Intern. J. Computer Math., Vol. 40, pp. 99–115
Reprints available directly from the publisher
Photocopying permitted by license only

PARALLEL ALGORITHMS FOR NEVANLINNA–PICK INTERPOLATION: THE SCALAR CASE*

ÇETIN K. KOÇ and GUANRONG CHEN

Department of Electrical Engineering, University of Houston, Houston, TX 77204, USA.

(Received 2 October 1990)

We study the parallel computational complexity of the Nevanlinna–Pick interpolation and introduce several parallel algorithms suitable for implementation on shared-memory multiprocessors and systolic/wavefront arrays. The classical algorithm for the Nevanlinna–Pick interpolation requires $O(n^2)$ arithmetic operations to compute the entries of the Fenyves array and $O(n)$ arithmetic operations to evaluate the interpolatory rational function at a given point. We propose an algorithm for parallel computation of the Fenyves array using $O(n)$ arithmetic operations and $O(n)$ processors. Furthermore, we propose a modification of the classical algorithm for fast and parallel implementation of the evaluation step. The resulting parallel algorithm requires $O(n)$ processors in evaluating the interpolatory rational function using $O(\log n)$ arithmetic operations. Finally, we introduce time-optimal and spacetime-optimal systolic algorithms for computing the entries of the Fenyves array and evaluating the interpolatory rational function.

KEY WORDS: Optimal control, Nevanlinna–Pick interpolation, Fenyves array, computational complexity, parallelism, systolic computation.

C.R. CATEGORIES: F.2, G.1.0, G.1.1, G.1.2.

1 INTRODUCTION

The classical Nevanlinna–Pick interpolation problem is an old subject [16, 15], related to which there is a huge volume of research papers in the literature. Recently, there is a renewal of interest on this subject in both Mathematics and Systems Control Engineering due to its fundamental importance in the so-called H^∞-control theory [3, 5], stable systems design [2], and system sensitivity minimization [20].

In mathematical considerations, the Nevanlinna–Pick interpolation has already been generalized from an operator-theoretic point of view to obtain very abstract results [19, 9, 17, 3, 7], which have also been directly and indirectly applied to systems control engineering and many other practical problems [6, 20, 2, 3, 5]. On the other hand, some computational issues in numerical analysis for such an interpolation has also been studied [1]. In the literature, however, there does not seem to have any (theoretical and/or practical) approach for investigating the Nevanlinna–Pick interpolation problem from a viewpoint of computational complexity and parallelism.

---

* Report No. 215, Center for Approximation Theory, Texas A&M University, May 1990. This work is partially supported by the US Army Research Office Grant No. DAAL03-91-G-0106 and the RIG program at the University of Houston.

In this paper, we study some algorithmic and complexity issues on the Nevanlinna–Pick interpolation. We first describe the interpolation problem precisely in Section 2 and then introduce the classical sequential algorithm in Section 3, which is to the best of our knowledge the only computational scheme available in the literature, except perhaps some with very minor modifications. As suggested by the title of the present article, we only investigate the scalar case in what follows, leaving the matrix case to a forthcoming paper. In Sections 4 and 5, we propose new parallel algorithms for the Nevanlinna–Pick interpolation. The parallel algorithms described therein are especially suitable for implementation on shared-memory parallel computers and systolic/wavefront arrays. More precisely, in Section 4 we describe parallel algorithms suitable for implementation on a shared-memory parallel computer. The PRAM model of a shared-memory machine [11] and its concurrent-read and exclusive-write (CREW) version [18] is most suitable for the algorithms that we present. The algorithms are also suitable for the exclusive-read and exclusive-write (EREW) model with the requirement of a few additional operations. For a general discussion on different parallel models, the reader is referred to [14]. In Section 5, we describe several algorithms for the parallel implementation of the Nevanlinna–Pick interpolation on systolic arrays. The algorithms for the systolic model that we propose are intrinsically different from the ones for the shared-memory model in terms of algorithm design philosophy, processor-time tradeoffs, and communication complexity considerations. The systolic arrays and their corresponding schedules are derived by embedding the process dependence graph of the problem in spacetime. We describe a 2-dimensional (triangular) and two 1-dimensional (linear) systolic arrays. Finally, we close the paper with some brief conclusions.

2 THE NEVANLINNA–PICK INTERPOLATION PROBLEM

Let H^∞ be the Hardy space of complex-variable functions analytic in the open unit disk $|z| < 1$ and belonging to $L^\infty(|z| = 1)$ on the unit circle. Let H_1^∞ denote the collection of all functions in H^∞ with $\|f\|_{H^\infty} \leq 1$. For n given distinct points $z_1, z_2, \ldots, z_n$ in the unit disk $|z| < 1$ and n complex numbers $w_1, w_2, \ldots, w_n$ satisfying the so-called Pick condition that the $n \times n$ matrix

$$\left[\frac{1 - w_i \bar{w}_j}{1 - z_i \bar{z}_j}\right]_{1 \leq i, j \leq n}$$

is non-negative definite, find a function $f(z)$ in H_1^∞ which satisfies the interpolation constraints:

$$f(z_i) = w_i \qquad \text{for} \quad i = 1, 2, \ldots, n.$$

3 THE CLASSICAL SEQUENTIAL ALGORITHM

For the given distinct points $z_1, z_2, \ldots, z_n$ in $|z| < 1$ and complex numbers $w_1, w_2, \ldots, w_n$ satisfying the Pick condition, the Nevanlinna–Pick algorithm first computes

the elements of the so-called Fenyves array [10] and then evaluate $f(z)$ for an arbitrarily given value of z. More precisely, if we introduce the notation

$$G(u, v) = \frac{v - u}{1 - \bar{u}v} \quad \text{and} \quad a_{ij} = \frac{1 - \bar{z}_i z_j}{z_j - z_i} \qquad \text{for} \quad 1 \le i, j \le n,$$

then the algorithm can be described as follows [6, 12, 5]:

The Nevanlinna–Pick Algorithm

Step 1 Set $w_{1i} = w_i$ for $1 \le i \le n$ and compute the Fenyves array $\{w_{ij}: 2 \le i \le j \le n\}$ where

$$w_{ij} = a_{i-1,j} G(w_{i-1,i-1}, w_{i-1,j}). \tag{1}$$

Step 2 Set $f_0(z) = w_{nn}$ and for $1 \le i \le n - 1$ compute

$$f_i(z) = \frac{w_{n-i,n-i} + G(z_{n-i}, z) f_{i-1}(z)}{1 + \bar{w}_{n-i,n-i} G(z_{n-i}, z) f_{i-1}(z)}. \tag{2}$$

Given a particular value of z, the final value $f_{n-1}(z)$ computed in Step 2 is equal to $f(z)$. The function $f(z)$ is in H_1^∞ and satisfies $f(z_i) = w_i$ for $1 \le i \le n$ as required.

