

A Processor-Time-Minimal Systolic Array for Cubical Mesh Algorithms

*Peter Cappello**

Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

Using a directed acyclic graph (dag) model of algorithms, the paper focuses on time-minimal multiprocessor schedules that use as few processors as possible. Such a *processor-time-minimal* scheduling of an algorithm's dag first is illustrated using a triangular shaped 2D directed mesh (representing, for example, an algorithm for solving a triangular system of linear equations). Then, algorithms represented by an $n \times n \times n$ directed mesh are investigated. This cubical directed mesh is fundamental; it represents the standard algorithm for computing matrix product as well as many other algorithms. Completion of the cubical mesh requires $3n - 2$ steps. It is shown that the number of processing elements needed to achieve this time bound is at least $\lceil 3n^2/4 \rceil$. A systolic array for the cubical directed mesh is then presented. It completes the mesh using the minimum number of steps and exactly $\lceil 3n^2/4 \rceil$ processing elements: it is processor-time-minimal. The systolic array's topology is that of a hexagonally shaped, cylindrically connected 2D directed mesh.

*This work supported by the National Science Foundation under grant MIP89-20598, and the Lawrence Livermore National Laboratories.

1 Introduction

Although it is only one of many styles of concurrent computation, systolic computation is both:

- physically suited to VLSI technology [29], and
- natural for many algorithms [25, 36].

These properties suggest that systolic computation has an important, enduring role to play in concurrent computation. The systolic paradigm of concurrent computation has been applied to a wide variety of problems [25] (e.g., bit-level arithmetic, database operations, digital filtering [47], graph problems, matrix computations, etc.). Systolic realizations are natural for any algorithm that has an iterative dependence graph; such graphs can be realized using only local communication in space and time. Indeed, the dependence graph is taken as the *starting point* in our search for an optimal systolic array: we adopt the view expressed by Papadimitriou and Ullman [35],

We model the computational problem to be solved as a directed acyclic graph (dag), with nodes corresponding to computed values, and arcs denoting dependencies (i.e., the children of a node are the values used to compute the node). *Naturally, in most cases this dag is part of the algorithm design sought, but it seems reasonable, at least at this point, to consider the dag fixed and given.* [emphasis added]

Such dags can be extracted automatically from, for example, uniform recurrent equations [37], a system of uniform recurrence equations [24], regular iterative arrays [43, 20], a system of linear recurrence equations [41, 40, 11], as well as computations specified in the model of Moldovan and Fortes [32, 12, 33, 14].

Given a dag, we are interested in a multiprocessor schedule that assigns node v for processing during step $\tau(v)$ on processor $\pi(v)$. The *time* (or number of steps) used to compute a dag $G = (N, A)$ is determined when we assign a multiprocessor schedule to the nodes that is subject to 2 constraints:

1. A node can be computed only when its children have been computed at previous time steps:

$$(u, v) \in A \Rightarrow \tau(u) < \tau(v). \tag{1}$$

2. No processor can compute 2 different nodes during the same time step:

$$\tau(v) = \tau(u) \Rightarrow \pi(v) \neq \pi(u). \quad (2)$$

Definition: A multiprocessor schedule for a dag is **time-minimal** when the number of steps in the schedule equals the number of nodes in a longest directed path in the dag.

This is a *machine-independent* measure of the *maximum parallelism* in the dag. We now turn our attention to a specific problem: matrix product. Starting with Kung and Leiserson’s seminal paper [26], there has been a steady stream of successful research on systolic arrays, especially for computing a matrix product. Kung and Leiserson were the first to present a systolic array for banded matrix product [26]. This soon was followed by Weiser and Davis’s systolic array [48] that uses $1/3$ as many steps. This was followed by a time-minimal design for full matrix product completing in $3n - 2$ steps, for $n \times n$ matrices, using n^2 processors [5].

All these systolic arrays share two things:

1. They all use the same dependence dag, an $n \times n \times n$ directed mesh¹.
2. They all map this cubical mesh into processor-time with a transformation of the indices that is linear. That is, if we are given the computation

$$\text{for } 1 \leq i, j \leq n \text{ do } \{c(i, j) \leftarrow \sum_{k=1}^n a(i, k) \cdot b(k, j)\}$$

then each term of the summation — each inner product step — corresponds to an index vector, $[i \ j \ k]^T$. The *time step* and *processor location* of each of these inner-product steps is given by:

$$\begin{bmatrix} \text{step} \\ \text{location}_1 \\ \text{location}_2 \end{bmatrix} = M \begin{bmatrix} i \\ j \\ k \end{bmatrix},$$

for some matrix $M \in \mathbf{Z}^{3 \times 3}$. (The resultant systolic arrays thus are all 2-dimensional.)

Such linear maps of iterative dependence dags have been researched intensely. They are implicit in the research of Johnsson and Cohen [23, 22], Weiser and Davis [48], and Ramakrishnan, Fussell,

¹The dependence dag associated with banded matrix product is a subgraph of the cubical mesh.

and Silberschatz [42]. Linear maps are explicit in the research of Moldovan [32, 33], Cappello and Steiglitz [2, 4, 5], Quinton [37, 38], and Gachet et al. [16]. Such investigations are surveyed by Fortes, Fu, and Wah [13], and Quinton [39]. Work on enumerating, or otherwise exploring, the various linear maps associated with an iterative dependence dag has been reported by Moldovan [34], Miranker and Winkler [31], Danielsson [8], Moldovan and Fortes [14], Rao [43], Delosme and Ipsen [10], Rajopadhye et al. [41, 40], and Lee and Kedem [27].

