Optimal Design of Lower Dimensional Processor Arrays for Uniform Recurrences[†]

Kumar N. Ganapathy and Benjamin W. Wah
Coordinated Science Laboratory,
1101, West Springfield Avenue,
Urbana, IL 61801.
e-mail: kumar@aquinas.csl.uiuc.edu

Abstract

In this paper we present a parameter-based approach for synthesizing systolic architectures from uniform recurrence equations. The scheme presented is a generalization of the Parameter Method proposed by Li and Wah [1]. The approach synthesizes optimal arrays of any lower dimension from a general uniform recurrence description of the problem. In other previous attempts for mapping uniform recurrences into lower-dimensional arrays [2][3], optimality of the resulting designs is not guaranteed. As an illustration of the technique, optimal linear arrays for matrix multiplication are given. Due to space limitation, proofs to theorems in this paper are not shown. In a companion paper [4], we present details of the proof of Theorems 1 and 3 and a different approach for defining the constraints shown in Theorem 2. A detailed design for solving path-finding problems is also presented there.

1 Introduction

In synthesizing systolic architectures for algorithms, most existing methods are restricted to the case where n-dimensional algorithms are mapped into $(n-1)$-dimensional processor arrays. This paper presents a systematic method of generating optimal lower-dimensional processor arrays for general n-dimensional recurrences. The approach presented is a generalization of the Parameter Method proposed by Li and Wah[1].

Throughout this paper, matrices are denoted by boldface capital letters, vectors are represented with an overbar, and scalars correspond to lower-case letters. The calligraphic typeface is used to denote recurrence variables. A lower-case letter with a hat is used to represent subscript-access functions for input matrices. The transpose of a vector or matrix is denoted with a superscript letter t, and the rank of matrix M is denoted by $rank(\mathrm{M})$.

[†]Research Supported by National Science Foundation, grant MIP 88-10584 and Joint Services Electronics Program contract N00014-90-J-1270, and an IBM fellowship grant.
Proceedings International Conference on Application-Specific Array Processors, IEEE, August 1992.

0-8186-2967-3/92 $03.00 © 1992 IEEE

In mapping algorithms to processor arrays, the representation of the algorithm is often in the form of a recurrence equation. Mathematically, a recurrence over domain $\mathcal{D}$ is given as

$$\mathcal{Z}(\vec{p}) = \phi[\mathcal{Z}(\vec{q}_1), \mathcal{Z}(\vec{q}_2), \ldots, \mathcal{Z}(\vec{q}_k), \psi(\vec{p})] \quad (1)$$

where $\vec{p}, \vec{q}_i \in \mathcal{D}$, ϕ is a single-valued function strictly dependent on each of its arguments, and ψ represents the input. A recurrence equation is called *uniform* if $\vec{q}_i = \vec{p} + \vec{d}_i$, for $i = 1 \ldots m$, where $\vec{d}_i$'s are constant n-dimensional vectors independent of $\vec{p}$ and $\vec{q}_i$. Matrix multiplication of two matrices, $\mathbf{X}$ and $\mathbf{Y}$, is an example of uniform recurrence and is represented as

$$\mathcal{Z}(i,j,k) = \mathcal{Z}(i,j,k-1) + \mathbf{X}(i,k) \cdot \mathbf{Y}(k,j) \quad (2)$$

$$\mathcal{Z}(i,j,0) = \mathbf{Z}(i,j) \quad (3)$$

In this paper, we are concerned only with algorithms represented as *uniform recurrence equations.*

In the last decade, a number of systematic techniques have been proposed to generate systolic architectures. The Dependency Method of Moldovan and Fortes [5] laid the foundation for a number of the systematic methods of array synthesis from uniform recurrences. A critical overview of these methods can be found in the reference [6].

In the Dependency Method (denoted as DM), a feasible design is obtained by a linear transformation, represented as an $n \times n$ matrix $\mathbf{T}$ ($\in Z^{n \times n}$) of the index space. Thus, $\mathbf{T} = [\vec{\pi}\ \mathbf{S}]^t$, where $\vec{\pi}$ is $1 \times n$ schedule or time vector and $\mathbf{S}$ is the processor allocation matrix. The design of a systolic array is then equivalent to determining the n^2 parameters of the transformation matrix, $\mathbf{T}$.

Another approach to array synthesis was the *Parameter Method*, originally proposed by Li and Wah [1] for a special case of uniform recurrences. They had considered specific 3-D (*resp.* 2-D) recurrences, *i.e.,* recurrences whose domain or index space is 3-D (*resp.* 2-D) and synthesized 2-D (*resp.* 1-D) arrays to solve them. In this paper, we present a generalization of the Parameter Method, called the *Generalized Parameter Method* (*GPM*), for synthesizing systolic structures for uniform recurrences. The recurrence model considered is a general n-dimensional uniform recurrence instead of a specific 3-D one. In addition, the scheme attempts to synthesize arrays of any lower dimension m ($m < n$) instead of the usual $n-1$ dimensional array.

