Systolic Computation of Interpolating Polynomials*

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

E. Gallopoulos‡
Center for Supercomputing Research and Development
University of Illinois at Urbana Champaign
Urbana, IL 61801

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

Abstract

Several time-optimal and spacetime-optimal systolic arrays are presented for computing a process dependence graph corresponding to the Aitken algorithm. It is shown that these arrays also can be used to compute the generalized divided differences, i.e., the coefficients of the Hermite interpolating polynomial. Multivariate generalized divided differences are shown to be efficiently computed on a 2-dimensional systolic array. The techniques also are applied to the Neville algorithm, producing similar results.

AMS Subject Classification: 65D05, 68Q80, 68N99.

Keywords: Newton interpolation, Hermite interpolation, Aitken's algorithm, Neville's algorithm, systolic array.

*This paper was presented at the Fourth SIAM Conference on Parallel Processing for Scientific Computing, Chicago, Illinois, December 11–13, 1989.

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

‡Supported by the National Science Foundation under Grants Nos. US NSF-MIP-8410110, US NSF DCR85-09970, and US NSF CCR-8717942, and by AT&T Grant AT&T AFFL67Sameh.

1 Introduction

McKeown [12] establishes a rationale for performing iterated interpolation using a systolic array:

> "Another consequence of developments in VLSI technology is that interpolation in a table as a means of function approximation could well rehabilitate itself, at least for certain 'difficult' functions. ... the continually falling cost and increasing capacity of semiconductor memory chips could soon make it feasible to consider storing on such a chip a table of values for a function requiring complex direct computation. The set of 'chip tables' together with a systolic array for iterated linear interpolation could thus have attractive possibilities."

To obtain a cost-effective design, we must consider the trade-off between the cost of chip tables plus the cost of the systolic array versus the cost of the evaluation of this 'difficult' function at many points using a fast arithmetic processor. However, when we have access to the values of the function at certain points but not to the function itself, systolic arrays will unquestionably provide parallelism. Polynomial interpolation is widely used for curve and surface fitting or, in general, the fitting of multidimensional curves to scattered data. In many applications of the surface-fitting techniques, the aim is to use the data to construct a *contour map* of the unknown function [20]. Since the function is known only at the data points, one must construct a contour map for one of the fitted surfaces. Contour map construction has applications in the oil industry (petroleum explorations), geological maps, and cardiology (heart potentials).

McKeown presents a systolic implementation of Aitken's method of iterated interpolation. In this paper, we consider a systolic version of Newton and Hermite polynomial interpolation using the algorithms of Aitken and Neville. This work builds on that of McKeown by:

1. presenting several alternative systolic implementations of Aitken's algorithm, some of which are optimal;
2. applying these implementations to Neville's algorithm;
3. presenting a 2-level systolic array for computing generalized divided differences;
4. presenting a systolic scheme for the multivariate case;
5. showing that the interpolating polynomials can be implicitly evaluated by only reprogramming the processors.

2 Newton and Hermite Polynomial Interpolation

The polynomial interpolation problem is defined as follows:

Input: $n+1$ pairs of points $(x_i, f_i) \in F \times F$ for $0 \leq i \leq n$, where F is a field.

Problem: Find a polynomial $p_n(x) \in F[x]$ of degree n such that $p_n(x_i) = f_i$ for $0 \leq i \leq n$.

The interpolating polynomial of degree n exists and is unique, provided that $x_i \neq x_j$ for $i \neq j$. Polynomial interpolation is applied to the problem of function approximation. If the points f_i are the values of a function $f(x)$ at the points x_i for $0 \leq i \leq n$, then the value $p_n(\bar{x})$ is used to approximate the value $f(\bar{x})$ for some $\bar{x} \neq x_i$ for $0 \leq i \leq n$, but which usually is in the same interval as the x_is.

The polynomial of degree n passing through the points (x_i, f_i) for $0 \leq i \leq n$ is given in the Newton form as

$$\begin{aligned} p_n(x) &= f_0 + f_{01}(x - x_0) + f_{012}(x - x_0)(x - x_1) + \cdots \\ &\quad + f_{012\cdots n}(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \ , \end{aligned} \tag{1}$$

where the coefficients $f_{012\cdots i}$ are called the *divided differences*. The coefficients of the Newton polynomial interpolating the function $f(x)$ at the node points x_i for $0 \leq i \leq n$ can be computed using the well-known algorithms of Neville and Aitken [9, 11]. These algorithms use recursions to compute what is known as the *divided difference table* whose diagonal entries are the desired coefficients of the Newton interpolating polynomial. The Aitken algorithm uses the recursion:

$$f_{012\cdots i,k} = \frac{f_{012\cdots i} - f_{012\cdots i-1,k}}{x_i - x_k} \tag{2}$$

for $0 \leq i \leq n-1$ and $i+1 \leq k \leq n$. The Neville algorithm uses the recursion:

$$f_{i,i+1,\ldots,k} = \frac{f_{i,i+1,\ldots,k-1} - f_{i+1,i+2,\ldots,k}}{x_i - x_k} \tag{3}$$

for $0 \leq i \leq n-1$ and $i+1 \leq k \leq n$.