We wish to separate these two steps of the algorithm, and rename Step 1 as the *interpolation step* and Step 2 the *evaluation step*. The interpolation step uses only the pairs (z_i, w_i) for $1 \le i \le n$ to compute the diagonal entries of the Fenyves array: $\{w_{ii}: 1 \le i \le n\}$. However, due to the dependencies involved, all elements of the Fenyves array $\{w_{ij}: 1 \le i \le j \le n\}$ need to be computed. The evaluation step uses the diagonal entries of the Fenyves array to evaluate the function $f(z)$ at a particular value of z. We first have the following result on the computational complexity of this classical sequential algorithm:

THEOREM 1 *The interpolation step of the classical sequential algorithm for the Nevanlinna–Pick interpolation requires $O(n^2)$ arithmetic operations. Furthermore, once the diagonal entries of the Fenyves array have been computed, the evaluation step requires $O(n)$ arithmetic operations to compute $f(z)$ for a given value of z.*

Proof There are exactly $\frac{1}{2}n(n + 1)$ values of w_{ij} since the indices i and j range from 1 to n and from i to n, respectively. A single value of w_{ij} is computed using $O(1)$ arithmetic operations via Eq. (1). Thus, Step 1 requires $\frac{1}{2}n(n + 1)O(1) = O(n^2)$ arithmetic operations altogether.

In order to compute the number of arithmetic operations required in Step 2, assume that a particular value of z is given and $f_{i-1}(z)$ has been computed. The value of $f_i(z)$ is then computed in $O(1)$ arithmetic operations using Eq. (2). It thus follows that the evaluation step of the Nevanlinna–Pick algorithm requires $\sum_{i=1}^{n-1} O(1) = O(n)$ arithmetic operations. □

4 PARALLEL ALGORITHMS FOR SHARED-MEMORY MODEL

In order to develop parallel algorithms for the Nevanlinna–Pick interpolation, we take a closer look at the dependencies involved in the computation of the entries of the Fenyves array and the evaluation of the function $f(z)$ for a particular value of z. Starting with $w_{1i} = w_i$ for $1 \le i \le n$, the entries of the Fenyves array, i.e., the w_{ij} values for $1 \le i \le j \le n$, are computed by using the previously computed values $w_{i-1,i-1}$, $w_{i-1,j}$, and $a_{i-1,j}$. For example, the Fenyves array for $n = 4$ is given as follows:

$$
\begin{array}{llll}
w_{11} = w_1 & & & \\
w_{12} = w_2 & w_{22} = a_{12} G(w_{11}, w_{12}) & & \\
w_{13} = w_3 & w_{23} = a_{13} G(w_{11}, w_{13}) & w_{33} = a_{23} G(w_{22}, w_{23}) & \\
w_{14} = w_4 & w_{24} = a_{14} G(w_{11}, w_{14}) & w_{34} = a_{24} G(w_{22}, w_{24}) & w_{44} = a_{34} G(w_{33}, w_{34})
\end{array}
$$

The computation of this table of values exhibits a certain degree of parallelism. Our first observation is that the values a_{ij} for $1 \le i < j \le n$ depend only on the inputs z_i for $1 \le i \le n$, and thus can be computed in parallel. However, the computation of w_{ij} requires that all the values $w_{p-1,q}$ and $w_{p-1,p-1}$ for $2 \le p \le q \le j$ be computed in advance, for $2 \le i \le j \le n$. A closer inspection of the Fenyves array reveals that once the values in the ith column (w_{ij} for $j = i, i+1, \ldots, n$) have been computed, the values in the $(i+1)$st column ($w_{i+1,j}$ for $j = i+1, i+2, \ldots, n$) can be computed in parallel. This approach yields a parallel algorithm which computes the entries of the Fenyves array using $O(n)$ arithmetic operations with n processors. This parallel algorithm is described in the following:

Interpolating Algorithm

Step 1 Set $w_{1i} = w_i$ for $1 \le i \le n$ in parallel using n processors.

Step 2 For $i = 2, 3, \ldots, n$ compute w_{ij} for $j = i, i+1, \ldots, n$ in parallel using $n - i + 1$ processors.

For this algorithm, we have the following result:

THEOREM 2 *The elements of the Fenyves array can be computed in $O(n)$ time using n processors.*

Proof The computation of the first column requires $O(1)$ parallel operations using n processors. Similarly, the second column requires $O(1)$ arithmetic operations but uses only $n - 1$ processors. The total number of parallel arithmetic operations required for the computation of the Fenyves array is therefore equal to $O(n)$ and at most n processors are needed. □ □

We now describe a parallel algorithm for the evaluation step of the Nevanlinna–

Pick interpolation. We define

$$f_i(z) = \frac{p_i(z)}{q_i(z)}.$$

Then Eq. (2) can be written as

$$\frac{p_i(z)}{q_i(z)} = \frac{w_{n-i,n-i} + G(z_{n-i}, z)\dfrac{p_{i-1}(z)}{q_{i-1}(z)}}{1 + \bar{w}_{n-i,n-i}G(z_{n-i}, z)\dfrac{p_{i-1}(z)}{q_{i-1}(z)}}$$

$$= \frac{w_{n-i,n-i}q_{i-1}(z) + G(z_{n-i}, z)p_{i-1}(z)}{q_{i-1}(z) + \bar{w}_{n-i,n-i}G(z_{n-i}, z)p_{i-1}(z)}.$$