There have been several earlier attempts to map algorithms onto lower dimensional arrays [7][2][8]. Important steps toward a formal solution were made by Lee and Kedem [7]. The concept of data-link collisions (that is, two data tokens contending for the same link simultaneously) and conditions to avoid them were presented. However, detection of computational conflicts (that is, two computations scheduled to execute simultaneously in the same processor) is by analysis of all the computations in the domain of the recurrence. Although they had given necessary and sufficient conditions for the feasibility of the design, no systematic procedure for finding the optimal design was presented.

Shang and Fortes [3] have developed closed form conditions for a mapping to be free of computational conflicts. They have considered data-link collisions only between active data[1] participating in the computations. Consequently, it is possible to have collisions between active and passive data or among passive data. Further, although a procedure for determining the optimal schedule vector is given, the allocation matrix **S** is chosen heuristically. The number of choices for matrix **S** could be very large or even infinite, making it difficult (or impossible) to enumerate over them. The resulting designs are, therefore, "optimal" only up to the choice of **S**.

In other relevant previous work, it has been shown [9] that there are precise relationships between the original restricted version of the Parameter Method (denoted as OPM) and DM. The DM with identity dependency matrix was shown to be *equivalent* to OPM. Therefore, the search procedure used in OPM was adopted in the DM, making the complexity of the modified DM also, polynomial in the size of the problem.

In summary, there are two broad approaches to designing systolic arrays: Dependency Method (DM) and Parameter Method (PM). Figure 1 presents a pictorial description of the two approaches for the general synthesis problem. The area to the left of the dashed line represents the problem of mapping n-dimensional recurrences to $(n-1)$-dimensional arrays. The dependence-based methods pose the problem as determining the matrix **T**, while a parametric representation is used in OPM. The solution method for optimal designs in the OPM [1] is by a polynomial-time search. In the dependency framework, two ways of finding the matrix **T** exist: DM-I and DM-II. In the first one (DM-I), the processor allocation matrix **S** is chosen heuristically, and the time vector $\vec{\pi}$ that satisfies the constraints is subsequently found [5]. The second way (DM-II) is to formulate an equivalent search procedure in the Dependency Method using the relationship established between DM and OPM. The key difference between DM-I and DM-II is that DM-II solves for **S** by enumerating over a sequence of time vectors $\vec{\pi}$ rather than solving for $\vec{\pi}$ assuming a given **S**. The area to the right of the dashed line represents the methods for mapping n-dimensional recurrences to lower-dimensional arrays. There are 3 possible solution methods: DM-I*, GPM, and DM-II*. In the first method DM-I* (extension to DM-I), additional conditions on matrix **T** are developed and solution is obtained along the lines of DM-I, where the **S** matrix is chosen heuristically and the optimal $\vec{\pi}$ vector is found [3, 7, 2]. However, the designs found are not guaranteed to be optimal. The second method (GPM), is an extension to OPM for mapping general recurrences to lower-dimensional arrays. It retains the search technique in OPM for finding the optimal solutions efficiently. A third method (DM-II*) could extend the equivalencies established between OPM and DM, and utilize them to develop a search-based technique along the lines of DM-II.

We have followed the second of the above three methods and developed the GPM,

[1]The lifetime of a data token in the array can be viewed as consisting of an active phase, where the token is involved in its chain of computations, and a passive phase, where the token either is moving from the input peripheral processor to become active, or is moving to an output peripheral processor after its active phase.

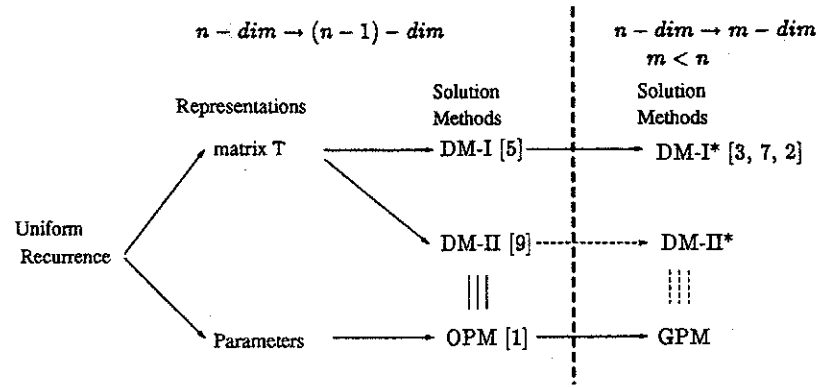

Figure 1 Approaches to the general array synthesis of uniform recurrences

which permits us to optimize a number of objectives efficiently. The remainder of this paper describes the details of the GPM.