For Hermite interpolation (the confluent case of Newton interpolation), we construct a polynomial that passes through the values of $f(x)$, as well as its derivatives, at the node points x_i. Let $\mathbf{m}$ be an ordered collection of n natural numbers: $\mathbf{m} = (m_0, m_1, \ldots, m_n)$. We denote the kth derivative of $f(x)$ evaluated at x_i by $f^{(k)}(x_i)$, with $f^{(0)}(x_i) = f(x_i) = f_i$. If we are given

$$x_i, f_i, f^{(1)}(x_i), f^{(2)}(x_i), \ldots, f^{(m_i)}(x_i)$$

for all $0 \leq i \leq n$, then we can construct the Hermite polynomial to interpolate this data set. That is, we can construct

$$p_N^{(k_i)}(x_i) = f^{(k_i)}(x_i) \text{ for } \quad 0 \leq i \leq n \text{ and } \quad 1 \leq k_i \leq m_i \ .$$

The Hermite interpolating polynomial also is unique, provided $x_i \neq x_j$ for $i \neq j$ as is the case for Newton interpolation [2]. The degree of the Hermite interpolating polynomial is

$$N = m_0 + m_1 + \cdots + m_{n-1} + m_n - 1 \ . \tag{4}$$

The Hermite interpolating polynomial can be given as

$$\begin{aligned} p_N(x) &= f_0 + f_{0^2}(x - x_0) + f_{0^3}(x - x_0)^2 + \cdots + f_{0^{m_0}}(x - x_0)^{m_0 - 1} + \\ &\quad f_{0^{m_0}1}(x - x_0)^{m_0} + f_{0^{m_0}1^2}(x - x_0)^{m_0}(x - x_1) + \cdots + f_{0^{m_0}1^{m_1}}(x - x_0)^{m_0}(x - x_1)^{m_1 - 1} + \\ &\quad f_{0^{m_0}1^{m_1}2}(x - x_0)^{m_0}(x - x_1)^{m_1} + f_{0^{m_0}1^{m_1}2^2}(x - x_0)^{m_0}(x - x_1)^{m_1}(x - x_2) + \cdots + \cdots + \\ &\quad f_{0^{m_0}1^{m_1}2^{m_2}\cdots n^{m_n}}(x - x_0)^{m_0}(x - x_1)^{m_1}(x - x_2)^{m_2}\cdots(x - x_n)^{m_n - 1} \ . \end{aligned} \tag{5}$$

The simplest instance of the Hermite interpolating polynomial is when $\mathbf{m} = (1, 1, \ldots, , 1)$, which corresponds to the Newton interpolating polynomial. The coefficients of (5) are referred to as the *generalized divided differences* [9, 22, 11]. The Aitken and Neville recursions can be used to compute the generalized divided differences, provided that we avoid division by zero in the recursion formulae (2) and (3) by defining

$$f_{iii\cdots i} = \frac{1}{(k-1)!} f^{(k-1)}(x_i) \ , \tag{6}$$

where the subscript i is repeated k times [2]. We also denote the generalized divided difference (6) by f_{i^k}.

The Aitken algorithm for generalized divided differences uses the recursion:

$$f_{0^{a_0}\ldots i^{a_i},k^{a_k}} = \frac{f_{0^{a_0}\ldots i^{a_i},k^{a_k-1}} - f_{0^{a_0}\ldots i^{a_i-1},k^{a_k}}}{x_i - x_k} \tag{7}$$

for $0 \le i \le n-1$ and $i+1 \le k \le n$; the corresponding Neville algorithm uses the recursion:

$$f_{i^{a_i}\ldots k^{a_k}} = \frac{f_{i^{a_i}\ldots k^{a_k-1}} - f_{i^{a_i-1}\ldots k^{a_k}}}{x_i - x_k} \tag{8}$$

for $0 \le i \le n-1$ and $i+1 \le k \le n$ where $a_i \le m_i$ for $0 \le i \le n$. The recursion formulae (7) and (8) specialize to (2) and (3) respectively when $\mathbf{m} = (1, 1, \ldots, 1)$.

3 Aitken Algorithm

In this section we give the Aitken algorithm in a Pascal-like notation which allows us to construct its process dependence graph. To illustrate, let $n = 3$ and $\mathbf{m} = (2, 2, 2, 2)$. The input data, then, consists of the function and derivative values x_i, f_i, f_{ii} for $0 \le i \le 3$. The following matrices, $A_{ij} \in F^{m_i \times m_j}$, contain the generalized divided differences for $0 \le i \le 2$ and $i < j \le 3$ as follows:

$$A_{03} = \begin{bmatrix} f_{033} & f_{0033} \\ f_{03} & f_{003} \end{bmatrix}, \quad A_{13} = \begin{bmatrix} f_{00133} & f_{001133} \\ f_{0013} & f_{00113} \end{bmatrix}, \quad A_{23} = \begin{bmatrix} f_{0011233} & f_{00112233} \\ f_{001123} & f_{0011223} \end{bmatrix},$$

$$A_{02} = \begin{bmatrix} f_{022} & f_{0022} \\ f_{02} & f_{002} \end{bmatrix}, \quad A_{12} = \begin{bmatrix} f_{00122} & f_{001122} \\ f_{0012} & f_{00112} \end{bmatrix},$$

$$A_{01} = \begin{bmatrix} f_{011} & f_{0011} \\ f_{01} & f_{001} \end{bmatrix}.$$

We denote an entry of the above matrices with $A_{ij}(p, q)$ where $1 \le p \le m_i$ and $1 \le q \le m_j$. For notational convenience we assume that the index p increases from left to right, and the index q increases from bottom to top.

In the general case, the input data is given as (x_i, f_{i^k}) for $0 \le i \le n$ and $1 \le k \le m_i$. The following procedure given in a Pascal-like notation computes $A_{ij}(p, q)$ for all $0 \le i \le n-1$, $1 \le j \le n$, and $1 \le p \le m_i$, $1 \le q \le m_j$ using recursion (7).

