Systolic Arrays for Integer Chinese Remaindering

Çetin K. Koç
Department of Electrical Engineering
University of Houston
Houston, TX 77204

Peter R. Cappello *
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

This paper presents several time-optimal, and spacetime-optimal systolic arrays for computing a process dependence graph corresponding to the mixed-radix conversion algorithm. The arrays are particularly suitable for software implementations of algorithms from the applications of residue number systems on a programmable systolic/wavefront array. Examples of such applications are exact solution of linear systems and matrix problems over integral domains. We also describe a decomposition strategy to treat a mixed-radix conversion problem whose size exceeds the arrays size.

1 Introduction

Residue Number Systems (RNS) provide an alternative to the weighted number systems for doing arithmetic on large integers. The advantages of residue representation are that addition, subtraction, and multiplication can be performed in a modular fashion without carry propagation. The work on residue arithmetic is mostly oriented towards hardware design since carry-free properties of RNS makes it attractive to implement digital signal processing algorithms using table-lookup techniques [36,38]. Also redundant moduli RNS provides fault tolerance in specialized signal processing architectures due to its error detection and correction properties [24,3,12]. RNS has applications in software as well. For example, it can be used to find solutions of equations over integral domains [4,2,25,23]. This allows computation of the exact solution of linear equations when the matrix of the coefficients is ill-conditioned.

One of the drawbacks of the RNS is that a number represented in residue notation does not have magnitude information. Thus it should be converted to a weighted number system to extract this information. The methods for conversion of a residue number to a weighted number system are based on two different constructive proofs of the *Chinese Remainder Theorem* (CRT). In the first case, the number is converted to a *single-radix* weighted number system, whereas in the second case it is converted to a *mixed-radix* weighted number system. The similarity between interpolation of polynomials and Chinese remaindering of integers is well-known. Lipson's paper provides an excellent source on this subject [22]. As it is also noted by Bareiss [2], the Lagrange polynomial interpolation formula is the analogue of the single-radix conversion algorithm for integer Chinese remaindering (see, page 270 in [19]). The mixed-radix conversion algorithm given in the book by Szabo and Tanaka [36] is attributed to Garner [15] by Knuth [19] and Lipson [21]. The mixed-radix conversion algorithm is in fact the analogue of the Aitken algorithm for the Newton polynomial interpolation. Henceforth, we refer to this algorithm as the *Garner Algorithm*.

*This work was supported in part by the Office of Naval Research under contracts N00014-84-K-0664 and N00014-85-K-0553.

Several hardware implementations of the Garner algorithm are reported in the literature [37,17,9]. A parallel implementation of the single-radix conversion algorithms also is reported [39]. In this paper we present time-optimal, and spacetime-optimal systolic arrays for computing a process dependence graph corresponding to the Garner algorithm. The arrays are especially suitable for software implementations of algorithms for exact solution of linear systems on a programmable systolic/wavefront array.

This paper is organized as follows: in §2, we give the Garner algorithm and point out its relationship to the Aitken algorithm. We also present the process dependence graph of the Garner algorithm in this section. In §3, 4, and 5, we embed the process dependence graph of the Garner algorithm in spacetime, obtaining several alternative systolic implementations of the Garner algorithm, some of which are optimal. In §6, we present a novel scheme, based on the mathematics of the problem, for decomposing a large mixed-radix conversion problem so that it can be computed on smaller systolic arrays. Finally in §7, we summarize the spacetime embeddings and the resulting systolic arrays, and discuss possible implementations.

2 The Garner Algorithm

We are given:

- The moduli set $\{m_0, m_1, m_2, \ldots, m_n\}$ consisting of $n+1$ pairwise relatively prime numbers, and
- The residues of a weighted number u with respect to each modulus

$$u_i \equiv u \pmod{m_i} \text{ for } 0 \le i \le n. \tag{1}$$

The number u then can be computed using the Garner algorithm as follows:

Garner Algorithm

Step 1. Compute constants c_{ij} for $0 \le i < j \le n$ where

$$c_{ij} m_i \equiv 1 \pmod{m_j}\ .$$

Step 2. Compute

$$\begin{aligned}
v_0 &\equiv u_0 \pmod{m_0}\ , \\
v_1 &\equiv (u_1 - v_0)c_{01} \pmod{m_1}\ , \\
v_2 &\equiv ((u_2 - v_0)c_{02} - v_1)c_{12} \pmod{m_2}\ , \\
&\vdots \\
v_n &\equiv (\cdots((u_n - v_0)c_{0n} - v_1)c_{1n} - \cdots - v_{n-1})c_{n-1,n} \pmod{m_n}\ .
\end{aligned}$$

Thus, the residue representation $(u_0, u_1, \ldots, u_n)$ now can be converted to mixed-radix representation $(v_0, v_1, \ldots, v_n)$. This representation of u has magnitude information since

$$u = v_0 + v_1 m_0 + v_2 m_0 m_1 + \cdots + v_n m_0 m_1 \cdots m_{n-1} \ . \qquad (2)$$

A mixed-radix representation can be converted to a single-radix representation by applying Horner's algorithm to formula (2).

The constants c_{ij} are the inverses of m_i modulo m_j for all $0 \leq i < j \leq n$, i.e.,

$$c_{ij} m_i \equiv 1 \pmod{m_j} \ , \qquad (3)$$

and can be computed using Euclid's algorithm (see [19,21]).

It becomes evident that Garner's algorithm is the integer analogue of Aitken's algorithm for Newton interpolation when we arrange the above computations (Step 2) in a table (the *multiplied difference table*) similar to the divided difference table for computing the coefficients of the Newton interpolating polynomial (i.e., the divided differences). The entries of this table are called the *multiplied differences* [22].