2 Generalized Parameter Method

The structure of the n-dimensional uniform recurrence considered in this paper is

$$\mathcal{Z}(\vec{I}) = f\left[\mathcal{Z}(\vec{I}-\vec{d}_1), \mathcal{Z}(\vec{I}-\vec{d}_2), \ldots, \mathcal{Z}(\vec{I}-\vec{d}_q), \mathbf{X}_1(\hat{x}_1(\vec{I})), \ldots, \mathbf{X}_r(\hat{x}_r(\vec{I}))\right], \qquad (4)$$

where $\vec{I}$ denotes a point (n-dimensional vector) in the domain of the recurrence; $f()$ is a single-valued function (to be executed in each PE); $\vec{d}_i$, $1 \le i \le q$, is the i-th dependence vector associated with any point in the domain (for uniform recurrences); and $\mathbf{X}_j(\hat{x}_j(\vec{I}))$, $1 \le j \le r$, is the j-th distinct input that is needed in the computation of function $\mathcal{Z}$. $\hat{x}_j(\vec{I})$ is the linear subscript-access (indexing) function for the array indices of input $\mathbf{X}_j$. Note that without loss of generality the dependence vectors $\vec{d}_i$ can be taken to be distinct.

Pipelining If the subscript-access function of input $\mathbf{X}_j$ does not involve a particular index variable i_l, then each input token of input $\mathbf{X}_j$ is used at all index points that differ along the index i_l. Therefore, each token of the input has to be pipelined through all the points it is used. This pipeline vector can be automatically deduced from the equations [10]. After pipelining, a total of r dependencies may have to be introduced.

Example 1. Consider the following 3-dimensional recurrence that describes matrix multiplication with $n = 3$, $q = 3$, $r = 2$

$$\mathcal{Z}(i,j,k) = \mathcal{Z}(i,j,k-1) + \mathbf{X}(i,k)\,\mathbf{Y}(k,j) \tag{5}$$

The domain of the recurrence is a 3-dimensional cube of size $N \times N \times N$. After pipelining the inputs $\mathcal{X}$ and $\mathcal{Y}$, we get

$$\begin{aligned}\mathcal{Z}(i,j,k) &= \mathcal{Z}(i,j,k-1) + \mathcal{X}(i,j-1,k)\,\mathcal{Y}(i-1,j,k)\\ \mathcal{X}(i,0,k) &= \mathbf{X}(i,k)\\ \mathcal{Y}(0,j,k) &= \mathbf{Y}(k,j)\end{aligned}$$

2.1 Parameters

The crux of the Parameter Method is the characterization of the behavior, correctness, and performance of the systolic array by a set of vector and scalar parameters.

Parameter 1: Periods. Suppose the time at which a computation is performed is defined by function τ_c, the period of computation along dependence $\vec{d}_j$ is defined as

$$t_j = \tau_c(\mathcal{Z}(\vec{I} + \vec{d}_j)) - \tau_c(\mathcal{Z}(\vec{I})) \tag{6}$$

Parameter 2: Velocity. Velocity of a datum moving along direction $\vec{d}_j$ is defined as the directional distance passed during a clock cycle and is denoted as $\vec{V}_j$. Since PEs are at unit distance from its neighbors, and buffers (if present) must be equally spaced between PEs, the magnitude of the velocity must be a rational number of the form i/j where i, j are integers and $i \leq j$ (to prevent broadcasting).

Parameter 3: Spacing or Data Distribution. Consider the data tokens $\mathcal{X}_i(\vec{I} - \vec{d}_i)$ and $\mathcal{X}_i(\vec{I} - \vec{d}_j)$ moving through the points $(\vec{I} - k\,\vec{d}_i)$. The directional distance from token $\mathcal{X}_i(\vec{I} - \vec{d}_j)$ to $\mathcal{X}_i(\vec{I} - \vec{d}_i)$ is defined as spacing parameter $\vec{S}_{i,j}$. The notation $\vec{S}_{i,j}$ denotes that it is the j-th spacing parameter of the i-th variable. Since there are $q + r$ dependencies (and variables), there are $q + r - 1$ spacing parameters for each variable $(\vec{S}_{i,i} = 0)$.