Procedure Aitken
Input: (x_i, f_{i^k}) for $0 \le i \le n$ and $1 \le k \le m_i$.
Output: $A_{ij}(p, q)$ for $0 \le i \le n-1$, $1 \le j \le n$, $1 \le p \le m_i$, and $1 \le q \le m_j$.
BEGIN
FOR $i = 0$ TO $n-1$ DO
  FOR $j = i+1$ TO n DO
    FOR $p = 1$ TO m_i DO
      FOR $q = 1$ TO m_j DO
        $A_{ij}(p, q) = \frac{A_{ij}(p,q-1) - A_{ij}(p-1,q)}{x_i - x_j}$.
END PROCEDURE

In the above procedure, p and q may take the value 0, corresponding to boundary and interface values of the Aitken table. When this happens, that value of the matrix entry is replaced with the appropriate one of the following:

$$\begin{aligned}
A_{0j}(p,0) &= f_{0^p} \text{ for } \quad 1 \le p \le m_0 \ ; \\
A_{ij}(p,0) &= A_{i-1,i}(m_{i-1},p) \text{ for } \quad 1 \le i \le n-1 \ , \quad 2 \le j \le n \ , \text{ and } \quad 1 \le p \le m_i \ ; \\
A_{0j}(0,q) &= f_{j^q} \text{ for } \quad 1 \le j \le n \text{ and } \quad 1 \le q \le m_j \ ; \\
A_{ij}(0,q) &= A_{i-1,j}(m_{i-1},q) \ , \quad 1 \le i \le n-1 \ , \quad 2 \le j \le n \ , \text{ and } \quad 1 \le q \le m_j \ .
\end{aligned}$$

The outputs of this procedure are the generalized divided differences:

$$A_{ij}(p,q) \quad = \quad f_{0^{m_0}1^{m_1}2^{m_2}\ldots i^p j^q}$$

for $0 \le i \le n-1$, $1 \le j \le n$, and $1 \le p \le m_i$, $1 \le q \le m_j$. The coefficients of the Hermite interpolating polynomial are the entries of the matrices $A_{i-1,i}$ for $1 \le i \le n$, and expressed in this notation as

$$\begin{aligned}
f_{0^{m_0}1^q} &= A_{01}(m_0,q) \text{ for } \quad 1 \le q \le m_1 \ ; \\
f_{0^{m_0}1^{m_1}2^q} &= A_{12}(m_1,q) \text{ for } \quad 1 \le q \le m_2 \ ; \\
&\vdots \\
f_{0^{m_0}1^{m_1}2^{m_2}\ldots n^q} &= A_{n-1,n}(m_{n-1},q) \text{ for } \quad 1 \le q \le m_n \ .
\end{aligned}$$

4 Process Dependence Graph of the Aitken Algorithm

The data dependences among the entries in the divided difference tables computed by the Aitken and Neville algorithms lend themselves to systolic implementation. In order to exploit the properties of these tables we look at the algorithms to compute the entries. This allows us to form the em process dependence graph of each algorithm to compute the divided differences.

We start with $n = 4$ and $\mathbf{m} = (1,1,1,1,1)$. In this case, the matrices A_{ij} are simply scalars. In order to see the functional dependence of each entry on the previously computed entries and on the node points x_i in the table, we use the recursion (2) and fill the Aitken table as follows:

$$A_{04} = \tfrac{f_0 - f_4}{x_0 - x_4} \quad A_{14} = \tfrac{A_{01} - A_{04}}{x_1 - x_4} \quad A_{24} = \tfrac{A_{12} - A_{14}}{x_2 - x_4} \quad A_{34} = \tfrac{A_{23} - A_{24}}{x_3 - x_4}$$

$$A_{03} = \tfrac{f_0 - f_3}{x_0 - x_3} \quad A_{13} = \tfrac{A_{01} - A_{03}}{x_1 - x_3} \quad A_{23} = \tfrac{A_{12} - A_{13}}{x_2 - x_3}$$

$$A_{02} = \tfrac{f_0 - f_2}{x_0 - x_2} \quad A_{12} = \tfrac{A_{01} - A_{02}}{x_1 - x_2}$$

$$A_{01} = \tfrac{f_0 - f_1}{x_0 - x_1}$$

In the denominator terms, $x_i - x_j$, the node point x_i is repeated along a column whereas the node point x_j is repeated along a row. The positions of the numerator terms, (e.g. the divided difference terms) are similar. First, a divided difference term of the form $A_{i-1,i}$ is computed on the diagonal, then this term is used in every operation along the ith column. Based on these observations, we illustrate the process dependence graph of the Aitken algorithm for $n = 4$ and $\mathbf{m} = (1,1,1,1,1)$ in Figure 1. The graph is drawn on the (i,j) coordinate system. The nodes of this acyclic directed graph represent the operations, and the

oriented edges correspond to dependences between the variables used in the operations. The node at point (i, j) computes A_{ij} by performing the operation

$$A_{ij} = \frac{A_{i-1,i} - A_{i-1,j}}{x_i - x_j} \ . \tag{9}$$

A comparison of Figure 1 with the above table reveals the fact that the necessary operands are present for the operations performed in the nodes.