We now define V_{ij} for $0 \leq i < j \leq n$ such that $v_i = V_{i-1,i}$ for $1 \leq i \leq n$, and $v_0 = u_0$. The V_{ij} for $i < j - 1$ are the temporary values of v_j resulting from the operations in Step 2. We build a triangular table of values whose diagonal entries $V_{i-1,i} = v_i$, for $1 \leq i \leq n$. This table is similar to the Aitken table produced by the Aitken algorithm. It is presented in Table 1, for $n = 4$.

The data dependences among the entries in the above table lend themselves to systolic implementation. The first step in achieving a systolic implementation is to form the *process dependence graph* of the Garner algorithm. Coefficient c_{ij} is in column i and row j. The positions of the numerator terms (i.e., the V_{ij}) are arranged as follows: First, a term of the form $V_{i-1,i}$ is computed on the diagonal, then this term is used in every operation along the ith column. Based on these observations, Figure 1 presents the process dependence graph of the Garner algorithm, for $n = 4$. The graph is drawn on the (i, j) coordinate system. The nodes of this directed acyclic graph (dag) represent the operations, and the arcs correspond to dependences between the variables used in the operations. The node at point (i, j) computes V_{ij} by performing the operation

$$c_{ij} \equiv \frac{1}{m_i} \pmod{m_j} \ , \qquad (4)$$

$$V_{ij} \equiv (V_{i-1,j} - V_{i-1,i}) c_{ij} \pmod{m_j} \ . \qquad (5)$$

The above formulation of the processes at each node allows *preconditioned* Chinese remaindering as well. In this case, the constants c_{ij} are precomputed and saved at the nodes. The node at point (i, j) now executes only (5).

In the following sections, we embed the process dependence graph of the Garner algorithm, G, in spacetime to produce systolic arrays some of which are optimal (for spacetime embedding techniques, see [28,29,8,27,13,31,32]).

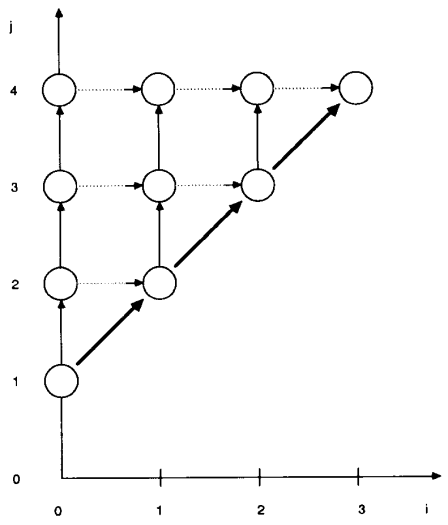

Figure 1: *The process dependence dag for Garner's algorithm ($n = 4$).*

3 A Time Optimal Systolic Array

In this section, we present a time-optimal array. We embed the process dependence graph for the Garner algorithm, G, in spacetime. The abscissa is interpreted as time (t); the ordinate as space (s). The linear embedding, E_1, is as follows:

$$\begin{bmatrix} t \\ s \end{bmatrix} := T_1 \begin{bmatrix} i \\ j \end{bmatrix} \ , \text{ where } \ T_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \ . \qquad (6)$$

The result, depicted in Figure 2 for $n = 6$, is a time-optimal array. Data that flows south $\rightarrow$ north in Figure 1, flows in the direction of time (perpendicular to space) in this design: It is in the processors' memory. Data that flows west $\rightarrow$ east in Figure 1, flows up through the array. Data that flows south $\rightarrow$ east in Figure 1, also flows up through the array, but at half the speed of the west $\rightarrow$ east data.

Process (i, j) is executed at time $i + j$ in processor j. By inspection, we see that the array uses n processors, finishing the computation in $2n - 1$ steps. The number of vertices (processes) in a longest directed path in any process dependence graph is a lower bound on the number of steps of any schedule for computing the processes. In our graph, the number of vertices in a longest path is $2n - 1$. This array thus uses a spacetime embedding that is optimal with respect to the number of steps used. Such an embedding is referred to as *time-optimal*.

Table 1: *The multiplied difference table, for $n = 4$.*

$V_{04} \equiv (u_4 - v_0)c_{04} \pmod{m_4}$ $V_{14} \equiv (V_{04} - V_{01})c_{14} \pmod{m_4}$ $V_{24} \equiv (V_{14} - V_{12})c_{24} \pmod{m_4}$ $V_{34} \equiv (V_{24} - V_{23})c_{34} \pmod{m_4}$

$V_{03} \equiv (u_3 - v_0)c_{03} \pmod{m_3}$ $V_{13} \equiv (V_{03} - V_{01})c_{13} \pmod{m_3}$ $V_{23} \equiv (V_{13} - V_{12})c_{23} \pmod{m_3}$

$V_{02} \equiv (u_2 - v_0)c_{02} \pmod{m_2}$ $V_{12} \equiv (V_{02} - V_{01})c_{12} \pmod{m_2}$

$V_{01} \equiv (u_1 - v_0)c_{01} \pmod{m_1}$

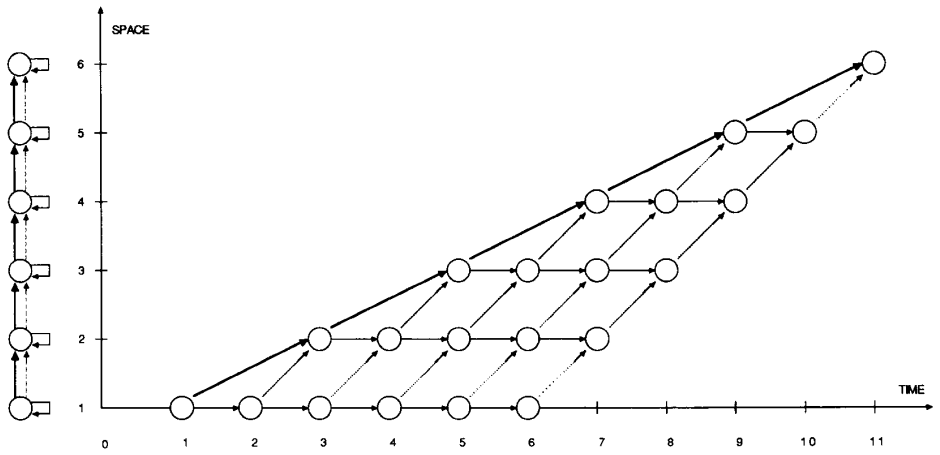

Figure 2: *A spacetime embedding (E_1) of the process dependence dag for Garner's algorithm ($n = 6$). The resultant linear systolic array is time-optimal.*

