* sendo acessado, teremos:

$$R^t\,(T_i)^t = (L_i)^t$$

onde T_i é a linha i da matriz de transformação T e L_i é a linha i da matriz de acesso L.

Vemos então que problema foi transformado num sistema da forma $Ax = b$, que pode ser resolvido com um método de redução [MEY 00], aplicado nos i sistemas ao mesmo tempo (inclusive no vetor coluna b, que conterá ao final o resultado do sistema). Após a aplicação da operação, a matriz de coeficientes será então chamada de matriz na forma escada reduzida.

No entanto, apenas a redução à forma de matriz escada reduzida não significa que o sistema possui solução, devemos também analisar o sistema resultante e verificar se ele é consistente, isto é, possui pelo menos uma solução. Isto pode ser realizado segundo um dos critérios de consistência apresentados por Meyer [MEY 00].

Realizado o alinhamento dos dados então o passo seguinte seria o mapeamento de computações e dados nos processadores virtuais.

A.2 Particionamento

A formulação buscando o mapeamento de dados e computações segue a representação de Anderson e Lam, diferindo, entretanto na forma como é feita a escolha da distribuição dos dados e computações. Na nossa proposta, esta verificação é realizada após a busca de uma matriz de transformação linear que alinhe os acessos aos *arrays* do aninhamento. Caso não tenha sido possível o alinhamento, a fase do particionamento definirá quais dados e computações devem ser mantidos juntos de modo a reduzir a quantia de comunicação entre os processadores.

O particionamento será representado por duas funções, uma nomeada decomposição de dados afim e a outra decomposição de computação afim. A primeira, $d : A \mapsto P$ para mapear um *array* de m dimensões num espaço de processadores de p dimensões, e dada por $d(\vec{a}) = D\,\vec{a} + \vec{\delta}$ onde D é uma matriz de transformação linear e $\vec{\delta}$ é um vetor constante. A outra, $c : I \mapsto P$, para mapear um aninhamento de *loops* de l dimensões no espaço de processadores, é dada por $c(\vec{i}) = C\,\vec{i} + \vec{\gamma}$ onde C é uma matriz de transformação linear e $\vec{\gamma}$ é um vetor constante.

Para relacionar dado e computação, uma nova relação é definida tendo como agente de ligação a função de acesso a *array*, $f_l(\vec{i})$ que representa a função de acesso ao *array* x num aninhamento de *loops* j. Temos, portanto

$$D.\vec{a} + \vec{\delta} = C\,\vec{i} + \vec{\gamma}$$
$$D.f_l(\vec{i}) + \vec{\delta} = C\,\vec{i} + \vec{\gamma}$$
$$D.L\,\vec{i} + D.l + \vec{\delta} = C\,\vec{i} + \vec{\gamma}$$

que indica quando dado e computação serão associados ao mesmo processador virtual.

Deixando de lado a comunicação devido ao deslocamento (que envolve apenas comunicação entre vizinhos) temos como resultado:

$$D.L\,\vec{i} = C\,\vec{i}$$

que é a equação de sincronização.

Consideremos agora que para um aninhamento de *loops* j de profundidade n, os *loops* de $1 \ldots s$ sejam os *loops* paralelos mais externos e os *loops* $q = (s+1) \ldots n$ aqueles não paralelos. Temos então que as iterações $\vec{i}$ e $\vec{i} + \vec{e}_q$ devem ser associados ao mesmo processador, onde $\vec{e}_q$ é o $q - ésimo$ vetor elementar de dimensão n [AND 97]:

$$C\,\vec{e}_q = 0$$

No caso do *forall* de *SISAL* essa situação não será um problema visto que ele é paralelo por natureza. Usando essa propriedade temos a liberdade para a escolha do nível de paralelismo e podemos assim adotar uma política diferente para a distribuição das "iterações". A política para a distribuição das iterações pode selecionar quais iterações dos

loops deixar juntos e quais deixar separado, segundo a carga de processamento existente, ou seja estamos livres para escolher o valor de s em $q=(s+1)$.

O cálculo da decomposição de deslocamento, desconsiderada no primeiro momento, é dado por:

$$\vec{\gamma} = D\vec{l} + \vec{\delta}$$

Caso as duas análises, alinhamento e particionamento tenham atingido seus objetivos, fica disponível para o mapeamento, quais as dimensões de dados e computações devem ser distribuídas entre os processadores. O passo seguinte então é o cálculo dos índices de dados e computações que serão distribuídos nos processadores físicos.

