Computers Math. Applic. Vol. 35, No. 3, pp. 71–80, 1998

Fast Parallel Algorithms for Graeffe's Root Squaring Technique

P. K. JANA
Department of Computer Engineering
BIT, Mesra, 835215, India

B. P. SINHA*
Electronics Unit, Indian Statistical Institute
203 Barrackpore Trunk Road. Calcutta. 700035, India

(Received August 1994; accepted May 1997)

Abstract—This paper presents two parallel algorithms for the solution of a polynomial equation of degree n, where n can be very large. The algorithms are based on Graeffe's root squaring technique implemented on two different systolic architectures, built around mesh of trees and multitrees, respectively. Each of these algorithms requires $O(\log n)$ time using $O(n^2)$ processors.

Keywords—Root extraction, Graeffe's root squaring method, Matrix-vector multiplication, Mesh of trees, Multitrees.

I. INTRODUCTION

In many real-time applications, e.g., automatic control, digital signal processing, etc., we often need fast extraction of the roots of a polynomial equation with a very high degree. A common technique for finding the roots of a polynomial equation is through iterations [1–3]. In recent years, many parallel algorithms have been proposed for the extraction of the roots of a polynomial equation. Mirankar [4,5], Schedler [6], and Winograd [7] have developed parallel algorithms for this purpose which are based on the approach of reducing the total number of iterations. Another approach for developing parallel algorithms for solving a polynomial equation is based on reducing the computation time per iteration. Rice and Jamieson [8] have developed a parallel algorithm following this latter approach. Their algorithm is based on the Graeffe's root squaring technique [9] and requires approximately $2n$ arithmetic steps and n communication steps, using $(n+1)$ processors.

The Graeffe's root squaring technique offers some inherent parallelism in computing the new coefficients at each step of iteration, and also in finding all the roots at the final step. In this paper, we propose two parallel algorithms exploiting this parallelism on two different architectures using mesh of trees and multitrees, respectively. Both the algorithms are developed with the objective of reducing the execution time per iteration. Each of these algorithms requires $O(\log n)$ time per iteration step employing $O(n^2)$ processors.

The paper is organized as follows. A sequential algorithm for the Graeffe's root squaring method is discussed in Section 2, followed by the two parallel implementations in Section 3.

*Author to whom all correspondence should be addressed.

2. GRAEFFE'S ROOT SQUARING METHOD

Graeffe's root squaring method for finding the roots, say, $\alpha_1, \alpha_2, \ldots, \alpha_n$ of a polynomial equation

$$f(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n = 0,$$

consists of forming a sequence of polynomials $f_1(x), f_2(x), \ldots$, such that the roots of the equation $f_{i+1}(x) = 0$, $i \geq 1$ are the squares of the roots of the equation $f_i(x) = 0$. Thus, if we assume that

$$f_i(x) = A_0x^n + A_1x^{n-1} + A_2x^{n-2} + \cdots + A_{n-1}x + A_n = 0,$$
$$f_{i+1}(x) = C_0x^n + C_1x^{n-1} + C_2x^{n-2} + \cdots + C_{n-1}x + C_n = 0,$$

then the C_j's, $0 \leq j \leq n$, can be computed as

$$C_j = A_j^2 - 2A_{j-1}A_{j+1} + 2A_{j-2}A_{j+2} - 2A_{j-3}A_{j+3} + \cdots.$$

There is a chance of getting an overflow error while computing the new coefficients because of the rapid growth of the relevant coefficients in the prolonged sequence of root squarings. But this problem can be overcome by using suitable floating point operations [9].

SEQUENTIAL ALGORITHM.
Input : $A_0, A_1, A_2, \ldots, A_n$
Output: $C_0, C_1, C_2, \ldots, C_n$
begin
  $C_0 := A_0^2$;
  for $j := 1$ to n **do**
  begin
    $C_j := A_j{}^2$;
    $i := 1$;
    while $(((i + j) \leq n)$ and $(i \leq j))$ **do**
    begin
      $C_j := C_j + (-1)^i * 2 * A_{j-i} * Aj + i$;
      $i := i + 1$;
    end
  end
end.