There has been a great deal of work on optimizing systolic arrays. The work pursued by Li and Wah [21], Fortes and Parisi-Presicce [15], Rao [43], Delosme and Ipsen [9], Chen [6, 7], Lee and Kedem [27], Shang and Fortes [45], and most recently by Wong and Delosme [49, 50], all contribute to methods for optimizing systolic arrays. These efforts constrain the processor-time mapping to be a linear or affine transformation of the problem's index set. The first reason that this constraint is used is because it yields systolic arrays that are both intuitively appealing and practical to implement. The question nonetheless arises as to whether relaxing the linearity constraint results in an even more efficient use of time and space. This question leads to the second reason that extant optimization efforts constrain the processor-time mapping to be linear or affine: the general problem of *precedence constrained scheduling* onto a set of processors is NP-complete (see [17] for references to a variety of such problems). Problems that are NP-complete may be dealt with in several ways. One way is to isolate a fundamental, indexed family of problem instances, and find an optimal parameterized solution for that family. This idea is illustrated below. First, the particular optimum under consideration is defined.

Definition: A multiprocessor schedule for a dag is **processor-time-minimal** when it uses as few processors as any time-minimal schedule for the dag.

Although only one of many performance measures, processor-time-minimality is useful because it indicates the *minimum number of processing elements* that are sufficient to extract the *maximum parallelism* from a dag. Being machine-independent, it is a more fundamental measure than those that depend on a particular architecture.

1.1 A simple example

We now illustrate a processor-time-minimal multiprocessor schedule for a simple family of dags: a triangular shaped directed mesh. This family represents the standard algorithm for solving a triangular system of linear equations. For this family of dags, the instance index (or size parameter) is n ; the parameterized time-minimal schedule is $2n - 1$; we will see that the parameterized processor-time-minimal multiprocessor schedule is a systolic array that uses $\lceil n/2 \rceil$ processing elements.

Figure 1[a] depicts a dag of processes for solving a triangular system of linear equations by forward substitution. Cappello and Laub [3] note that the multiprocessor schedule, depicted in Figure 1[b], is processor-time-minimal.



Figure 1: [a] *A process dag for solving a triangular system of 6 linear equations by forward substitution.* [b] *A processor-time-minimal mapping of the dag of [a].*

Why is this map processor-time-minimal? Using Figure 1[b], we can visualize a partition of the node set into layers according to the step for which the node is scheduled. If we name these layers with their step, the names would be $1, 2, \dots, 11$. Notice that every node has an arc to some node in the next higher layer (except the sink node, labeled ‘6 6’ in layer 11): Every node is on a longest path from the source to the sink. We now use this fact to argue that this layering is unique. Let v be some node depicted in Figure 1[b] that is in layer i ($2 \leq i \leq 11$). We cannot reschedule v for an earlier layer without violating Constraint 1. Similarly, if we reschedule v for a later layer, then we must either move all nodes that come later in v ’s path to later layers (violating time-minimality) or leave them in their present layers (violating Constraint 1). The layering thus is unique. In particular, consider layer 5 which consists of the nodes labeled ‘3 3’, ‘2 4’, and ‘1 5’. From the foregoing, we conclude that, in any time-minimal schedule, these nodes all must be scheduled for the same step. By Constraint 2, these 3 nodes must be scheduled onto 3 distinct processors. ■

The example above is a very simple illustration of an optimal parameterized solution for an indexed family of problem instances.

2 A processor-time-minimal systolic array for the cubical mesh

We now consider a fundamental family of dags: the $n \times n \times n$ directed mesh. Beyond matrix product, Ibarra and Palis [19] point out that the cubical mesh is the dependence dag of a variety of recurrences over 3 variables (e.g., finding the longest common subsequence among 3 strings). Other computations include $L - U$ factorization, a 3-pass transitive closure [18], matrix triangulation, matrix inversion, and 2-dimensional tuple comparison [21]. This cubical mesh can be defined as follows.

$G_n = (N_n, A_n)$, where

- $N_n = \{(i, j, k) \mid 1 \leq i, j, k \leq n\}$.
- $A_n = \{[(i, j, k), (i', j', k')] \mid \text{where exactly 1 of the following conditions holds}$
 1. $i' = i + 1$
 2. $j' = j + 1$
 3. $k' = k + 1$

for $1 \leq i, j, k \leq n\}$.

Time-minimal schedule: $3n - 2$ steps. It is clear that the longest directed path in this dag has $3n - 2$ nodes.

2.1 The processor-time lower bound

Definition: Let $G = (N, A)$ be a dag. We uniquely label each node $v \in N$ with number:

- i , when v is the i th node on a longest directed path in G ;
- 0, otherwise.

This labeling partitions N into equivalence classes:

$$Q_0 = \{v \in N \mid \nexists \text{ a longest directed path in } G \text{ containing } v\}$$

$$Q_{i \neq 0} = \{v \in N \mid v \text{ is the } i\text{th node on a longest directed path in } G\}$$

We refer to each nonzero equivalence class as a **concurrent** set of nodes.

Using this definition, we state a simple but useful theorem.