These observations can be used to compute the generalized divided differences. In this case we have $\mathbf{m} = (m_0, m_1, \ldots, m_n)$ with $m_i > 1$ and A_{ij} are matrices of dimension $m_i \times m_j$. If the nodes represent the operations to compute the entries of the matrix A_{ij}, then the process dependence graph for the generalized divided difference tables is the same as the Newton case, illustrated in Figure 1. Let $m_i = 3$ and $m_j = 4$. By using Procedure Aitken, these entries are computed as follows:

$A_{ij}(1,4) = \frac{A_{ij}(1,3) - A_{i-1,j}(m_{i-1},4)}{x_i - x_j}$ $\quad$ $A_{ij}(2,4) = \frac{A_{ij}(2,3) - A_{i,j}(1,4)}{x_i - x_j}$ $\quad$ $A_{ij}(3,4) = \frac{A_{ij}(3,3) - A_{i,j}(2,4)}{x_i - x_j}$

$A_{ij}(1,3) = \frac{A_{ij}(1,2) - A_{i-1,j}(m_{i-1},3)}{x_i - x_j}$ $\quad$ $A_{ij}(2,3) = \frac{A_{ij}(2,2) - A_{i,j}(1,3)}{x_i - x_j}$ $\quad$ $A_{ij}(3,3) = \frac{A_{ij}(3,2) - A_{i,j}(2,3)}{x_i - x_j}$

$A_{ij}(1,2) = \frac{A_{ij}(1,1) - A_{i-1,j}(m_{i-1},2)}{x_i - x_j}$ $\quad$ $A_{ij}(2,2) = \frac{A_{ij}(2,1) - A_{i,j}(1,2)}{x_i - x_j}$ $\quad$ $A_{ij}(3,2) = \frac{A_{ij}(3,1) - A_{i,j}(2,2)}{x_i - x_j}$

$A_{ij}(1,1) = \frac{A_{i-1,i}(m_{i-1},1) - A_{i-1,j}(m_{i-1},1)}{x_i - x_j}$ $\quad$ $A_{ij}(2,1) = \frac{A_{i-1,i}(m_{i-1},2) - A_{i,j}(1,1)}{x_i - x_j}$ $\quad$ $A_{ij}(3,1) = \frac{A_{i-1,i}(m_{i-1},3) - A_{i,j}(2,1)}{x_i - x_j}$

The process dependence graph of the above operations is illustrated in Figure 2. The node at the point (i, j) here represents $m_i \times m_j$ smaller nodes connected in a rectangular mesh where the entries of the matrix A_{ij} are computed. If $m_i = 1$ for all $0 \leq i \leq n$, then this graph consists of only one node and the process dependence graph of the generalized Aitken table reduces to the simple case depicted in Figure 1. We thus can view Figure 1 as the process dependence graph of the general case, where the nodes represent rectangular meshes of simpler nodes (operations). This hierarchical view simplifies the treatment of each case, and allows us to embed the process dependence graph in spacetime, producing various systolic arrays. In the following sections, the process dependence graph, G_A, for Aitken's algorithm is embedded in spacetime, obtaining several different systolic arrays (for spacetime embedding techniques, see [14, 15, 3, 13, 16, 4, 17]).

5 The McKeown Array

In this section, we present McKeown's array [12]. We embed the process dependence graph for Aitken's algorithm in spacetime. The abscissa is interpreted as time (t); the ordinate as space (s). The linear embedding, E_1, is as follows:

$$\begin{bmatrix} t \\ s \end{bmatrix} \ = \ T_1 \begin{bmatrix} i \\ j \end{bmatrix} \ \text{where} \quad T_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

The result, depicted in Figure 3 for $n = 6$, is McKeown's array. Data that flows south $\rightarrow$ north in Figure 1, flows in the direction of time (perpendicular to space) in the McKeown design: It is in the processors' memory. Data that flows west $\rightarrow$ east in Figure 1, flows up through the McKeown array. Data that flows south $\rightarrow$ east in Figure 1, also flows up through the McKeown 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 $2n - 1$. McKeown's 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*.

6 A Spacetime-Optimal Version of the McKeown Array

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 McKeown 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:

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

This embedding, E_2, is illustrated, for $n = 6$, in Figure 4. This design has 2 phases of data movement. In the 1st phase, data moves as in the McKeown design. As the 1st phase ends, and the 2nd begins, there is a transition: When n is even (as depicted in Figure 4), 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_A is spacetime-optimal.*

Proof The embedding E_2 is identical to E_1 with respect to time: it too is time-optimal. We argue space minimality as follows. To reduce space, we must reschedule a processor's processes to other processors. Let us focus on the points in time where all processors are used. In Figure 4, they occur for $t = 5, 6, 7$. During these points all 3 processors are used. We refer to such points in time as the *space-maximal* points of the embedding. In order to reduce the spatial extent of this embedding, it is necessary that the processes from some processor's space-maximal points be rescheduled onto other processors. We can reduce the spatial extent of the embedding depicted in Figure 4, for example, if processes embedded at coordinates (5,3), (6,3), and (7,3) can be rescheduled onto other processors. Notice that these processes are on a longest directed path in the process dependence graph. This means that none can be rescheduled for earlier completion without violating an order constraint. Neither can they be scheduled for later completion without either violating an order constraint or extending the overall completion time, violating the time-optimality property. In fact, in this embedding every process is on some longest path, and hence can be rescheduled onto neither an earlier nor a later cycle. For this process dependence graph, processes

occurring during the space-maximal points, in particular, cannot be rescheduled. Therefore the number of processors used cannot be reduced without violating time-optimality. We conclude that the embedding 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.