A.3 Mapeamento

Em razão das novas formulações, vamos redefinir o domínio de índices como a inequação

$$\vec{l} \le \vec{a} \le \vec{h} \qquad (1)$$

onde $\vec{l}$ e $\vec{h}$ são vetores de dimensão $m\times 1$ que representam os limites inferior e superior do *array*. Da mesma forma redefiniremos os iteradores de um *loop* como a inequação

$$\vec{\lambda} \le \vec{i} \le \vec{\eta} \qquad (2)$$

onde $\vec{\lambda}$ e $\vec{\eta}$ são vetores de dimensão $n\times 1$ que representam respectivamente os limites inferior e superior do *loop*.

Os p processadores do espaço de processadores, ρ, serão arranjados como os seguintes sub-espaços $\rho \mapsto \underline{\rho_1}\times\ldots\times\underline{\rho_q}$ onde ρ_x é o número de processadores na *x-ésima* dimensão, com $1 \le x \le q$ e $1 \le q_{\max} \le \min(m,n) = Q$.

A carga média será calculada por:

$$b = \left\lceil \frac{\sum_{x=1}^{q} (\eta_x - \lambda_x + 1)}{p} \right\rceil$$

λ_x e η_x são os limites superior e inferior de cada uma das iterações do aninhamento de *loops*.

Com a carga média, b, podemos calcular o número de processadores associados a cada dimensão do *array* como:

$$p_x = \left\lceil \frac{\eta_x - \lambda_x + 1}{b} \right\rceil$$

Os limites dos *loops* locais ao processador z_x consistem de seqüências de valores distintos dados por $\underline{i_{xz_x}} = \underline{i^1_{xz_x}} \times\ldots\times \underline{i^d_{xz_x}}$ sendo que d é o número de vezes que o *array* é empacotado nos processadores, geralmente uma vez. E os limites dos *arrays* locais ao processador z_x consistem de seqüências de valores distintos dados por $\underline{a_{xz_x}} = \underline{a^1_{xz_x}} \times\ldots\times \underline{a^d_{xz_x}}$.

Reformulando a inequação (1) para definir o valor dos índices locais em algum processador z, temos então como novo conjunto de inequações

$$\begin{bmatrix} -e_x \\ \hline e_x \end{bmatrix} \begin{bmatrix} i^{k_x}_{1z_x} \\ i^{k_x}_{2z_x} \\ \vdots \\ i^{k_x}_{Qz_x} \end{bmatrix} \le \begin{bmatrix} -[(z_x-1)\times b + (k_x-1)\times r_x + \lambda_x] \\ \hline z_x \times b + (k_x-1)\times r_x + \lambda_x - 1 \end{bmatrix} \qquad (3)$$

e reformulando a inequação (2) temos

$$\begin{bmatrix} -e_x \\ \hline e_x \end{bmatrix} \begin{bmatrix} a^{k_x}_{1z_x} \\ a^{k_x}_{2z_x} \\ \vdots \\ a^{k_x}_{Qz_x} \end{bmatrix} \le \begin{bmatrix} -[(z_x-1)\times b_x + (k_x-1)\times r_x + l_x + (\lambda_x - 1) + \delta_x] \\ \hline z_x \times b_x + (k_x-1)\times r_x + l_x + (\lambda_x - 1) + \delta_x - 1 \end{bmatrix} \qquad (4)$$

para o índices dos *arrays* onde δ_x é o deslocamento encontrado no acesso aos dados:

$$1 \le z_x \le p_x \qquad 1 \le k_x \le d_x \qquad d_x = \left\lceil \frac{\eta_x - \lambda_x + 1}{b \times p_x} \right\rceil.$$

IV. Estudo dos métodos com SISAL

A. Alinhamento

Apresentaremos nessa seção a aplicação da transformação que busca verificar se é possível o alinhamento de dados de exemplos de trechos de código SISAL. No código da Fig. 1 abaixo, podemos ver que o *array* a é criado com os índices (i, j, k) e o *array* b é acessado pelos índices $(i+j, k+j, k)$, o que mostra que os acessos não estão com um alinhamento perfeito.

No código da Fig. 1, temos como funções de acesso aos *arrays* a e b.

$$f_l(\vec{i}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{e} \quad f_r(\vec{i}) = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Teremos então dois sistemas distintos $T\,R = L$ e $\vec{t} = \vec{l} - T\,\vec{r}$. Após a redução dos sistemas para a forma de matriz escada temos então:

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

E dessas matrizes podemos constatar que temos soluções consistentes para os dois alinhamentos.