3. PARALLEL IMPLEMENTATIONS

In this section, two different parallel implementations of an iteration step in the Graeffe's root squaring technique are discussed. The implementations have been done on the mesh of trees and the multitrees with the main objective of reducing the time for computing the new coefficients C_j's, that is, to reduce the time per iteration.

We observe that the computations of C_j's can be obtained from the the following matrix by vector multiplication:

$$\begin{pmatrix} A_0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & A_1 & -2A_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & A_2 & -2A_1 & 2A_0 & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & A_n \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_n \end{pmatrix}.$$

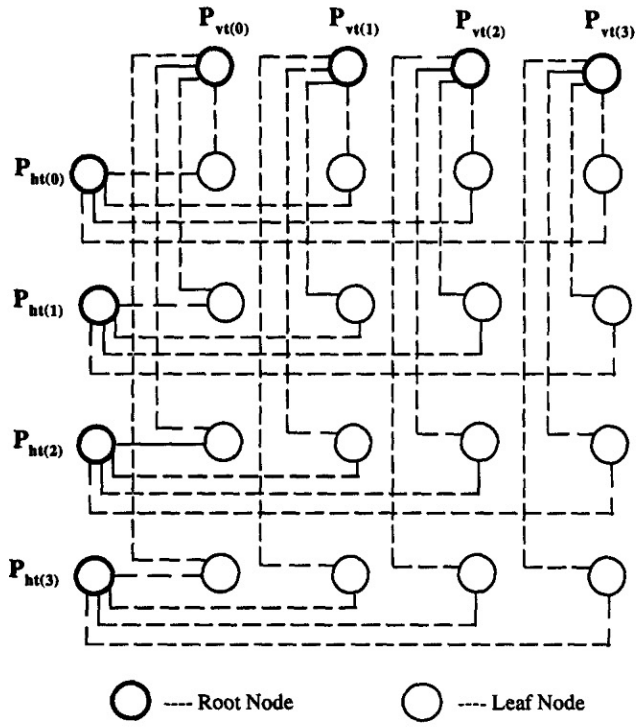


Figure 1. Mesh of trees.

In recent years, a few general techniques for parallel sparse-matrix by vector multiplication have been reported in the literature [10,11]. But in our present case, the sparse matrix has a special structure. We can exploit this structural characteristic to compute the C_j's in an efficient way by the following two implementations.

3.1. Implementation on Mesh of Trees

For an n-degree polynomial equation, let us assume that $m = \lceil n/2 + 1 \rceil$. Then we arrange m^2 processors in the form of an $m \times m$ square array. Using the processors in the i^{th} row, $0 \leq i \leq m-1$, of this array as the leaf nodes, a binary tree is constructed with the help of additional $(m-1)$ internal nodes (processors). There will be m such trees which will be termed as horizontal binary trees. Similarly, with the processors in the j^{th} column of the $m \times m$ square array as the leaf nodes, $0 \leq j \leq m-1$, a vertical tree is also constructed using $(m-1)$ additional nonleaf nodes (processors). The scheme is shown in Figure 1 for $n = 6$, where $P_{ht(i)}$ indicates the root of the i^{th} horizontal tree and $P_{vt(j)}$ denotes the root of the j^{th} vertical tree. The internal nodes other than the root of a tree are, however, not shown in the figure and the presence of the links connecting a root of a tree to its leaf nodes is indicated by dotted lines.

Let $P(i,j)$ denote the processor at the position of the i^{th} row and the j^{th} column. Consider now the main diagonal connecting the processors $P(m-1,0)$ and $P(0,m-1)$. The processors on every diagonal parallel to this main diagonal, including itself, are also interconnected to form a binary tree such that:

(i) $P(0,j)$ is a root for $0 \leq j \leq m-1$,
(ii) $P(i,m-1)$ is a root for $1 \leq i \leq m-1$,
(iii) $P(i,j-i)$ is directly linked to $P(2i+1,j-2i-1)$ and $P(2i+2,j-2i-2)$ whenever they exist, for $i \geq 0$ and $0 \leq j \leq m-1$,
(iv) $P(i+j,m-j)$ is directly linked to $P(i+2j,m-2j)$ and $P(i+2j+1,m-2j-1)$ whenever they exist, for $0 \leq i \leq m-2$ and $j \geq 1$.