Theorem 1: *Let $G = (N, A)$ be a dag, $Q_i \subseteq N$ be a concurrent set of nodes, and p be the number of processors implementing a time-minimal schedule. Then $|Q_i| \leq p$.*

Proof. Since Q_i is a concurrent set of nodes, it is a nonzero equivalence class of the labeling: every node in Q_i is the i th node on a longest directed path in G . Let $v \in Q_i$. By multiprocessor scheduling Constraint 1, v cannot be scheduled for processing before step i . Consider all nodes that come later in a longest path that goes through v . If we move v to a later layer, then all these nodes must also move to later layers (or else Constraint 1 is violated). But then the schedule is no longer time-minimal. Therefore, every node in Q_i must be scheduled for execution during step i in order for the schedule to be time-minimal. By Constraint 2, each node scheduled for step i must be processed on a different processor. Therefore, $p \geq |Q_i|$. ■

So, the number of nodes in a concurrent set contained in the cubical mesh is a lower bound on the number of processors used in any time-minimal multiprocessor schedule for it. The cubical mesh, it turns out, contains a concurrent set of size $\lceil (3/4)n^2 \rceil$. Fig. 2 depicts a $6 \times 6 \times 6$ mesh. We



Figure 2: A $6 \times 6 \times 6$ mesh, where each node is labeled with its step in a time-minimal schedule.

argue this lower bound as follows. Each node in this dag is on some longest path. Consequently, by the argument given in the proof of Theorem 1, the concurrent sets of this dag are unique. In Fig. 2 each node is labeled with its step in a time-minimal schedule. By inspection, we can see that a maximum concurrent set corresponds to the set of nodes scheduled for step 8. In general, this is the midpoint of the computation: step $\lceil (3n - 2)/2 \rceil$. The ceiling notation is used in case n is odd. Since the dag of Fig. 2 contains 27 nodes scheduled for step 8, according to Theorem 1, we need at least 27 processors to schedule this dag for completion in 16 steps. General expressions for the

number of processors needed for the midpoint step follow:

$$\text{even } n: \sum_{i=(n/2)+1}^n i + \sum_{i=n/2}^{n-1} i = (3/4)n^2.$$

$$\text{odd } n: \sum_{i=\lceil n/2 \rceil}^n i + \sum_{i=\lceil n/2 \rceil}^{n-1} i = \lceil (3/4)n^2 \rceil.$$

Thus, according to Theorem 1, we need at least $\lceil (3/4)n^2 \rceil$ processors to complete this computation dag in $3n - 2$ steps.

2.2 The processor-time upper bound

The question we pursue now is whether there is a systolic array that achieves this lower bound on processors. To do so, we introduce a more succinct representation of the cubical mesh. All the nodes in a vertical column of the mesh in Fig. 2 are represented by a single node (in Fig. 3) that is labeled with an interval of steps. These are the steps used by the n nodes in the vertical column represented. For example, in Fig. 2, the front, left, vertical column of 6 nodes are labeled with steps 1, 2, 3, 4, 5, and 6. This vertical column of nodes is represented in Fig. 3 by 1 node labeled 1–6.

The strategy for mapping the set of nodes onto the processor array is as follows. There is a processor for every node labeled with step 8 in Fig. 2. A distinct processor is assigned to each vertical column of nodes (in Fig. 2) that contains a node labeled with step 8 (in general, every column with a node labeled with step $\lceil (3n - 2)/2 \rceil$). These processors correspond to the circular nodes in Fig. 3. The processor array developed so far (i.e., the array of circular nodes in Fig. 3) is



Figure 3: *A partial mapping of columns onto the 27 processors. Each node in the figure is labeled with the interval of steps used by the nodes in the column it represents. Columns containing a node assigned to step 8 are represented by a disk; others are represented by a square.*

shaped hexagonally.

To complete the mapping, we need to assign the 9 remaining (square) columns in Fig. 3 (in general, $\lfloor (1/4)n^2 \rfloor$ columns) to processors. Fig. 4 presents such a completion. In the figure, these

columns are labeled **A, B, C, D, E, F, G, H, and I**; their assigned processors are labeled correspondingly (in lower case). There thus are 9 processors that have 2 columns assigned to them. When a processor is assigned 2 columns, its first column finishes execution *just before* its second column begins execution (i.e., Constraint 2 is satisfied). The connectivity implied by this mapping requires,



Figure 4: *The complete assignment of columns to processors: Virtual processors with Upper-case names are assigned to processors with corresponding lower-case names. The resulting connectivity between boundary processors is indicated.*

for example, that the processor named **d** must communicate directly to the processor named **c**. In general, these boundary processors communicate directly with the processors on the opposite (parallel) boundary. To bring these directly communicating boundary processors into proximity, we map the hexagonally shaped array onto the surface of a cylinder. One simply wraps the $n \times n$ mesh of Fig. 5 (where each node represents a column of processes) so that the each node with an upper case label is superimposed onto the node with the corresponding lower case label. In this way, each column of processes maps onto a processor.



Figure 5: *The systolic array before wrapping.*

This processor-time map generalizes to any even n . To visualize this map:

1. construct a transparency from Fig. 5;
2. wrap it into a cylinder, superimposing the appropriate nodes.