In order to distinguish data moving in different dependence directions, Eq. 4 is rewritten in the following equivalent form.

$$\mathcal{Z}(\vec{I}) = f\left[\mathcal{Z}(\vec{I} - \vec{d}_1), \mathcal{A}_2(\vec{I} - \vec{d}_2), \mathcal{A}_3(\vec{I} - \vec{d}_3), \ldots, \mathcal{A}_q(\vec{I} - \vec{d}_q), \right. \tag{7}$$

$$\left. \mathcal{X}_1(\vec{I}), \ldots, \mathcal{X}_r(\vec{I})\right] \tag{8}$$

$$\mathcal{A}_i(\vec{I}) = \mathcal{Z}(\vec{I}), \quad i = 2, \ldots, q \tag{9}$$

For ease of notation, the variable set $\{\mathcal{Z}, \mathcal{A}_2, \mathcal{A}_3, \cdots, \mathcal{A}_q, \mathcal{X}_1, \mathcal{X}_2, \cdots, \mathcal{X}_r\}$ is ordered such that

$$i-th \text{ variable refers to } \begin{cases} \mathcal{Z} & if \;\; i = 1,\\ \mathcal{A}_i & if \;\; i = 2, 3, \ldots, q\\ \mathcal{X}_{i-q} & if \;\; i = q+1, q+2, \ldots, q+r \end{cases}$$

Example 2. For the 3-D recurrence in Eq. 5, the parameters t_1, t_2, t_3 are the periods, $\vec{V}_1, \vec{V}_2, \vec{V}_3$ are the velocities, and $\vec{S}_{1,2}, \vec{S}_{1,3}, \vec{S}_{2,1}, \vec{S}_{2,3}, \vec{S}_{3,1}, \vec{S}_{3,2}$ are the six spacings.

$$\begin{aligned}&\vec{S}_{1,2} : \vec{\mathcal{P}}(\mathcal{B}_{k,j+1}|t) - \vec{\mathcal{P}}(\mathcal{B}_{k,j}|t) \quad \vec{S}_{1,3} : \vec{\mathcal{P}}(\mathcal{B}_{k+1,j}|t) - \vec{\mathcal{P}}(\mathcal{B}_{k,j}|t)\\ &\vec{S}_{2,1} : \vec{\mathcal{P}}(\mathcal{A}_{i+1,k}|t) - \vec{\mathcal{P}}(\mathcal{A}_{i,k}|t) \quad \vec{S}_{2,3} : \vec{\mathcal{P}}(\mathcal{A}_{i,k+1}|t) - \vec{\mathcal{P}}(\mathcal{A}_{i,k}|t)\\ &\vec{S}_{3,1} : \vec{\mathcal{P}}(\mathcal{C}_{i+1,j}|t) - \vec{\mathcal{P}}(\mathcal{C}_{i,j}|t) \quad \vec{S}_{3,2} : \vec{\mathcal{P}}(\mathcal{C}_{i,j+1}|t) - \vec{\mathcal{P}}(\mathcal{C}_{i,j}|t),\end{aligned}$$

where $\vec{\mathcal{P}}(x|t)$ is a function that gives the location or position of data token x at time t. Note that even though, $\vec{\mathcal{P}}(x|t)$ is a function of time, the spacings are independent of time. ■

The design of the systolic array is now reduced to an appropriate choice of the parameters. The following theorem, stated without proof, shows the relationships that must hold among the parameters for correct execution of the recurrence [4].

Theorem 1. *The parameters velocities, spacings, and periods must satisfy the following constraint equations for correct systolic processing of the general recurrence.*

$$\vec{V}_i t_i = \vec{V}_j t_i + \vec{S}_{j,i}, \quad i, j = 1, 2, \cdots, q + r \tag{10}$$

2.2 Lower dimensional arrays

Theorem 1 in the last section provides constraint equations based on the recurrence being mapped, irrespective of the dimension of the target systolic array. The key observation is that the systolic constraint equations are vector equations. If the target systolic array is m-dimensional, the velocity and spacing parameters are m-dimensional as well. Hence, the constraint equations must be solved in m-dimensions in order to get m-dimensional solutions.

The number of parameters, $(q + r)(q + r + 1)$, exceed the number of equations by $2(q + r)$. Hence, $2(q + r)$ periods and velocities are chosen to optimize a performance criterion, and $(q + r)(q + r - 1)$ spacings are determined from the constraint equations. For each variable, only $(g - 1)$ of the $(q + r - 1)$ spacings are independent, where g is the rank of the dependency matrix $\mathbf{D}$. Hence, the additional $(q + r - g)$ of the spacings must be consistent with the independent $(g - 1)$ spacings. Theorem 2 (stated without proof) captures this notion mathematically.