Thus, we obtain a matrix recursion for the numerator and denominator of the function $f(z)$ as

$$\begin{bmatrix} p_i(z) \\ q_i(z) \end{bmatrix} = \begin{bmatrix} G(z_{n-i}, z) & w_{n-i,n-i} \\ \bar{w}_{n-i,n-i}G(z_{n-i}, z) & 1 \end{bmatrix} \begin{bmatrix} p_{i-1}(z) \\ q_{i-1}(z) \end{bmatrix}. \tag{3}$$

Moreover, we define

$$\mathscr{F}_i(z) = \begin{bmatrix} p_i(z) \\ q_i(z) \end{bmatrix}$$

and

$$\mathscr{L}_i(z) = \begin{bmatrix} G(z_{n-i}, z) & w_{n-i,n-i} \\ \bar{w}_{n-i,n-i}G(z_{n-i}, z) & 1 \end{bmatrix}.$$

The above recursion can then be simply rewritten as

$$\mathscr{F}_i(z) = \mathscr{L}_i(z)\mathscr{F}_{i-1}(z)$$

for $i = 1, 2, \ldots, n-1$, where

$$\mathscr{F}_0(z) = \begin{bmatrix} p_0(z) \\ q_0(z) \end{bmatrix} = \begin{bmatrix} w_{nn} \\ 1 \end{bmatrix}.$$

The computation of $\mathscr{F}_{n-1}(z)$ can be achieved as follows:

$$\begin{aligned} \mathscr{F}_{n-1}(z) &= \mathscr{L}_{n-1}(z)\mathscr{F}_{n-2}(z) \\ &= \mathscr{L}_{n-1}(z)\mathscr{L}_{n-2}(z)\mathscr{F}_{n-3}(z) \\ &\vdots \\ &= \mathscr{L}_{n-1}(z)\mathscr{L}_{n-2}(z)\cdots\mathscr{L}_1(z)\mathscr{F}_0(z). \end{aligned}$$

The advantage of this scheme over the recursion (2) is that it allows parallelism. The product of $n-1$ elements can be done in $O(\log n)$ time using $\lfloor (n-1)/2 \rfloor$ processors via the well-known binary tree algorithm [4, 14], where $\lfloor \cdot \rfloor$ denotes the integer part of its argument.

We summarize the above discussions in the following algorithm and theorem:

Evaluating Algorithm

Step 1 Compute the 2×2 matrices $\mathscr{L}_i(z)$ for $i = 1, 2, \ldots, n-1$ with $n-1$ processors given a particular value of z, the diagonal elements w_{ii} of the Fenyves array, and z_i for $1 \le i \le n$.

Step 2 Compute the product $\mathscr{L}(z) = \mathscr{L}_{n-1}(z)\mathscr{L}_{n-2}(z) \cdots \mathscr{L}_1(z)$ using $\lfloor (n-1)/2 \rfloor$ processors via the binary tree algorithm. Then compute $f(z) = f_{n-1}(z)$ by first multiplying the above 2×2 matrix $\mathscr{L}(z)$ with the 2×1 vector $\mathscr{F}_0(z)$ and then performing a single complex division operation.

THEOREM 3 *Given the diagonal elements of the Fenyves array and a particular value for z, the evaluation step of the Nevanlinna–Pick algorithm requires $O(\log n)$ parallel arithmetic operations using $n-1$ processors.*

Proof The computation of the entries of the 2×2 matrix $\mathscr{L}_i(z)$ requires $O(1)$ arithmetic operations. Given a particular value of z, the diagonal elements w_{ii} of the Fenyves array, and z_i for $1 \le i \le n$, the matrices $\mathscr{L}_i(z)$ for $1 \le i \le n-1$ can all be computed in parallel in $O(1)$ time using a total of $n-1$ processors. We apply the binary tree algorithm to compute the product matrix $\mathscr{L}(z)$, which requires $O(\log n)$ time using $\lfloor (n-1)/2 \rfloor$ processors. We then perform $O(1)$ complex arithmetic operations to compute $f(z)$. The total number of arithmetic operations required by the Evaluating Algorithm is thus bounded by $O(\log n)$. □

If these two algorithms are combined, then the resulting parallel algorithm for the Nevanlinna–Pick interpolation requires $O(n) + O(\log n) = O(n)$ arithmetic operations using n processors. Furthermore, the Evaluating Algorithm can be used to evaluate the function $f(z)$ at as many as $O(n/\log n)$ points in $O(n)$ time with n processors. Combining all the results and discussions given above, we thus have the following consequence:

THEOREM 4 *Given z_i and w_i for $1 \le i \le n$, the Nevanlinna–Pick interpolating function $f(z)$ can be evaluated at any regular points $\tilde{z}_k$ for $1 \le k \le O(n/\log n)$ in $O(n)$ time using n processors.*

Finally, we close this section by a discussion of some important related topics. An important issue arises when the number of processors in the parallel computer system does not match the input size. Depending on the number of processors, denoted by p in the following, several variations of the above parallel algorithms can be proposed.

Fewer processors ($p < n$) We compute the ith column of the Fenyves array using $p < n$ processors. The ith column contains $n - i + 1$ elements. If $n - i + 1 \le p$, then this computation requires $O(1)$ arithmetic operations. If $n - i + 1 > p$ then the

number of arithmetic operations required is $\lceil (n-i+1)/p \rceil O(1)$, where $\lceil \cdot \rceil$ denotes the smallest integer which is larger than or equal to its argument. Thus, it can be verified that the total number of arithmetic operations required in the Interpolating Algorithm is proportional to

$$\sum_{i=1}^{p} 1 + \sum_{i=p+1}^{n} \left\lceil \frac{i}{p} \right\rceil = p + \left\lceil \frac{1}{p} \left(\frac{n(n+1)}{2} - \frac{p(p+1)}{2} \right) \right\rceil = O\left(\frac{n^2}{p} \right).$$

For the parallel execution of the Evaluating Algorithm, we partition the 2×2 matrices $\mathscr{L}_i(z)$ into p groups. Let $n-1 = (p-1)k + q$ for integers k and q with $q \leq k$. Each of the first $p-1$ groups contains k elements while the last group contains $q \leq k$ elements. By assigning a single processor to each group, we can compute the product of the elements in each group in $O(k) = O(n/p)$ time. We then compute the product of the group products using $\lceil p/2 \rceil$ processors in $O(\log p)$ time via the binary tree algorithm. The total number of arithmetic operations required by the Evaluating Algorithm thus becomes $O(n/p + \log p)$.