The interconnections of two such trees rooted at $P(0,4)$ and $P(1,5)$ for $m = 6$ are shown in Figure 2. These trees will be termed as diagonal trees.

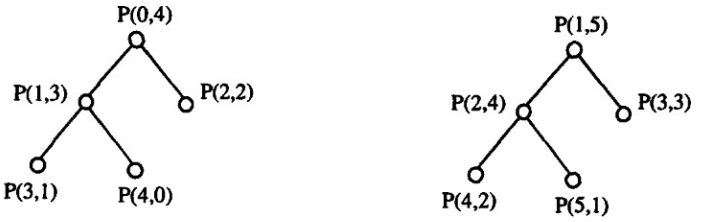


Figure 2. Links of the diagonal trees rooted at $P(0,4)$ and $P(1,5)$.

Every processor in the $m \times m$ array, as shown in Figure 1, is a leaf node of the horizontal as well as the vertical binary trees, and also it may be an internal node of the diagonal tree. Hence, each processor will have a maximum of five links. However, for large n, about 83% of the total number of processors need to have maximum of only three links.

Each of the processors $P(i,j)$, $0 \leq i, j \leq m-1$, will be assumed to have four local registers $V(i,j)$, $H(i,j)$, $D(i,j)$, and $R(i,j)$. The registers V, H, and D will be used for the communications along the vertical, horizontal, and diagonal trees, respectively. The intermediate results will be stored in the R register. The parallel algorithm is formally described in Algorithm A. The main idea is to divide the coefficients A_i's into two groups with even and odd values of the index i, perform the computations with each group independently, and then combine the results together.

We assume that the coefficient values $A_{-1} = A_n = A_{n+1} = 0$. Steps 1–6 compute the coefficients $C_0, C_1, \ldots, C_{n-1}$, and Steps 7–10 are used to transfer these computed coefficients to the root processors of the corresponding horizontal and vertical binary trees, to effect initialization for the next step of iteration.

ALGORITHM A.

begin

/* Computations of new coefficients C_i's from A_i's */

Step 1 :

/* Inputting the even coefficients and broadcasting them */

do Steps 1.1 and 1.2 **in parallel**

1.1 for all j, $0 \leq j \leq m-1$ **do in parallel**

$P_{vt(j)}$ receives A_{2j}, multiplies it by $(-1)^j$, and broadcasts the result to its leaf processors for being stored in $V(i,j)$, $0 \leq i \leq m-1$.

1.2 for all i, $0 \leq i \leq m-1$ **do in parallel**

$P_{ht(i)}$ receives A_{2i}, multiplies it by $(-1)^i$, and broadcasts the result to its leaf processors for being stored in $H(i,j)$, $0 \leq j \leq m-1$.

Step 2 :

for all $P(i,j)$, $0 \leq i \leq m-1$, $0 \leq j \leq m-1$, **do in parallel**

$D(i,j) := V(i,j) * H(i,j)$.

Step 3 :

Sum up the $D(i,j)$'s using the links of the respective diagonal trees for being stored in the R registers of the root processors of the corresponding diagonal trees.

Step 4 :

/* Inputting the odd coefficients and broadcasting them */

do Steps 4.1 and 4.2 in parallel

4.1 for all j, $0 \leq j \leq m-1$ **do in parallel**

$P_{vt(j)}$ receives A_{2j-1}, multiplies it by $(-1)^{j-1}$, and broadcasts the result to its leaf processors for being stored in $V(i,j)$, $0 \leq i \leq m-1$.

4.2 for all i, $0 \leq i \leq m-1$ **do** in parallel

$P_{ht(i)}$ receives A_{2i+1}, multiplies it by $(-1)^i$, and broadcasts the result to its leaf processors for being stored in $H(i,j)$, $0 \leq j \leq m-1$.

Step 5 :

Repeat Steps 2 and 3, except that the results are now stored in the D registers instead of the R registers.