7 Some Other Spacetime-Optimal Arrays

We now present 2 other spacetime-optimal embeddings of the process dependence graph of Figure 1. The first is another variation of McKeown's array. 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:

$$\begin{aligned} t &:= i + j \ ; \\ s &:= j \bmod \lfloor \frac{n}{2} \rfloor \ . \end{aligned}$$

This embedding, E_3, is illustrated, for $n = 6$, in Figure 5. This design has data flow characteristics that are identical to the McKeown array, 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 6, as follows:

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

   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:

$$T_2 = \lfloor C \rfloor \ ,$$

   where

$$C = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \ , \text{ and } \quad y = \lfloor C \rfloor x = \lfloor Cx \rfloor \ , \text{ where } \quad \left\lfloor \begin{bmatrix} i \\ j \end{bmatrix} \right\rfloor = \begin{bmatrix} \lfloor i \rfloor \\ \lfloor j \rfloor \end{bmatrix} \ .$$

   Processor efficiency (i.e., the percentage of time that a processor is used) is doubled asymptotically by this nonlinear transformation. Figure 7 illustrates the result.

This design has 2 phases of data movement which alternate with each time step. Regardless of the phase, data that flows south $\rightarrow$ east in Figure 1, flows in the direction of time: It is remembered.

In phase A, data that flows south $\rightarrow$ north in Figure 1, flows up through the array; data that flows west $\rightarrow$ east in Figure 1, flows in the direction of time.

In phase B, data that flows south $\rightarrow$ north in Figure 1, flows in the direction of time; data that flows 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.

8 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:

$$\begin{bmatrix} t \\ s_1 \\ s_2 \end{bmatrix} := T_3 \begin{bmatrix} i \\ j \end{bmatrix} \text{ where } T_3 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} .$$

Figure 8 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 9 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. Consequently, executing k such process dependence graphs uses $2n+k-2$ steps.

The 2-d systolic array is useful when one is computing many interpolation polynomials with possibly different set of points. Such is the case for multivariate interpolation, which we consider in §11.

9 Computing Generalized Divided Differences

In this section, we incorporate extensions to Aitken's algorithm, enabling it to be used to compute generalized divided differences. As mentioned in §4, the process dependence graph for the generalized divided differences has a two-level structure. At the top level, its structure is that of the divided differences process dependence graph, illustrated in Figure 1. Each node in this top level structure comprises a rectangular mesh of nodes that perform scalar operations, illustrated in Figure 2. Figure 10 illustrates how these rectangular meshes are interconnected to form the 2-level process dependence graph G_{GA}. The spacetime embeddings used for computing the divided differences (see §5-8) can be applied to the process dependence graph for computing generalized divided differences. The McKeown embedding for the generalized case is illustrated in Figure 11. Table 1 contains the time and space complexities of these embeddings. Embeddings E_2, E_3, and E_4 also are spacetime-optimal for the process dependence graph, G_{GA}, that computes generalized divided differences.

There are several other ways to embed this 2-level process dependence graph. Some of these design alternatives are touched upon in the Conclusion.

10 Neville Algorithm and its Process Dependence Graph

In this section we describe the Neville algorithm and its process dependence graph. For simplicity of illustration, we assume that $n = 3$ and $\mathbf{m} = (2, 2, 2, 2)$. Analogous to the Aitken algorithm, we define the matrices $N_{ij} \in F^{m_i \times m_j}$ for $0 \le i \le 2$ and $i < j \le 3$ as follows:

$$N_{03} = \begin{bmatrix} f_{00112233} & f_{0112233} \\ f_{0011223} & f_{011223} \end{bmatrix}, \quad N_{13} = \begin{bmatrix} f_{112233} & f_{12233} \\ f_{11223} & f_{1223} \end{bmatrix}, \quad N_{23} = \begin{bmatrix} f_{2233} & f_{233} \\ f_{223} & f_{23} \end{bmatrix},$$

$$N_{02} = \begin{bmatrix} f_{001122} & f_{01122} \\ f_{00122} & f_{0112} \end{bmatrix}, \quad N_{12} = \begin{bmatrix} f_{1122} & f_{122} \\ f_{112} & f_{12} \end{bmatrix},$$

$$N_{01} = \begin{bmatrix} f_{0011} & f_{011} \\ f_{001} & f_{01} \end{bmatrix}.$$

In general we have

$$N_{ij}(p, q) \quad = \quad f_{i^p (i+1)^{m_{i+1}} \cdots (j-1)^{m_{j-1}} j^q}$$

where index p is assumed to increase from right to left and index q from bottom to top. Here, the diagonal entries, N_{01}, N_{12}, and N_{23}, are computed first, followed by the entries above the main diagonal, N_{02}, and N_{13}, and so on. This process is formalized in the following procedure.

Procedure Neville
Input: (x_i, f_{i^k}) for $0 \le i \le n$ and $1 \le k \le m_i$.
Output: $N_{ij}(p, q)$ for $0 \le i \le n-1$, $1 \le j \le n$, $1 \le p \le m_i$, and $1 \le q \le m_j$.
BEGIN
FOR $j = 0$ TO n DO
  FOR $i = 0$ TO $n - j$ DO
    FOR $p = 1$ TO m_i DO
      FOR $q = 1$ TO m_j DO
        $N_{i,i+j}(p, q) = \frac{N_{i,i+j}(p,q-1) - N_{i,i+j}(p-1,q)}{x_i - x_{i+j}}$.
END PROCEDURE

