Synthesizing Optimal Lower Dimensional Processor Arrays [†]

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. *Most existing methods for synthesizing systolic architectures can only map n-dimensional recurrences to $n-1$-dimensional arrays. In this paper, we generalize the parameter-based approach of Li and Wah [1] to map n-dimensional uniform recurrences to any k-dimensional processor arrays, where $k < n$. In our approach, operations of the target array are captured by a set of parameters, and constraints are derived to avoid computational conflicts and data collisions. We show that the optimal array for any objective function expressed in terms of these parameters can be found by a systematic enumeration over a polynomial search space. In contrast, previous attempts [2, 3] do not guarantee the optimality of the resulting designs. We illustrate our method with optimal single-pass linear arrays for re-indexed Warshall-Floyd path-finding algorithm. Finally, we show the application of GPM to practical situations characterized by restriction on resources, such as processors or completion time, rather than a well-defined objective.*

1 Introduction

Advances in VLSI technology in the last decade have led to development of application-specific high performance systems, called systolic arrays. A lot of research has been devoted to systematically synthesize systolic arrays from an algorithm description [4]. However, most of the techniques are restricted to mapping n-dimensional uniform recurrences to $(n-1)$-dimensional processor arrays. This paper proposes an efficient solution to the problem of synthesizing k-dimensional arrays for n-dimensional uniform recurrences ($k < n$).

The notation used in this paper is as follows: matrices are denoted by boldface capital letters; vectors with an overbar; recurrences variables by calligraphic letters; and scalars by normal lower and upper case letters. Subscript-access functions for matrices, are represented as lower-case letters with a hat. The rank of matrix $\mathbf{M}$ is denoted by $rank(\mathbf{M})$, and the transpose of a vector or matrix is written with a superscript letter t.

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

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

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

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

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

One of the earliest attempts to systematically generate systolic arrays was the dependency method of Moldovan and Fortes [5]. Most existing design methods are patterned after the dependency method. In the dependency method (denoted as DM), a feasible design is obtained by a linear transformation of the index space, represented as an $n \times n$ matrix $\mathbf{T}$. The matrix $\mathbf{T}$ is composed of a time vector $\vec{\pi}$ and a processor-allocation matrix $\mathbf{S}$. The design of a systolic array is then equivalent to determining the n^2 parameters of $\mathbf{T}$. This is an integer programming problem and could be of exponential complexity in the worst case.

An alternative and more efficient approach to array synthesis, called the *parameter method*, was proposed by Li and Wah [1] for a special class of uniform recurrences. Instead of the transformational approach (like the DM), they took an operational view of the systolic array. The systolic array, that executes the uniform recurrence has to be regular and, therefore, was characterized by a set of parameters. The design of an array is then reduced to an appropriate choice of the parameters. Using the parameter method (denoted as OPM), they presented optimal 2-D (*resp.* 1-D) systolic arrays for a specific class of 3-D (*resp.* 2-D) recurrences.

In this paper, we present a generalization of the parameter method, called the general parameter method (GPM), to synthesize systolic structures for uniform recurrences. In the GPM, we consider general n-dimensional uniform recurrences instead of specific 3-D ones. The GPM can be used to synthesize optimal arrays of any lower-dimension m (where $m < n$), instead of the usual $n-1$ dimensional array. The objective could be a function described in terms of the parameters, and a polynomial-time search is used to find the optimal array.

There have been several earlier attempts to map algorithms onto lower dimensional arrays [2, 6, 3, 7]. Lee and Kedem [2, 6] gave a set of necessary and sufficient conditions for the feasibility of a design and conditions to avoid data-link collisions (when two data tokens contend for the same link simultaneously). However, detection of computational conflicts (when two computations are scheduled to execute simultaneously in the same processor) is by analysis of all the computations in the domain of the recurrence. In addition, no systematic procedure to find the optimal designs were presented. Shang and Fortes [3]

[†]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 of International Conference on Parallel Processing, August 1992.

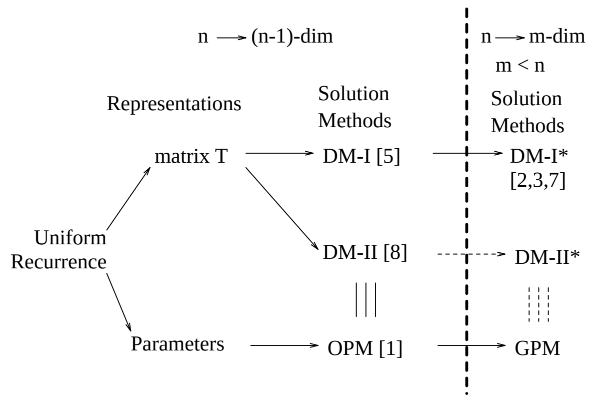

Figure 1: Approaches to the general array synthesis of uniform recurrences

developed a set of closed-form conditions for a design to be feasible (*i.e.,* free of computational conflicts). However, they consider data-link collisions only between active data[1] and not between active and passive data or among passive data. Although they present a procedure to determine the optimal schedule vector, the allocation matrix $\mathbf{S}$ is chosen heuristically. The number of choices for matrix $\mathbf{S}$ could be very large, making it difficult to enumerate over them.

In summary, Figure 1 shows two broad approaches for designing systolic arrays: Dependency Method (DM) and Parameter Method (PM). The area to the left of the dashed line represents methods that map n-dimensional recurrences to $(n-1)$-dimensional arrays. The dependence-based methods denote the mapping by a matrix $\mathbf{T}$, while OPM employs a parametric representation. The efficiency of OPM [1] stems from constraining the parameters, thereby limiting the search space to polynomial in the size of the problem. In the dependency framework, two ways of finding $\mathbf{T}$ exist: (i) DM-I, which chooses the processor-allocation matrix $\mathbf{S}$ heuristically and finds vector $\vec{\pi}$ [5], (ii) DM-II, which uses a search procedure similar to that in OPM based on the relationships between OPM and DM [8]. The DM with an identity dependency matrix was shown to be equivalent to OPM. The main difference is that DM-II enumerates over a sequence of time vectors $\vec{\pi}$ and solves for $\mathbf{S}$, rather than assuming a given $\mathbf{S}$ and finding a feasible $\vec{\pi}$.

