for $m = 1$ and fixed initial request rate r for all processors. The factor f and the bandwidth can be calculated from (14) and (16).

For fixed priority, (14) becomes

$$\mathrm{BW} = \sum_{i=1}^{n} f_i r \tag{17}$$

i.e., the bandwidth is given by the summation of the contribution made by each processor Dependencies are separated by the use of the factor f. The bandwidth is also given by

$$\mathrm{BW} = 1 - \prod_{j=1}^{n} f_i(1 - f_i r) \tag{18}$$

by reduction of the final bandwidth equation in [4] with $m = 1$. Hence, f can be calculated by iteration. We have

$$\mathrm{BW}_i = 1 - \prod_{j=1}^{i} f_j(1 - f_j r). \tag{19}$$

From (19), we obtain

$$\mathrm{BW}_i = 1 - (1 - \mathrm{BW}_{i-1}) f_i (1 - f_i r) \tag{20}$$

i.e., an expression for BW_i in terms of BW_{i-1}. We also have

$$\mathrm{BW}_i - \mathrm{BW}_{i-1} = f_i r \tag{21}$$

i.e., the contribution of the ith processor on the bandwidth, from (17). Equations (20) and (21) can be used iteratively to obtain BW and f. By using (1), (6), and (15), we obtain the access time in terms of f_i as

$$T_i = \frac{1 - f_i}{f_i r}. \tag{22}$$

We have computed the access time for the lowest priority processor using the performance degradation factor model and find the model gives values close to the other models described in Section IV for low numbers of processors or low request rate. We include the method here because it is a method previously proposed for fixed priority taking into account dependencies, yet it only gives satisfactory results for low numbers of processors or low request rate when applied to access time.

VI. Conclusions

In this correspondence, we have presented new simulation results for the access time of a shared memory multiprocessor system. We have shown that the access time reduces with reduced requests in all cases including equal priority, in contrast to previously published work and that there are no significant differences between random, equal, and rotating priority protocols when requests are generated in a random order. Previously published theoretical equations for the access time have been corrected and modified to become closer to simulation results. We have also highlighted that great care should be exercised in deriving probabilistic equations to ensure that joint probabilities are from independent events. This is especially important when the priority protocol is not the random or an equivalent protocol. Differences between theoretical and simulation results, where they exist, can be attributed to the dependences in probabilities in the theoretical equations.

References

[1] F. El Guibaly, "Design and analysis of arbitration protocols," *IEEE Trans. Comput.*, vol. 38, pp. 161–171, Feb. 1989.
[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.
[3] S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find," *Commun. ACM*, vol. 31, pp. 1192–1201, Oct. 1988.
[4] D. W. L. Yen, J. H. Patel, and E. S. Davidson, "Memory interference in synchronous multiprocessor systems," *IEEE Trans. Comput.*, vol. C-31, pp. 1116–1121, Nov. 1982.
[5] T. N. Mudge, J. P. Hayes, G. D. Buzzard, and D. C. Winsor, "Analysis of multiple bus interconnection networks," *Proc. 1984 Int. Conf. Parallel Processing*, IEEE, 1984, pp. 228–232.
[6] Y-C Liu and C-J Jou, "Effective memory bandwidth and processor blocking probability in multiple-bus systems," *IEEE Trans. Comput.*, vol. C-36, pp. 761–764, June 1987.

Data Flow Representation of Iterative Algorithms for Systolic Arrays

Chein-Wei Jen and Ding-Ming Kwai

***Abstract*—An algebraic representation is proposed for regular iterative algorithms that can be described as bundles of data flows with different wavefronts. Modeling data flows with a generating function of power series, this form corresponds to the geometric representation such as a dependence graph. The main attributes of systolic algorithms and arrays are revealed by a unique data flow representation. This provides the ability to pipeline two or more systolic arrays solving different subproblems without intermediate buffering. We use an example to show the case with which our technique can be used.**

***Index Terms*—Algebraic representation, chaining systolic arrays, data flow, dependence graph, flow velocity, generating function, pipelining.**

I. Introduction

Regular iterative algorithms are often shown by a geometric representation, called a dependence graph, that corresponds vertices to variables and edges to dependence between variables [4]. The dependence graph constructed in an n-dimensional integer Euclidean space basically is a locally recursive and single assignment form of the iterative algorithm. It consists of a finite set of variables that each variable takes on a certain value at each point in the index space and assumes a unique value supplied by the variables at the neighboring points. Using dependence graphs has the advantage that they are easy to understand and convenient to display. The transformation of iterative algorithms into systolic arrays is also well defined via dependence graphs [3], [7]–[11], [14].