4 Space-Time Optimal Systolic Arrays

Definition 1 *A graph's embedding is spacetime-optimal when it is space-minimal among those embeddings that are time-optimal.*

We now make a slight modification to the above array, producing a *spacetime-optimal* array. There is unused time on the lower numbered processors. We reschedule the computation done on the upper processors onto these lower processors. More formally, we embed the process dependence graph as follows:

$$\begin{bmatrix} t \\ s \end{bmatrix} := T_1 \begin{bmatrix} i \\ j \end{bmatrix} \quad \text{for} \quad i \leq n - j \; ; \tag{7}$$

$$\begin{bmatrix} t \\ s \end{bmatrix} := T_1 \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ n - j \end{bmatrix} \quad \text{for} \quad i > n - j \; . \tag{8}$$

This embedding, E_2, is illustrated, for $n = 6$, in Figure 3. This design has 2 phases of data movement. In the 1st phase, data moves as in the embedding E_1. As the 1st phase ends, and the 2nd begins, there is a transition: When n is even (as depicted in Figure 3), there are 2 time steps in which the south $\rightarrow$ east data flows in the direction of time: It is in the uppermost processor's memory for these 2 steps. When n is odd, the south $\rightarrow$ east data always moves through the array.

In the 2nd phase, data moves as follows. Data that flows south $\rightarrow$ north in Figure 1, flows down through the array. Data that flows west $\rightarrow$ east in Figure 1, flows in the direction of time: It is remembered in this phase. Data that flows south $\rightarrow$ east in Figure 1, also flows down through this array, but at half the speed of the south $\rightarrow$ north data.

Of those spacetime embeddings that are time-optimal, this embedding also is space-minimal:

Theorem 1 *Embedding E_2 of G is spacetime-optimal.*

Proof The embedding E_2 is identical to E_1 with respect to time: it too is time-optimal. We now argue that the embedding is space-minimal among time-optimal embeddings. Let us focus on the time steps in which all 3 processors are used (which we refer to as the *processor-maximal* time steps). They are time step 6, 7, and 8. In order to reduce the number of processors, during the processor-maximal time steps the nodes scheduled for some processor must be rescheduled onto the other 2 processors. Two processors suffice, for example, if the nodes named (1,5), (1,6), and (2,6) can be rescheduled from processor 2 onto processors 0 and 1. These nodes are in a longest directed path in the process dependence dag. This means that none can be rescheduled for earlier completion without violating a dependence. Neither can they be scheduled for later completion without either violating a

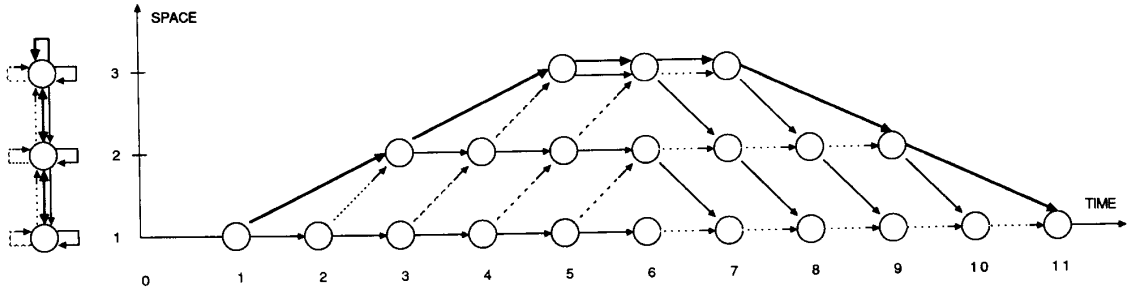

Figure 3: *A spacetime embedding (E_2) of the process dependence dag for Garner's algorithm ($n = 6$). The resultant linear systolic array of $\lceil n/2 \rceil$ processors is spacetime-optimal.*

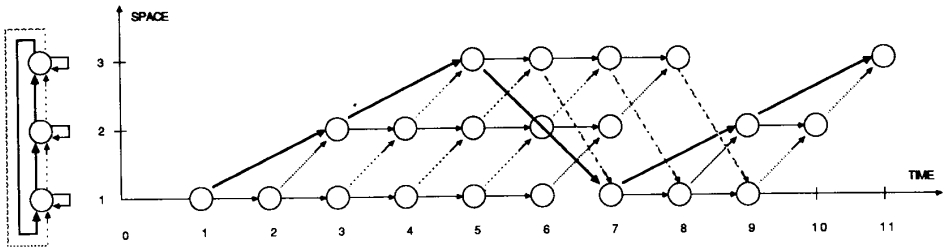

Figure 4: *A spacetime embedding (E_3) of the process dependence dag for Garner's algorithm ($n = 6$). The resultant ring systolic array of $\lceil n/2 \rceil$ processors is spacetime-optimal.*

dependence, or extending the overall completion time, violating time-optimality.In fact, in this dag, every node is on some longest directed path, and hence can be rescheduled onto neither an earlier nor a later time step. In particular, the nodes scheduled for the processor-maximal time steps cannot be rescheduled. The number of processors therefore cannot be reduced: The design is spacetime-optimal.

Any spacetime embedding of this process dependence graph that completes in $2n-1$ cycles, must use at least $\lceil \frac{n}{2} \rceil$ processors. □

Moreover, the nonlinearity of our spacetime transformation is necessary: There does not exist a linear embedding of the initial indices that is spacetime-optimal.