The cylindrical wrapping is the same when n is odd, except that the boundary connectivity is

skewed slightly. This connectivity is illustrated, for $n = 5$, in Fig. 6. The ‘transparency method’ mentioned above is an especially useful way to visualize this skewed wrapping.



Figure 6: *The cylindrical connectivity for $n = 5$.*

This strategy is formalized by the following map, $m : N \mapsto \mathbf{Z}^3$. Given a node represented by a vector $[i \ j \ k]^T$, m produces a step number (its first component) and a processor location (its last 2 components). It can be defined formally as follows.

$$\begin{pmatrix} \text{step} \\ \text{location}_1 \\ \text{location}_2 \end{pmatrix} = \begin{pmatrix} \tau(i, j, k) \\ \pi_1(i, j) \\ \pi_2(i, j) \end{pmatrix}, \text{ where}$$

$$\begin{aligned} \tau(i, j, k) &= i + j + k - 2 \\ \pi_1(i, j) &= (i + j - \lceil n/2 \rceil - 1) \bmod n \\ \pi_2(i, j) &= \begin{cases} i - j, & \text{if } n \text{ is even or } \lceil n/2 \rceil + 1 \leq i + j \leq \lceil 3n/2 \rceil \\ i - j + 1, & \text{if } n \text{ is odd and } \lceil n/2 \rceil + 1 > i + j \\ i - j - 1, & \text{if } n \text{ is odd and } i + j > \lceil 3n/2 \rceil \end{cases} \end{aligned}$$

The step function is linear whereas the location function is not. The first component of the location function, π_1 , gives rise to the cylindrical wrapping. The second component of the location function, π_2 , decomposes into cases. These cases distinguish whether n is even (a cylindrical mesh) or odd (a skewed cylindrical mesh).

Three lemmas now are presented which, taken together, prove that this map results in a processor-time-minimal systolic array.

Lemma 1: Applying map m to G_n results in a valid multiprocessor schedule.

Proof. In order for a schedule to be valid, 2 constraints must be satisfied:

1. A node is computed only after its children have been computed:

Node (i, j, k) may have 3 children: $(i - 1, j, k)$, $(i, j - 1, k)$, and $(i, j, k - 1)$. The schedule honors all precedences because

$$\tau(i, j, k) = i + j + k - 2 > i + j + k - 3 = \tau(i - 1, j, k) = \tau(i, j - 1, k) = \tau(i, j, k - 1).$$

2. No processor computes 2 different nodes during the same time step:

Since the spatial components of the map (i.e., π_1 and π_2) do not depend on k , a processor is the image of all of a k -column or none of it. The n nodes in k -column (i, j) are executed at n different steps, depending on the node's k value. The only case we need consider, then, is when a processor is the image of more than 1 column. Let column (i_1, j_1) and column (i_2, j_2) both be mapped to the same processor. Then

$$(i_1 + j_1 - \lceil n/2 \rceil - 1) \bmod n = (i_2 + j_2 - \lceil n/2 \rceil - 1) \bmod n.$$

That is, $|(i_1 + j_1) - (i_2 + j_2)| = n$. (Columns that map to the same processor cannot have index sums that differ by a multiple of n greater than 1; the difference between the smallest column index sum, $1 + 1$, and the largest, $n + n$, is less than $2n$.) Let us assume, without loss of generality, that $i_1 + j_1 + n = i_2 + j_2$. Then column (i_1, j_1) finishes at step $\tau(i_1, j_1, n) = i_1 + j_1 + n - 2$ and column (i_2, j_2) starts at step $\tau(i_2, j_2, 1) = i_1 + j_1 + n - 1$. When 2 columns are mapped to the same processor, their scheduling thus does not overlap. ■

Lemma 2: Applying map m to G_n results in a multiprocessor schedule that is processor-time-minimal.

Proof.

Processor-time-minimality decomposes into 2 claims:

1. The schedule is time-minimal:

The node that maps to the least point in time is node $(1, 1, 1)$. The node that maps to the greatest point in time is node (n, n, n) . Their time coordinates are respectively $\tau(1, 1, 1) = 1$ and $\tau(n, n, n) = 3n - 2$. Since the number of steps used, $3n - 2$, equals the number of nodes in a longest path, the schedule is time-minimal.

2. The schedule uses as few processors as any that is time-minimal:

We know that there are $\lceil (3/4)n^2 \rceil$ nodes that are labeled with step $\lceil (3n-2)/2 \rceil$. We now show that every column of nodes that does not contain a node labeled with step $\lceil (3n-2)/2 \rceil$ is mapped to a processor that also is the image of a column that *does* contain a node labeled with step $\lceil (3n-2)/2 \rceil$. This enables us to place an upper bound of $\lceil (3/4)n^2 \rceil$ on the number of processors. If column (i, j) does not contain a node labeled with step $\lceil (3n-2)/2 \rceil$, then either $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$ or $\tau(i, j, 1) > \lceil (3n-2)/2 \rceil$. We consider each of these cases:

Case $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$: This case decomposes into 2, depending on whether n is even or odd:

even n : We will show that:

- (a) Column (i, j) maps to the same processor as column $(i + (n/2), j + (n/2))$.
- (b) Column $(i + (n/2), j + (n/2))$ contains a node with time label $\lceil (3n-2)/2 \rceil$.

This is sufficient because it means that column (i, j) maps to one of at most $\lceil (3/4)n^2 \rceil$ processors.