The systolic array structure that includes PE's locations and communication links between PE's can be obtained simply by projecting the dependence graph onto a lower dimensional processor space. We denote the processor space as $\boldsymbol{S}$. The projection direction is represented by an integer n-vector $\mathbf{Pd}$, and is defined as the iteration vector by [3]. Choosing more than one projection directions (under

Manuscript received April 5, 1989; revised September 25, 1990. This work was supported in part by the National Science Council, Taiwan, Republic of China, under Grant NSC78-0404-E009-23.

The authors are with the Institute of Electronics, National Chiao-Tung University, Hsin-Chu, Taiwan 30039, Republic of China.

IEEE Log Number 9102597.

some conditions [7]), multiple projection is made. Thus, a systolic algorithm can be mapped into an array structure of desired dimension. It is obvious that $\boldsymbol{S}$ is orthogonal to $\mathbf{Pd}$.

$$\boldsymbol{S} \cdot \mathbf{Pd} = 0. \tag{1}$$

Once the processor space is determined, a schedule is required for the computations. A correct scheduling assures that a right data reaches its destination at the right time step. The schedule space Π is a linear transformation that maps vertices of the dependence graph to scalar time steps. The linear scheduling is equivalent to draw parallel isotemporal hyperplanes through the dependence graph; the progression ticks away the time along the direction normal to these hyperplanes.

According to the linear properties and causality constraints on the dependence graphs, the following inequalities should be satisfied.

$$\Pi \cdot \vec{d} \geq 1 \tag{2}$$

where dependence vector $\vec{d}$ is a direct dependence among variables with respect to the index points or an index displacement vector in the index space, and

$$\Pi \cdot \mathbf{Pd} \neq 0\,.$$

The derivations of processor space $\boldsymbol{S}$ and schedule space Π could be found in [3], [7], [11], and [14].

However, we can regard the iterative algorithms as moving data items rhythmically and extend this concept to systolic arrays. The entities that we consider then consist of bundles of data streams having individual velocities, sources, and ends. The data items interact as they meet and generate intermediate results. Describing this process in an algebraic notation, the pipelinability and parallelism that an iterative algorithm possesses can be shown clearly [12], [13].

We would like to explore the relationship between the geometric representation and the kind of algebraic representation in this short paper. In Section II, the algebraic notation of data flow and wavefront are defined by generating functions of power series, moreover, the parameters of characterization are derived. In Section III, the system chaining two or more systolic arrays is considered and the condition for such coalescence is examined. Section IV gives a conclusion.

II. Data Flows and Wavefronts

A. Representation of Data Flows

Let $(\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_t, \cdots)$ be the symbolic representation of a sequence of events. The function

$$F(\tau) = \boldsymbol{p}_0 \mu_0(\tau) + \boldsymbol{p}_1 \mu_1(\tau) + \cdots + \boldsymbol{p}_t \mu_t(\tau) + \cdots$$

is called the generating function of the sequence $(\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_t, \cdots)$ where $\mu_0, \mu_1, \cdots, \mu_t, \cdots$ is a sequence of function of τ used as indication. Clearly the generating function of a sequence is just an alternative representation of the sequence $(\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_t, \cdots)$ [6]. In our case, the form of $\mu_t(\tau)$ is chosen as τ^t, then for the sequence we have

$$F(\tau) = \boldsymbol{p}_0 + \boldsymbol{p}_1 \tau + \cdots + \boldsymbol{p}_t \tau^t + \cdots.$$