We now present 2 other spacetime-optimal embeddings of the process dependence graph of Figure 1. The first is another variation of the array resulted from embedding E_1. We again reschedule the computation done on the upper processors onto the lower processors. To do this, we connect the endpoints of the linear array, making a *ring* of processors. More formally, we nonlinearly embed the process dependence graph as follows:

$$t := i + j \; ; \tag{9}$$

$$s := j \bmod \lfloor \frac{n}{2} \rfloor \; . \tag{10}$$

This embedding, E_3, is illustrated, for $n = 6$, in Figure 4. Its data flow characteristics are identical to those of embedding E_1, except that the upper processor is attached to the lower processor, and data movement wraps around.

Since this embedding results in a computation of the process dependence graph that uses $2n-1$ steps and $\lceil \frac{n}{2} \rceil$ processors, it too is spacetime-optimal.

Finally, we present a bilateral array in which the south $\rightarrow$ north data of Figure 1 moves up through the array, while the west $\rightarrow$ east data moves down through the array. Such a data movement scheme may be useful, depending on the larger context of which this computation is a part. The spacetime embedding, E_4, is presented in 2 steps:

1. First, we embed the process dependence graph, illustrated in Figure 5, as follows:

$$\begin{bmatrix} t \\ s \end{bmatrix} := T_2 \begin{bmatrix} i \\ j-1 \end{bmatrix} \text{ , where } \quad T_2 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} . \tag{11}$$

   In this spacetime embedding, when processors are used, they are used every other time step.

2. We now compress the spatial extent of this embedding with the following nonlinear transformation:

$$T_2 := \lfloor C \rfloor \text{ , where } \quad C = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \tag{12}$$

$$y := \lfloor C \rfloor x \equiv \lfloor Cx \rfloor \text{ , where } \quad \left\lfloor \begin{bmatrix} i \\ j \end{bmatrix} \right\rfloor \equiv \begin{bmatrix} \lfloor i \rfloor \\ \lfloor j \rfloor \end{bmatrix} . \tag{13}$$

   Processor efficiency (i.e., the percentage of time that a processor is used) is doubled asymptotically by this nonlinear transformation. Figure 6 illustrates the result.

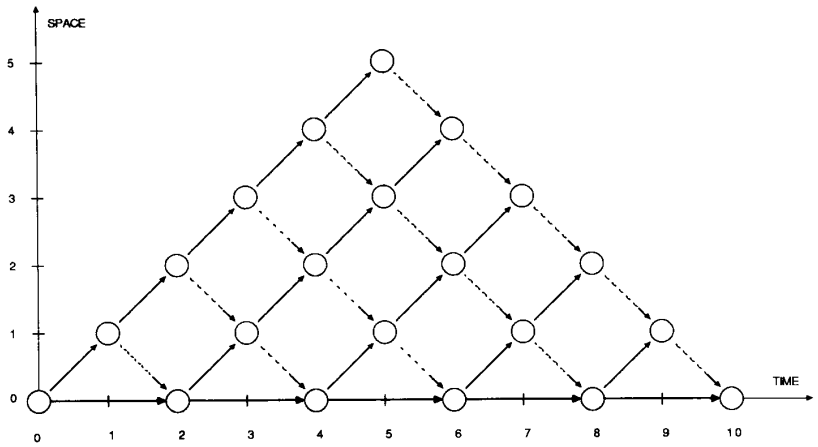

Figure 5: *A spacetime embedding of the process dependence dag for Garner's algorithm ($n = 6$). The resultant bidirectional linear systolic array has $\lceil n \rceil$ processors.*

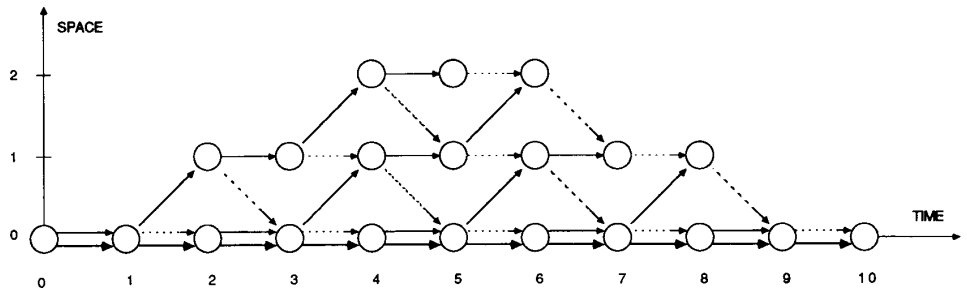

Figure 6: *A spacetime embedding (E_4) of the process dependence dag for Garner's algorithm ($n = 6$). The resultant bidirectional linear systolic array of $\lceil n/2 \rceil$ processors is spacetime-optimal.*

This design has 2 phases of data movement which alternate with each time step. Regardless of the phase, data flowing south $\rightarrow$ east in Figure 1, flows in the direction of time: It is remembered. In phase A, data flowing south $\rightarrow$ north in Figure 1, flows up through the array; data flowing west $\rightarrow$ east in Figure 1, flows in the direction of time. In phase B, data flowing south $\rightarrow$ north in Figure 1, flows in the direction of time; data flowing west $\rightarrow$ east in Figure 1, flows up through the array. Since this second transformation results in an embedding that uses $2n - 1$ steps and $\lceil \frac{n}{2} \rceil$ processors, it too is spacetime-optimal.