First, we show part (a) by substituting directly into the definition of π_1 and π_2 , the spatial components of the map.

$$\begin{aligned} \pi_1(i, j) &= (i + j - \lceil n/2 \rceil - 1) \bmod n \\ &= (i + (n/2) + j + (n/2) - \lceil n/2 \rceil - 1) \bmod n \\ &= \pi_1(i + (n/2), j + (n/2)) \end{aligned}$$

$$\pi_2(i, j) = i - j = i + (n/2) - j - (n/2) = \pi_2(i + (n/2), j + (n/2))$$

We now show part (b): column $(i + (n/2), j + (n/2))$ must contain a node whose step is $\lceil (3n-2)/2 \rceil$. That is, if $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$, then

$$\tau(i + (n/2), j + (n/2), 1) \leq \lceil (3n-2)/2 \rceil \leq \tau(i + (n/2), j + (n/2), n).$$

The first inequality is established as follows: Since $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$, we have $i + j \leq n/2$. Therefore,

$$\tau(i + (n/2), j + (n/2), 1) = i + j + n - 1 \leq \lceil (3n-2)/2 \rceil.$$

For the second inequality,

$$\begin{aligned}
\lceil (3n-2)/2 \rceil &\leq \tau(i + (n/2), j + (n/2), n) \\
&= i + (n/2) + j + (n/2) + n - 2 \Leftrightarrow \\
1 &\leq i + j + (n/2).
\end{aligned}$$

odd n : We will show that:

- (a) Column (i, j) maps to the same processor as column $(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor)$.
- (b) Column $(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor)$ contains a node with time label $\lceil (3n-2)/2 \rceil$.

This is sufficient because it means that column (i, j) maps to a processor that already was allocated (see (a) above).

First, we show part (a) by substituting directly into the definition of π_1 and π_2 , the spatial components of the map.

$$\begin{aligned}
\pi_1(i, j) &= (i + j - \lceil n/2 \rceil - 1) \bmod n \\
&= (i + \lceil n/2 \rceil + j + \lfloor n/2 \rfloor - \lceil n/2 \rceil - 1) \bmod n \\
&= \pi_1(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor)
\end{aligned}$$

In order to use the definition of π_2 , we first establish that $i + j < \lceil n/2 \rceil + 1$. This follows from the inequality $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$. Therefore, $\pi_2(i, j) = i - j + 1$. On the other hand,

$$\lceil n/2 \rceil + 1 \leq i + \lceil n/2 \rceil + j + \lfloor n/2 \rfloor \leq \lceil 3n/2 \rceil.$$

The first inequality is clear; the second inequality holds since given $\tau(i, j, n) < \lceil (3n-2)/2 \rceil$, then $i + j \leq \lceil n/2 \rceil$. Therefore,

$$\pi_2(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor) = i + \lceil n/2 \rceil - j - \lfloor n/2 \rfloor = i - j + 1.$$

Summarily,

$$\pi_2(i, j) = i - j + 1 = i + \lceil n/2 \rceil - j - \lfloor n/2 \rfloor = \pi_2(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor).$$

We now show part (b): column $(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor)$ must contain a node with time label $\lceil (3n - 2)/2 \rceil$. That is, if $\tau(i, j, n) < \lceil (3n - 2)/2 \rceil$, then

$$\tau(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor, 1) \leq \lceil (3n - 2)/2 \rceil \leq \tau(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor, n).$$

The first inequality is established as follows: Again, since $\tau(i, j, n) < \lceil (3n - 2)/2 \rceil$, we have that $i + j \leq \lfloor n/2 \rfloor$. Therefore,

$$\tau(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor, 1) = i + j + n - 1 \leq \lceil (3n - 2)/2 \rceil.$$

For the second inequality,

$$\begin{aligned} \lceil (3n - 2)/2 \rceil &\leq \tau(i + \lceil n/2 \rceil, j + \lfloor n/2 \rfloor, n) \\ &= i + \lceil n/2 \rceil + j + \lfloor n/2 \rfloor + n - 2 \Leftrightarrow \\ 1 &\leq i + j + \lfloor n/2 \rfloor. \end{aligned}$$

Case $\tau(i, j, 1) > \lceil (3n - 2)/2 \rceil$: The details of this case are handled similarly. ■

We now show that the multi-processor schedule defined by m results in a systolic array.

Lemma 3: Applying map m to G_n results in a multiprocessor schedule for a systolic array.

Proof.

We show that communication is local in both time and space. Direct communication is represented by arcs in the graph.

Time: Communication in time is local; if (u, v) is an arc in the mesh, then $\tau(u) = \tau(v) - 1$.

Space: There are 3 types of arcs, ‘ i ’ arcs, ‘ j ’ arcs, and ‘ k ’ arcs (refer to the definition of the arc set of the cubical mesh). Let there be an arc directed from node $v = (i, j, k)$ to node $v' = (i', j', k')$. We may assume that this is either an i arc or a j arc: $i' = i + 1$, or $j' = j + 1$, but not both. (Recall that if 2 nodes differ only in coordinate k , then they are mapped to the same processor.) For these 2 cases, we now show that the spatial components of the processor-time map, π_1 and π_2 , preserve locality.