Let $\mathbf{N} = [\vec{\alpha}_1\ \vec{\alpha}_2\ \cdots\ \vec{\alpha}_{q+r-g}]$ be a $(q+r) \times (q+r-g)$ matrix, where $\vec{\alpha}_i$ are basis vectors of the Null Space of $\mathbf{D}$, *i.e.*,

$$\mathbf{D} \cdot \vec{\alpha}_i \ = \ 0, \quad 1 \leq i \leq (q+r-g) \tag{11}$$

Theorem 2. *The additional constraints on the spacings $\vec{S}_{i,j}$ are*

$$\mathbf{S}\,\mathbf{N} \ = \ 0 \tag{12}$$

where $\mathbf{N}$ is the matrix of basis vectors of Null Space of $\mathbf{D}$ (see Eq. 11)

Example 3. In the 3-D recurrence (Eq. 5), the dependence matrix $\mathbf{D}$ is an identity matrix. Therefore, the Null Space $\mathbf{N}$ is an all-zero matrix, and the additional constraints on spacings are trivially satisfied. ■

Since the goal is to generate any lower m-dimensional array ($m < n$), all the inputs have to be fed in m-dimensions. If $m \geq (g-1)$, the $(g-1)$ consistent spacings have to be augmented by arbitrarily choosing $(m-g)$ additional ones. If $m < (g-1)$, it is possible that the spacings are such that two elements of the input are sent into the same PE of the array simultaneously, and travel together throughout the execution (since they have the same velocity). This is referred to as a *data conflict*. Data conflicts are not permissible within the framework, as extra control bits that travel with the two tokens are necessary to determine which of the two inputs is to be used in the computation. In addition, the two tokens also contend for the data links as they move through the array. Theorem 3 below expresses this notion mathematically.

Consider the spacings of variable i. Let $\mathbf{S}' = \left[\vec{S}_{i,1}, \vec{S}_{i,2}, \ldots, \vec{S}_{i,g-1}\right]^t$, where $\vec{S}_{i,1}$, $\vec{S}_{i,2}$, $\cdots$, $\vec{S}_{i,g-1}$ are the $(g-1)$ consistent spacings. Let $L_k, U_k, k = 1, 2, \cdots, g-1$, be defined such that the position of all the tokens of the input matrix can be represented by $\sum_{k=1}^{g-1} \beta_k \vec{S}_{i,k}$, $L_k \leq \beta_k \leq U_k$.

Theorem 3. *Data conflicts occur in the input matrix if and only if $\vec{\alpha} \cdot \mathbf{S}' = \vec{0}, \vec{\alpha} \neq \vec{0}$, where $\vec{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_{g-1}]^t$ and $\alpha_i \in [(L_i - U_i), \ldots, (L_i + U_i)], \forall i$ such that $1 \leq i \leq (g-1)$ [4].*

Example 4. For the recurrence in Eq. 5, let $\vec{S}_{2,1}, \vec{S}_{2,3}$ be the two spacings for input $\mathbf{X}$. Data conflicts occur in input $\mathbf{X}$ if and only if there exist $\alpha_1, \alpha_2 \neq 0$ and $-(n-1) \leq \alpha_1, \alpha_2 \leq (n-1)$ such that $\alpha_1 \vec{S}_{2,1} + \alpha_2 \vec{S}_{2,3} = 0$. For instance, if $L = 4$, $\vec{S}_{2,1} = 4$, and $\vec{S}_{2,3} = 2$, then $\alpha_1 = 1$ and $\alpha_2 = -2$ lead to data conflicts between tokens $\mathbf{X}(2,1), \mathbf{X}(1,3)$ and $\mathbf{X}(3,1), \mathbf{X}(2,3)$. ■

2.3 Design Methodology

The design of the optimal systolic array can be formulated as an optimization problem. The objective function to be optimized is expressed in terms of the parameters and problem size. To limit the complexity of the search, the space is ordered along increasing parameter values by choosing a monotonically increasing objective function. This results in the first feasible solution found being the optimal solution. Typical objective functions that are of interest to array designers are: completion time T, or number of processors #PE, or $\#PE \times T^2$, or $\#PE \times T$.

The condition that the systolic array has to be faster than a single processor forces the periods t_i to be chosen between 1 and t_i^{max} for some finite t_i^{max}. Since there is no broadcasting of data, $1 \leq \left|\vec{k}_i\right| = \left|\vec{V}_i t_i\right| \leq t_i^{max}$, where $\vec{k}_i$ represents the distance traveled by a token between computations. Hence, the size of the space of parameters is bounded above by $O(\prod_{i=1}^{q+r}(t_i^{max})^2)$. The enumeration procedure for minimizing completion time is presented below.

1. Compute the upper bounds $t_i^{max}, 1 \leq i \leq (q+r-1)$.
2. Find values of periods that minimize T.
3. Choose the distances $\left|\vec{k}_i\right|$ as unity and compute the velocities.
4. Solve for the spacing parameters from the constraint equations.
5. Check that the spacings satisfy the constraints $\mathbf{S}\,\mathbf{N} = 0$ (Theorem 2).
6. Check for data conflicts among the spacing parameters (Theorem 3).
7. If no feasible solution is found, increment one of $\left|\vec{k}_i\right|$ and repeat steps 4 and 5 until all $\left|\vec{k}_i\right|$ equal t_i
8. If there is no feasible solution still, find another set of periods that increase the completion time by the lowest possible value. Go to step 3.