5 A Two-Dimensional Array

We now present a *2-d* array for computing the process dependence graph of Figure 1. This is done by embedding the process dependence graph into a *3-d* space. One way to do this is with a linear embedding, E_5:

$$\begin{bmatrix} t \\ s_1 \\ s_2 \end{bmatrix} := \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \qquad (14)$$

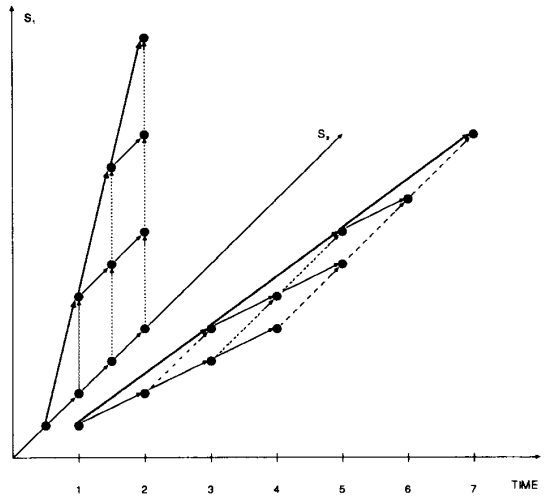

Figure 7: *A spacetime embedding (E_5) of the process dependence dag for Garner's algorithm ($n = 4$). The resultant triangular systolic array of $n(n+1)/2$ processors has a period of one time step.*

Figure 7 illustrates the result. In this array, there is a processor for every process (in the process dependence graph). The flow of data between processors corresponds to the arcs in the process dependence graph. Every processor whose corresponding vertex in Figure 1 has indices whose sum is k executes its process at step k. Execution completes after $2n - 1$ steps. This embedding has the property that each processor is used exactly once per execution of the process dependence graph. The array can start executing a new process dependence graph every step. Figure 8 is intended to illustrate the pipeline quality of this array; it shows 2 process dependence graphs embedded in spacetime such that execution of the 2nd starts 1 step after the 1st: Executing k such process dependence graphs uses $2n + k - 2$ steps.

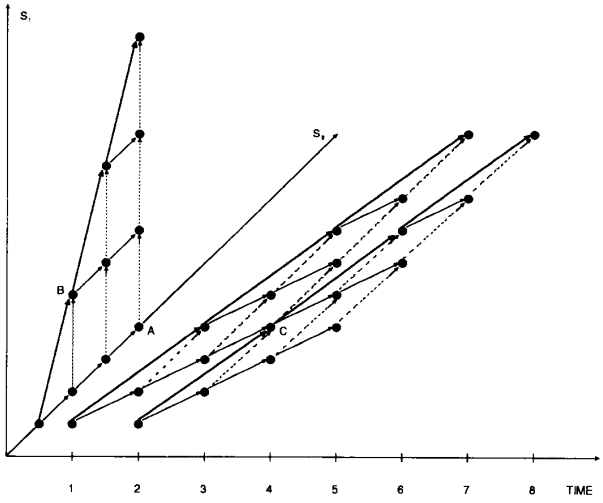

Figure 8: *A spacetime embedding of two copies of the process dependence dag for Garner's algorithm ($n = 4$). There are two distinct processes that appear as though they are embedded in the same point (C) of spacetime. They, in fact, are embedded in distinct spatial coordinates. The process from the first copy of the dag executes on processor A while the process from the second copy executes on processor B.*

6 A Decomposition Strategy

One of the well-known constraints in the VLSI implementation of algorithms is what we call the *extent problem*. Processor arrays are of fixed extent, and we must use them effectively. The size of the problem rarely is fixed beforehand. There are 3 cases depending on whether the size of the problem is smaller than, equal to, or larger than the array:

Equal to: In this case there should be no difficulty in using the processor array effectively.

Smaller than: In this case, much depends on the structure of the algorithm. In most cases this situation is handled easily, by disabling some units (assuming this is possible). Hence, a less efficient use of the array results. There are situations, however, in which this is not possible. For example, when a processor array recirculates data, or uses wrap-around connections (e.g., Cannon's array for matrix product [5,11]). Again, the situation is handled by suitably increasing the size of the problem with data elements that do not affect the result (e.g., identity elements with respect to the size of the problem). In any case, the array is used with less than full efficiency.

Larger than: This is the most interesting situation. Heller's corollary [16] to Murphy's law states:

- No matter what special-purpose device is available, there is a problem too large for it.
- The problem will manifest itself only after the device is acquired and can no longer be modified.
- The problem cannot be ignored.

We thus are advised to consider this case in some detail. The difficulty of this case is directly proportional to the algorithm's degree of decomposability. At best, the algorithm can be decomposed into blocks whose size equals the extent of the processor array. In this case, partial results can be composed using a processor array of the same size (perhaps the same array). Work on general methods for this problem has been reported by Moldovan and Fortes [30]. Problem-specific methods also have been given considerable attention (see, e.g., Schreiber and Kuekes [35] and Schreiber [34]).

We now formulate the mixed-radix conversion problem as a system of linear congruence equations. We then show that, in this formulation, the problem can be decomposed into smaller mixed-radix conversion problems, and small inner-products (of vectors modulo a number from the moduli set).

Recall equation (2)

$$u = v_0 + v_1 m_0 + v_2 m_0 m_1 + \cdots + v_n m_0 m_1 \cdots m_{n-1} .$$

The coefficients v_i, for $0 \le i \le n$, can be obtained by solving

$$\begin{aligned}
v_0 &= u_0 \pmod{m_0} \\
v_0 + v_1 m_0 &= u_1 \pmod{m_1} \\
v_0 + v_1 m_0 &= u_2 \pmod{m_2} \\
&\vdots \\
v_0 + v_1 m_0 + \cdots + v_n m_0 m_1 \cdots m_{n-1} &= u_n \pmod{m_n}
\end{aligned}$$

We represent the above linear system of equations in matrix notation as

$$\mathbf{M} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix} \pmod{\begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_n \end{bmatrix}} , \qquad (15)$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & m_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & m_0 & m_0 m_1 & \cdots & m_0 m_1 \cdots m_{n-1} \end{bmatrix} ,$$

(or more compactly, $\mathbf{Mv} = \mathbf{u} \pmod{\mathbf{m}}$). This system is solved by applying Garner's algorithm to the integers $u_0, u_1, \ldots, u_n$ using the moduli set $m_0, m_1, \ldots, m_n$. We denote this operation with

$$\textbf{Garner\_Algorithm}\,[u_j, m_j; 0 \le j \le n] ,$$

producing the set of multiplied differences $v_0, v_1, \ldots, v_n$.