The area to the right of the dashed line shows three methods of mapping n-dimensional recurrences to lower-dimensional arrays: The first, DM-I* extends DM-I with conditions on $\mathbf{T}$; $\mathbf{S}$ is still chosen heuristically, and $\vec{\pi}$ is found by enumeration [2, 3, 6]. The second, GPM, generalizes OPM to map general recurrences into lower-dimensional arrays; it retains OPM's search technique to find optimal solutions efficiently. A possible third, DM-II* could extend the equivalence between OPM and DM, by using a search-based technique similar to GPM.

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

2 Generalized Parameter Method

The n-dimensional recurrences considered in this work are represented as

$$\begin{aligned}\mathcal{Z}(\vec{I}) &= f\left[\mathcal{Z}(\vec{I}-\vec{d_1}),\mathcal{Z}(\vec{I}-\vec{d_2}),\ldots,\mathcal{Z}(\vec{I}-\vec{d_q}),\right.\\ &\qquad \left.\mathbf{X}_1(\hat{x}_1(\vec{I})),\ldots,\mathbf{X}_r(\hat{x}_r(\vec{I}))\right] \qquad (2)\end{aligned}$$

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 the function $\mathcal{Z}$. $\hat{x}_j(\vec{I})$ is the linear subscript-access function for the array indices of input $\mathbf{X}_j$. Without loss of generality, the dependence vectors $\vec{d_i}$ can be taken to be distinct. Let $\mathbf{D} = \left[\vec{d_1}\ \vec{d_2}\ \cdots\ \vec{d}_{q+r}\right]$ denote an $n \times (q+r)$ matrix of dependence vectors. Let g be the rank of matrix $\mathbf{D}$.

Pipelining. Let the linear subscript-access function of input $\mathbf{X}_j$ be represented as $\hat{x}_j(\vec{I}) = \mathbf{P}_j \cdot \vec{I} + \vec{Q}_j$. If the matrix $\mathbf{P}_j$ is not full rank, then each token $\mathbf{X}_j(\mathbf{P}_j \cdot \vec{I})$ is also used at other index points, $\vec{I_1} \ne \vec{I}$ (where $\mathbf{P}_j \cdot \vec{I_1} = \mathbf{P}_j \cdot \vec{I}$). We assume that broadcasting of an input token is not permissible and input tokens are not replicated. Therefore, each token of the input has to be pipelined through all the points it is used. A dependence vector is associated with each input such that each token moves along the dependence direction, and is sequenced through all the points. This pipeline vector can be automatically deduced from matrix $\mathbf{P}_j$ [9]. Let $\vec{d}_{q+j}$ be the dependence associated with input $\mathbf{X}_j$, Eq. 2 can be written as

$$\begin{aligned}\mathcal{Z}(\vec{I}) &= f\left[\mathcal{Z}(\vec{I}-\vec{d_1}),\mathcal{Z}(\vec{I}-\vec{d_2}),\ldots,\mathcal{Z}(\vec{I}-\vec{d_q}),\right.\\ &\qquad \left.\mathcal{X}_1(\vec{I}-\vec{d}_{q+1}),\ldots,\mathcal{X}_r(\vec{I}-\vec{d}_{q+r})\right] \qquad (3)\end{aligned}$$

where $\mathcal{X}_j$ is the recurrence variable associated with input matrix $\mathbf{X}_j$, $j = 1,\ldots,r$. Thus, a total r dependencies may have to be introduced to pipeline the r inputs. For points at (or close to) the boundary of the domain $\mathcal{D}$, the value of recurrence variable is given by the external input, as the dependence points are not present in the domain. Hence,

$$\mathcal{X}_j(\vec{I}) = \mathbf{X}_j(\hat{x}_j(I)), \quad \text{if } (\vec{I}-\vec{d}_{q+j}) \notin \mathcal{D}$$

Example 1. Consider a 3-dimensional recurrence with $n=3$, $q=3$, $r=2$.

$$\begin{aligned}\mathcal{Z}(k,i,j) &= \phi\left[\mathcal{Z}(k,i,j-1),\mathcal{Z}(k,i-1,j),\mathcal{Z}(k-1,\right.\\ &\qquad \left. i+1,j+1),\mathbf{X}_1(j,k+i-1),\mathbf{X}_2(k+j-1,i)\right] \ (4)\end{aligned}$$

The domain of the recurrence is a 3-dimensional cube of size $N \times N \times N$.

The subscript-access functions for the inputs represented as matrices are

$$\mathbf{P}_1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{P}_2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

where $\hat{x}_1(\vec{I}) = \mathbf{P}_1 \cdot \vec{I} + (0,-1)^t$ and $\hat{y}(\vec{I}) = \mathbf{P}_2 \cdot \vec{I} + (-1,0)^t$. The pipeline vectors for inputs $\mathbf{X}_1$ and $\mathbf{X}_2$ are $(1,-1,0)^t$

and $(1,0,-1)^t$, respectively. They are found as basis vectors of the Null Space of matrices $\mathbf{P}_1$ and $\mathbf{P}_2$. (Eq. 4) becomes

$$\begin{aligned}\mathcal{Z}(k,i,j) &= \phi\,[\mathcal{Z}(k,i,j-1),\mathcal{Z}(k,i-1,j),\mathcal{Z}(k-1,\\ &\quad i+1,j+1),\mathcal{X}_1(k-1,i+1,j),\mathcal{X}_2(k-1,i,j+1)]\,.\end{aligned}$$

For points at the boundary of the domain, $\mathcal{Z}(0,i+1,j+1) = \mathbf{Z}(i,j), \mathcal{Z}(k,0,j) = \mathcal{Z}(k,i,0) = 0, \mathcal{X}_1(0,i+1,j) = \mathbf{X}_1(j,i), \mathcal{X}_1(k-1,N+1,j) = \mathbf{X}_1(j,N+k-1), \mathcal{X}_2(0,i,j+1) = \mathbf{X}_2(j,i), \mathcal{X}_2(k-1,i,N+1) = \mathbf{X}_2(N+k-1,i)$, for all k, i, j such that $1 \le k,\, i,\, j \le N$. ■

The recurrence in Eq. 2 is rewritten in the following equivalent form to distinguish the data moving in different dependence directions.

$$\begin{aligned}\mathcal{Z}(\vec{I}) &= f\left[\mathcal{Z}(\vec{I}-\vec{d}_1),\mathcal{A}_2(\vec{I}-\vec{d}_2),\ldots,\mathcal{A}_q(\vec{I}-\vec{d}_q),\right.\\ &\quad \left.\mathcal{X}_1(\vec{I}),\ldots,\mathcal{X}_r(\vec{I})\right] &(5)\\ \mathcal{A}_i(\vec{I}) &= \mathcal{Z}(\vec{I}), \quad i = 2,\ldots,q, &(6)\end{aligned}$$

where variable $\mathcal{A}_i$ indicates the movement of $\mathcal{Z}$ along direction $\vec{d}_i$. With this form, each dependency is associated with exactly one variable.

