Semi-systolic Algorithm Synthesis for the Grid*

Kunio Okuda[1] and Marco Dimas Gubitoso[1]

[1] Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação
Rua do Matão, 1010
CEP 05508-900 São Paulo, SP, Brazil
Tel. + 55 11 3818 6135 Fax + 55 11 3814 4135
Comments and suggestions to: kunio@ime.usp.br, gubi@ime.usp.br

Abstract—

The systolic paradigm that appeared in the late seventies has contributed to parallel computing in two ways: the development of the method allowed the parallelization of nested loops algorithms and at the same time several practical applications like cryptography and multimedia using mostly 2-d grids.

Unifying these two tendencies we propose a new method for semi-systolic algorithm synthesis, for uniform nested loops. Our solution is better than the traditional approach if the target architecture is a grid.

Keywords— **grids, systolic, parallel computing, schedule**

I. Introduction

The idea of parallel systolic algorithms appeared in the late seventies [2, 4, 15] with the purpose to take advantage of the rapid development of integrated circuits for high performance parallel processing.

In the mid eighties, after a period when the algorithms were developed in an *ad hoc* way, the idea of systolic algorithm synthesis was proposed and a formal and systematic method [3, 7] became available. Later this idea was generalized and applied in a wider context, not restricted to systolic computing. From the new approaches, the main one is the hyperplane method [8, 9, 10, 13, 14].

On the other hand, still nowadays systolic computing is widely used in applications like cryptography and multimedia. Usually these applications use a 2-D grid with some limitations [11, 12].

This paper proposes the union of these approaches. The hyperplane method will be used to map uniform nested loop algorithms to a n-dimensional grid using only first-neighbor communication.

Uniform nested loops are very common in high performance parallel computing and guaranteeing that the target machine is always a grid makes the implementation more straightforward and natural to use.

The paper has the following structure: In the section II we describe briefly the definitions and the hyperplane method. The semi-systolic algorithm is described in the section III. In section IV we show the existence of operator T, the mapping operator. Section V presents an example and section VI concludes.

II. The hyperplane method

Consider a nested loop algorithm having the following general structure:

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
      S_1
      $\vdots$
      S_k

where l_1 and u_1 are constants, $l_j(i_1, \ldots, i_{j-1})$ and $u_j(i_1, \ldots, i_{j-1})$ are the lower and upper limits of the loops and all variables used in statements S_1 to S_k have indices which are affine functions of i_1 to i_n.

*This paper has support from FAPESP, Project SIDAM, number No.98/06138-2

The computation set for this algorithm is defined as

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

The statements S_1 to S_k are attributions and the differences between the indexes of the variable on the left-hand side (result) and each variable on the right-hand side (operand) forms a n-dimensional vector called *dependency vector*. The set of dependency vectors can be arranged as columns of a *dependency matrix*.

Given two statements S_u and S_v, it may exist several pairs of indexes (I, J) in Dom such that $S_u(I)$ depends on $S_v(J)$. We will restrict ourselves to the important subclass of uniform nested loops, where the dependency vector between two statements depends only on the difference of the indexes but not on their particular values. The importance of this subclass comes from the following two reasons:

1. Many algorithms for scientific computing belong to this class.
2. The regularity of this structure allows a better exploitation of parallelism.

Example 1

for $i = 0$ to N do
  for $j = 0$ to N
    S_1: $a(i,j) = b(i-1, j-5) + d(i-1, j+4)$
    S_2: $b(i,j) = c(i-1, j+6)$
    S_3: $c(i,j) = a(i, j-2)$
    S_4: $d(i,j) = a(i, j-1)$

This is a simplified version of a well-known example considered in [5, 10] and others.

For this example, the computation set is

$$Dom = \{(i,j) \in Z^2 | 0 \leq i, j \leq N\}$$

and there are 5 dependency vectors:

$$\begin{array}{lll} S_1 \rightarrow S_3 & : & d_1 = (0, \ \ 2) \\ S_3 \rightarrow S_2 & : & d_2 = (1, -6) \\ S_2 \rightarrow S_1 & : & d_3 = (1, \ \ 5) \\ S_1 \rightarrow S_4 & : & d_4 = (0, \ \ 1) \\ S_4 \rightarrow S_1 & : & d_5 = (1, -4) \end{array}$$

and the dependency matrix is

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

To simplify the notation we will follow [1, 6, 16, 17] and suppose that l_j and u_j are constant and that l_j is always null.