The boundary and interface conditions for the Neville Procedure are found to be

$$\begin{aligned}
N_{i,i+1}(p, 0) &= f_{i^p} \text{ for } \quad 0 \le i \le n-1 \text{ and } \quad 1 \le p \le m_0 \ ; \\
N_{i,i+j}(p, 0) &= N_{i,i+j-1}(p, m_{i+j-1}) \text{ for } \quad 2 \le j \le n \ , \quad 0 \le i \le n-j \ , \text{ and } \quad 1 \le p \le m_i \ ; \\
N_{i,i+1}(0, q) &= f_{(i+1)^q} \text{ for } \quad 0 \le i \le n-1 \text{ and } \quad 1 \le q \le m_{i+1} \ ; \\
N_{i,i+j}(0, q) &= N_{i,i+j+1}(m_{i+j+1}, q) \text{ for } \quad 2 \le j \le n \ , \quad 0 \le i \le n-j \ , \text{ and } \quad 1 \le q \le m_{i+j+1} \ .
\end{aligned}$$

To illustrate, we take $n = 4$ and $\mathbf{m} = (1, 1, 1, 1, 1)$.

$$N_{04} = \tfrac{N_{03}-N_{14}}{x_0-x_4} \quad N_{14} = \tfrac{N_{13}-N_{24}}{x_1-x_4} \quad N_{24} = \tfrac{N_{23}-N_{34}}{x_2-x_4} \quad N_{34} = \tfrac{f_3-f_4}{x_3-x_4}$$

$$N_{03} = \tfrac{N_{02}-N_{13}}{x_0-x_3} \quad N_{13} = \tfrac{N_{12}-N_{23}}{x_1-x_3} \quad N_{23} = \tfrac{f_2-f_3}{x_2-x_3}$$

$$N_{02} = \tfrac{N_{01}-N_{12}}{x_0-x_2} \quad N_{12} = \tfrac{f_1-f_2}{x_1-x_2}$$

$$N_{01} = \tfrac{f_0-f_1}{x_0-x_1}$$

For the denominator terms $x_i - x_j$, the point x_i is repeated along a column whereas the point x_j is repeated along a row. Also when a divided difference term, N_{ij}, is computed, then this term is used to compute $N_{i-1,j}$ and $N_{i,j+1}$. The process dependence graph of the Neville algorithm, G_N, is given in Figure 12 drawn on the (i, j) coordinate system where the node at the point (i, j) computes N_{ij}, performing the operation

$$N_{ij} \quad = \quad \frac{N_{i,j-1} - N_{i+1,j}}{x_i - x_j} \tag{10}$$

For the generalized divided differences, similar to the Aitken case, we have $\mathbf{m} = (m_0, m_1, \ldots, m_n)$ with $m_i > 1$ and N_{ij} are matrices of dimension $m_i \times m_j$. Now we let the nodes in Figure 12 represent the set operations to compute the entries of the matrix N_{ij}. Figure 12, then, represents the process dependence graph for the generalized divided difference table as well. If $m_i = 3$ and $m_j = 4$, then by using Procedure Neville we compute the entries of N_{ij} as

$$N_{ij}(3,4) = \tfrac{N_{ij}(3,3)-N_{ij}(2,4)}{x_i-x_j} \qquad N_{ij}(2,4) = \tfrac{N_{ij}(2,3)-N_{ij}(1,4)}{x_i-x_j} \qquad N_{ij}(1,4) = \tfrac{N_{ij}(1,3)-N_{i+1,j}(m_{i+1},4)}{x_i-x_j}$$

$$N_{ij}(3,3) = \tfrac{N_{ij}(3,2)-N_{ij}(2,3)}{x_i-x_j} \qquad N_{ij}(2,3) = \tfrac{N_{ij}(2,2)-N_{ij}(1,3)}{x_i-x_j} \qquad N_{ij}(1,3) = \tfrac{N_{ij}(1,2)-N_{i+1,j}(m_{i+1},3)}{x_i-x_j}$$

$$N_{ij}(3,2) = \tfrac{N_{ij}(3,1)-N_{ij}(2,2)}{x_i-x_j} \qquad N_{ij}(2,2) = \tfrac{N_{ij}(2,1)-N_{ij}(1,2)}{x_i-x_j} \qquad N_{ij}(1,2) = \tfrac{N_{ij}(1,1)-N_{i+1,j}(m_{i+1},2)}{x_i-x_j}$$

$$N_{ij}(3,1) = \tfrac{N_{i,j-1}(3,m_{j-1})-N_{ij}(2,1)}{x_i-x_j} \quad N_{ij}(2,1) = \tfrac{N_{i,j-1}(2,m_{j-1})-N_{ij}(1,1)}{x_i-x_j} \quad N_{ij}(1,1) = \tfrac{N_{i,j-1}(1,m_{j-1})-N_{i+1,j}(m_{i+1},1)}{x_i-x_j}$$

The process dependence graph of the above operations is illustrated in Figure 13. Similar to the Aitken algorithm, the node at the point (i, j) represents $m_i \times m_j$ smaller nodes connected in a rectangular mesh.

The Neville algorithm's process dependence graph can be embedded in spacetime in a way that is similar to the embeddings of the Aitken algorithm described in §5-8. Here we describe one such embedding which is a variation of McKeown's array. The spacetime embedding of the process dependence graph given in Figure 12 is as follows:

$$\begin{bmatrix} t \\ s \end{bmatrix} \quad = \quad \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \quad .$$

This embedding is illustrated in Figure 14 for $n = 4$. Data that flows south $\rightarrow$ north in Figure 12, flows in the direction of time and space. It flows up through the array. Data that flows east $\rightarrow$ west in Figure 12, flows in the direction of time: it stays in the processors' memory. We see that the array uses n processors, finishing the computation in n steps.

11 Computation of Multivariate Divided Differences

In this section we consider the problem of multivariate interpolation. We illustrate the underlying algorithms and systolic arrays for computation of the bivariate divided differences. Extension to more than two variables is straightforward.