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. When a uniform recurrence is executed in a systolic array the computations are periodic along each dependence direction. 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,

$$t_j = \tau_c(\vec{I}+\vec{d}_j) - \tau_c(\vec{I}).$$

Since there are $q+r$ dependencies, there are a total of $q+r$ periods.

Parameter 2: Velocity. Since computations are equally-spaced along each dependence direction, a data token moves the same distance between consecutive computations. Hence, velocity of the data remains constant and is chosen as a parameter. Velocity of a datum moving along direction $\vec{d}_j$ is defined as the directional distance passed during a clock cycle and is denoted by $\vec{V}_j$. 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 such that $i \le j$ (to prevent broadcasting). The number of velocity parameters is $q+r$, as there are $q+r$ dependencies (and variables) in the recurrence.

Parameter 3: Spacing or Data distribution. If the velocity of all tokens of an input is constant, the relative distance between two tokens of the variable remains unchanged throughout the entire computation.

Consider the data tokens $\mathcal{X}_i(\vec{I}-\vec{d}_i)$, $\mathcal{X}_i(\vec{I}-\vec{d}_j)$ moving through the points $(\vec{I}-k\vec{d}_i)$ and $(\vec{I}-k\vec{d}_j)$, $k = 0,\pm1,\pm2,\cdots$, respectively. 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 ($\vec{S}_{i,i} = 0$).

Example 2. The rewritten form of the 3-D recurrence in Eq. 4 is

$$\begin{aligned}\mathcal{Z}(k,i,j) &= [\mathcal{Z}(k,i,j-1),\mathcal{A}_2(k,i-1,j),\mathcal{A}_3(k-1,\\ &\quad i+1,j+1),\mathcal{X}_1(k-1,i+1,j),\mathcal{X}_2(k-1,i,j+1)]\\ \mathcal{A}_2(k,i,j) &= \mathcal{Z}(k,i,j)\\ \mathcal{A}_3(k,i,j) &= \mathcal{Z}(k,i,j)\end{aligned}$$

There are five periods : t_1,t_2,t_3,t_4,t_5, five velocities : $\vec{V}_1,\vec{V}_2,\vec{V}_3,\vec{V}_4,\vec{V}_5$, and 20 spacing parameters (4 for each variable). For instance, the spacings $\vec{S}_{3,1}$ and $\vec{S}_{3,2}$ are given as

$$\begin{aligned}\vec{S}_{3,1} &: \vec{\mathcal{P}}(\mathbf{Z}_{i,j+1}|t) - \vec{\mathcal{P}}(\mathbf{Z}_{i,j}|t),\\ \vec{S}_{3,2} &: \vec{\mathcal{P}}(\mathbf{Z}_{i+1,j}|t) - \vec{\mathcal{P}}(\mathbf{Z}_{i,j}|t),\end{aligned}$$

where $\vec{\mathcal{P}}(x|t)$ is a function that gives the location 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 shows the relationships that must hold among the parameters for correct execution of the recurrence. All theorems are presented without proofs due to space limitations.

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

$$\vec{V}_i\, t_i = \vec{V}_j\, t_i + \vec{S}_{j,i}, \quad i,\, j = 1,2,\cdots,q+r \qquad (7)$$

Let $\vec{T} = \{t_1,t_2,\cdots,t_{q+r}\}$ be a vector the periods, and $\mathbf{K} = \left[\vec{k}_1,\vec{k}_2,\cdots,\vec{k}_{q+r}\right]$ be a matrix of displacements, where $\vec{k}_i = \vec{V}_i\, t_i$ is the (vector) distance traversed by variable i in a single period. The displacement $\vec{k}_i$ is synonymous with velocity $\vec{V}_i$, because choosing one immediately determines the other. Hence, in searching for parameter values, we consider $\vec{k}_i$ and not $\vec{V}_i$.

Let $\mathbf{S} = \left[\vec{S}_{i,j}\right]$, $\quad i,\, j = 1,2,\cdots,q+r$, be a $(q+r)\times(q+r)$ matrix of these spacings such that the (i,j)-th element of the matrix is $\vec{S}_{i,j}$. Note by definition $\vec{S}_{i,i} = 0$. If g is the rank of matrix $\mathbf{D}$, the dimension of the Null Space of $\mathbf{D}$ is $q+r-g$ (as $\mathbf{D}$ has $q+r$ columns). 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 $\mathbf{D}$. Hence,

$$\mathbf{D}\cdot\vec{\alpha}_i = 0, \quad 1 \le i \le (q+r-g) \qquad (8)$$

The following theorem gives constraints on the periods and displacements that depend on the dependence structure of the recurrence.

Theorem 2. *The periods t_i and the displacements $\vec{k}_i$ are related as follows:*

1. $\vec{T} \cdot \mathbf{N} = 0$,
2. $\mathbf{K}\,\mathbf{N} = 0$,

where N is the matrix consisting of the basis vector of Null Space of $\mathbf{D}$.

The following corollary shows the constraints on spacings that follow from Theorem 1. In fact, it can be shown that the constraints in Theorem 1 and Corollary 1 are equivalent, *i.e.,* they can be derived from each other.

Corollary 1. *The spacing parameters $\vec{S}_{i,j}$ are constrained by the vector equations* $\mathbf{S}\,\mathbf{N} = 0$.