Resolvendo para a parte do deslocamento temos $\vec{t} = [0 \;\; 0 \;\; 0]^t$ e portanto, a solução do sistema seria

$$T = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \vec{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

B. Particionamento

Para o exemplo da Fig. 1, supondo que nossa escolha seja distribuir apenas o *loop* mais externo do *forall*, encontraríamos $C_1(\vec{e}_2) = \vec{0}$ e $C_1(\vec{e}_3) = \vec{0}$, que representa os *foralls* que serão colocados no mesmo processador e $D_1\,L = C_1$ e $D_1\,R = C_1$ para dados e computações que serão colocados juntos, ou seja temos os sistemas

$$C_1 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad D_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = C_1$$

$$C_1 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad D_1 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = C_1$$

e temos como uma possível solução $C_1 = [1 \;\; 0 \;\; 0]$ e $D_1 = [1 \;\; 0 \;\; 0]$. Isto indica que uma opção seria distribuir dados e computação pelo primeiro índice.

Entretanto vemos que essa não é a única solução possível. Outra solução seria $C_1 = [0 \;\; 0 \;\; 0]$ e $D_1 = [0 \;\; 0 \;\; 0]$ entretanto esta solução significa que não distribuiremos dados e computações entre os processadores, mas colocaríamos todos os elementos no mesmo processador. Com essa solução não teríamos distribuição de computação e dados, que vai contra exatamente ao que procuramos.

C. Mapeamento

Pelo resultado obtido anteriormente a distribuição poderia ser pelo primeiro índice do aninhamento de *loops* e primeira dimensão dos *arrays*, e supondo 4 processadores físicos teríamos

$$b = \left\lceil \frac{100-1+1}{4} \right\rceil = 25 \qquad \text{e} \qquad p_1 = \left\lceil \frac{100-1+1}{25} \right\rceil = 4$$

ou seja, o número de elementos por processador será 25 e o número de processadores associado à dimensão que será particionada são 4.

Desenvolvendo as inequações dadas por (3) e (4) temos então

$x = 1 \qquad b_1 = b = 25 \qquad r_1 = b_1 \times p_1 = 25 \times 4 = 100$

$d = (d_1) \quad d_1 = \left\lceil \frac{100_1 - 1 + 1}{100} \right\rceil = 1 \quad z_1 = (1,2,3,4) \quad k_1 = (1)$

$$\left[\frac{-e_1}{e_1}\right] \begin{bmatrix} \vec{i}_{1z_1}^{\,k_1} \\ \vec{i}_{2z_1}^{\,k_1} \\ \vec{i}_{3z_1}^{\,k_1} \end{bmatrix} \leq \left[\frac{-[(z_1-1)\times b_1 + (k_1-1)\times r_1 + \lambda_1]}{[z_1 \times b_1 + (k_1-1)\times r_1 + \lambda_1 - 1]}\right] =$$

$$\left[\frac{-1 \;\; 0 \;\; 0}{1 \;\; 0 \;\; 0}\right] \begin{bmatrix} \vec{i}_{1z_1}^{\,1} \\ \vec{i}_{2z_1}^{\,1} \\ \vec{i}_{3z_1}^{\,1} \end{bmatrix} \leq \left[\frac{-[(z_1-1)\times 25 + 1]}{[z_1 \times 25]}\right]$$

$z_1 = 1 \qquad z_1 = 2 \qquad z_1 = 3 \qquad z_1 = 4$

$1 \leq i_{11}^1 \leq 25 \qquad 26 \leq i_{12}^1 \leq 50 \qquad 51 \leq i_{13}^1 \leq 75 \qquad 76 \leq i_{14}^1 \leq 100$

e

$$\left[\frac{-e_1}{e_1}\right] \begin{bmatrix} \vec{a}_{1z_1}^{\,k_1} \\ \vec{a}_{1z_1}^{\,k_1} \\ \vec{a}_{1z_1}^{\,k_1} \end{bmatrix} \leq \left[\frac{-[(z_1-1)\times b_1 + (k_1-1)\times r_1 + l_1 + (\lambda_1-1) + \delta_1]}{[z_1 \times b_1 + (k_1-1)\times r_1 + l_1 + (\lambda_1-1) + \delta_1 - 1]}\right] =$$

$$\left[\frac{-1 \;\; 0 \;\; 0}{1 \;\; 0 \;\; 0}\right] \begin{bmatrix} \vec{a}_{1z_1}^{\,1} \\ \vec{a}_{1z_1}^{\,1} \\ \vec{a}_{1z_1}^{\,1} \end{bmatrix} \leq \left[\frac{-[(z_1-1)\times 25 + 1]}{[z_1 \times 25]}\right]$$