The general problem of multivariate interpolation of an arbitrary set of points is difficult, and the algorithms are complicated. The main reason for this is that *unisolvence* cannot be satisfied for arbitrarily spaced data [2]. In one dimension, the distribution of the points is restricted enough to satisfy unisolvence [21]. In more than one dimension, one usually needs to make assumptions regarding the distribution of the points (see, e.g., [18, 8, 7]). Otherwise it is required that certain determinants which appear in constructing the interpolating polynomials do not vanish [19]. We consider bivariate interpolation only on a two-dimensional grid. This case is common, and also offers a natural generalization of the divided difference algorithms to higher dimensions. The machinery developed so far for the computation of the univariate divided differences is well suited to this case. We assume that there exists a bivariate function $f(x, y)$ evaluated at $(n+1)(n+1)$ points on the Euclidean plane (x_i, y_j) for $0 \leq i, j \leq n$. Denote $f(x_i, x_j)$ by $f_{i,j}$; then the points are arranged as follows:

$$\begin{array}{ccccc} f_{0,n} & f_{1,n} & f_{2,n} & \cdots & f_{n,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{0,2} & f_{1,2} & f_{2,2} & \cdots & f_{n,2} \\ f_{0,1} & f_{1,1} & f_{2,1} & \cdots & f_{n,1} \\ f_{0,0} & f_{1,0} & f_{2,0} & \cdots & f_{n,0} \end{array}$$

The bivariate Newton polynomial interpolating this data set can be given as

$$P(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} (x - x_0) \cdots (x - x_{i-1})(y - y_0) \cdots (y - y_{j-1}) f_{01\cdots i, 01\cdots j}$$

where the coefficients $f_{01\cdots i, 01\cdots j}$ are bivariate divided differences, found by applying the univariate divided difference algorithms repeatedly in the x and y directions. We first simultaneously apply the Aitken (or the Neville) Algorithm in the x direction for all $0 \leq j \leq n$, finding

$$\begin{array}{ccccc} f_{0,n} & f_{01,n} & f_{012,n} & \cdots & f_{012\cdots n,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{0,2} & f_{01,2} & f_{012,2} & \cdots & f_{012\cdots n,2} \\ f_{0,1} & f_{01,1} & f_{012,1} & \cdots & f_{012\cdots n,1} \\ f_{0,0} & f_{01,0} & f_{012,0} & \cdots & f_{012\cdots n,0} \end{array}$$

The application of the Aitken (or the Neville) Algorithms in the y direction for all $0 \leq i \leq n$ yields the bivariate divided differences:

$$\begin{array}{ccccc} f_{0,012\cdots n} & f_{01,012\cdots n} & f_{012,012\cdots n} & \cdots & f_{012\cdots n,012\cdots n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_{0,012} & f_{01,012} & f_{012,012} & \cdots & f_{012\cdots n,012} \\ f_{0,01} & f_{01,01} & f_{012,01} & \cdots & f_{012\cdots n,01} \\ f_{0,0} & f_{01,0} & f_{012,0} & \cdots & f_{012\cdots n,0} \end{array}$$

The 2-d array developed in §8 can be used to perform the above computations efficiently. Assume initially that $B_{ij} = f_{i,j}$ for $0 \leq i, j \leq n$. The following procedure computes the bivariate divided differences using a 2-d Aitken array that is attached to a host computer.

Procedure Bivariate$(x_i, y_j, B_{ij}; 0 \le i, j \le n)$
BEGIN
FOR $j = 0$ TO n DO
    B_{ij} =**Univariate**$(x_i, B_{ij}; 0 \le i \le n)$.
FOR $i = 0$ TO n DO
    B_{ij} =**Univariate**$(y_i, B_{ij}; 0 \le j \le n)$.
END PROCEDURE

At the end of all computations the arrays B_{ij} holds the bivariate divided differences: $B_{ij} = f_{01\cdots i,01\cdots j}$, for $0 \le i, j \le n$.

The execution of k process dependence graphs on a 2-d Aitken array takes $2n + k - 2$ time steps where each step is a divided difference operation (9). Thus, the above procedure uses $3n - 1$ steps to execute the first loop, and the same number of steps for the second loop; which gives the total parallel time as $T_{par} = 6n - 2$. Since the sequential computation of the bivariate divided differences takes $T_{seq} = n(n-1)(n+1)$, and 2-d array contains $p = \frac{n(n-1)}{2}$ processing elements, the efficiency of this scheme will be

$$E = \frac{T_{seq}}{pT_{par}} = \frac{n+1}{3n-1} \approx 0.33 \ ,$$

which is asymptotically optimal. Computation of bivariate (or *multivariate*) generalized divided differences can be done by a straightforward extension of the above algorithm.

12 Discussion and Conclusion

The graph used to represent the Aitken algorithm is very similar to that of the Neville algorithm. Since both graphs have a triangular shape, their space-time embeddings are similar.

We have presented several systolic designs for implementing the Aitken and the Neville algorithms. There are, in fact, an exponential number (in the number of indices) of possible designs. Indeed, it is well known that given 1) a process dependence graph (i.e., a directed acyclic graph), 2) a completion time, and 3) a number of processors, finding a multiprocessor schedule for the dag meeting the time and processor bound is NP-complete [6]. Thus, it is unlikely that an efficient algorithm exists for synthesizing space-time optimal designs, given a process dependence graph.

As a complexity measure, space-time optimality indicates how many processors are needed [sufficient] to extract the maximum amount of parallelism from an algorithm. Since this measure depends only on the [graph representation of the] algorithm, it is machine-independent.