Example 3. For the recurrence in Eq. 4, the matrix $\mathbf{N}$ comprising of vectors in Null Space of $\mathbf{D}$ is

$$\mathbf{N} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \Longrightarrow \begin{array}{ll} t_4 = t_1 + t_3, & \vec{k}_4 = \vec{k}_1 + \vec{k}_3 \\ t_5 = t_2 + t_3, & \vec{k}_5 = \vec{k}_2 + \vec{k}_3 \end{array}$$

Consider the 4 spacings of 3-rd variable $\mathcal{A}_3$. From Corollary 1, $\vec{S}_{3,4} = \vec{S}_{3,1}$, and $\vec{S}_{3,5} = \vec{S}_{3,2}$ ($\vec{S}_{3,3} = 0$). Hence, only 2 of the 4 spacings are independent. ■

2.2 Lower dimensional arrays

The previous section presents the constraints that must be obeyed by the parameters of any feasible solution. An important observation is that these constraints are vector equations, and are independent of the dimensions of the target systolic array. Hence, the constraint equations can be solved in m-dimensions for an m-dimensional array ($m < n$) as the parameters are m-dimensional.

Theorem 1 provides $(q+r)(q+r-1)$ constraints and Theorem 2 imposes $2(q+r-g)$ constraints, giving a total of $(q+r)(q+r+1) - 2g$ constraints. The number of parameters is $(q+r)(q+r+1)$. Therefore, $2g$ of the parameters have to be chosen. The performance of the design is usually only a function of its periods and displacements. Hence, we can choose g of the periods and g of the displacements to optimize a given performance criterion, and determine the spacings from the constraint equations.

From the view of dependence-based methods, a mapping is feasible if and only if there are no computational conflicts or data-link collisions. Computational conflicts can be between two computations that have at least one input in common, or between two computations that have no inputs in common. Conflicts between two computations with common data tokens can be avoided by choosing the periods to be strictly positive ($t_i \geq 1$). If a conflicts occurs between two computations with no common data tokens, then two distinct tokens of each variable is present in the same processor. The colliding tokens of each variable must have been sent into a peripheral processor simultaneously, since all tokens of a variable move with the same velocity in our array model. These colliding tokens also contend for the same data links as they travel through the array. The condition of co-existing or colliding data tokens is termed *data conflict.* The presence of data conflicts imply the existence of computational conflicts, that in turn imply the presence of data-link collisions. Hence, a feasible design can only be obtained by avoiding data conflicts. Data conflicts also require the presence of extra control information in the processors to disambiguate between the conflicting tokens.

The trivial case of data conflicts is when a spacing parameter $\vec{S}_{i,j} = 0$ for some $j \neq i$. However, data conflicts can occur even if all spacings are non-zero in lower-dimensional array synthesis. For non-zero spacings, the conditions under which data conflicts can occur is expressed mathematically in Theorem 3.

Consider the spacing parameters of variable i. Let $\mathbf{S}' = \left[\vec{S}_{i,1}, \vec{S}_{i,2}, \ldots, \vec{S}_{i,g-1}\right]^t$, where $\vec{S}_{i,1}, \ldots, \vec{S}_{i,g-1}$ be the $g-1$ non-zero independent 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 $\vec{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_{g-1}]$ and $\alpha_i \in [(L_i - U_i), \ldots, (L_i + U_i)], \forall i$ such that $1 \leq i \leq g-1$.*

Example 4. For the recurrence in Eq. 4, let $\vec{S}_{3,1}, \vec{S}_{3,2}$ be the two independent spacings for variable $\mathcal{A}_3$. Data conflicts occur 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 $\alpha_1 \vec{S}_{3,1} + \alpha_2 \vec{S}_{3,2} = 0$. For instance, if $N = 4$, then $\vec{S}_{3,1} = 4$, $\vec{S}_{3,2} = 2$, $\alpha_1 = 1$, and $\alpha_2 = -2$ lead to data conflicts between tokens $\mathbf{Z}(2,1)$, $\mathbf{Z}(1,3)$ and $\mathbf{Z}(3,1)$, $\mathbf{Z}(2,3)$. ■

2.3 Design Method

The design of the systolic array is equivalent to an appropriate choice of the parameters that satisfy the constraints imposed by Theorems 1, 2 and 3. Since the target array is systolic, the displacements $\left|\vec{k}_i\right|$ should not exceed the periods t_i in order to prevent data broadcasting. In addition, the constraints $t_i \geq 1$, $i = 1, 2, \cdots q+r$ are necessary to enforce the dependencies in the recurrence.

The target systolic array could be designed in two possible ways. In the first way, the target array could be generated for a specific well-defined objective expressed in terms of the parameters and problem size. These objectives may be chosen to increase monotonically with the parameters. This choice orders the search space and results in the first feasible solution found being the optimal solution, thereby limiting the complexity of the search process. Typical monotonic objective functions of interest to array designers are the completion time, T_c, or the number of processors #PE. However, for other specific objectives, such as $\#PE \times T_c$ or $\#PE \times T_c^2$, the search space can still be limited by bounding the parameters from above. This is illustrated in Section 3 for the example on designing a array for solving the transitive closure.

Second, the designer might not be aware of the specific function to be optimized. He might be only able to impose additional constraints that bound the number of PEs or completion time from above. In these cases, the GPM can still be employed, as the method can incorporate additional constraints easily into the search procedure.

The condition that the systolic array has to be faster than a single processor bounds the periods t_i between 1 and t_i^{max} for some finite t_i^{max}. Since the magnitudes of periods and displacements (velocities) are chosen from a finite set, the optimization problem has finite complexity $O(\prod_{i=1}^{q+r}(t_i^{max})^2)$.

Next, we present the enumeration procedure to minimize the completion time of the array. This procedure can be modified easily if the objective is to minimize T_c for a given bound the number of PEs in the array. The constraint on $\#PE$ would be included in step 8 of the enumeration procedure given below by checking to see whether the bound has been exceeded when any one of the displacements is increased.