Step 6 :

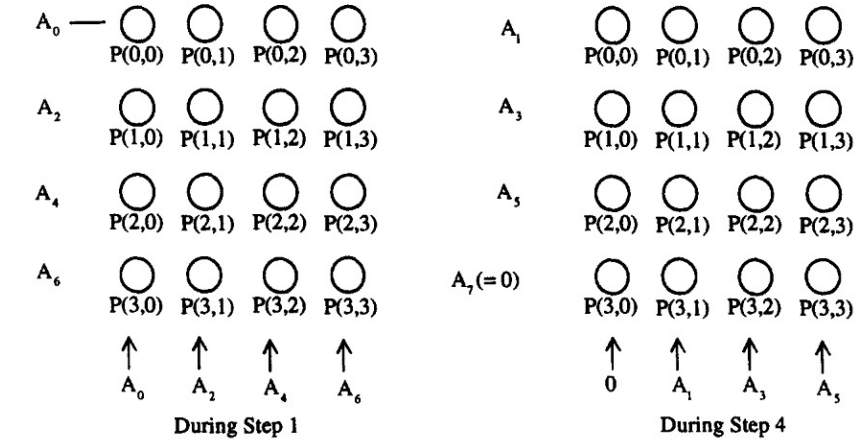
do Steps 6.1 and 6.2 **in parallel**

6.1 for all j, $0 \leq j \leq m-1$, **do in parallel**

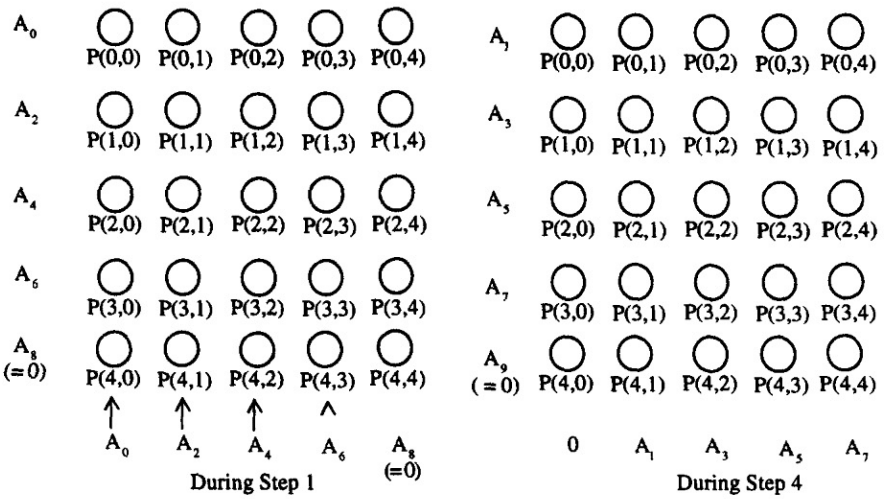
$D(0,j) := D(0,j) + R(0,j)$.

6.2 for all i, $1 \leq i \leq m-1$, **do in parallel**

$D(i,m-1) := D(i,m-1) + R(i,m-1)$.



(a) Input data for $n = 6$, $m = 4$.



(b) Input data for $n = 7$, $m = 5$.

Figure 3. Distribution of input coefficient values for different n.

EXAMPLE 1. The even and odd coefficient values received by different rows and columns of the processors, during Step 1 and Step 4, respectively, are shown in Figure 3 for two different values of n. The situation for $n = 6$ (i.e., $m = 4$) is shown in Figure 3a and that for $n = 7$ (i.e., $m = 5$) is shown in Figure 3b.

REMARK. It appears from the illustrations in Figure 3 that we need, in fact, an $m \times m$ array for even m and an $(m-1) \times m$ array for odd n.

/* Moving C_i's to the roots of trees to initialize for the next iteration */

Step 7 :

do Steps 7.1 and 7.2 **in parallel**

7.1 for all i, $0 \le i \le \lceil m/2 \rceil - 1$, **do** Steps 7.1.1 and 7.1.2 **in parallel**

7.1.1 Using the respective diagonal trees, the content of $D(0, 2i)$ is sent to the V register of $P(i, i)$.

7.1.2 Using the respective diagonal trees, the content of $D(0, 2i+1)$, whenever it exists, is sent to the V register of $P(i, i+1)$.