or

$$F(\tau) = \sum_{t=0} \boldsymbol{p}_t \tau^t.$$

Without loss of generality, a 3-D coordinate is used. A data stream $F(\tau)$ represents a data item moving in a specific path. Assume that at time $t = s$, the data stream originates from the source and at time $t = e$, the data stream ends to the outside world. At time t, let the variable be located at position $\boldsymbol{X}^{i(t)} \boldsymbol{Y}^{j(t)} \boldsymbol{Z}^{k(t)}$. The data stream is described as a generating function of power series as

$$F(\tau) = \sum_{t=s}^{e} f \boldsymbol{X}^{i(t)} \boldsymbol{Y}^{j(t)} \boldsymbol{Z}^{k(t)} \tau^t.$$

Since systolic algorithms preserve uniformity and regularity, we have $i(t) = i(s) + t\Delta i$, $j(t) = j(s) + t\Delta j$, $k(t) = k(s) + t\Delta k$ where $0 \leq t \leq e' = e - s$. Then

$$\begin{aligned} F(\tau) &= \sum_{t=0}^{e'} f \boldsymbol{X}^{i(s)+t\Delta i} \boldsymbol{Y}^{j(s)+t\Delta j} \boldsymbol{Z}^{k(s)+t\Delta k} \tau^{(t+s)} \\ &= f \boldsymbol{X}^{i(s)} \boldsymbol{Y}^{j(s)} \boldsymbol{Z}^{k(s)} \tau^s \sum_{t=0}^{e'} \alpha^{t\Delta i} \beta^{t\Delta j} \gamma^{t\Delta k} \tau^t \\ &= p_s \sum_{t=0}^{e'} \alpha^{t\Delta i} \beta^{t\Delta j} \gamma^{t\Delta k} \tau^t \\ &= p_s \frac{1 - \left(\alpha^{\Delta i} \beta^{\Delta i} \gamma^{\Delta k} \tau\right)^{e'+1}}{1 - (\alpha^{\Delta i} \beta^{\Delta j} \gamma^{\Delta k} \tau)}. \end{aligned} \tag{3}$$

α, β, and γ are unit-step moving operators in $\boldsymbol{X}, \boldsymbol{Y}$, and $\boldsymbol{Z}$ directions, respectively. The term $\alpha^{\Delta i} \beta^{\Delta j} \gamma^{\Delta k} \tau$ stands for the distance that a data item moves in index space per time step and is exactly the meaning of velocity. Obviously a data stream consists of its flow velocity $\alpha^{\Delta i} \beta^{\Delta j} \gamma^{\Delta k} \tau$, source position $\boldsymbol{p}_s$ and end position $\boldsymbol{p}_e$. For uniform recurrence equations [4], a special class of iterative algorithms that appear in a form as a mathematical expression, the ends can be ignored in the representation. We can let e' approximate infinity, the data stream in (3) becomes

$$\boldsymbol{p}_s \frac{1}{1 - (\alpha^{\Delta i} \beta^{\Delta j} \gamma^{\Delta k} \tau)}.$$

Wavefronts of a bundle of data streams moving in a uniform manner (with the same flow velocity) can be represented in an ensemble by

$$W = \sum F(\tau).$$

Different wavefronts (with different flow velocities) will interact and values they carry may change. This occurs only when the associated space positions and times are equal. A systolic algorithm is described as a collection of wavefronts of data streams in the data flow representation. Each data stream is characterized algebraically by three parameters: the flow velocity, the location of source, and the location of end.

B. Derivation of Flow Velocity

For a given iterative algorithm, the source and end of every data stream can be obtained directly from the description of the locally recursive and single assignment form. The key point to have a complete data flow representation is to derive the flow velocity. In the following theorem, some restrictions are given.

Theorem 1: Given the dependence vector $\vec{d}$ of a variable and a timing schedule Π of index space, the flow velocity of the data stream with this variable must satisfy the following equations:

$$\vec{V} = c \cdot \vec{d}, \qquad c \leq 1. \tag{4}$$

$$\Pi \cdot \vec{V} = 1. \tag{5}$$

It states that in (4) the flow velocity is a multiple of the dependence vector of the data stream, and this multiple is not larger than one, and in (5) the flow velocity is a reciprocal of the schedule.