1. Choose g periods and g displacements to be unconstrained parameters.
2. Compute the upper bounds $t_i^{max}, 1 \leq i \leq g$.
3. Find values of g independent periods that minimize the completion time T_c.
4. Compute the values of the dependent periods and displacements using the conditions of Theorem 2.
5. If $t_i > t_i^{max}$ for some $i = 1, 2 \cdots, q + r$, exit (since it is faster to execute the recurrence on a single processor).
6. Choose the magnitude of displacements $\left|\vec{k}_i\right|$ as unity and solve for the spacing parameters from the equations of Theorem 1.
7. Check for data conflicts in the inputs using Theorem 3 on the spacing parameters.
8. If the solution is not feasible, then increment one of $\left|\vec{k}_i\right|$ and repeat steps 6 and 7 until all $\left|\vec{k}_i\right|$ equal t_i, else exit (the optimal design has been found).
9. 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 4.

The procedure to minimize $\#PE$ or minimize $\#PE$ for a given bound on T_c would be similar to one presented above.

3 Example: Path-Finding Problems

Path-finding problems are an important class of problems that occur in optimization. Typical examples include the computing the transitive closure and the shortest paths of a graph. Two-dimensional arrays for transitive closure have been presented before [10, 11, 12]. In this section we synthesize a one-pass linear systolic array for the Warshall-Floyd path-finding algorithm. The discussion below is with respect to the transitive closure problem.

The transitive closure problem is defined as follows. Compute the transitive closure $\mathbf{C}^+[i,j]$ of an n-node directed graph with an $n \times n$ Boolean adjacency matrix $\mathbf{C}[i,j]$, where $\mathbf{C}[i,j] = 1$ if there is an edge from vertex i to vertex j or $i = j$, and $\mathbf{C}[i,j] = 0$ otherwise. Since the dependence structure is irregular and difficult to map, S.Y. Kung *et al.*, have converted the transitive closure algorithm into a reindexed form and mapped it to 2-D spiral and orthogonal arrays [10]. After pipelining the variables, we obtain the following five dependence vectors.

1. $\vec{d}_1 = (0,0,1)^t$ for $(k,i,j)^t \leftarrow (k,i,j-1)^t, \;\; 2 \leq j \leq N$.
2. $\vec{d}_2 = (0,1,0)^t$ for $(k,i,j)^t \leftarrow (k,i-1,j)^t, \;\; 2 \leq i \leq N$
3. $\vec{d}_3 = (1,-1,-1)^t$ for $(k,i,j)^t \leftarrow (k-1,i+1,j+1)^t, \;\; 2 \leq k \leq N, \;\; 1 \leq i,j \leq N-1$.
4. $\vec{d}_4 = (1,-1,0)^t$ for $(k,i,n)^t \leftarrow (k-1,i+1,n)^t, \;\; 2 \leq k \leq N, \;\; 1 \leq i \leq N-1$.
5. $\vec{d}_5 = (1,0,-1)^t$ for $(k,n,j)^t \leftarrow (k-1,n,j+1)^t, \;\; 2 \leq k \leq N, \;\; 1 \leq j \leq N-1$,

where $\vec{I}_1 \leftarrow \vec{I}_2$ means that the data at point $\vec{I}_2$ is used at point $\vec{I}_1$. For nodes on the boundary of the dependence graph (G) where $i = N$ (*resp.* $j = N$), dependence $\vec{d}_4$ (*resp.* $\vec{d}_5$) is present instead of dependence $\vec{d}_3$. For other interior points, only 3 dependencies $\vec{d}_1, \vec{d}_2, \vec{d}_3$ exist.

The running example used in the discussion of the GPM is a recurrence with the 5 dependencies as listed above. The dependence graph of the recurrence used in the example (Eq. 4) is regular and homogeneous with 5 dependencies at each point. However, for transitive closure the dependence graph is not completely regular. Hence, control bits are used to modify the flow (or velocity) of the tokens in order to execute the dependence graph on the processor array correctly.

The key observation is as follows. Matrix $\mathbf{C}$ (whose transitive closure is to be found) is input along dependence direction $\vec{d}_3$. The inputs along other dependence directions $\vec{d}_1$, $\vec{d}_2$, $\vec{d}_4$, $\vec{d}_5$ are non-existent, *i.e.,* they are never sent into the array from the external host. Hence, there are no data conflicts along these dependency directions as the generated outputs are sent at most once on each link in every cycle of the array. Therefore, we need to consider only data conflicts along dependence $\vec{d}_3$. Since, dependencies $\vec{d}_3$, $\vec{d}_4$ and $\vec{d}_5$ never co-exist, there are only two spacings for data along direction $\vec{d}_3$, namely, $\vec{S}_{3,1}$ and $\vec{S}_{3,2}$.

Thus, a total of 8 parameters are defined for the transitive closure problem: 3 periods t_1, t_2, t_3, 3 displacements $\vec{k}_1, \vec{k}_2, \vec{k}_3$, and 2 spacings $\vec{S}_{3,1}$, $\vec{S}_{3,2}$. For a linear array all the parameters are scalars. Applying Theorem 2, the periods along dependencies, $\vec{d}_4$ and $\vec{d}_5$, are given as $t_4 = t_1 + t_3$ and $t_5 = t_2 + t_3$. Similarly, displacements $\vec{k}_4 = \vec{k}_1 + \vec{k}_3$ and $\vec{k}_5 = \vec{k}_2 + \vec{k}_3$. From Theorem 1 we get,

$$\vec{S}_{3,1} = \frac{t_3\vec{k}_1 - t_1\vec{k}_3}{t_3}, \quad \vec{S}_{3,2} = \frac{t_3\vec{k}_2 - t_2\vec{k}_3}{t_3}.$$

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

Lemma 1. *The completion time T_c for an $N \times N$ transitive closure is given by*

$$T_c \quad = \quad (N-1)(2\,t_1 + 2\,t_2 + t_3) + 1$$

Proof. The critical path in the execution is as follows:

$$(1,1,1) \overset{(N-1)t_1}{\longrightarrow} (1,1,N) \overset{(N-1)t_2}{\longrightarrow} (1,N,N) \overset{(N-1)t_3}{\longrightarrow}$$
$$(N,1,1) \overset{(N-1)t_1}{\longrightarrow} (N,1,N) \overset{(N-1)t_2}{\longrightarrow} (N,N,N)$$

Thus, T_c is $(N-1)(2\,t_1 + 2\,t_2 + t_3) + 1$. ∎

Lemma 2. *The number of processor $\#PE$ needed for an $N \times N$ transitive closure is given by*

$$\#PE \quad = \quad (N-1)(\left|\vec{k}_1\right| + \left|\vec{k}_2\right| + \left|\vec{k}_1 + \vec{k}_2 + \vec{k}_3\right|) + 1$$

Table 1: Transitive Closure: Optimal linear arrays for minimum completion time T_c. Run time is the CPU time in seconds on a Sun/4 IPC workstation.