More processors ($p > n$) The product of n elements cannot be performed in asymptotically less than $O(\log n)$ time. Thus, the proposed algorithm (Evaluating Algorithm) for the evaluation step of the Nevanlinna–Pick interpolation achieves the lower bound. The number of operations required by this algorithm cannot be further (asymptotically) reduced by simply increasing the number of processors.

We have some comments at this point. Because of the data dependency, it is not clear if (and how) the computation of the Fenyves array can be performed in asymptotically less than $O(n)$ time. This question is left for future research. For example, if there exists a formula for the diagonal element w_{ii} in terms of the input values z_i and w_i for $i = 1, 2, \ldots, n$, then it may be possible to compute the diagonal elements in $O(\log n)$ time using the techniques given in [8]. Another approach may be to give a formulation of the Nevanlinna–Pick interpolation to compute the function $f(z)$ without directly computing the elements of the Fenyves array.

We note that the Interpolating Algorithm achieves optimal efficiency since

$$E_i = \frac{n^2}{pO\left(\frac{n^2}{p} \right)} = O(1).$$

The efficiency of the Evaluating Algorithm is equal to

$$E_e = \frac{O(n)}{pO\left(\frac{n}{p} + \log p \right)} = \frac{O(n)}{O(n + p \log p)},$$

which is equal to $O(1/\log n)$ when $p = O(n)$. The efficiency is optimal when $p =$

$O(n/\log n)$ since in this case we have

$$E_e = \frac{O(n)}{O\left(n + \frac{n}{\log n}\log\left(\frac{n}{\log n}\right)\right)} = \frac{O(n)}{O\left(n - \frac{n \log\log n}{\log n}\right)} = O(1).$$

5 PARALLEL ALGORITHMS FOR SYSTOLIC MODEL

The first step in designing an efficient systolic algorithm is to construct a process (data) dependence graph of the problem, requiring only local communication. We then embed this graph in spacetime to produce systolic schedules. We show that the Nevanlinna–Pick interpolation problem can be solved efficiently on 1-dimensional (linear) 2-dimensional (triangular) systolic arrays. We derive time-optimal and space-time-optimal systolic schedules for solving the Nevanlinna–Pick interpolation problem in this section.

We may again separate the interpolation and evaluation steps of the algorithm and then construct the process dependence graphs of these two steps separately. Note, however, that these two steps can be combined and the resulting algorithm is actually more regular, and thus more suitable for implementation on systolic arrays.

An inspection of the Fenyves array reveals that the diagonal entries are computed in the forward order, i.e., w_{11} is computed first, w_{22} secondly, and so on. However, the evaluation step needs w_{nn} to start, which is the last computed diagonal entry. The last element used by the evaluation step is w_{11} which was available in the very beginning. This property of the classic Nevanlinna–Pick algorithm hinders an efficient implementation, since we have to save the values $w_{11}, w_{22}, \ldots, w_{nn}$ in the memory of the host processor before starting the evaluation step.

Here we show that the evaluation step of the Nevanlinna–Pick algorithm can be performed "on the fly" as the diagonal entries of the Fenyves array are computed. In order to achieve this goal, we modify the evaluation step of the algorithm in such a way that it uses the diagonal entries in the same order as they are computed. We use the scheme proposed in the preceding section and define the 2×2 matrices $\mathcal{M}_i(z)$ such that

$$\mathcal{M}_i(z) = \mathcal{L}_{n-1}(z)\mathcal{L}_{n-2}(z) \cdots \mathcal{L}_{n-i}(z) \tag{4}$$

for $i = 1, 2, \ldots, n-1$ and with $\mathcal{M}_0(z) = \mathcal{I}$ (the 2×2 identity matrix). Note that $\mathcal{L}_{n-i}(z)$ is a function z_i, w_{ii}, and z, since

$$\mathcal{L}_{n-i} = \begin{bmatrix} G(z_i, z) & w_{ii} \\ \bar{w}_{ii} G(z_i, z) & 1 \end{bmatrix}.$$

Hence $\mathcal{M}_i(z)$ is a function of z, z_k and w_{kk} for $k = 1, 2, \ldots, i$. The definition (4)

implies the recursion

$$\mathscr{M}_i(z) = \mathscr{M}_{i-1}(z)\begin{bmatrix} G(z_i, z) & w_{ii} \\ \bar{w}_{ii}G(z_i, z) & 1 \end{bmatrix} \tag{5}$$

for $i = 1, 2, \ldots, n-1$. We thus have obtained an algorithm which computes the diagonal entries and the 2×2 matrices $\mathscr{M}_i(z)$ in the forward order. Once $\mathscr{M}_{n-1}(z)$ and w_{nn} have been obtained, we can evaluate $f(z)$ by first computing

$$\begin{bmatrix} p_{n-1}(z) \\ q_{n-1}(z) \end{bmatrix} = \mathscr{F}_{n-1}(z) = \mathscr{M}_{n-1}(z)\mathscr{F}_0(z) = \mathscr{M}_{n-1}(z)\begin{bmatrix} w_{nn} \\ 1 \end{bmatrix} \tag{6}$$

and then setting

$$f(z) = \frac{p_{n-1}(z)}{q_{n-1}(z)}. \tag{7}$$

We summarize this new scheme in the following:

The modified Nevanlinna–Pick algorithm

Step 1 Set $w_{1i} = w_i$ for $1 \le i \le n$ and $\mathscr{M}_0(z) = \mathscr{I}$, and compute $\mathscr{M}_i(z)$ for $i = 1, 2, \ldots, n-1$ and w_{ij} for $2 \le i \le j \le n$ using the recursions (5) and (1), respectively.

Step 2 Compute $f(z)$ using formulas (6) and (7).