Assuming $n+1 = (p+1)(q+1)$, we partition matrix $\mathbf{M}$ into $(p+1) \times (p+1)$ dimension matrices $\mathbf{M}_{ij}$ for $0 \le j \le i \le q$, and vectors $\mathbf{v}$, $\mathbf{u}$, and $\mathbf{m}$ into $(p+1)$ dimension vectors $\mathbf{v}_i, \mathbf{u}_i, \mathbf{m}_i$ for $0 \le i \le q$, respectively. Thus,

$$\begin{bmatrix} \mathbf{M}_{00} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{M}_{10} & \mathbf{M}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{M}_{20} & \mathbf{M}_{21} & \mathbf{M}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{M}_{q0} & \mathbf{M}_{q1} & \mathbf{M}_{q2} & \cdots & \mathbf{M}_{qq} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_q \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_q \end{bmatrix} \left(\bmod \begin{bmatrix} \mathbf{m}_0 \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_q \end{bmatrix} \right) \qquad (16)$$

where matrices $\mathbf{M}_{ii}$ are lower-triangular and the zero matrices are of the same dimension as the $\mathbf{M}_{ij}$ matrices. System (16) can be solved via forward substitution.

Procedure Partition_and_Solve

1: $\mathbf{v}_0 = \text{Solve}[\mathbf{M}_{00}\mathbf{v}_0 = \mathbf{u}_0 \pmod{\mathbf{m}_0}]$
FOR $i = 1$ TO q DO
BEGIN
2: $\mathbf{e}_i = \mathbf{u}_i$
FOR $j = 0$ TO $i - 1$ DO
BEGIN
3: $\mathbf{e}_i = \mathbf{e}_i - \mathbf{M}_{ij}\mathbf{v}_j \pmod{\mathbf{m}_i}$
END FOR
4: $\mathbf{v}_i = \text{Solve}[\mathbf{M}_{ii}\mathbf{v}_i = \mathbf{e}_i \pmod{\mathbf{m}_i}]$
END FOR
END PROCEDURE

We consider the equation $\mathbf{M}_{00}\mathbf{v}_0 = \mathbf{u}_0 \pmod{\mathbf{m}_0}$, i.e.,

$$\mathbf{M} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_p \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_p \end{bmatrix} \pmod{\begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_p \end{bmatrix}},$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & m_0 & 0 & \cdots & 0 \\ 1 & m_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & m_0 & m_0 m_1 & \cdots & m_0 m_1 \cdots m_{p-1} \end{bmatrix}$$

A comparison of this system to (15) shows that the solution of this system of equations is equivalent to application of Garner's algorithm to the set (u_j, m_j), for $0 \le j \le p$. That is,

$$\mathbf{Solve}[\mathbf{M}_{00}\mathbf{v}_0 = \mathbf{u}_0 \pmod{\mathbf{m}_0}]$$

$$\equiv \mathbf{Garner\_Algorithm}\,[u_j, m_j; 0 \le j \le p]\,.$$

Now, we consider solving $\mathbf{M}_{ii}\mathbf{v}_i = \mathbf{e}_i \pmod{\mathbf{m}_i}$ for $1 \le i \le q$. Observe that $\mathbf{M}_{ii} =$

$$\begin{bmatrix} m_0m_1\cdots m_{i(p+1)-1} & 0 & \cdots & 0 \\ m_0m_1\cdots m_{i(p+1)-1} & m_0m_1\cdots m_{i(p+1)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ m_0m_1\cdots m_{i(p+1)-1} & m_0m_1\cdots m_{i(p+1)} & \cdots & m_0m_1\cdots m_{i(p+1)+p-1} \end{bmatrix}$$

$$= \mu \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & m_{i(p+1)} & \cdots & 0 \\ 1 & m_{i(p+1)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & m_{i(p+1)} & \cdots & m_{i(p+1)}m_{i(p+1)+1}\cdots m_{i(p+1)+p-1} \end{bmatrix}$$

where $\mu = m_0m_1\cdots m_{i(p+1)-1}$. So, solving $\mathbf{M}_{ii}\mathbf{v}_i = \mathbf{e}_i \pmod{\mathbf{m}_i}$ is equivalent to applying Garner's algorithm to vectors $(\mathbf{d}_i, \mathbf{m}_i)$ where

$$\mathbf{d}_i = \left(m_0m_1\cdots m_{i(p+1)-1}\right)^{-1}\mathbf{e}_i \pmod{\mathbf{m}_i}\,,$$

or more explicitly

$$\begin{bmatrix} d_{i(p+1)} \\ d_{i(p+1)+1} \\ \vdots \\ d_{i(p+1)+p} \end{bmatrix} = \mathbf{C}\begin{bmatrix} e_{i(p+1)} \\ e_{i(p+1)+1} \\ \vdots \\ e_{i(p+1)+p} \end{bmatrix} \left(\mathrm{mod}\begin{bmatrix} m_{i(p+1)} \\ m_{i(p+1)+1} \\ \vdots \\ m_{i(p+1)+p} \end{bmatrix}\right)$$

where

$$\mathbf{C} = \begin{bmatrix} c_{0,i(p+1)}c_{1,i(p+1)}\cdots c_{i(p+1)-1,i(p+1)} \\ c_{0,i(p+1)+1}c_{1,i(p+1)+1}\cdots c_{i(p+1)-1,i(p+1)+1} \\ \vdots \\ c_{0,i(p+1)+p}c_{1,i(p+1)+p}\cdots c_{i(p+1)-1,i(p+1)+p} \end{bmatrix}$$