$i' = i + 1$: The difference in their first coordinate,

$$\begin{aligned}\pi_1(v') - \pi_1(v) &= (i' + j - \lceil n/2 \rceil - 1) \bmod n - (i + j - \lceil n/2 \rceil - 1) \bmod n \\ &= (i' - i) \bmod n = 1\end{aligned}$$

The difference in their second coordinate clearly is 1 when n is even. When n is odd, the map into the second coordinate, π_2 , decomposes into 3 cases, depending on what interval the sum $i + j$ falls into:

- $\lceil n/2 \rceil + 1 > i + j$;
- $\lceil n/2 \rceil + 1 \leq i + j \leq \lceil 3n/2 \rceil$;
- $i + j > \lceil 3n/2 \rceil$.

When n is odd, the difference in the nodes' second coordinate depends on whether both $i + j$ and $i' + j$ fall into the same interval (in which case their difference is 1), or they fall into different intervals (in which case the difference in the nodes' second coordinate is 0):

$$\pi_2(i', j') - \pi_2(i, j) = \begin{cases} 1 & \text{if } n \text{ is even or} \\ & \text{if } \lceil n/2 \rceil + 1 \leq i' + j, i + j \leq \lceil 3n/2 \rceil \text{ or} \\ & \text{if } \lceil n/2 \rceil + 1 > i' + j, i + j \text{ or} \\ & \text{if } i' + j, i + j > \lceil 3n/2 \rceil \\ 0 & \text{otherwise} \end{cases}$$

$j' = j + 1$: This case is similar to the one above. The difference in the nodes' first coordinate,

$$\begin{aligned}\pi_1(v') - \pi_1(v) &= (i + j' - \lceil n/2 \rceil - 1) \bmod n - (i + j - \lceil n/2 \rceil - 1) \bmod n \\ &= (j' - j) \bmod n = 1\end{aligned}$$

The difference in the nodes' second coordinate,

$$\pi_2(i', j') - \pi_2(i, j) = \begin{cases} 1 & \text{if } n \text{ is even or} \\ & \text{if } \lceil n/2 \rceil + 1 \leq i + j', i + j \leq \lceil 3n/2 \rceil \text{ or} \\ & \text{if } \lceil n/2 \rceil + 1 > i + j', i + j \text{ or} \\ & \text{if } i + j', i + j > \lceil 3n/2 \rceil \\ -2 & \text{otherwise} \end{cases}$$

In Fig. 6, the skewed wrap around connections (for odd n) correspond to the analysis above: an arc that wraps around also drops down 2 rows (i.e., the difference in rows between the destination of the arc and the source of the arc is -2).

From Lemmas 1 – 3, we have the following.

Theorem 2: Applying the map m to G_n results in a processor-time-minimal systolic array.

2.3 Array Layout

Cylindrically connected meshes are directly implementable in current PCB technology. They also are easy to embed in more densely connected systems, (e.g., in a hypercube, using a suitable gray code). Processors along the left and bottom edges of the square array depicted in Fig. 3 receive the input matrices. In the cylindrical processor-time-minimal array, these left [bottom] processors form a line that is inscribed on the surface of the cylinder. For these processors, the input schedule is given by the rule: $a(i, j)$ and $b(j, i)$ are input during step $i + j - 1$. The output (i.e., the product matrix) is held in place by the processors.

There are several simple ways to embed this cylinder in the Euclidean plane: fold it along the line drawn in Fig. 7, resulting in a trapezoidally shaped array. When n is even, the trapezoid encompasses exactly $3n^2/4$ processors. When n is odd, this fold results in an array that is similar (see Fig. 8). Connectivity differences occur along the right boundary processors. When n is even and $n/2$ is odd, there is another natural way to embed the hexagonally shaped, cylindrically connected mesh in the Euclidean plane. As Fig. 9 makes plain, the array can be folded into a rectangular array whose dimensions are $n/2 \times 3n/2$.



Figure 7: *The hexagonally shaped cylindrically connected 2D mesh, for $n = 14$. [a] Before folding. [b] After folding.*



Figure 8: *The hexagonally shaped cylindrically connected 2D mesh, for $n = 13$. [a] Before folding. [b] After folding.*

In the embeddings illustrated, the $3n^2/4$ processors are placed compactly, and routed with short wires. All of these organizations are feasible, for example, on the CHiP [46].

3 Conclusion

A definition was introduced for a *processor-time-minimal* embedding of a dag. Although only one of many performance measures, processor-time-minimality is, by definition, a *machine-independent* measure of the *minimum number of processors* sufficient to extract the *maximum parallelism* from a dependence dag. This first was illustrated on the triangular shaped 2D mesh, and then on the cubical mesh. This latter dag is fundamental, modeling a large variety of important computations, including matrix product. When the cubical mesh is representing matrix product, $C \leftarrow A \cdot B$, the ‘column’ of nodes that are mapped to a processor represent the inner-product steps associated with either a single a_{ij} , b_{ij} , or c_{ij} . That is, exactly 1 (*but any 1*) of the 3 matrices is stationary.

The presented bounds are not merely asymptotic, they are precise. For example, the $20 \times 20 \times 20$ mesh for computing a 20×20 matrix product requires at least 58 steps. Any parallel processor that achieves this time-minimal schedule needs at least 300 inner-product step processing elements. The systolic array presented here realizes this processor-time lower bound.