Proof: (In the Appendix.)

We use the matrix multiplication as an example. Multiplying an N_i by N_k matrix A and an N_k by N_j matrix B to get an N_i by N_j product matrix C is mathematically as follows

$$c_{ij} = \sum_{k=0}^{N_k - 1} a_{ik} b_{kj}$$

where i, j are integers between 0 and $N_i - 1, N_j - 1$ inclusive, respectively. Rewrite this expression to be in a locally recursive and single assignment form as follows [5]:

$$c(i,j,k+1) = c(i,j,k) + a(i,j,k) \cdot b(i,j,k)$$
$$a(i,j+1,k) = a(i,j,k)$$
$$b(i+1,j,k) = a(i,j,k)$$

with inputs initially

$$a(i,0,k) = a_{ik}$$
$$b(0,j,k) = b_{kj}$$
$$c(i,j,0) = 0$$

and desired outputs

$$c_{ij} = c(i,j,N_k).$$

The dependence graph is shown in Fig. 1 with $N_i = N_j = N_k = 3$. The data dependence vectors for matrices A, B, and C (denoted as $\vec{d}_A$, $\vec{d}_B$, and $\vec{d}_C$) can easily be seen as

$$\vec{d}_A = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \vec{d}_B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad \vec{d}_C = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Choose schedule space $\Pi = [1\ 1\ 1]$, the available flow velocities for data streams of wavefronts A, B, and C are

$$\vec{V}_A = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \vec{V}_B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{and} \quad \vec{V}_C = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

In an algebraic representation, the wavefronts of data streams are

$$\begin{aligned} \boldsymbol{A} &= \sum_{i,k} \boldsymbol{A}(0) \sum_{t=0} \beta^t \tau^t \\ &= \sum_{i,k} a_{ik} \boldsymbol{X}^i \boldsymbol{Y}^0 \boldsymbol{Z}^k \tau^{i+k} \sum_{t=0} \beta^t \tau^t \\ &= \sum_{i,k} \sum_{t=(i+k)} a_{ik} \boldsymbol{X}^i \boldsymbol{Y}^{t-(i+k)} \boldsymbol{Z}^k \tau^t. \end{aligned}$$

$$\begin{aligned} \boldsymbol{B} &= \sum_{k,j} \boldsymbol{B}(0) \sum_{t=0} \alpha^t \tau^t \\ &= \sum_{k,j} b_{kj} \boldsymbol{X}^0 \boldsymbol{Y}^j \boldsymbol{Z}^k \tau^{k+j} \sum_{t=0} \alpha^t \tau^t \\ &= \sum_{k,j} \sum_{t=(k+j)} b_{kj} \boldsymbol{X}^{t-(k+j)} \boldsymbol{Y}^j \boldsymbol{Z}^k \tau^t. \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{C} &= \sum_{i,j} \boldsymbol{C}(0) \sum_{t=0} \gamma^t \tau^t \\ &= \sum_{i,j} c_{ij} \boldsymbol{X}^i \boldsymbol{Y}^j \boldsymbol{Z}^0 \tau^{i+j} \sum_{t=0} \gamma^t \tau^t \\ &= \sum_{i,j} \sum_{t=(i+j)} c_{ij} \boldsymbol{X}^i \boldsymbol{Y}^j \boldsymbol{Z}^{t-(i+j)} \tau^t. \end{aligned}$$

Assume that these three bundles of data streams meet at time t, the space position they meet are derived by equating the locations of data items in $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$. We have the space–time correlation

$$i + j + k = t$$

which is exactly the schedule of each point in the index space.