The above algorithm produces an *extended* Fenyves array. For $n = 6$, it is given as follows:

$$\begin{array}{cccccc}
w_{11} & \mathscr{M}_1 & & & & \\
w_{12} & w_{22} & \mathscr{M}_2 & & & \\
w_{13} & w_{23} & w_{33} & \mathscr{M}_3 & & \\
w_{14} & w_{24} & w_{34} & w_{44} & \mathscr{M}_4 & \\
w_{15} & w_{25} & w_{35} & w_{45} & w_{55} & \mathscr{M}_5 \\
w_{16} & w_{26} & w_{36} & w_{46} & w_{56} & w_{66} \quad \mathscr{F}_5
\end{array}$$

The process dependence graph of this table is shown in Figure 1. In this graph the circle nodes represent the operations required to compute w_{ij} for $i = 2, 3, \ldots, n$ and $j = i, i+1, \ldots, n$. Thus, the circle node (i, j) receives z_{i-1}, z_j, $w_{i-1,i-1}$, and $w_{i-1,j}$ as inputs and computes w_{ij} using Eq. (1). The arcs shown in Figure 1 indicate the data dependencies. The square node $(i+1, i)$ receives $z, z_i, w_{ii}, \mathscr{M}_{i-1}(z)$ as inputs and computes $\mathscr{M}_i(z)$ using the recursion (5). The last square node $(n+1, n)$ receives $\mathscr{M}_{n-1}(z)$ and w_{nn} as inputs and computes the final output $f(z)$ using Eqs. (6) and (7).

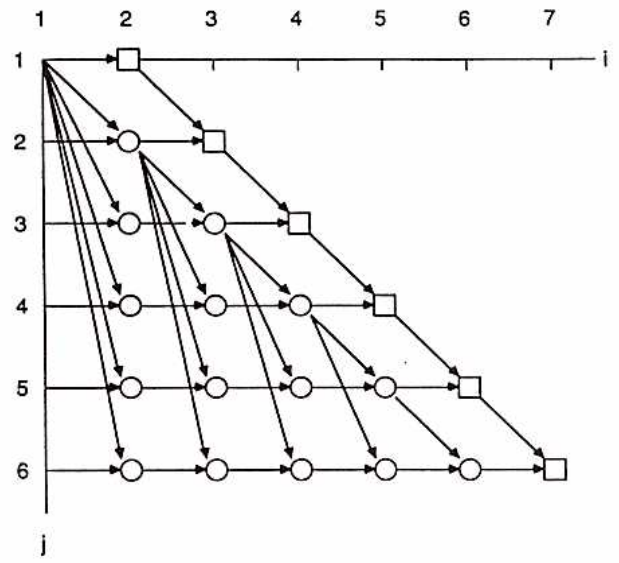

Figure 1 The process dependence graph of the extended Fenyves array for $n = 6$.

Thus, the square node $(i + 1, i)$ needs the data computed in the nodes $(i, i - 1)$ and (i, i).

The process dependence graph of the extended of Fenyves array shown in Figure 1 requires global communication. For example, the value w_{ii} computed in node (i, i) needs to be broadcast to nodes $(i + 1, j)$ for $j = i + 1, i + 2, \ldots, n$. However, with a few modifications, one can obtain a process dependence graph requiring only local communication. The graph given in Figure 2 has this property. The reader is referred

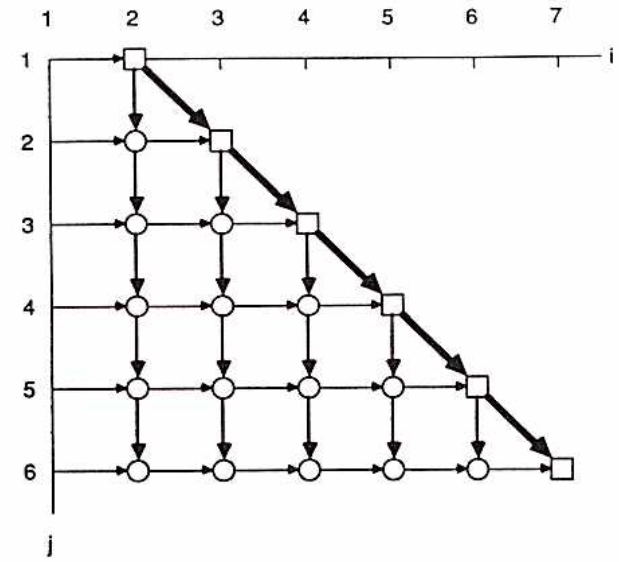

Figure 2 The modified process dependence graph requiring local dependence.

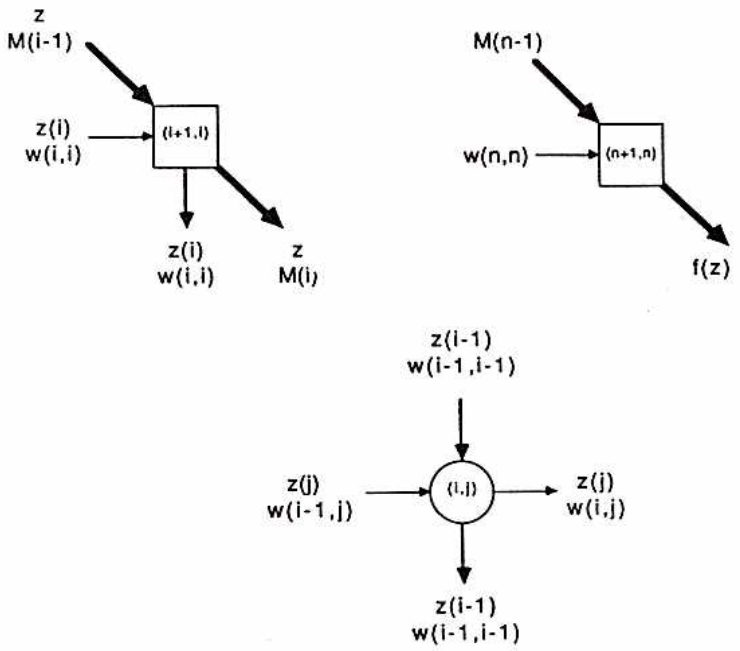

Figure 3 The functions of the nodes.

to [13] for some available techniques in transforming the process dependence graphs for the purpose of achieving local dependency. The functions of the circle and square nodes are described in Figure 3.