7.2 for all i, $0 \le i \le \lceil m/2 \rceil - 1$, **do** Steps 7.2.1 and 7.2.2 **in parallel**

7.2.1 Using the respective diagonal trees, the content of $D(m-2i-1, m-1)$, if it exists, is sent to the V register of $P(m-i-1, m-i-1)$.

7.2.2 Using the respective diagonal trees, the content of $D(m-2i-2, m-1)$, if it exists, is sent to the V register of $P(m-i-2, m-i-1)$.

Step 8 :

for all i, $0 \le i \le m-1$, **do in parallel**

$H(i, i) := V(i, i)$

$H(i, i+1) := V(i, i+1)$

Step 9 :

for all i, $0 \le i \le m-1$, **do** Steps 9.1 and 9.2 **in parallel**

9.1 The content of $V(i, i)$ is sent to its root processor $P_{vt(i)}$ for being stored as the even coefficient for the next iteration.

9.2 The content of $H(i, i)$ is sent to its root processor $P_{ht(i)}$ for being stored as the even coefficient for the next iteration.

Step 10 :

for all i, $0 \le i \le m-2$, **do** Steps 10.1 and 10.2 **in parallel**

10.1 The content of $V(i, i+1)$ is sent to its root processor $P_{vt(i)}$ for being stored as the odd coefficient for the next iteration.

10.2 The content of $H(i, i+1)$ is sent to its root processor $P_{ht(i)}$ for being stored as the odd coefficient for the next iteration.

end.

Time complexity

In the above algorithm, Steps 1, 3, 5, 7, 9, and 10 require $O(\log n)$ time each and the remaining steps take constant time. So the overall time complexity of the algorithm is $O(\log n)$.

EXAMPLE 2. Let us assume that $n = 6$. Then the new coefficients are given as follows:

$$
\begin{aligned}
C_0 &= A_0^2, \\
C_1 &= A_1^2 - 2A_0A_2, \\
C_2 &= A_2^2 - 2A_1A_3 + 2A_0A_4, \\
C_3 &= A_3^2 - 2A_2A_4 + 2A_1A_5 - 2A_0A_6, \\
C_4 &= A_4^2 - 2A_3A_5 + 2A_2A_6, \\
C_5 &= A_5^2 - 2A_4A_6, \\
C_6 &= A_6^2.
\end{aligned}
$$

Figure 4 shows the contents of different registers of the processors after Step 3, where a '–' denotes a *don't care* value. After the execution of Step 8, the contents of the registers are shown in Figure 5.

The above algorithm has the advantage that it works with the time complexity $O(\log n)$ and it uses the processors with roughly one-sixth of them having the maximum degree of 5, and the rest

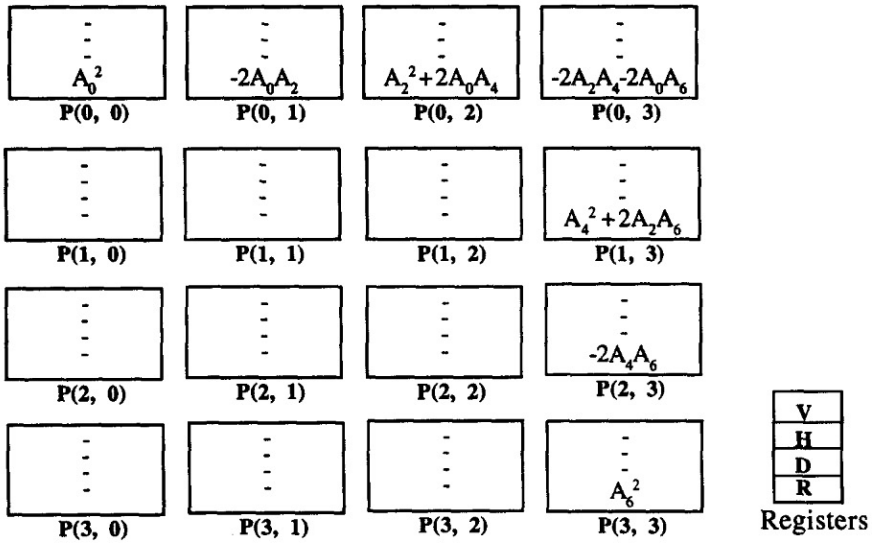


Figure 4. Contents of registers after Step 3 of Algorithm A ($n = 6$).