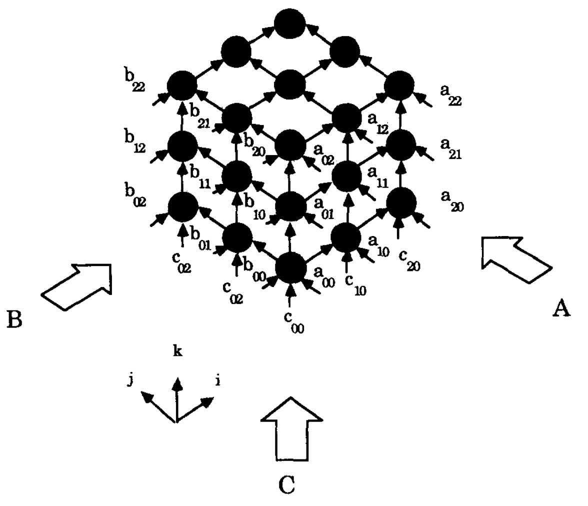

Fig. 1. The dependence graph of the matrix multiplication ($N_i = N_j = N_k = 3$) denoting three bundles of data streams.

III. Coalescent Systolic Array Systems

There are generally two ways of data input to a systolic array, either supplied by a host computer such as those in attached subsystems, or directly by other subsystems. In the latter, the form is like a chained pipeline used in vector computers. Chaining is a link process that occurs when results obtained from one pipeline unit are directly fed into the operand registers of another unit. In other words, intermediate results do not have to be retrieved in the host memory unit as buffering. However, the interface between the units is an issue [1]. Using the language of data flow representation, the supplied bundle of data streams must match exactly in space and time the demanded bundle of data streams. The relationship between the two bundles of data streams can be describing formally by the following equation:

$$\boldsymbol{p}_{s(\text{dem})} \cdot \tau^{s(\text{dem}) - e(\text{sup})} = \boldsymbol{p}_{e(\text{sup})}. \tag{6}$$

The $s(\text{dem}) - e(\text{sup})$ is the time interval between the end of supplier and the source of demander.

Consider two iterative algorithms. One is for convolution (or equivalently, the finite impulse response (FIR) filtering) of two sequences of $\boldsymbol{d} = [d_k]$ and weighting coefficients $\boldsymbol{g} = (g_0, g_1, \cdots, g_{N-1})$:

$$b_i = \sum_{k=0}^{N-1} g_{i-k} d_k, \qquad 0 \le i \le 2N - 1. \tag{7}$$

The other is for the matrix–vector multiplication of matrix $\boldsymbol{A} = (a_{ik})$ and vector $\boldsymbol{b} = [b_k]$ or (or equivalently, discrete Fourier transformation (DFT) [5]):

$$c_i = \sum_{k=0}^{N_k} a_{ik} b_k, \qquad 0 \le i \le N_i - 1. \tag{8}$$

Their dependence graphs are shown in Fig. 2 and Fig. 3. We transform them, respectively, such that the weighting coefficients of the convolution are stored in the PE's, and the resulting vector $\boldsymbol{c} = [c_i]$ of matrix–vector multiplication is stored in the PE's. The algebraic representations of the two algorithms will be as follows: (We use the apostrophes to denote the data streams of the arrays.)

for (7) $$\boldsymbol{d}' = \sum_k d_k \boldsymbol{X}^0 \boldsymbol{Y}^0 \tau^{2k} \sum_{t=0}^{N-1} \alpha^{-t} \tau^t.$$

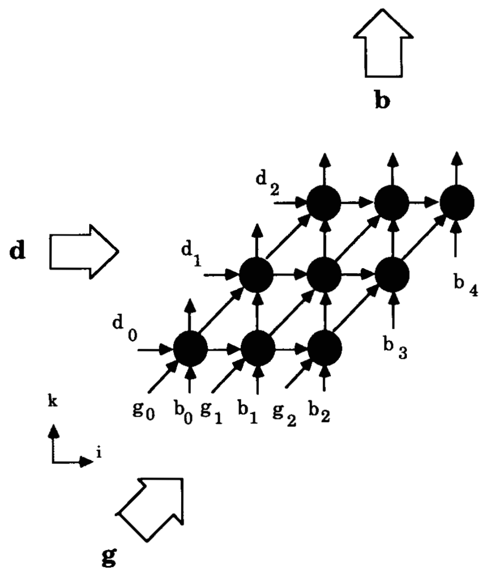

Fig. 2. The dependence graph of the convolution algorithm denoting three bundles of data streams.