The hyperplane method [10, 14]

Consider uniform nested loops with dimension n and let $D = (d_1, \ldots, d_m)$ be a dependency matrix $n \times m$. The problem is to find a scheduling vector $\pi = (\pi_1, \ldots, \pi_n)$ which minimizes $max\{\pi \cdot I - \pi \cdot J | I, J \in Dom\}$, under the following restrictions:

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

All the points I and J in Dom resting on the same hyperplane perpendicular to π, that is $\pi \cdot I = \pi \cdot J$, will be processed simultaneously at time $\pi \cdot I (= \pi \cdot J)$. This hyperplane will be called *time hyperplane*.

This method identifies all points in the computation set which must be processed simultaneously, while minimizing the total execution time. However, it is possible that the solution is the best mapping for a given machine.

A dependency vector $\vec{d}$ going from p to q, $\{p, q\} \in Dom$, cannot lie completely on a hyperplane: if $p \in H_1$, then q must be in H_2 to be executed later, otherwise the dependency constraint would be violated. This condition follows directly from the method.

The projection of (p', q') of $\vec{d}$ on a hyperplane indicates that a communication must be done from the processor associated to p' to the processor associated to q'. It may be possible, by the hyperplane method, that p' and q' are not neighbors.

Our method avoids this problem by mapping any uniform nested loop algorithm into a grid network using only first neighbor links.

III. The New Method

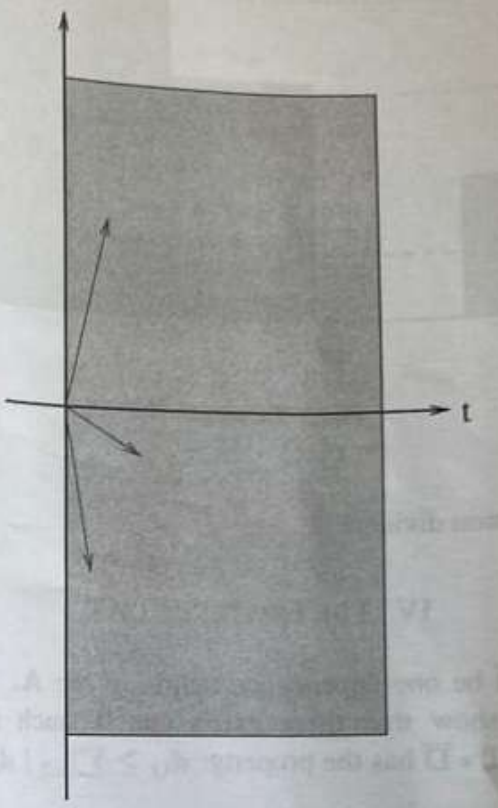
The main characteristics of the method are the following:

- It is semi-systolic because the data moves through intermediate processors without participating in any operation.
- **A** is a n-dimensional uniform nested loop algorithm.
- **T** is the transformation matrix of **A**.
- The first line of **T** is the scheduling vector (π

in the hyperplane method) and the $(n-1)$ last lines form the mapping matrix.

- The mapping is done to a virtual $(n-1)$-dimensional of virtual processors.
- $\overline{\mathbf{D}}$ is the dependency matrix of $\mathbf{A}$.
- $\mathbf{D} = \mathbf{T}^{*}\overline{\mathbf{D}}$ is the transformed dependency matrix.
- $d \in \mathbf{D}$ is a column of $\mathbf{D}$ and its first element represents the time (number of cycles) needed for the data arrive the destination. The last $(n-1)$ elements indicate the communication path.
- $\mathbf{D} = d_{ij}$ is such that $d_{1j} \geq \sum_{i=2}^{n} \mid d_{ij} \mid$. This mean that the "large" dependency vectors reach a later hyperplane. Figure 1 illustrates the case of dimension 2.
- Let x be a date with the dependency vector $d = (d_1, d_2, \cdots, d_n)$ generated in the processor $P(2, \cdots, n)$ at the cycle t_0. It means that x must be in the processor $P(2 + d_2, \cdots, n + d_n)$ at the cycle $t_0 + d_1$.
- x walks in the grid in the following way:
  x walks with a n-ple $(d_1, d_2, \cdots, d_n)$ where d_1 represents the number of necessary steps to enter the new operation and $d_2, \cdots, d_n$ represent the number of connections that x must across in each direction $2, \cdots, n$ (note that the direction 1 was used to represent the time). At each cycle, d_1 receives decrease of one unit. x and $(d_1, d_2, \cdots, d_n)$ walk to the direction of the first not null d_k only one connection (positive or negative sense conformable to the sign of the d_k). d_k receives decrease of one unit. This walking of the x the $(d_1, d_2, \cdots, d_n)$ continues until the $d_2, \cdots, d_n$ be all null. In this instance if d_1 is also null x enters in the operation at the processor $P(2 + d_2, \cdots, n + d_n)$. However if $x > 0$ then x stays in this processor and d_1 receives decrease of one unit in each cycle. In the cycle in which, x enters in the operation.
  As $d_{1j} \geq \sum_{i=2}^{n} \mid d_{ij} \mid$, it assures that x reach to the target processor without delay.
- **The hyperplane method for Semi-systolic Algorithm**
  Find $\mathbf{T}$ such that:

1. minimize $max\{\pi \cdot I - \pi \cdot J | I, J \in Dom\}$ where π is the first line of the $\mathbf{T}$.
2. $\mathbf{D} = \mathbf{T}^{*}\overline{\mathbf{D}}$ satisfy $d_{1j} \geq \sum_{i=2}^{n} \mid d_{ij} \mid$
3. $\pi \cdot D > 0$
4. $(\quad\quad) = 1$ (mcd=maximum com-



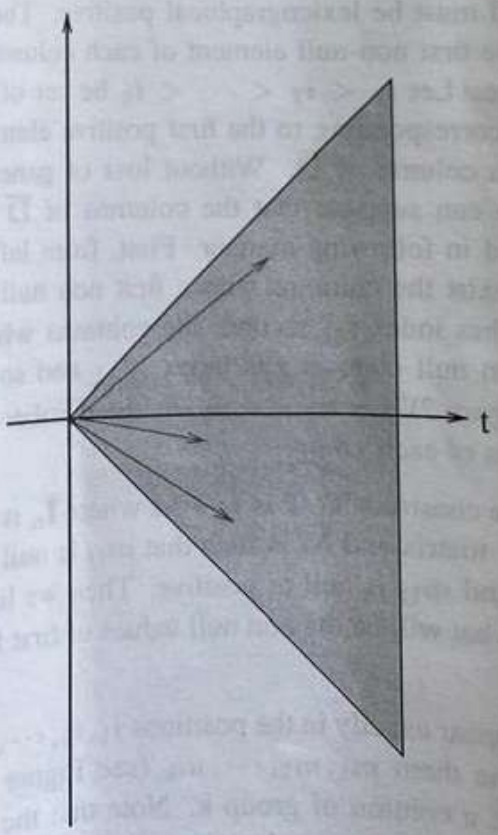


Fig. 1. before and after

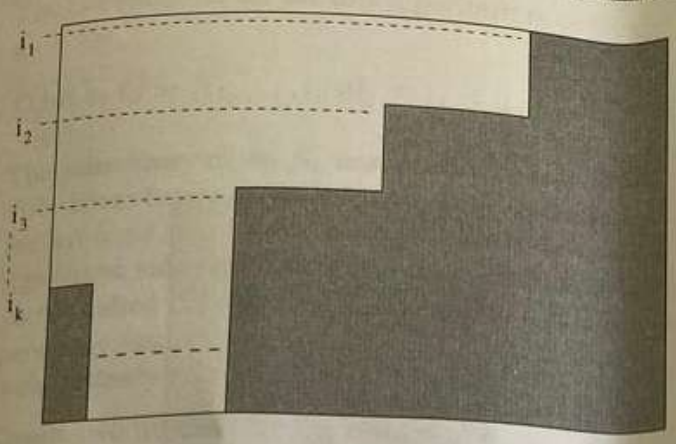


Fig. 2. The matrix $\overline{\mathbf{D}}$

mon divider)

IV. The Existence of T

Let $\overline{\mathbf{D}}$ be one dependence matrix of the $\mathbf{A}$. We shall show that there exists one $\mathbf{T}$ such that $\mathbf{D} = \mathbf{T} * \overline{\mathbf{D}}$ has the property: $d_{1j} \geq \sum_{i=2}^{n} \mid d_{ij} \mid$.