and where the c_{ij} are the inverse terms defined by (3). Thus,

$$\mathbf{Solve}[\mathbf{M}_{ii}\mathbf{v}_i = \mathbf{u}_i \pmod{\mathbf{m}_i}]$$

$$\equiv \mathbf{Garner\_Algorithm}\,[d_j, m_j; i(p+1) \le j \le i(p+1)+p]\,.$$

The next step is the construction of the elements of matrices $\mathbf{M}_{ij}$ for $0 \le j < i \le q$. Since $\mathbf{M}_{ij}$ is a matrix with repeated row

$$m_0m_1\cdots m_{i(p+1)-1}\,,\; m_0m_1\cdots m_{i(p+1)}\,,\; \cdots\,,\; m_0m_1\cdots m_{i(p+1)+p-1}\,,$$

we can construct $\mathbf{M}_{ij}$ by first computing the prefix product of the terms

$$m_{i(p+1)-1}\,,\; m_{i(p+1)}\,,\; m_{i(p+1)+1}\,,\; \cdots\,,\; m_{i(p+1)+p-1}$$

using a linear array of size $p+1$, and then multiplying every element above by $m_0m_1\cdots m_{(i-1)(p+1)+p-1}$ which is the element in the last column of matrix $\mathbf{M}_{i-1,j}$.

In sum, our decomposition of the mixed-radix conversion problem leads to the following computations:

Step 1 a mixed-radix problem of size $p+1$;

Step 3.1 construction of matrices $\mathbf{M}_{ij}$, which can be computed on a linear array of size $(p+1)$ when the entries of matrix $\mathbf{M}_{i-1,j}$ are used for the computation of the entries of $\mathbf{M}_{ij}$;

Step 3.2 matrix-vector product operation $\mathbf{M}_{ij}\mathbf{v}_j$ which also can be computed on a $(p+1)$ linear array (see, e.g., [20]);

Step 4.1 construction of the terms $c_{0,i(p+1)+k}c_{1,i(p+1)+k}\cdots c_{i(p+1)-1,i(p+1)+k}$ for $0 \le k \le p$, which has a $p+1$ mesh as its process dependence graph when the result from the *(i-1)st* step is used in the *ith* step;

Step 4.2 a mixed-radix problem of size $p+1$.

Our decomposition thus yields process dependence graphs that can be embedded in spacetime, producing linear arrays of size $p+1$.

7 Conclusions

We summarize the spacetime embeddings, and the resulting systolic arrays, in the following table:

Embedding	Space	Time	Optimality	Figure
E_1	n	$2n-1$	time	2
E_2, E_3, E_4	$\lceil \frac{n}{2} \rceil$	$2n-1$	spacetime	3, 4, 6
E_5	$\frac{n(n+1)}{2}$	$2n-1$	period	7

The systolic array resulting from the E_5 embedding is optimal with respect to period: it is completely pipelined.

Several implementations of the Garner algorithm are given in the literature. These implementations are mostly hardware oriented, using table lookup techniques. They thus put restrictions on the size and cardinality of the moduli [37,17,9]. The systolic arrays described in this paper have no such restrictions. Each of the various design options have a place, indicating the potential usefulness of software implementations on a programmable systolic/wavefront array. Examples of such software-oriented systolic computing systems include 1) an array of Transputers [1] [18], 2) the Warp [1], and 3) the Matrix-1 [14].

An RNS simplifies large-number computations by resolving a problem into a set of parallel, independent computations performed in modularly formed channels. Consequently, combining RNS with systolic arrays is particularly suitable for a VLSI implementation. A fault-tolerant systolic array using redundant residue system was reported in [10]. The systolic arrays in this paper are also suitable for implementing fault-tolerant computing systems employing residue arithmetic.

A systolic implementation of the Aitken algorithm for iterated interpolation is reported by McKeown in [26]. McKeown's work is extended by Cappello, Gallopoulos, and Koç [6]. They give systolic versions of Newton and Hermite polynomial interpolation using the algorithms of Aitken and Neville. A decomposition strategy, similar to the one in §6, for solving large Newton interpolation problems on smaller systolic arrays is reported by the same authors [7].

[1] Transputer is a trademark of INMOS, Ltd.

References

[1] A. M. Annaratone, E. Arnould, T. Gross, H-T Kung, M. Lam, O. Menzilcioglu, J. Webb, "The WARP Computer: Architecture, Implementation, and Performance," *IEEE Trans. on Computers*, Vol. C-36, No. 12, pp. 1523-1538, December 1987.

[2] E. H. Bareiss, "Computational Solutions of Matrix Problems Over an Integral Domain," *J. Inst. Maths. Applics.*, No. 10, pp. 68–104, 1972.

[3] F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems," *IEEE Transactions on Computers*, Vol. C-22, No. 3, pp. 307–315, March 1973.

[4] I. Borosh and A. S. Fraenkel, "Exact Solutions of Linear Equations with Rational Coefficients by Congruence Techniques," *Mathematics of Computation*, Vol. 20, No. 93, pp. 107–112, January 1966.

[5] L. E. Cannon, *A Cellular Computer to Implement the Kalman Filtering Algorithm*, Ph. D. Dissertation, Montana State University, 1969.

[6] P. R. Cappello, E. Gallopoulos, and Ç. K. Koç, "Systolic Computation of Interpolating Polynomials," Technical Report No. TRCS88-20, Department of Computer Science, University of California, Santa Barbara, August 1988.

[7] Ç. K. Koç, P. R. Cappello, and E. Gallopoulos, "Decomposing Polynomial Interpolation for Systolic Arrays," Technical Report No. TRCS89-1, Department of Computer Science, University of California, Santa Barbara, January 1989.