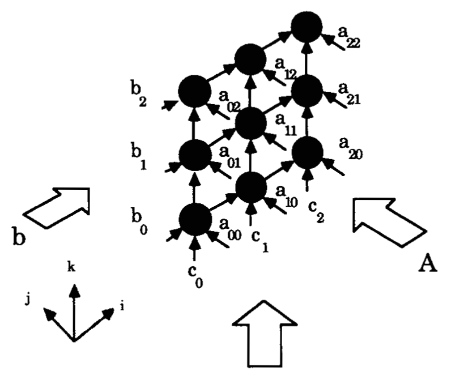

Fig. 3. The dependence graph of the matrix–vector multiplication.

$$g' = \sum_i g_i \boldsymbol{X}^{-i} \boldsymbol{Y}^0 \tau^i \sum_{t=0}^{N-1} \tau^t.$$

$$\boldsymbol{b}' = \sum_i b_i \boldsymbol{X}^{-(N-1)} \boldsymbol{Y}^0 \tau^{2i-(N-1)} \sum_{t=0}^{N-1} \alpha^t \tau^t;$$

and for (9)

$$\boldsymbol{A}' = \sum_i a_{ik} \boldsymbol{X}^i \boldsymbol{Y}^0 \tau^{i+k} \sum_{t=0} \beta^t \tau^t.$$

$$\boldsymbol{b}' = \sum_k b_k \boldsymbol{X}^0 \boldsymbol{Y}^0 \tau^k \sum_{t=0} \alpha^t \tau^t.$$

$$\boldsymbol{c}' = \sum_i c_i \boldsymbol{X}^i \boldsymbol{Y}^0 \tau^i \sum_{t=0} \tau^t. \tag{9}$$

All of $\boldsymbol{d}'$, $\boldsymbol{g}'$, $\boldsymbol{b}'$, $\boldsymbol{A}'$, and $\boldsymbol{c}'$ are wavefronts carrying data items of $\boldsymbol{d}$, $\boldsymbol{g}$, $\boldsymbol{b}$, $\boldsymbol{A}$, and $\boldsymbol{c}$, respectively. The systolic arrays are shown in Fig. 4 and Fig. 5. Notice that the projection directions (i.e., the processor spaces) we choose for each dependence graph are different. Now the output $\boldsymbol{b}'$ of convolution is the input vector of matrix–vector multiplication, and the two arrays are to be concatenated in a chain

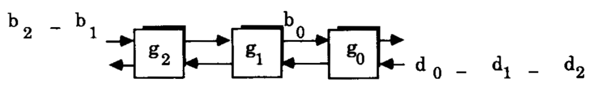

Fig. 4. The systolic array executing the convolution algorithm. The weighting coefficients are stored in the PE's.

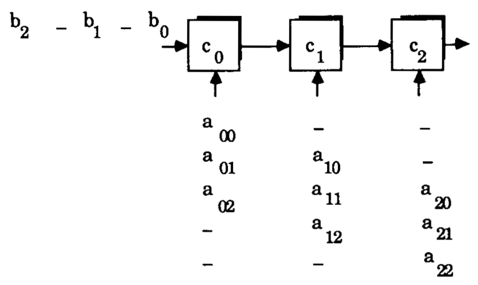

Fig. 5. The systolic array executing the matrix–vector multiplication.

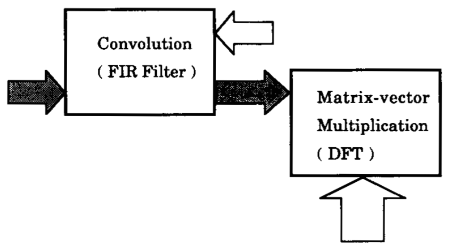

Fig. 6. Concatenation of systolic arrays. The output result of convolution is directed to the input operand of matrix–vector multiplication.