The Aitken and the Neville algorithms, explained in §2, compute the coefficients of interpolating polynomials. Their recursions can be modified to evaluate the interpolating polynomial at a point $\bar{x}$ without actually computing its coefficients. This implicit evaluation of the interpolating polynomial is named *iterated interpolation*. In this case, recursions (9) and (10) become

$$A_{ij} = \frac{(\bar{x} - x_j)A_{i-1,i} - (\bar{x} - x_i)A_{i-1,j}}{x_i - x_j} \tag{11}$$

$$N_{ij} = \frac{(\bar{x} - x_j)N_{i,j-1} - (\bar{x} - x_i)N_{i+1,j}}{x_i - x_j} \tag{12}$$

We can use these systolic arrays for iterated interpolation as follows:

1. Instead of using x_i as input, we use $\bar{x} - x_i$.

2. instead of programming the processors according to recursion (9) and (10), we program them according to recursions (11) and (12).

There are several extensions of the material that we have presented. We sketch some of them; a presentation of details is tedious but straightforward, hence omitted. First, the 2-level process dependence graph for computing generalized divided differences admits composite embeddings, each of which possesses its own data flow and spacetime tradeoffs. For example, the 2-level process dependence graph for computing generalized divided differences can be embedded in spacetime so that the top level graph has a spatial projection that is a 1-d array of processing elements, while the bottom level graph — the rectangular mesh process dependence graph — is embedded so that its spatial projection is a 2-d array of processing elements. Its dual embedding, a 2-d triangular array of 1-d arrays, also is possible.

Second, one can 'compose' our solutions for generalized divided differences and multivariate computation. That is, the schemes and arrays presented can be implemented hierarchically.

All of these 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] [10], 2) the Warp [1], and 3) the Matrix-1 [5].

In this paper, we have:

1. presented 3 variations on the McKeown linear systolic array, each of which is spacetime optimal;

2. presented a 2-D interpolation systolic array, and shown its applicability in the multivariate case;

3. generalized these arrays to 2-level arrays that compute generalized divided differences;

4. indicated how these techniques can be applied to Neville's algorithm;

5. sketched the reprogramming that enables these same systolic arrays to implicitly evaluate the interpolating polynomials.

[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] I. S. Berezin and N. P. Zhidkov, *Computing Methods*, Vol. 1, Addison-Wesley, 1965.

[3] 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.

[4] 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.

[5] 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.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, 1979.

[7] M. Gasca and J. I. Maeztu, "On Lagrange and Hermite Interpolation in R^k," *Numerische Mathematik*, Vol. 39, No. 1, pp. 1-14, 1982.

[8] R. B. Guenther and E. L. Roetman, "Some Observations on Interpolation in Higher Dimensions," *Mathematics of Computation*, Vol. 24, No. 111, pp. 517-527, July 1970.

[9] F. B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill, 1956.

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

[11] F. Krogh, "Efficient Algorithms for Polynomial Interpolation and Divided Differences," *Mathematics of Computation*, Vol. 24, No. 109, pp. 185-190, January 1970.

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

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

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

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

[16] P. Quinton, "Automatic synthesis of systolic arrays from uniform recurrent equations", *Proc. 11th Ann. Symp. on Computer Architecture*, pp. 208-214, 1984.

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

[18] H. E. Salzer, "Some New Divided Difference Algorithms," in *On Numerical Approximation*, edited by R. E. Langer, pp. 61-98, The University of Wisconsin Press, 1956.

[19] H. E. Salzer, "Divided Differences for Functions of Two Variables for Irregularly Spaced Arguments," *Numerische Mathematik*, Vol. 6, No. 2, pp. 68-77, 1964.

[20] L. L. Schumaker, "Fitting Surface to Scattered Data," in *Approximation Theory*, Vol. II, edited by G. G. Lorentz, C. K. Chui, and L. L. Schumaker, pp. 203–268, Academic Press, 1976.

[21] H. C. Thacher, Jr., "Derivation of Interpolation Formulas in Several Independent Variables," *Annals of New York Academy of Sciences*, Vol. 86, No. 3, pp. 758-775, May 1960.

[22] N. K. Tsao and R. Prior, "On Multipoint Numerical Interpolation," *ACM Transactions on Mathematical Software*, Vol. 4, No. 1, pp. 51-56, March 1978.

Table 1. Complexities for various embeddings of G_{GA}, a 2-level process dependence graph for computing generalized divided differences.

Embedding	**Time Complexity**	**Space Complexity**
McKeown(E_1)	$m_0 + m_n - 1 + 2\sum_{i=1}^{n-1} m_i$	$\sum_{i=0}^{n-1} m_i$
Optimal McKeown(E_2)	$m_0 + m_n - 1 + 2\sum_{i=1}^{n-1} m_i$	$\left\lceil \frac{1}{2} \sum_{i=0}^{n-1} m_i \right\rceil$
Optimal Ring(E_3)	$m_0 + m_n - 1 + 2\sum_{i=1}^{n-1} m_i$	$\left\lceil \frac{1}{2} \sum_{i=0}^{n-1} m_i \right\rceil$
Optimal Bilateral (E_4)	$m_0 + m_n - 1 + 2\sum_{i=1}^{n-1} m_i$	$\left\lceil \frac{1}{2} \left[\sum_{i=0}^{n} m_i - 1\right] \right\rceil$
2-d (E_5)	$m_0 + m_n - 1 + 2\sum_{i=1}^{n-1} m_i$	$\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} m_i m_j$