Similar research has recently been conducted by Benaini and Robert [1] on the algebraic path problem, by Louka and Tchunte [28] on Gauss-Jordon elimination, by Benaini and Robert [1] on



Figure 9: *The hexagonally shaped 2D mesh, for $n = 13$. [a] Before folding. [b] After folding.*

Gaussian elimination, and by Scheiman and Cappello on [44] transitive closure. With respect to matrix product, Melkemi and Tchunte [30] make the following conjecture:

In all optimal algorithms exhibited here, and which compute the product of two dense square matrices of order n in time $T = 3n - 2$, on an array of size $n \times n$, some data variables a_{ik}, b_{kj} must be input several times to the array. We conjecture that this multiple reading of some data variables, is a necessary condition for the designing of algorithms which run in time $t = 3n - 2$ on arrays of size $S \leq n(n + 1)$.

The cylindrically connected mesh presented here constructively disproves this conjecture (Melkemi and Tchunte may have meant to exclude cylindrically connected meshes).

Investigating fundamental algorithms with respect to processor-time-minimality may increase our basic understanding of their limits and potential for concurrent realization.

References

- [1] Abdelhamid Benaini and Yves Robert. Spacetime-minimal systolic arrays for gaussian elimination and the algebraic path problem. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 746–757, Princeton, September 1990. IEEE Computer Society.
- [2] Peter R. Cappello. *VLSI Architectures for Digital Signal Processing*. PhD thesis, Princeton University, Princeton, NJ, Oct 1982.
- [3] Peter R. Cappello and Alan J. Laub. Systolic computation of multivariable frequency response. *IEEE Trans. Autom. Control*, 33(6):550–558, June 1988.

- [4] Peter R. Cappello and Kenneth Steiglitz. Unifying VLSI array design with geometric transformations. In H. J. Siegel and Leah Siegel, editors, *Proc. Int. Conf. on Parallel Processing*, pages 448–457, Bellaire, MI, Aug. 1983.
- [5] Peter R. Cappello and Kenneth Steiglitz. *Advances in Computing Research*, volume 2: VLSI theory, chapter Unifying VLSI Array Design with Linear Transformations of Space-Time, pages 23–65. JAI Press, Inc., Greenwich, CT, 1984.
- [6] Marina C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *J. of Parallel and Distributed Computing*, pages 461–491, Dec. 1986.
- [7] Marina C. Chen. The generation of a class of multipliers: synthesizing highly parallel algorithms in VLSI. *IEEE Trans. on Computers*, 37(3):329–338, Mar. 1988.
- [8] Per E. Danielsson. Serial/parallel convolvers. *IEEE Trans. on Computers*, C-33(7):652–667, July 1984.
- [9] Jean-Marc Delosme and Ilse C. F. Ipsen. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *Proc. 2nd Int. Symp. on VLSI Technology, Systems and Applications*, pages 268–273, Taipei, 1985.
- [10] Jean-Marc Delosme and Ilse C. F. Ipsen. Systolic array synthesis: Computability and time cones. Technical Report Yale/DCS/RR-474, Yale, May 1986.
- [11] Vincent Van Dongen and Patrice Quinton. Uniformization of linear recurrence equations: a step towards the automatic synthesis of systolic arrays. In *Proc. Int. Conf. on Systolic Arrays*, pages 473–482, San Diego, May 1988. IEEE Computer Society.
- [12] José A. B. Fortes. *Algorithm Transformations for Parallel Processing and VLSI Architecture Design*. PhD thesis, University of Southern California, Los Angeles, Dec. 1983.
- [13] José A. B. Fortes, King-Sun Fu, and Benjamin W. Wah. Systematic design approaches for algorithmically specified systolic arrays. In Veljko M. Milutinović, editor, *Computer Architecture: Concepts and Systems*, chapter 11, pages 454–494. North-Holland, Elsevier Science Publishing Co., New York, NY 10017, 1988.

- [14] José A. B. Fortes and Dan I. Moldovan. Parallelism detection and algorithm transformation techniques useful for VLSI architecture design. *J. Parallel Distrib. Comput.*, 2:277–301, Aug. 1985.
- [15] José A. B. Fortes and F. Parisi-Presicce. Optimal linear schedules for the parallel execution of algorithms. In *Int. Conf. on Parallel Processing*, pages 322–328, Aug. 1984.
- [16] Pierrick Gachet, Brigitte Jouannault, and Patrice Quinton. Synthesizing systolic arrays using DIASTOL. In W. Moore, A. McCabe, and R. Urquhart, editors, *Proc. Int. Workshop on Systolic Arrays*, pages 25–36, University of Oxford, July 1986. Adam Hilger.
- [17] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.
- [18] Leonidas J. Guibas, H.-T. Kung, and Clark D. Thompson. Direct VLSI implementation of combinatorial algorithms. In *Proc. Caltech Conf. on VLSI*, pages 509–525, 1979.
- [19] Oscar H. Ibarra and Michael Palis. VLSI algorithms for solving recurrence equations and applications. *IEEE Trans. on Acoust., Speech, and Signal Processing*, ASSP-35(7):1046–1064, July 1987.
- [20] H. V. Jagadish, Sailash K. Rao, and Thomas Kailath. Multi-processor architectures for iterative algorithms. *Proc. IEEE*, September 1987.
- [21] Guo jie Li and Benjamin W. Wah. The design of optimal systolic algorithms. *IEEE Trans. on Computers*, C-34(1):66–77, 1985.
- [22] S. Lennart Johnsson and Danny Cohen. A mathematical approach to modeling the flow of data and control in computational networks. In H. T. Kung, R. Sproull, and G. Steele, editors, *VLSI Systems and Computations*, pages 213–225. Computer Science Press, Rockville, MD, 1981.
- [23] S. Lennart Johnsson, Uri Weiser, Danny Cohen, and Al L. Davis. Towards a formal treatment of VLSI arrays. In *2nd Caltech Conf. on VLSI*, pages 375–398, 1981.
- [24] Richard M. Karp, Richard E. Miller, and Shmuel Winograd. The organization of computations for uniform recurrence equations. *J. ACM*, 14:563–590, 1967.