as shown in Fig. 6. However, one may see in Fig. 4 and Fig. 5, the intervals between the two arrays are different. The supplying end at $t = N - 1$ is $b_k \boldsymbol{X}^0 \boldsymbol{Y}^0 \tau^{2k}$, but the demanding source at $t = 0$ is $b_k \boldsymbol{X}^0 \boldsymbol{Y}^0 \tau^k$. According to (6), τ^k is added to adjust the interval of data items and τ^{N-1} is added to delay the array $N - 1$ time steps until the pipelining operand arrives. Equation (9) becomes

$$\boldsymbol{A}' = \sum_i a_{ik} \boldsymbol{X}^i \boldsymbol{Y}^0 \tau^{i+2k+N-1} \sum_{t=0} \beta^t \tau^t.$$

$$\boldsymbol{b}' = \sum_k b_k \boldsymbol{X}^0 \boldsymbol{Y}^0 \tau^{2k+N-1} \sum_{t=0} \alpha^t \tau^t.$$

$$\boldsymbol{c}' = \sum_i c_i \boldsymbol{X}^i \boldsymbol{Y}^0 \tau^{i+N-1} \sum_{t=0} \tau^t.$$

The interval is expanded, and thus matches the data stream from the convolution array (shown in Fig. 7). The coalescent system of systolic arrays seems more efficient for computations when chaining.

IV. Conclusion

The algebraic representation proposed in this short paper describes an iterative algorithm as bundles of data streams with different velocities. Each data stream consists of the data item, its source, and end. The only condition for chaining systolic arrays is to satisfy the interfacing of flow velocities of data items along with correct positions of demanding sources and supplying ends. By this means, several array subsystems could be easily coalesced.

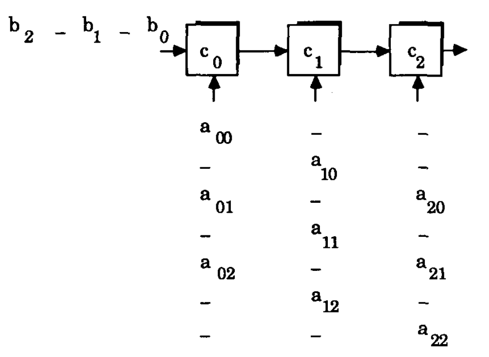
b_2 - b_1 - b_0
c_0
c_1
c_2
a_{00} - a_{01} - a_{02} - -
- a_{10} - a_{11} - a_{12} -
- - a_{20} - a_{21} - a_{22}

Fig. 7. The systolic array executing the matrix–vector multiplication by expanding the interval of the data items.

Appendix

Proof of Theorem 1: 1) The distance from the position at time t to the source is assumed to take q time steps for the data item with flow velocity $\vec{V}$, that is,

$$\boldsymbol{p}_t - \boldsymbol{p}_s = q\vec{V}.$$

or

$$\begin{bmatrix} i(t) \\ j(t) \\ k(t) \end{bmatrix} - \begin{bmatrix} i(s) \\ j(s) \\ k(s) \end{bmatrix} = q\vec{V}.$$

but the position vector $\boldsymbol{p}_t$ on the dependence graph is a multiple of the dependence vector plus the source, then

$$\left(r\vec{d} + \begin{bmatrix} i(s) \\ j(s) \\ k(s) \end{bmatrix} \right) - \begin{bmatrix} i(s) \\ j(s) \\ k(s) \end{bmatrix} = q\vec{V}.$$

Letting $c = r/q$ gives (4).

2) Measuring the time to go through the distance by the timing schedule Π, the time needed is q in 1), then by the above

$$q = \Pi(r\vec{d}) = \Pi(q\vec{V}) = q(\Pi \cdot \vec{V}).$$

and gives (5). By (4), substitute $\vec{d}$ with $c^{-1}\vec{V}$ into (2), it becomes

$$\Pi c^{-1}\vec{V} \geq 1.$$

Since $\Pi \cdot \vec{V} = 1$, $c \leq 1$. Q.E.D.

Acknowledgment

The authors are grateful to C.-M. Liu for his delightful help and the anonymous referees for their constructive suggestions.