Triangular array The graph given in Figure 2 immediately suggests a 2-dimensional systolic array for the Nevanlinna–Pick interpolation. The resulting array, shown in Figure 4, is triangular and contains $p = \frac{1}{2}n(n+1)$ processors. It is easily obtained by

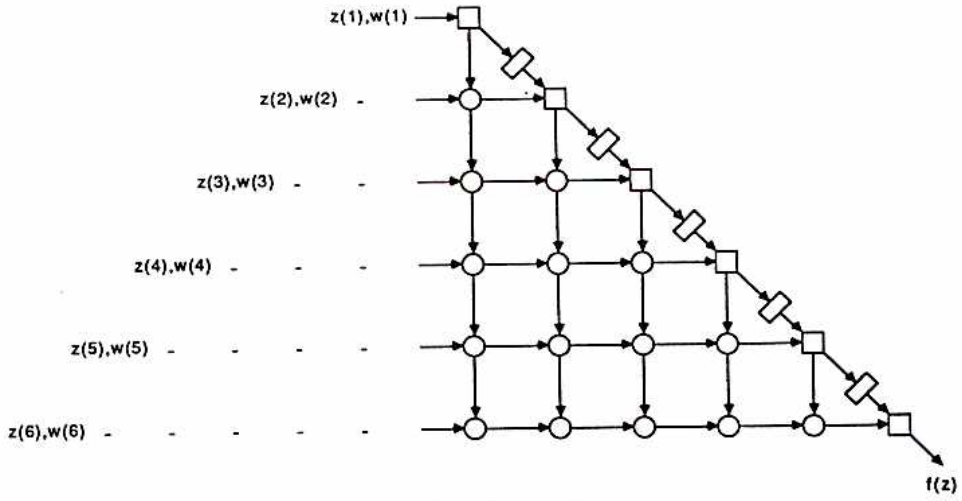

Figure 4 Triangular systolic array.

putting delay elements between the square nodes. The data (z_i, w_i) for $1 \leq i \leq n$ enter the array from the west in a skewed manner. The Nevanlinna–Pick interpolation can be performed in $2n - 1$ time steps, where a time step is equal to the time required to compute (1) or (5), whichever is maximum. The 2-dimensional array is completely pipelined and can be used to solve several Nevanlinna–Pick interpolation problems successively. The first problem requires $2n - 1$ time steps. However, the second problem is solved only one time step thereafter. Thus, the solution of k such problems requires a total of $T = 2n - 1 + k - 1$ time steps. Since the sequential time required by the algorithm is equal to $T_{\text{seq}} = \frac{1}{2}n(n + 1)$, the efficiency of the triangular array is found as

$$E_t = \frac{kT_{\text{seq}}}{p(2n + k - 2)} = \frac{k}{2n + k - 2}.$$

When $k = n - 1$, the efficiency is $E_t = \frac{1}{3}$, and when $k = 2(n - 1)$, it is equal to $\frac{1}{2}$. As $k \to \infty$ we have $E_t \to 1$.

The process dependence graph of the extended Fenyves array can also be embedded in spacetime to produce systolic arrays requiring $p = O(n)$ processors instead of $p = O(n^2)$. The book [13] by S. Y. Kung provides an excellent overview of spacetime embedding techniques. Following those ideas, we give two systolic schedules in the following.

Linear array A We embed the process dependence graph for the extended Fenyves array in spacetime using the following linear embedding:

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i - 2 \\ j \end{bmatrix}$$

where t denotes the time and s denotes the space, i.e., the index of the processor. The embedding shown in Figure 5 provides a schedule for computing the process dependence graph. The process denoted by the pair (i, j) is mapped to the pair (t, s). This implies that process (i, j) will be executed by processor s at time t. Since $\max(s) = n$ and $\max(t) = 2n - 1$, the resulting systolic array requires n processors and completes the computations using $2n - 1$ time steps. The processor represented by the square node computes the matrices $\mathcal{M}_i$ while the other processors (represented by circle nodes) continually compute w_{ij} for all possible values of i and j. Since the length of the longest directed path in the process dependence graph (Figure 2) is equal to $2n - 1$, this embedding is time-optimal (as is the triangular array). However, as opposed to the 2-dimensional array, the execution of k interpolation problems requires $(2n - 1)k$ time steps here. The efficiency of this embedding is therefore equal to

$$E_a = \frac{\frac{1}{2}n(n + 1)}{n(2n - 1)} \approx \frac{1}{4}.$$

An inspection of the schedule shows that the processors are idle more than a half of the time. For example, the first circle processor ($s = 2$) executes during the even