Size (N)	Optimal Time Design - GPM								Run Time	LK[2] Design		SF[3] Design	
	Periods			Distances			Min	#PE					
	t_1	t_2	t_3	$\vec{k}_1$	$\vec{k}_2$	$\vec{k}_3$	T		(Sun4)	T_c	$\#PE$	T_c	$\#PE$
3	1	1	2	0	-1	1	13	3	-	15	5	11*	3
4	1	1	3	0	-1	1	22	4	-	28	7	19*	4
8	1	2	3	0	-2	1	64	22	-	120	15	71	8
16	1	2	5	0	-2	3	166	46	-	496	31	271	16
32	1	3	6	0	-3	5	435	156	-	2,016	63	1,055	32
64	1	5	7	0	-5	6	1,198	379	-	8,128	127	4,159	64
100	1	5	11	0	-5	9	2,278	892	2	19,900	199	10,099	100
200	1	8	13	1	-8	12	6,170	2,787	14	79,800	399	40,199	200
300	1	9	18	0	-9	17	11,363	5,084	46	179,700	599	90,299	300

Proof. Let $\vec{k}_3''$ denote the displacement from the execution location of index (k,i,j) to index $(k+1,i,j)$. Therefore, the displacement $\vec{k}_3''$ equals $\vec{k}_1+\vec{k}_2+\vec{k}_3$ as depicted below.

$$(k,i,j) \xrightarrow{\vec{k}_1} (k,i,j+1) \xrightarrow{\vec{k}_2} (k,i+1,j+1) \xrightarrow{\vec{k}_3} (k+1,i,j)$$

Consider the displacements $\vec{k}_1$, $\vec{k}_2$, and $\vec{k}_3''$. Two of the above 3 displacements should be in the same direction, since the array is 1-dimensional. Assume that $\vec{k}_1$ and $\vec{k}_2$ are positive displacements, *i.e.,* they correspond to velocities flowing to the right (refer to Figure 2). Let A be the PE where the computation indexed by $(1,1,1)$ occurs. Therefore, computation $(1,1,N)$ is executed at PE B that is at a distance $(N-1)\left|\vec{k}_1\right|$ from PE A. Similarly, computation $(1,N,N)$ is executed at PE C that is $(N-1)\left|\vec{k}_2\right|$ PEs to the right of B. Computation $(N,1,1)$ is executed at PE D that is at a distance of $(N-1)\left|\vec{k}_3''\right|$ to the left of PE A (since $\vec{k}_3$ corresponds to the left moving variable). All other computations in the domain are executed by PEs between C and D. Therefore, the total number of PEs required is $(N-1)(\left|\vec{k}_1\right|+\left|\vec{k}_2\right|+\left|\vec{k}_3''\right|)+1$ $=(N-1)(\left|\vec{k}_1\right|+\left|\vec{k}_2\right|+\left|\vec{k}_1+\vec{k}_2+\vec{k}_3\right|)+1.$ ∎

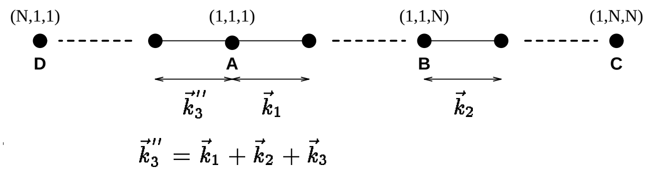

Figure 2: PE allocation with $\vec{k}_1$, $\vec{k}_2 \geq 0$ and $\vec{k}_3'' \leq 0$.

For linear array synthesis, since the spacings are scalars, let $s_{3,1} = \left|\vec{S}_{3,1}\right|$ and $s_{3,2} = \left|\vec{S}_{3,2}\right|$. In addition, the condition for data conflict (Theorem 3) can be refined as follows.

Theorem 4. *Data conflicts occur in input matrix* **C** *if and only if,*

$$\frac{s_{3,1}}{m} < N, \quad \text{and} \quad \frac{s_{3,2}}{m} < N \qquad (9)$$

where $m = GCD(s_{3,1}, s_{3,2})$ and $GCD(a,b)$ is the greatest common divisor of a and b.

Table 1 shows the optimal linear designs found by the enumeration procedure of GPM when the objective is to minimize completion time. The column LK in the table gives the completion time T_c and $\#PE$, for the linear array designed by the heuristic procedure of Lee and Kedem [2]. The schedule vector for their mapping is $(2N-1,1,1)$, and the PE allocation matrix (vector in the case of a linear array) is $(0,1,1)$ for $N \times N$ matrices. Table 1 also shows T_c and $\#PE$ for the Shang and Fortes (SF) design[3]. This design uses $(N,1,1)$ as the schedule vector $(0,0,-1)$ for PE allocation matrix.

Table 1 shows that for a problem of size around 200, the minimum T_c array is roughly 13 times faster than the LK design and 7 times faster than SF design. However, LK and SF designs use fewer $\#PE$ than the minimum T_c design found by GPM.

If the objective is to minimize the PE count, $\#PE$, in the linear array, then Theorem 5 below characterizes the optimal design.

Theorem 5. *The parameters $(t_1,t_2,t_3) = (1,1,N-1)$ and $(\vec{k}_1,\vec{k}_2,\vec{k}_3) = (0,\pm1,\mp1)$ or $(\pm1,0,\mp1)$ result in a linear array with a primary objective of minimizing the number of PEs, and a secondary objective minimizing the completion time.*

Theorem 5 shows that the SF and LK designs are processor-optimal with $\#PE$ equal to N. The SF design has periods $(t_1,t_2,t_3) = (1,1,N-2)$ and displacements $(k_1,k_2,k_3) = (1,0,-1)$. Hence, the SF design has a completion time $T_c = (N-1)(N+2)+1$. This completion time is lower than that of the minimum-processor design characterized by Theorem 5 (*e.g.,* N = 3,4 in Table 1). The above periods and velocities lead to spacings $\vec{S}_{3,1} = \frac{N-1}{N-2}$ and $\vec{S}_{3,2} = \frac{1}{N-2}$. These values of spacings result in data conflicts between the tokens $(\mathbf{C}_{1,j}, \mathbf{C}_{N-1,j-1})$ and $(\mathbf{C}_{2,j}, \mathbf{C}_{N,j-1})$, $j = 2,3,\cdots,N$, of the input matrix **C** (Theorem 4). Note that these conflicts occur between passive data as mentioned in Section 1. Hence, the above solution is not an acceptable solution in the GPM, although it was found by the search procedure in GPM.