[8] P. R. Cappello and K. Steiglitz, "Unifying VLSI Array Designs with Linear Transformations of Space-Time," in *Advances in Computer Research*, edited by F. P. Preparata, Vol. 2, pp. 23-65, JAI Press, 1984.

[9] N. B. Chakraborti, J. S. Soundararajan, and A. L. N. Reddy, "An Implementation of Mixed-Radix Conversion for Residue Number Applications," *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp. 762–764, Augusts 1986.

[10] R. J. Cosentino, "Fault Tolerance in a Systolic Residue Arithmetic Processor Array," *IEEE Trans. on Computers*, Vol. 37, No. 7, pp. 886–890, July 1988.

[11] E. Dekel, D. Nassimi, and S. Sahni, "Parallel Matrix and Graph Algorithms," *SIAM Journal on Computing*, Vol. 10, No. 4, pp. 657-675, November 1981.

[12] M. H. Etzel and W. K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," *IEEE Transactions on Acoust., Speech, Signal Processing*, Vol. ASSP-28, No. 538–545, October 1980.

[13] J. A. B. Fortes and D. I. Moldovan, "Parallelism Detection and Algorithm Transformation Techniques useful for VLSI Architecture Design", *J. Parallel Distrib. Comput.*, Vol. 2, pp. 277-301, August 1985.

[14] D. E. Foulser and R. Schreiber, "The Saxpy Matrix-1: A General-Purpose Systolic Computer," *IEEE Computer*, Vol. 20, No. 7, pp. 35-43, July 1987.

[15] H. L. Garner, "The Residue Number System," *IRE Trans. Electronic Computers*, Vol. EL-8, No. 6, pp. 140–147, June 1959.

[16] D. Heller, "Partitioning Big Matrices for Small Systolic Arrays," in *VLSI and Modern Signal Processing*, edited by S. Y. Kung, H. J. Whitehouse and T. Kailath, pp. 185-199, Prentice-Hall, 1985.

[17] C. H. Huang, "A Fully Parallel Mixed-Radix Conversion Algorithm for Residue Number Applications," *IEEE Trans. on Computers*, Vol. C-32, No. 4, pp. 398–402, April 1983.

[18] *IMS T800 Transputer*, Rpt. 72 TRN 117 01, INMOS Ltd., Almondsbury, Bristol, UK, November 1986.

[19] D. E. Knuth, *The Art of Computer Programming*, Volume 2, *Seminumerical Algorithms*, 2nd Edition, Addison-Wesley Publishing Company, 1981.

[20] H. T. Kung and C. E. Leiserson "Algorithms for VLSI Processor Arrays," in *Introduction to VLSI Systems*, by C. Mead and L. Conway, pp. 271-292, Addison-Wesley, 1980

[21] J. D. Lipson, *Elements of Algebra and Algebraic Computing*, Addison-Wesley Publishing Company, 1981.

[22] J. D. Lipson, "Chinese Remainder and Interpolation Algorithms," *Proc. 2nd Symp. Symbolic Algebraic Manipulation*, pp. 372–391, 1971.

[23] G. Mackiw, *Applications of Abstract Algebra*, John-Wiley & Sons, Inc., 1985.

[24] D. Mandelbaum, "Error Correction in Residue Arithmetic," *IEEE Trans. on Computers*, Vol. C-21, No. 6, pp. 538–545, June 1972.

[25] M. T. McClellan, "The Exact Solution of Systems of Linear Equations with Polynomial Coefficients," *Journal of the ACM*, Vol. 20, No. 4, pp. 563–588, October 1973.

[26] G. P. McKeown, "Iterated Interpolation using a Systolic Array," *ACM Transactions on Mathematical Software*, Vol. 12, No. 2, pp. 162-170, June 1986.

[27] W. L. Miranker, and A. Winkler, "Spacetime Representations of Computational Structures," *Computing*, Vol. 32, pp. 93-114, 1984.

[28] D. I. Moldovan, "On the Analysis and Synthesis of VLSI Algorithms," *IEEE Transactions on Computers*, Vol. C-31, pp. 1121-1126, November 1982.

[29] D. I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays," *Proc. IEEE* , Vol. 71, No. 1, pp. 113-120, January 1983.

[30] D. I. Moldovan and J. A. B. Fortes, "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays," *IEEE Transactions on Computers*, Vol. C-35, No. 1, pp. 1-12, January 1986.

[31] P. Quinton, "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations", *Proc. 11th Ann. Symp. on Computer Architecture*, pp. 208-214, 1984.

[32] S. K. Rao, *Regular Iterative Algorithms and Their Implementation on Processor Arrays*, Ph.D. Dissertation, Stanford University, October, 1985.

[33] I. J. Schoenberg, "The Chinese Remainder Problem and Polynomial Interpolation," Tech. Rep. No. 2954, Mathematics Research Center, University of Wisconsin-Madison, Auguts 1986.

[34] R. Schreiber, "Solving Eigenvalue and Singular Value Problems on an Undersized Systolic Array", *SIAM J. on Scientific and Statistical Computing*, Vol. 7, No. 2, pp. 441-451, April 1986.

[35] R. Schreiber and P. J. Kuekes, "Systolic Linear Algebra Machines in Digital Signal Processing." In *VLSI & Modern Signal Processing*, edited by S-Y Kung, H. J. Whitehouse, and T. Kailath, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[36] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill, 1967.

[37] F. J. Taylor, "An Efficient Residue-To-Decimal Converter," *IEEE Trans. on Circuits and Systems*, Vol. CAS-28, No. 12, pp. 1164–1169, December 1981.

[38] F. J. Taylor, "Residue Arithmetic: A Tutorial with Examples," *IEEE Computer Mag.*, Vol. 17, pp. 50-62, May 1984.

[39] C. N. Zhang, B. Shirazi, and D. Y. Y. Yun, "Parallel Designs for Chinese Remainder Theorem," *Proc. of Intern. Conf. on Parallel Processing*, pp. 557–559, 1987.