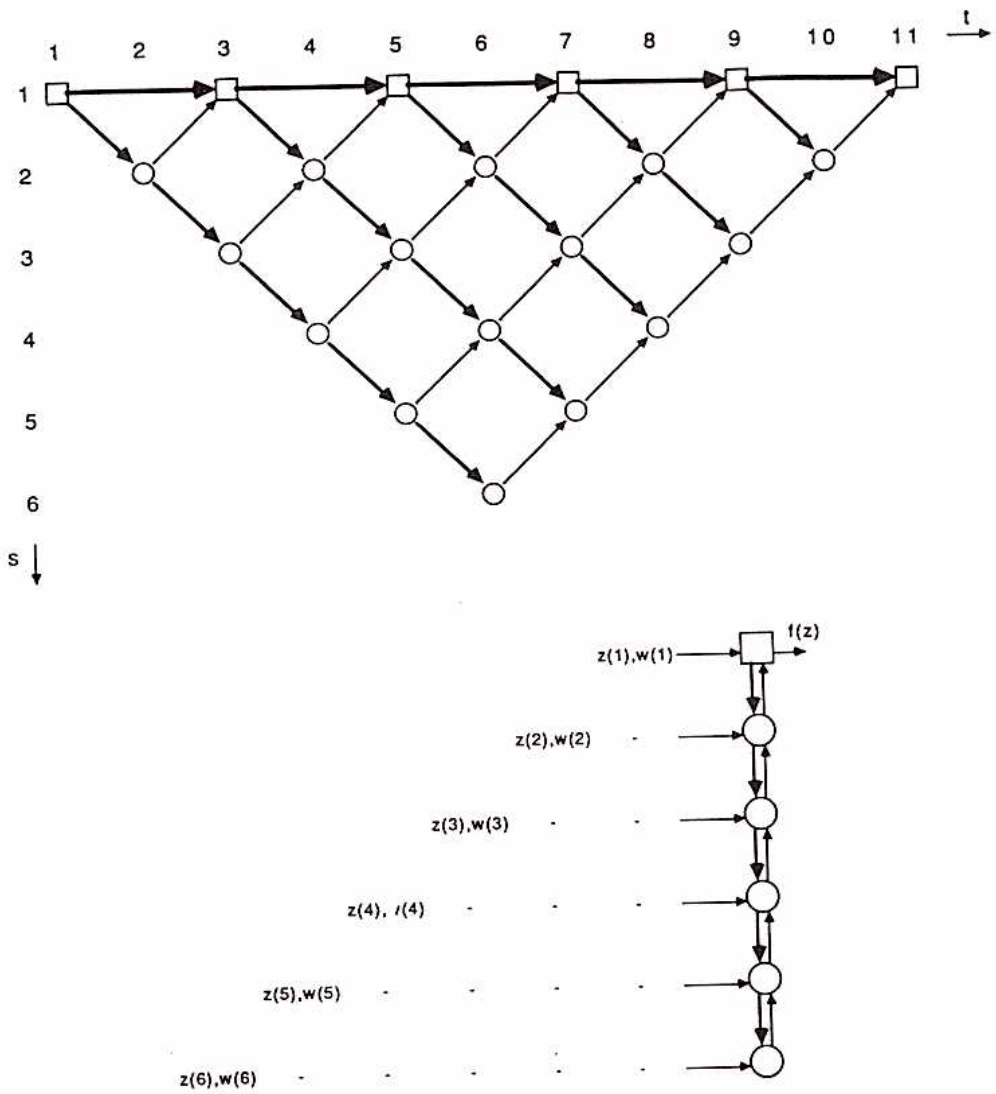

Figure 5 Linear Array A.

numbered time steps and it is idle during the odd numbered time steps. Similarly the second circle processor ($s = 3$) operates only during the time steps: $3, 5, 7, \ldots, 2n - 3$. The efficiency of the processing can be doubled by rescheduling the jobs in such a way that most of the processors are made busy during as many time steps as possible. For example, the jobs of the second circle processor can be allocated to the first circle processor. In this case, this processor executes during the even as well as the odd numbered time steps. Note, however, that we treat the square processor differently since it is unique and its function is different from other processors. The jobs allocated to the square processor do not get reallocated. We achieve these goals with the following embedding.

Linear array B This embedding is achieved in two steps. First we use the linear embedding given above, namely:

$$\begin{bmatrix} t' \\ s' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} i-2 \\ j \end{bmatrix}.$$

Then we map the variables t' and s' to the time and space variables t and s, respectively, using the nonlinear embedding:

$$t = t'$$
$$s = \left\lfloor \frac{1}{2} s' \right\rfloor + 1.$$

The resulting systolic array, as shown in Figure 6, requires $\lceil (n-1)/2 \rceil + 1$ processors and completes in $2n-1$ time steps. Thus, we have doubled the efficiency by using only half as many processors and performing the same amount $(2n-1)$ time steps. The efficiency of the linear array B is computed as

$$E_b = \frac{\frac{1}{2}n(n+1)}{\frac{1}{2}n(2n-1)} \approx \frac{1}{2}.$$

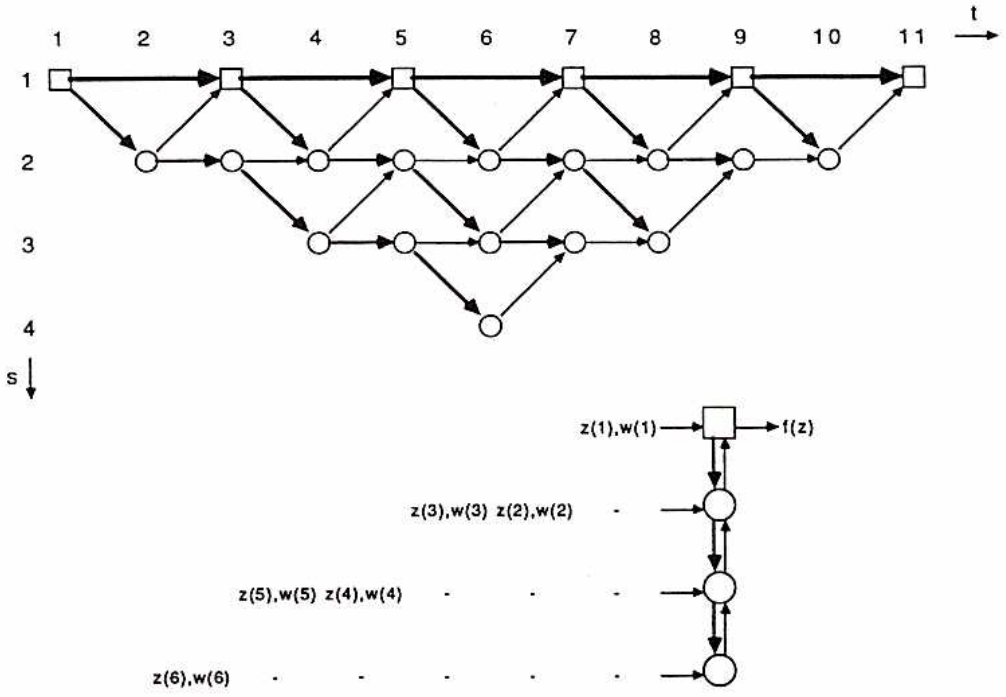

Figure 6 Linear Array B.

The efficiency of this array cannot be further improved. To see this we note that this array is also time-optimal since it requires $2n - 1$ time steps. Furthermore, there exist time steps (for example, time step 6) in which all $\lceil (n-1)/2 \rceil$ circle processors are busy and the jobs of these processors cannot be rescheduled to another time without increasing the total amount of time, i.e., violating time-optimality, and/or without violating data-dependency. Thus, the number of processors cannot be reduced. A systolic schedule with these properties is called spacetime-optimal.