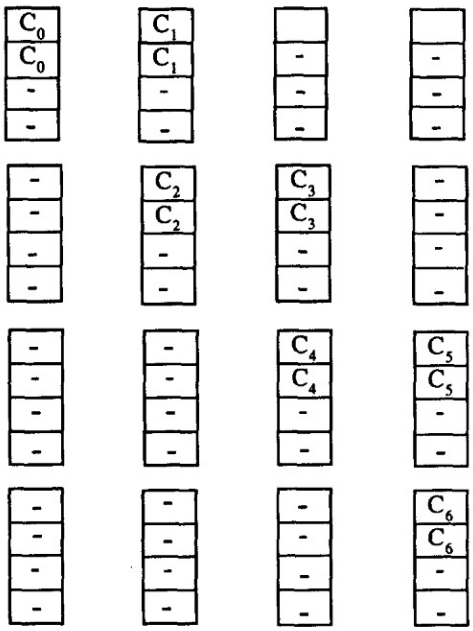


Figure 5. Contents of registers after Step 8 of Algorithm A ($n = 6$).

having the maximum degree of only 3. But it has the disadvantage that only about one-half of the processors are utilized for the main computation, and the remaining processors are needed only for data communication purposes. The processor utilization can, however, be increased if we use an alternative architecture with an increased number of links per processor. An implementation with this idea is shown below.

3.2. Implementation on Multitrees

Let us assume that $q = \lfloor n/2 \rfloor$. Then we arrange $\lfloor (n+2)^2/4 \rfloor$ processors in the form of an $(n+1) \times (q+1)$ triangular array such that there are $n - 2j + 1$ processors in the j^{th} column, $0 \leq j \leq q$. The interconnection scheme is described below, with an example for $n = 6$ in Figure 6.