References

[1] S. Horiike, S. Nishida, and T. Sakaguchi, "Systematic design of systolic arrays using mapping algorithm," in *Proc. IEEE Int. Symp. Circuit Syst.*, 1988, pp. 2505–2508.
[2] H. V. Jagadish and T. Kailath, "A family of new efficient arrays for matrix multiplication," *IEEE Trans. Comput.*, vol. 38, pp. 149–155, Jan. 1989.
[3] H. V. Jagadish, S. K. Rao, and T. Kailath, "Array architectures for iterative algorithms," *Proc. IEEE*, vol. 75, pp. 1304–1321, Sept. 1987.
[4] R. M. Karp, R. E. Miller, and S. Winograd, "The organization of computers for uniform recurrence equations," *J. ACM*, vol. 14, pp. 563–590, July 1967.
[5] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," in *Introduction to VLSI Systems*, C. Mead and L. Conway, Eds. Reading, MA: Addison-Wesley, 1980.
[6] C. L. Liu, *Introduction to Combinatorial Mathematics*, New York: McGraw-Hill, 1968.
[7] C. M. Liu and C. W. Jen, "Design of the algorithm-based fault-tolerant VLSI array processor," *IEE Proc.*, vol. 136, pt. E, pp. 539–547, Nov. 1989.
[8] D. I. Moldovan, "On the analysis and synthesis for VLSI algorithms," *IEEE Trans. Comput.*, vol. C-31, pp. 1121–1126, Nov. 1982.
[9] ——, "On the design of algorithms for VLSI systolic arrays," *Proc. IEEE*, vol. 71, pp. 113–120, Jan. 1983.
[10] P. Quinton, "Synthesizing systolic arrays using DIASTOL," in *Proc. Int. Workshop Systolic Arrays*, Univ. Oxford, July 1986, pp. 4.1–4.12.
[11] S. K. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor array," *Proc. IEEE*, vol. 76, pp. 259–269, Mar. 1988.
[12] J. C. Tsay and Y. C. Hou, "On equivalent systolic designs of matrix multiplication and its algebraic representation," in *Proc. Int. Comput. Symp.*, Taiwan, Dec. 1988, pp. 247–252.
[13] ——, "Generating function and equivalent transformation for systolic arrays," *Parallel Comput.*, vol. 10, pp. 347–356, May 1989.
[14] Y. Yaacoby and P. R. Cappello, "Scheduling a system of affine recurrence equations onto a systolic array," in *Proc. Int. Conf. Systolic Arrays*, 1988, pp. 373–382.

Some Combinatorial Aspects of Parallel Algorithm Design for Matrix Multiplication

Jong-Chuang Tsay and Sy Yuan

***Abstract*—In this paper, some combinatorial characteristics of matrix multiplication on regular two-dimensional arrays are studied. From the studies, we are able to design many efficient varieties of the cylindrical array and the two-layered mesh array for matrix multiplication. To design a cylindrical array for matrix multiplication, a systematic design procedure is proposed. In this design procedure, Latin square (a special type of matrix) plays an important role. To design a two-layered mesh array, we find that there is a transformation procedure to transform a cylindrical array to a two-layered mesh array.**

***Index Terms*—Cylindrical array, Latin square, matrix multiplication, parallel algorithm design, systolic array, two-layered mesh array.**

I. Introduction

Systolic array, as defined by Kung [3], is a synchronous operating parallel processing array consisting of simple processing elements interconnected in a local and regular manner. Systolic array has an unnoticed characteristic—its architecture should be planar, in other words, the communication lines among the processing elements should not cross over each other. If we relax this restriction, it is possible to obtain more efficient designs than systolic designs for a certain class of problems. Usually for this class of problems, operators used in their algorithm satisfy the associative and commutative properties. Typical examples in this class are matrix multiplication

Manuscript received April 28, 1989; revised November 11, 1990.
J.-C. Tsay is with the Institute of Computer Science and Information Engineering, College of Engineering, National Chiao Tung University, Hsinchu, Taiwan 30049, Republic of China.
S. Yuan is with the Advanced Technology Center of Computer and Communication Research Laboratories, Industrial Technology Research Institute, Chutung, Hsinchu, Taiwan 31015, Republic of China.
IEEE Log Number 9102596.