3 Example: Matrix Multiplication

Since the running example used in the discussion of the GPM in Section 2 is matrix multiplication, recurrence in pipelined form, its parameters, and constraint equations have been given. In this section, we present optimal linear arrays for matrix multiplication.

The performance objectives of the target design are related to the parameters as follows.

Lemma 1. *For multiplying two $L \times L$ matrices using a linear array, T, the completion time, and $\#PE$, the number of PEs, are given by*

$$\begin{aligned} T &= 1 + (L-1)(t_1 + t_2 + t_3) \\ \#PE &= 1 + (L-1)\left(\left|\vec{k}_1\right| + \left|\vec{k}_2\right| + \left|\vec{k}_3\right|\right) \end{aligned}$$

Table 1 Matrix Multiplication: Optimal linear-array synthesis for minimum completion time T or $\#PE \cdot T^2$. Run time is the time in seconds on a Sun IPC workstation using GPM. Results obtained by LK method are the same as those obtained by GPM minimizing $\#PE$ alone.

Size (L)	Optimal Time Design - GPM								Run Time	LK [7] Design		SF [3] Design	
	Periods			Distances			Min T	#PE		T	#PE	T	#PE
	t_1	t_2	t_3	$\|k_1\|$	$\|k_2\|$	$\|k_3\|$			(sun4)				
4	1	2	3	1	1	1	19	10	-	19	10	-	-
5	1	2	3	1	1	2	25	17	-	29	13	25	13
10	2	3	4	1	1	3	82	46	-	109	28	-	-
17	3	3	5	1	2	4	177	113	-	305	49	289	49
25	3	4	6	2	7	1	313	193	-	649	73	625	73
51	4	5	8	3	2	7	851	601	-	2651	151	2601	151
100	6	7	10	5	4	9	2278	1783	4	10099	298	-	-
201	8	9	14	7	7	13	6201	5401	34	40601	601	40401	601
501	12	13	23	11	10	21	24001	21001	528	251501	1501	251001	1501
800	15	16	29	14	13	27	47941	43147	1807	640799	2398	-	-
1000	16	17	33	15	14	32	65935	60940	3281	1000999	2998	-	-

Table 1 shows the optimal linear designs found by the enumeration procedure when the objective is to minimize completion time. LK is the linear-array design proposed by Lee and Kedem [7]. Their approach is similar to the Dependency Method outlined in Section 1. The schedule vector is $(1, 2, n-1)$, and the PE allocation matrix (vector in the case of a linear array) is $(1, 1, -1)$ for $n \times n$ matrices. The above design was shown to be asymptotically optimal in terms of T. Table 1 also shows the completion time and #PE of the LK design, and the performance of the design proposed by Shang and Fortes (SF) [3]. The schedule vector used in the mapping is $(1, L-1, 1)$ and the space or PE allocation matrix is $(1, 1, -1)$. The above solution is feasible only for odd L, and the designs for even values of L are not given [3].

From Table 1, we see that the LK design is about 10-15 times slower, for problem of size around 1000, than the best array found by GPM for minimizing completion time T. However, #PEs used in the LK design is smaller than that in the designs found by GPM. In fact, we found by our design method that the LK design is the best possible design for minimizing the #PE, as shown in Theorem 4 below.

If the objective is to minimize the number of PEs in the array, then the following theorem characterizes the optimal design.

Theorem 4. *The set of parameters $(t_1, t_2, t_3) = (1, 2, L-1)$ and $(\vec{k}_1, \vec{k}_2, \vec{k}_3) = (1, 1, -1)$ results in the linear array with the minimum number of PEs.*

Table 2 presents the results for the objective of minimizing processor-time product, $\#PE \cdot T$. The minimum-processor design is the best design for $\#PE \cdot T$ up to a problem size of 52. For sizes over that, the design that minimizes $\#PE \cdot T$ is different from the

Table 2 Matrix Multiplication: linear-array design for minimizing $\#PE \cdot T$. Run time is the CPU time in seconds on a Sun IPC workstation.

Size (L)	Optimal $\#PE \cdot T$ Design - GPM								
	Periods			Distances			T	#PE	Run Time
	t_1	t_2	t_3	$\|k_1\|$	$\|k_2\|$	$\|k_3\|$			
4	1	2	3	1	1	1	19	10	-
5	1	2	4	1	1	1	29	13	-
10	1	2	9	1	1	1	109	28	-
17	1	2	16	1	1	1	305	49	-
53	3	5	25	2	2	1	1717	261	65
100	3	5	49	2	2	1	5644	496	1189
201	4	5	66	3	3	1	15001	1401	26208

minimum-processor and minimum-time designs.