If the objective is to minimize the processor-time product $\#PE \cdot T$, the optimal design values are identical to those that minimize $\#PE$ (Theorem 1) for problem sizes listed in Table 1.

Table 2 presents the results for the objective of min-

Table 2: Transitive Closure: Optimal linear array design for minimizing $\#PE \cdot T^2$.

Size (N)	Optimal $\#PE \cdot T^2$ Design - GPM								
	Periods			Distances			T_c	$\#PE$	Run Time
	t_1	t_2	t_3	$\vec{k}_1$	$\vec{k}_2$	$\vec{k}_3$			
3	1	1	2	0	-1	1	13	3	-
4	1	1	3	0	-1	1	22	4	-
8	1	1	7	0	-1	1	78	8	-
16	1	2	5	0	-2	3	166	46	-
32	1	3	7	0	-3	4	466	125	3
64	1	5	7	0	-5	6	1198	379	32
100	1	6	10	0	-6	7	2377	694	202
200	1	8	16	0	-8	9	6767	1792	2698

imizing $\#PE \cdot T^2$. It was found that the minimum-PE design is the best for minimizing $\#PE \cdot T^2$, up to a problem size of 13. For sizes over 13, the minimum T_c design or an intermediate design optimizes $\#PE \cdot T^2$. 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_1 be the completion time and $\#PE$ of the minimum-time design, respectively. The search then proceeds to find the minimum $\#PE \cdot T^2$ design with with completion time between T_{min} and $T_{up} = \frac{T_{min}\sqrt{P_1}}{\sqrt{N}}$. For every feasible design found in the search, the upper bound T_{up} is updated to $T_{up} = Min(\frac{T^{'}\sqrt{P^{'}}}{\sqrt{N}}, T_{up})$, where $T^{'}$ and $P^{'}$ are the completion time and $\#PE$ of the feasible design.

The optimal linear array and its space-time diagram for $N = 3$ is given in Figure 3. The parameters of the design are: $(t_1, t_2, t_3) = (1, 1, 2)$ and $(\vec{k}_1, \vec{k}_2, \vec{k}_3) = (0, 1, -1)$. This design minimizes both T_c and $\#PE$, and therefore, minimizes any objective of the form $\#PE^m \cdot T^n$ for $m, n \geq 1$. Note that for correct execution of the Warshall-Floyd algorithm, control signals are needed to govern the index-dependent assignments performed by the PEs in the array. These index-dependent assignments are given in Tables I and II of the reference [6].

In the following discussion, let T_c^{min} and $\#PE^{max}$, respectively, be the completion time and $\#PE$ of the minimum-time design. Designs with $\#PE > \#PE^{max}$ would not be useful as their completion times has to be at least T_c^{min}. Let T_c^{max} and $\#PE^{min}$, respectively, be the completion time and $\#PE$ of the minimum-processor design (from Theorem 5 and Lemma 2, $\#PE^{min} = N$). Again, there is no benefit in obtaining designs with $T_c > T_c^{max}$ as the number of PEs cannot be reduced below $\#PE^{min}$.

So far, we have presented ways of finding optimal arrays when the objective function can be clearly established ahead of time. However, in practical situations, the exact tradeoff is not easily translated into an objective function that can be minimized. A designer might be unwilling to settle for either the high number of PEs required in the minimum-time design or the long completion time of the minimum-processor design. Hence, designs with completion time greater than T_c^{min} and $\#PE$ less than $\#PE^{min}$ would be of interest. In realistic design situations there may be bounds on the number of processors or the completion time or both. Hence, possible objectives could be to have as few processors as possible, so long as the time

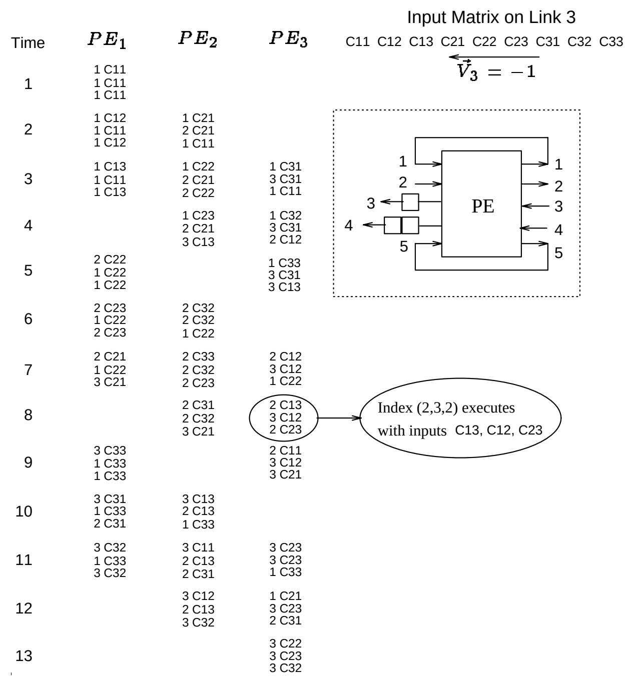
Time
PE_1
PE_2
PE_3
Input Matrix on Link 3
C11 C12 C13 C21 C22 C23 C31 C32 C33
$\vec{V}_3 = -1$
1: 1 C11, 1 C11, 1 C11
2: 1 C12, 1 C11, 1 C12 | 1 C21, 2 C21, 1 C11
3: 1 C13, 1 C11, 1 C13 | 1 C22, 2 C21, 2 C22 | 1 C31, 3 C31, 1 C11
4: 1 C23, 2 C21, 3 C13 | 1 C32, 3 C31, 2 C12
5: 2 C22, 1 C22, 1 C22 | 1 C33, 3 C31, 3 C13
6: 2 C23, 1 C22, 2 C23 | 2 C32, 2 C32, 1 C22
7: 2 C21, 1 C22, 3 C21 | 2 C33, 2 C32, 2 C23 | 2 C12, 3 C12, 1 C22
8: 2 C31, 2 C32, 3 C21 | 2 C13, 3 C12, 2 C23
9: 3 C33, 1 C33, 1 C33 | 2 C11, 3 C12, 3 C21
10: 3 C31, 1 C33, 2 C31 | 3 C13, 2 C13, 1 C33
11: 3 C32, 1 C33, 3 C32 | 3 C11, 2 C13, 2 C31 | 3 C23, 3 C23, 1 C33
12: 3 C12, 2 C13, 3 C32 | 1 C21, 3 C23, 2 C31
13: 3 C22, 3 C23, 3 C32
PE
1
2
3
4
5
Index (2,3,2) executes with inputs C13, C12, C23