6 CONCLUSIONS

In this paper we have studied the computational complexity of the classical algorithm for the Nevanlinna–Pick interpolation and proposed several parallel algorithms for solving the Nevanlinna–Pick interpolation problem on shared-memory multiprocessors and systolic arrays. The classical algorithm requires $O(n^2)$ arithmetic operations to compute the diagonal entries of the Fenyves array, which are needed in evaluating the interpolatory rational function at a given point using an additional $O(n)$ arithmetic operations. We have proposed an algorithm for parallel computation of the Fenyves array using $O(n)$ arithmetic operation and $O(n)$ processors. Furthermore, we have modified the classical algorithm for fast and parallel implementation of the evaluation step. The resulting parallel algorithm requires $O(n)$ processors to evaluate the interpolatory rational function using $O(\log n)$ arithmetic operations. In Table 1 we summarize the parallel algorithms for the shared-memory model in terms of the number of parallel arithmetic operations, the number of processors, and the efficiency.

Table 1 Parallel algorithms for the shared-memory model.

Algorithm	*Arithmetic Steps*	*Processors*	*Efficiency*
Interpolate	$O(n)$	n	$O(1)$
	$O\left(\frac{n^2}{p}\right)$	p	$O(1)$
	$O(\log n)$	$n-1$	$O\left(\frac{1}{\log n}\right)$
Evaluate	$O\left(\frac{n}{p} + \log p\right)$	p	$\frac{O(n)}{O(n + p \log p)}$
	$O(\log n)$	$\frac{n}{\log n}$	$O(1)$
Interpolate and Evaluate at $O\left(\frac{n}{\log n}\right)$ points	$O(n)$	n	$O(1)$

Table 2 Systolic algorithms.

Algorithm	*Time Steps*	*Processors*	*Efficiency*
Triangular array	$2n-1+k-1$	$\frac{1}{2}n(n+1)$	$\frac{k}{2n+k-2}$
(k problems)			1 as $k \to \infty$
Linear Array A	$2n-1$	n	$\frac{1}{4}$
Linear Array B	$2n-1$	$\left\lceil \frac{n-1}{2} \right\rceil + 1$	$\frac{1}{2}$

We have then introduced time-optimal and spacetime-optimal systolic arrays for the Nevanlinna–Pick interpolation. By incorporating the modified Nevanlinna–Pick algorithm proposed in Section 4, we have shown that the interpolation and evaluation steps of the classical algorithm can be combined. The resulting algorithm is very regular and turns out to be more suitable for implementation on systolic arrays. We have produced 3 systolic arrays by embedding the process dependence graph of the modified Nevanlinna–Pick algorithm in spacetime. In Table 2, we summarize the systolic algorithms. We would like to point out, however, that the systolic arrays introduced in this paper are not the only possible ones. There exist many other embeddings producing time-optimal and spacetime-optimal systolic schedules. Nevertheless, since the length of the longest path in the process dependence graph (Figure 2) of the modified Nevanlinna–Pick algorithm is $2n-1$, there does not exist any systolic schedule requiring fewer than $2n-1$ time steps. Also, it should be noticed that a spacetime-optimal systolic schedule requires at least $\lceil (n-1)/2 \rceil + 1$ processors. Hence, the systolic arrays proposed in this paper achieve both time and spacetime optimalities.

References

[1] A. C. Allison and N. J. Young. Numerical algorithms for the Nevanlinna–Pick problem. *Numerische Mathematik*, **42** (1983), 125–145.

[2] A. C. Antoulas and B. D. O. Anderson. On the problem of stable rational interpolation. *Linear Algebra and its Applications*, 122–124:301–329, September–November 1989.

[3] J. A. Ball. Nevanlinna–Pick interpolation: Generalizations and applications. In J. B. Conway and B. B. Morrel, editors, *Survey of Some Recent Results in Operator Theory I*, pages 51–94. Pitman, 1988.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods.* Prentice-Hall, 1989.

[5] C. K. Chui and G. Chen. *Signal Processing and Systems Theory. Selected Topics.* Springer-Verlag, 1991.

[6] Ph. Delsarte, Y. Genin, and Y. Kamp. On the role of the Nevanlinna–Pick problem in circuit and system theory. *Circuit Theory and Applications*, **9** (1981), 177–187.

[7] H. Dym. *J. Contractive Matrix Functions, Reproducing Kernel Hilbert Spaces and Interpolation*, volume 71. American Mathematical Society, 1989.

[8] Ö. Eğecioğlu, E. Gallopoulos, and Ç. K. Koç. Parallel Hermite interpolation: An algebraic approach. *Computing*, **42** (1989), 291–307.

[9] I. P. Fedčina. A criterion for the solvability of the Nevanlinna–Pick tangent problem (in Russian). *Mat. Issled.*, **7** (1972), 223–227.

[10] F. Fenyves. *Beitrag zur Realisierung von Zweipolen mit Vorgegebener Charakteristik.* Atheneum, Budapest, 1938.
[11] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 114–118, San Diego, CA, May 1978.
[12] J. B. Garnett. *Bounded Analytic Functions.* Academic Press, 1981.
[13] S. Y. Kung. *VLSI Array Processors.* Prentice-Hall, 1988.
[14] S. Lakshmivarahan and S. K. Dhall. *Analysis and Design of Parallel Algorithms.* McGraw-Hill, 1990.
[15] R. Nevanlinna. Über baschränkte analytische funtionen. *Annales Academiae Scientiarum Fennicae*, A32:1–75, 1929.
[16] G. Pick. Über die baschränkungen analytischer funtionen welche durch vorgegebene funktionswerte bewirkt werden. *Math. Ann.*, 77:7–23, 1916.
[17] M. Rosenblum and J. Rovnyak. *Hardy Classes and Operator Theory.* Oxford University Press, 1985.
[18] M. Snir. On parallel searching. *SIAM Journal on Computing*, **14** (1985), 688–708.
[19] B. Sz.-Nagy and A. Korányi. Relations d'un problème de Nevanlinna et Pick la théorie des opérateurs de l'espace Hilbertien. *Acta Mathematica Academiae Scientiarum Hungaricae*, **7** (1956), 295–302.
[20] M. Vidyasagar. *Control System Synthesis: A Factorization Approach.* MIT Press, 1985.