The search strategy for minimizing $\#PE \cdot T^2$ is to first find a design that minimizes the completion time. Let T_{min} and P_{max} be the completion time and $\#PE$ of the minimum-time design, respectively. Then, the search space (which is polynomial in L, the size of the problem) is searched for a design that minimizes the $PE \cdot T^2$ product with completion time between T_{min} and $\frac{T_{min}\sqrt{P_{max}}}{\sqrt{3L-2}}$. It was found that for the problem sizes in Table 1, the best $\#PE \cdot T^2$ design is identical to the minimum-time design given in Table 1.

The optimal linear array together with the data flows for $L = 4$ is depicted in Figure 2 and has parameters $(t_1, t_2, t_3) = (1, 2, 3)$ and $(\vec{k}_1, \vec{k}_2, \vec{k}_3) = (1, 1, 1)$. This design minimizes both the completion time and $\#PE$, and therefore, minimizes the $\#PE \cdot T^2$ product (or any objective of the form $\#PE^m \cdot T^n$, $m, n \geq 1$).

So far, we have presented ways for finding optimal solutions when the objective function is known precisely. But often in realistic situations an analytic objective function may not capture the needs of the designer. For instance, the designer might have bounds on the resources such as the number of processors or completion time. Hence, a possible objective could be to have as few PEs with the completion time lower than an upper bound T_{up}, or to minimize completion time with $\#PE < \#PE^{up}$. Plots similar to those in Figure 3 would be very helpful to the designer in designing realistic arrays.

Figure 3 shows how $\#PE$ varies with T for 2 sizes: L = 100 and 201. In each plot the y- and x-axes are scaled by $\#PE^{max}$ and T^{max}, respectively. This allows different problem sizes to be combined on the same figure, as the values are all between 0 and 1. Two kinds of plots are presented in the figure: "time" plots and "#PE" plots. The "time" plots, shown as stepped curves, depict the minimum number of PEs required for a given maximum completion time. Hence, a point (T_1, P_1) on these curves show that at

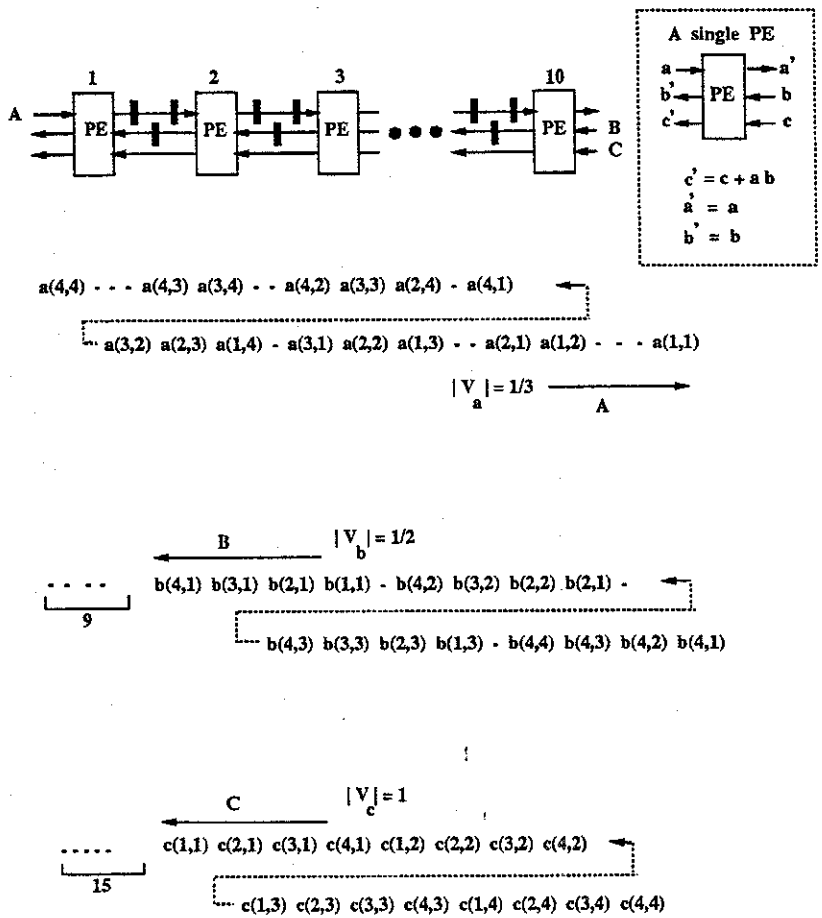

Figure 2 Linear array for multiplying two 4x4 matrices. The array is optimal for completion time, $\#PE$ and $\#PE \cdot T^2$ product