Each column of $\overline{\mathbf{D}}$ represents a dependency vector and must be lexicographical positive. Therefore the first non-null element of each column is positive. Let $i_1 < i_2 < \cdots < i_k$ be set of indexes corresponding to the first positive element of each column of $\overline{\mathbf{D}}$. Without loss of generality, we can suppose that the columns of $\overline{\mathbf{D}}$ are grouped in following manner. First, from left to right, exist the columns whose first non null element has index i_k, second, the columns whose first non null element has index i_{k-1} and so on (see Figure 2). Let $n_k, \cdots, n_2, n_1$ the numbers of columns of each group.

Now we construct $\mathbf{T}$. $\mathbf{T}$ is $\mathbf{I_n} + \mathbf{M}$ where $\mathbf{I_n}$ is the identity matrix and $\mathbf{M}$ is such that m_{ij} is null for $i > 1$ and m_{1j} is null or positive. Then we have to see what will be the non null values of first line of $\mathbf{T}$.

They appear exactly in the positions $i_1, i_2, \cdots, i_k$. We name them $m_1, m_2, \cdots, m_k$ (see Figure 3). Let d be a column of group k. Note that the elements of index larger than $i_k + 1$ in $\mathbf{T} * d$ are equal to elements of d. We can choose m_k such that $m_k \geq \sum_{i=i_k+1}^{n} \mid d_{ij} \mid$. Choose the lesser m_k satisfying the above inequality for each column of group k. Do the same for group $k-1$ and notice that the choice of m_{k-1} does not affect group k. Continue the process until the group 1 and we have $\mathbf{T}$.

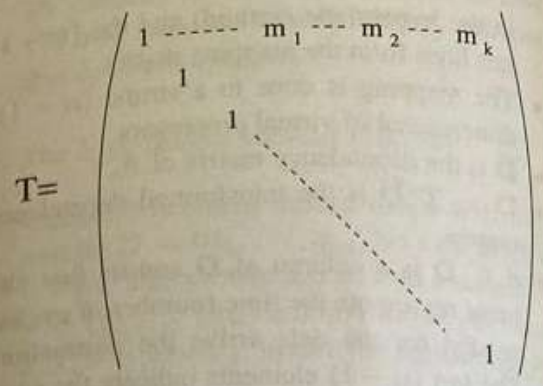


Fig. 3. The matrix $\mathbf{T}$

V. The Example

Let us go back to example 1.

We have

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

This $\overline{\mathbf{D}}$ does not satisfy the condition $d_{1j} \geq \sum_{i=2}^{n} \mid d_{ij} \mid$.

Let

$$T = \begin{pmatrix} 7 & 1 \\ 6 & 1 \end{pmatrix}$$

Then we have $\mathbf{D} = \begin{pmatrix} 2 & 1 & 12 & 1 & 3 \\ 2 & 0 & 11 & 1 & 2 \end{pmatrix}$ satisfying above condition. It needs $8N$ steps: the same number of the hyperplane method ([10]).

VI. The Conclusion

We presented the Semi-systolic Algorithm Synthesis of uniform nested loops for the Grid. It uses the hyperplane method to generate a Semi-systolic algorithm for the processor grid using only first-neighbor communication. As uniform nested loops are of great interest in the high-performance parallel computer field, the potential of this application is very large. In the other hand the fact that the resultant processor array is always

a grid (2-dimension generally) makes the implementation much more feasible.

REFERENCES

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

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

[3] 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.

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

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

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

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

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

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

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

[11] Schmidt, B., Schimmler, M., Schroder, H. Long Operand Arithmetic on Instruction Systolic Computer Arquitectures and Its Application in RSA Cryptography. *Euro-Par'98 Parallel Processing, Lecture Notes in Computer Science Series 1123*, Springer-Verlag, 1998, 916-922.

[12] Schmidt, B., Schimmler, M. A Parallel Accelerator Architecture for multmedia Video compression. *Euro-Par'99 Parallel Processing, Lecture Notes in Computer Science Series 1123*, Springer-Verlag, 1999, 950-960.

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

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

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

[16] Wolfe, M., *Optimizing supercompilers for supercomputers*. MIT Press, Cambridge, MA, 1989.

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