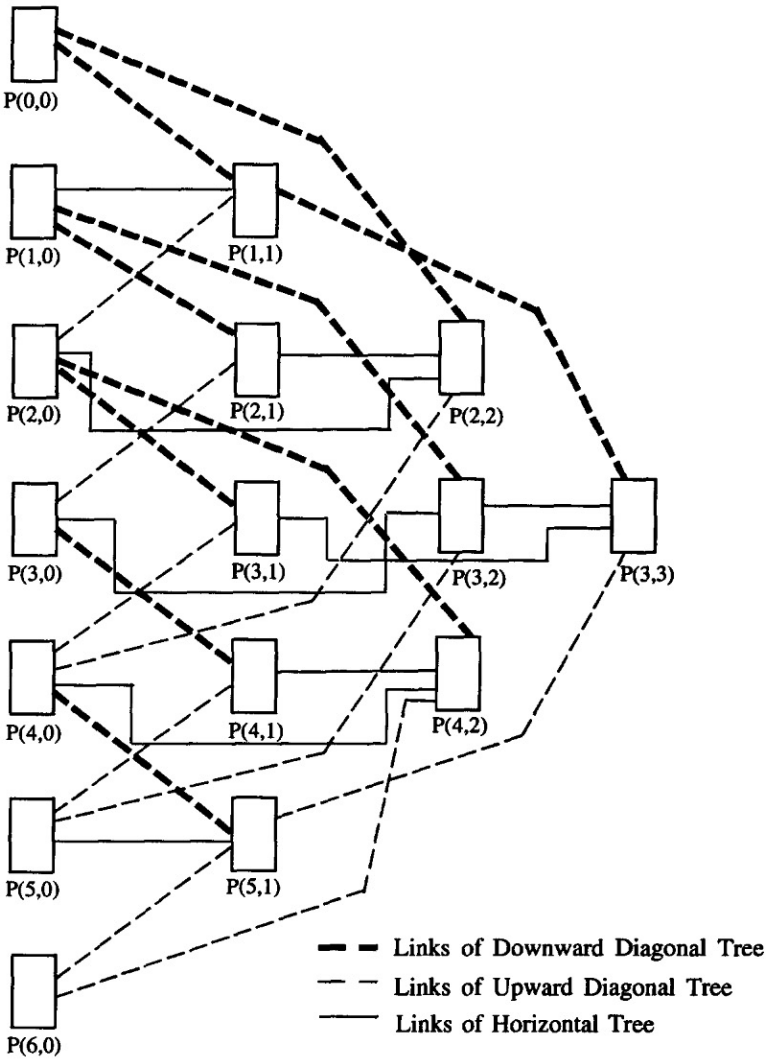


Figure 6. Multitrees structure.

(i) The processors in the i^{th} row, $0 \leq i \leq n$, are interconnected to form a horizontal complete binary tree, with the rightmost processor in that row as the root. Thus, $P(i,j)$ has the two children $P(i, 2j-q-1)$ and $P(i, 2j-q-2)$, for $j \leq q$, whenever they exist.

(ii) The processors on every upward diagonal are interconnected to form a complete binary tree with $P(i,0)$ as the root, $0 \leq i \leq n$. That is, $P(i,j)$ has the two children $P(i-j-1, 2j+1)$ and $P(i-j-2, 2j+2)$, for $j \geq 0$, whenever they exist.

(iii) Similarly, the processors on every downward diagonal are interconnected to form a complete binary tree. That is, $P(i,j)$ has the two children $P(i+j+1, 2j+1)$ and $P(i+j+2, 2j+2)$, for $j \geq 0$, whenever they exist.

The above interconnection scheme requires that the maximum number of links of a processor will be nine, since a processor can be an internal node of three different trees. Data inputting or outputting can be done only through the processors $P(i,0)$ for all i, $0 \leq i \leq n$. Every processor $P(i,j)$ has three local registers $U(i,j)$, $D(i,j)$, and $R(i,j)$. The registers $U(i,j)$ and $D(i,j)$ will be used for the communication along upward and downward diagonal trees, respectively, while the register $R(i,j)$ will be used for communication along the horizontal tree. The parallel algorithm is now formally described below.

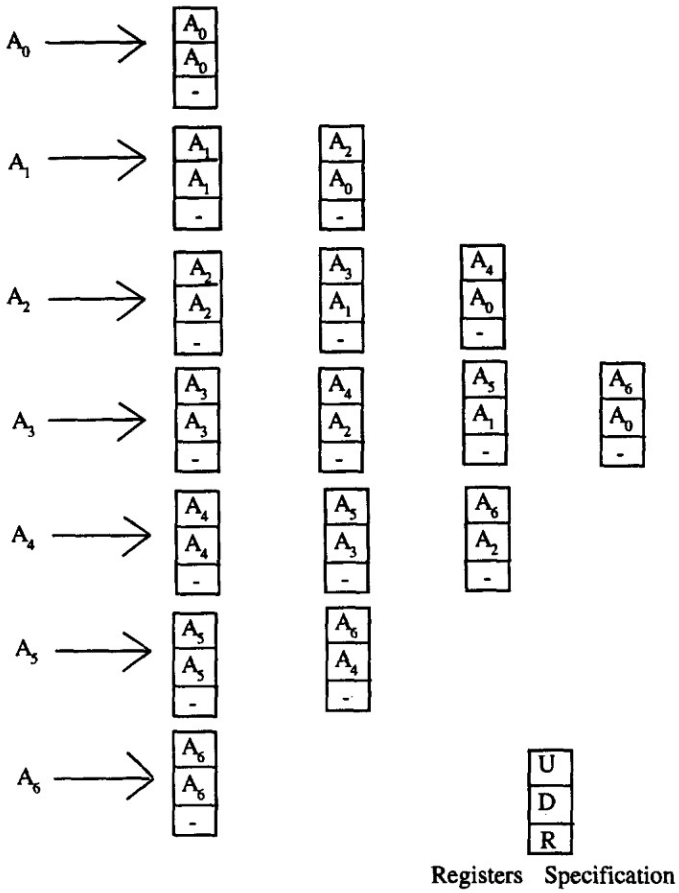


Figure 7. Contents of registers after Step 2 of Algorithm B.