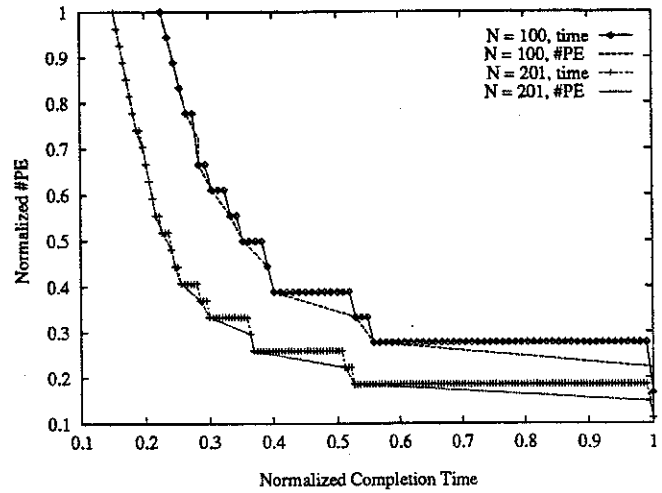

Figure 3 Performance tradeoffs: linear array for multiplication of two $L \times L$ matrices. The plots are given for problem sizes L = 100 and 201.

least P_1 PEs are needed so as to have completion time less than T_1. Similarly, a point (T_2, P_2) on the "#PE" curves denote any array with less than P_2 PEs will take more than T_2 steps to complete. Hence, given upper bounds on resources, the designer can determine from these plots which is the best design under the constraints.

The following observations can be made from Figure 3. The minimum-time (resp., minimum-processor) design uses a relatively high number of PEs (resp., time steps). Hence, the initial sharp decline in the plots could be exploited to get an alternative design that trades off performance for substantial cost reduction. For instance, the normalized $\#PE$ drops from 1 to 0.61, when the relative completion time increases from 0.225 to 0.303. Another observation from the plots is that the curves for larger values of L lie below those for smaller values of L. Hence, there is a substantial reduction for a relatively small deviation from the optimum. For large values of L, the tradeoffs become more attractive and important than the minimum-time or minimum-PE designs.

4 Final Remarks

This paper presents a parameter-based systematic search method for synthesizing systolic architectures from uniform recurrence equations. The design of the array is formulated as an optimization problem with the constraints of the search being the conditions for

correct systolic processing of the recurrence. The parameter space is bounded from above, resulting in a polynomial-time search. We present application of this technique by designing optimal linear arrays for 2-D matrix multiplication. We also show how to design processor arrays if the objective function cannot be established exactly as a function of the parameters.

In the future, we will apply the above search technique to map regular algorithms into existing vector and SIMD machines. We also plan to extend this method to cover recurrences with irregular or non-uniform dependencies.

References

[1] G.-J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Transactions on Computers*, vol. C-34, pp. 66–77, Jan. 1985.

[2] P.-Z. Lee and Z. M. Kedem, "Mapping nested loop algorithms into multidimensional systolic arrays," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 64–76, Jan. 1990.

[3] W. Shang and J. A. B. Fortes, "Time-optimal and conflict-free mappings of uniform dependence algorithms into lower dimensional processor arrays," *Proceedings of International Conference on Parallel Processing*, vol. 1, pp. 101–110, Pennsylvania State University Press, Aug. 1990.

[4] K. N. Ganapathy and B. W. Wah, "Synthesizing optimal lower dimensional processor arrays," *Proceedings of International Conference on Parallel Processing*, Pennsylvania State University Press, Aug. 1992.

[5] D. I. Moldovan, "On the analysis and synthesis of VLSI algorithms," *IEEE Transactions on Computers*, vol. C-31, pp. 1121–1126, Nov. 1982.

[6] J. A. B. Fortes, K.-S. Fu, and B. W. Wah, "Systematic design approaches for algorithmically specified systolic arrays," in *Computer Architecture : Concepts and Systems* (V. M. Milutinovic, ed.), pp. 454–494, North Holland, 1988.

[7] P.-Z. Lee and Z. M. Kedem, "Synthesizing linear array algorithms from nested For loop algorithms," *IEEE Transactions on Computers*, vol. C-37, pp. 1578–1597, Dec. 1988.

[8] V. P. Roychowdhury and T. Kailath, "Subspace scheduling and parallel implementation of non-systolic regular iterative algorithms," *Journal of VLSI Signal Processing*, vol. 1, 89.

[9] M. T. O'Keefe and J. A. B. Fortes, "A comparative study of two systematic design methodologies for systolic arrays," *Proceedings of International Conference on Parallel Processing*, pp. 672–675, Pennsylvania State University Press, Aug. 1986.

[10] S. V. Rajopadhye, "Synthesizing systolic arrays with control signals from recurrence equations," *Distributed Computing*, vol. 3, pp. 88–105, Springer Verlag, 1989.