- [25] H.-T. Kung. Why systolic architectures. *Computer*, 15(1):37–45, Jan 1982.
- [26] H.-T. Kung and C. E. Leiserson. Systolic arrays (for VLSI). In I. S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 256–282. SIAM, 1979.
- [27] Peizong Lee and Zvi Meir Kedem. Synthesizing linear array algorithms from nested for loop algorithms. *IEEE Trans. on Computers*, 37(12):1578–1598, Dec. 1988.
- [28] Basile Louka and Maurice Tchuente. An optimal solution for Gauss-Jordan elimination on 2D systolic arrays. In John V. McCanny, John McWhirter, and Earl E. Swartzlander Jr., editors, *Systolic Array Processors*, pages 264–274, Killarney, IRELAND, May 1989. Prentice-Hall.
- [29] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley Publishing Co., Menlo Park, CA, 1980.
- [30] Lamine Melkemi and Maurice Tchuente. Complexity of matrix product on a class of orthogonally connected systolic arrays. *IEEE Trans. on Comput.*, C-36(5):615–619, May 1987.
- [31] Willard L. Miranker and Andrew Winkler. Spacetime representations of computational structures. *Computing*, 32:93–114, 1984.
- [32] Dan I. Moldovan. On the analysis and synthesis of VLSI algorithms. *IEEE Trans. Comput.*, C-31:1121–1126, Nov. 1982.
- [33] Dan I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proc. IEEE*, 71(1):113–120, Jan. 1983.
- [34] Dan I. Moldovan. ADVIS: A software package for the design of systolic arrays. *IEEE Trans. Computer-Aided Design*, CAD-6(1):33–40, Jan. 1987.
- [35] Christos H. Papadimitriou and Jeffrey D. Ullman. A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646, 1987.
- [36] Nikolay Petkov. *Systolische Algorithmen und Arrays*. Akademie-Verlag, Berlin, 1989.
- [37] Patrice Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *Proc. 11th Ann. Symp. on Computer Architecture*, pages 208–214, 1984.

- [38] Patrice Quinton. *The Systematic Design of Systolic Arrays*, pages 229–260. Princeton University Press, 1987.
- [39] Patrice Quinton. *Mapping recurrences on parallel architectures*, chapter Vol. III: Supercomputer Design: Hardware & Software, pages 1–8. Int. Supercomputing Inst., Inc., 1988.
- [40] Sanjay V. Rajopadhye and Richard M. Fujimoto. Systolic array synthesis by static analysis of program dependencies. In J. W. DeBakker, A. J. Nijman, and P. C. Treleaven, editors, *Lecture Notes in Computer Science*, number 258: Parallel architectures and languages, Europe, pages 295–310. Springer Verlag, June 1987.
- [41] Sanjay V. Rajopadhye, S. Purushothaman, and Richard M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In K. V. Nori, editor, *Lecture Notes in Computer Science*, number 241: Foundations of Software Technology and Theoretical Computer Science, pages 485–503. Springer Verlag, December 1986.
- [42] I. V. Ramakrishnan, D. S. Fussell, and A. Silberschatz. Mapping homogeneous graphs on linear arrays. *IEEE Trans. Computers*, C-35(3):189–209, Mar. 1986.
- [43] Sailash K. Rao. *Regular Iterative Algorithms and Their Implementation on Processor Arrays*. PhD thesis, Stanford University, October 1985.
- [44] Chris Scheiman and Peter R. Cappello. A processor-time minimal systolic array for transitive closure. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 19–31, Princeton, September 1990. IEEE Computer Society.
- [45] Weijia Shang and José A. B. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. In *Int. Conf. on Systolic Arrays*, pages 393–402, San Diego, May 1988.
- [46] Larry Snyder. Introduction to the configurable highly parallel computer. *Computer*, 15(1):47–56, Jan 1982.
- [47] Earl E. Swartzlander, editor. *Systolic Signal Processing Systems*. Marcek Dekker Inc., New York, July 1987.

- [48] Uri Weiser and Al L. Davis. A wavefront notation tool for VLSI array design. In H. T. Kung, R. Sproull, and G. Steele, editors, *VLSI Systems and Computations*, pages 226–234. Computer Science Press, Rockville, MD, 1981.
- [49] Yiwan Wong and Jean-Marc Delosme. Optimization of computation time for systolic arrays. Dept. of Computer Sci. RR-651, Yale Univ., May 1989.
- [50] Yiwan Wong and Jean-Marc Delosme. Optimization of processor count for systolic arrays. Dept. of Computer Sci. RR-697, Yale Univ., May 1989.