ALGORITHM B.

begin

Step 1 :

for all i, $0 \leq i \leq n$, $P(i,0)$ receives A_i and stores it in $U(i,0)$ and $D(i,0)$.

Step 2 : do Steps 2.1 and 2.2 **in parallel**

2.1 for all i, $0 \leq i \leq n$, the content of $U(i,0)$ is broadcast to the processors in the corresponding upward diagonals following the binary tree connection.

2.2 for all i, $0 \leq i \leq n$, the content of $D(i,0)$ is broadcast to the processors in the corresponding downward diagonals following the binary tree connection.

Step 3 :

for all $P(i,j), 0 \leq i \leq n$, $0 \leq j \leq q$ **do in parallel**

$R(i,j) := U(i,j) * D(i,j)$.

Step 4 :

for all $P(i,j), 0 \leq i \leq n, 0 \leq j \leq q$, **do** Steps 4.1 and 4.2 **in parallel**

4.1 if j is odd, **then** $R(i,j) := -2 * R(i,j)$.

4.2 if j is even, **then** $R(i,j) := 2 * R(i,j)$.

Step 5 :

for all $P(i,j)$, $0 \leq i \leq n$, $0 \leq j \leq q$ **do in parallel**

sum up the contents of $R(i,j)$'s following the horizontal binary tree connections and put the result in $R(i,0)$.

end.

Time complexity

In Algorithm B, each of the Steps 1, 2, and 5 requires $O(\log n)$ time and the remaining steps require constant time. So the overall time complexity of the algorithm is $O(\log n)$.

EXAMPLE 3. We illustrate Algorithm B with an example for $n = 6$. After the execution of Step 2, we get the situation as shown in Figure 7, where a '$-$' denotes a *don't care* value.

4. CONCLUSION

In this paper, we have proposed two efficient parallel implementations for an iteration step of Graeffe's root squaring method, implemented on mesh of trees and multitrees, respectively. The time complexities of both these algorithms are $O(\log n)$ using $O(n^2)$ processors, resulting to an AT-value of $O(n^2 \log n)$. The mesh of trees has the advantage of low number of communication links per processor. But only about one-half of the processors are actually utilized for the main computation and the rest are needed only for data communication in this architecture. In the scheme using multitrees, the number of processors will be asymptotically about one-third of that used in the mesh of trees. However, in that case, a processor needs to have a maximum of nine links. The utilization of processor time in the multitree structure is, of course, much more than that in the mesh of trees architecture.

REFERENCES

1. P. Henrici, *Elements of Numerical Analysis*, John Wiley, New York, (1964).
2. M.A. Jenkins and J.F. Traub, A three-stage algorithm for real polynomials using quadratic iteration, *SIAM J. Numerical Analysis*, 545–566, (December 1970).
3. M.A. Jenkins and J.F. Traub, Zeros of a real polynomial, *ACM Trans. Math. Software*, (1975).
4. W.L. Miranker, Parallel methods for approximating the roots of a function, *IBM J. Res. Develop.*, 297–301, (1969).
5. W.L. Miranker, A survey of parallelism in numerical analysis, *SIAM Review*, 524–547, (October 1971).
6. G.S. Schedler, Parallel numerical methods for the solution of equations, *Communications ACM*, 286–290, (May 1967).
7. S. Winogard, Parallel iteration methods, In *Complexity of Computer Computations*, (Edited by R.E. Miller and J.W. Thatcher), pp. 53–60, Plenum, New York, (1972).
8. T.A. Rice and L.H. Jamieson, A highly parallel algorithm for root extraction, *IEEE Trans. on Computers* **38** (3), 443–449, (March 1989).
9. F.B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill, New York, (1974).
10. A. Peters, Sparse matrix vector multiplication techniques on the IBM 3090 VF, *Parallel Computing* **17** (12), 1409–1424, (December 1991).
11. J. Andersen, G. Mitra and D. Parkinson, The scheduling of sparse matrix vector multiplication on a massively parallel DAP computer, *Parallel Computing* **18** (6), 675–697, (June 1992).