$z_1 = 1 \qquad z_1 = 2 \qquad z_1 = 3 \qquad z_1 = 4$

$1 \leq a_{11}^1 \leq 25 \quad 26 \leq a_{12}^1 \leq 50 \quad 51 \leq a_{131}^1 \leq 75 \quad 76 \leq a_{14}^1 \leq 100$

V. Figuras

A. Figuras

```
%  dimensão de b
%  b[1..100][1..100][1..100]

  a := for i in 1, 100 cross j in 1, 100 cross k in 1, 100
       returns array of b[i+j,k+j,k]
       end for
```

Fig.1 Trecho de código *SISAL* com *forall*

VI. Conclusão

Apresentamos neste artigo propostas para análise da distribuição de dados e computações em sistemas de memória distribuída. Esses métodos são baseados em fundamentos matemáticos da álgebra linear (funções afins) e programação linear (condições de invariância).

A utilização de modelos matemáticos para a análise apresenta um grande benefício, visto que escolhemos métodos matemáticos bem conhecidos, com provas de sua correção, o que facilita a verificação da validade do modelo apresentado.

Basicamente o modelo consiste em traduzir os trechos de código em que existem *loops forall* com acessos a *arrays* em sistemas de equações ou inequações lineares, que devem então ser resolvidas. Esses sistemas são montados conforme as necessidades de cada uma das etapas da nossa análise.

Em nosso modelo atacamos duas frentes. Uma busca o alinhamento perfeito dos dados o que possibilitaria a distribuição livre de dados e computações. O ideal para a distribuição seria a existência de uma transformação linear que possibilitasse o alinhamento nos acessos aos dados o que tornaria livre a escolha de quais dimensões de arrays e aninhamentos de loops particionar. A outra trata o caso em que não existe um alinhamento possível. O particionamento verifica então quais dados devem ser deixados locais e quais podem ser distribuídos. Caso tenhamos uma solução nos sistemas, podemos então realizar uma distribuição livre de comunicações não locais.

O mapeamento realiza análises que associam dimensões de *arrays* e aninhamentos de *forall* a processadores físicos existentes, se as análises de alinhamento e particionamento tenham obtido sucesso, caso contrário o código analisado é mantido sem alterações.

O modelo para particionamento foi elaborado levando em conta que os valores para dimensões dos *arrays* e limites dos *loops forall* que aparecem no código analisado são constantes. Uma possível extensão do modelo apenas exigiria que as análises obtivessem no momento da execução os valores dos limites acima referidos, transferindo parte das análises do compilador para tempo de execução. A avaliação de quando é justificável realizar esse tipo de análise em tempo de execução não parece trivial.

A proposta apresentada também não leva em conta o balanceamento de carga (estático ou dinâmico), mas essa análise poderia perfeitamente ser encaixada logo após os métodos apresentados terem sido aplicados.

Referências

[AND 97] ANDERSON, J. M, Automatic Computation and Data Decomposition For Multiprocessors, Thesis CSL-TR-97-719, Stanford University, Mar 1997.

[BOO 97] BOOKMAN, J. B., Kogge, P. M., The Case for Processing-in-Memory, Technical Report TR-97-03, University of Notre Dame, Jan.1997

[CUL 99] CULLER, D., et. al, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, ISBN 1558603433, 1999.

[FEO 90] FEO, J. T. et. al, A Report on the Sisal Language Project, Journal of Parallel and Distributed Computing, 10, 349-366, Dec. 1990.

[MEY 00] MEYER, C. D., Matrix Analysis and Applied Linear Algebra, Society for Industrial & Applied Mathematics - SIAM, **ISBN:** 0898714540, Jun. 2000.

[MGR 85] MCGRAW, J. et al. SISAL: Stream and Iteration in a Single Assignment Language, Language Reference Manual, Technical Report M-146, Rev. 1, University of California, Lawrence Livermore National Laboratory, Mar. 1985.

[OBO 93] O'BOYLE, M. F. P., Program and Data Transformations for Efficient Execution on Distributed Memory Architectures, Technical Report UMCS-93-1-6, Department of Computer Science, University of Manchester, Jun. 1993.

[SKE 85a] SKEDZIELEWSKI, S. K. and Glauert, J., IF1 An Intermediate Form for Applicative Languages, Manual M170, Lawrence Livermore National Laboratory, Livermore, California, Jan. 1985.

[SKE 91] SKEDZIELEWSKI, S. K., Parallel and Functional Languages and Compilers, ACM Press, 1991

[WOL 92] WOLF, E. M., Improving Locality and Parallelism in Nested Loops, Department of Computer Science, Stanford University, Aug. 1992.