Figure 3: **Linear array to find the transitive closure of a 3×3 matrix. The array is optimal for completion time, $\#PE$, and $\#PE^m \cdot T^n$, $m, n \geq 1$.**

is within a preset upper limit, T_c^{up}, or to minimize T_c with $\#PE$ less than a given upper bound $\#PE^{up}$. Plots similar to those shown in Figure 4 would aid the designer to make these choices easily and create realistic designs.

Figure 4 shows how $\#PE$ varies with T_c for 3 different problem sizes: $N = 100, 200$, and 300. In all plots, the y-axis $\#PE$ is normalized by $\#PE^{max}$ and the x-axis T_c is scaled by T_c^{max}. This lets us combine the different problem sizes uniformly on the same scale. The stepped curves (labeled as "time" plots) show the $\#PE$ required for a given maximum completion time. A point (T,P) in any of these "time" plot denotes that at least P PEs are needed if the completion time must not exceed T. The smooth curves (labeled as "#PE" plots in the figure) show the completion time required for a given maximum $\#PE$. Thus, a point (T,P) in the "#PE" plots denotes that at least T time steps are needed if the $\#PE$ is to be less than P processors. Hence, given the bound T_c^{up} (*resp.* $\#PE^{up}$) the designer can read off the minimum $\#PE$ (*resp.* T_c) required and decide (possibly from a cost perspective) if it is acceptable. Again, the designer could exploit the initial steep decline in the plots to choose an alternative design that trades off performance for cost. For instance, the minimum $\#PE$ drops by 43% for only a 19% increase in completion time. In contrast, the SF and LK methods can only obtain a single design (with minimum-$\#PE$) and cannot perform tradeoffs similar to those presented here.

If both T_c and $\#PE$ are bounded from above, then the design with minimum $\#PE$ for the given time bound is determined using the "time" plots in Figure 4. If this $\#PE$ is within the given processor bound, then the minimum-

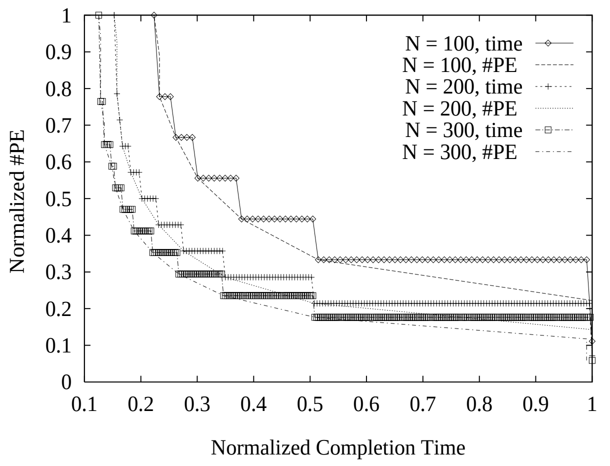

Figure 4: Performance tradeoffs : Variation in $\#PE$ with the time bound T_c^{up} and variation in T_c with processor bound $\#PE^{up}$. The plots are given for three problem sizes $N = 100, 200$ and 300.

T_c for the above $\#PE$ is found from the "#PE" plots in the figure. This now represents the best design under the time and processor constraints.

Another observation from Figure 4 is that the plots for larger N decrease more rapidly than those for smaller N. Hence, for larger values of N, there is a substantial reduction in $\#PE$ (*resp.* T_c) for a relatively small increase of the completion time (*resp.* $\#PE$) from the optimum. Therefore, for large N, there are more attractive alternatives than the optimal-time or optimal-$\#PE$ designs.

4 Concluding Remarks

This paper describes the generalized parameter method, a systematic technique for synthesizing systolic architectures from uniform recurrences (as well as algorithms with nested loop structures). We characterize the conditions for correct systolic processing of the recurrence, and by bounding the parameters from above, find the optimal solution in polynomial time. We illustrate the method by deriving optimal linear arrays for transitive closure problem with specific objectives such as completion time, $\#PE$, or $\#PE \cdot T$. The formulation of GPM as an optimization allows additional constraints on the target array to be incorporated easily. In realistic situations when the objective cannot be established exactly, the GPM permits the designer to trade performance for cost to best suit the given constraints and needs. In the future, we will extend and apply this parameter-based search technique to efficiently map algorithms to existing SIMD and vector machines, such as the CM2, MasPar, and Cray-Y-MP.

Acknowledgement. We would like to thank Vijay Karamcheti for a number of stimulating discussions and providing useful input to this work.

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," *International Conference on Parallel Processing*, vol. 1, pp. 101–110, Pennsylvania State University Press, Aug. 1990.

[4] 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.

[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] 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.

[7] 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.

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

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

[10] S. Y. Kung, S. C. Lo, and P. S. Lewis, "Optimal systolic design for transitive closure and shortest path problems," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 603–614, 1987.

[11] L. J. Guibas, H. T. Kung, and C. D. Thompson, "Direct VLSI implementation of combinatorial algorithms," *Proceedings of CALTECH Conference on VLSI*, pp. 509–525, Jan. 1979.

[12] G. Rote, "A systolic array for algebraic path problem," *Computing*, vol. 34, pp. 192–219, Springer Verlag, 1985.