J Supercomput (2010) 54: 139–153
DOI 10.1007/s11227-009-0312-7

Parallel algorithms for finding polynomial Roots on OTIS-torus

Keny T. Lucas · Prasanta K. Jana

Published online: 7 July 2009

Abstract We present two parallel algorithms for finding all the roots of an N-degree polynomial equation on an efficient model of Optoelectronic Transpose Interconnection System (OTIS), called OTIS-2D torus. The parallel algorithms are based on the iterative schemes of Durand–Kerner and Ehrlich methods. We show that the algorithm for the Durand–Kerner method requires $(N^{0.75} + 0.5N^{0.25} - 1)$ electronic moves $+ 2(N^{0.5} - 1)$ OTIS moves using N processors. The parallel algorithm for Ehrlich method is shown to run in $(N^{0.75} + 0.5N^{0.25} - 1)$ electronic moves $+$ $2(N^{0.5} - 1)$ OTIS moves with the same number of processors. The algorithms have lower AT cost than the algorithms presented in Jana (Parallel Comput 32:301–312, 2006). The scalability of the algorithms is also discussed.

Keywords Parallel algorithms · Optoelectronic parallel computer · OTIS-2D torus · Polynomial roots · Durand–Kerner scheme · Ehrlich scheme

1 Introduction

Optical Transpose Interconnection System (OTIS) [12, 28, 34, 40] forms the basis of an efficient architecture of optoelectronic parallel computers that takes benefits from both optical and electronic communications. In this architecture, processors are divided into groups. The processors within the same group are connected through electronic links, whereas the optical interconnections are used for inter-group communication following the OTIS rule: pth processor of the gth group is connected to the

K.T. Lucas (✉)
Department of Information Management, Xavier Institute of Social Service, Ranchi 834001, India
e-mail: kennylucas@xiss.ac.in

P.K. Jana
Department of Computer Science and Engineering, Indian School of Mines University,
Dhanbad 826 004, India
e-mail: prasantajana@yahoo.com

gth processor of the pth group. Depending on the interconnection pattern within each group, an OTIS model can be obtained. OTIS-Mesh, OTIS-Ring, OTIS-Hypercube, OTIS-Mesh of trees and OTIS-Torus are some of the examples of this interconnection network. In the recent years, OTIS architecture has brought the attention among researchers. Several parallel algorithms have been developed for various computations, such as matrix multiplication [38], prefix computation [22], BPC permutation [35], routing, selection and sorting [32], polynomial interpolation and polynomial root finding [20]. The issues regarding scalability and large-scale optic-based implementation of interconnection networks can also be found in [4, 5, 8].

In this paper, we propose two parallel algorithms for finding all the roots of an $N(=n^4)$-degree polynomial equation. The algorithms are based on Durand–Kerner [1, 13, 26] and Ehrlich [14] methods and mapped on OTIS-2D torus network. OTIS-2D torus is built around two-dimensional torus (mesh with wraparound connections) which is one of the most popular instances of n-ary m-cube (for $m = 2$) [10] that have been extensively used to build multicomputer such as J-Machine [31], CRAY T3D [27] and CRAY T3E [3].

Finding polynomial roots has many real-time applications, such as in digital signal processing, automatic control, petroleum exploration, in which we often require very fast computation of all the roots of a very high degree polynomial [23]. Researchers usually adopt one of the two following approaches to parallelize root finding algorithms. One approach is to reduce the total number of iterations as implemented by Miranker [29, 30], Schedler [36] and Winogard [39]. Another approach is to reduce the computation time per iteration, as reported in [7, 20, 24, 33]. There are many schemes for simultaneous approximations of all roots of a given polynomial. Several works on different methods and issues of root finding have been reported in [6, 17, 25, 37, 41, 42]. However, Durand–Kerner and Ehrlich methods are the most practical choices among them [9]. These two methods have been extensively studied for parallelization due to their following advantages. The computation involved in these methods has some inherent parallelism that can be suitably exploited by SIMD machines. Moreover, they have fast rate of convergence (quadratic for the Durand–Kerner method and cubic for the Ehrlich). Various parallel algorithms reported for these methods can be found in [11, 15, 16, 21, 24]. Freeman and Bane [16] presented two parallel algorithms on a local memory MIMD computer with the compute-to-communication time ratio $O(n)$. However, their algorithms require each processor to communicate its current approximation to all other processors at the end of each iteration. Therefore they cause a high degree of memory conflict. Recently the author in [20] proposed two versions of parallel algorithm for the Durand–Kerner method on an OTIS-Mesh. For N-degree polynomial using row-column mapping, the first version requires $6(N^{0.5} - 1)$ electronic moves + 2 OTIS moves per iteration using N^2 processors. However, with the assumption that data points are already stored in the processors, the same algorithm requires $2(N^{0.5} - 1)$ electronic moves + 1 OTIS move. The second version using the group mapping requires $4(N^{0.5} - 1)$ electronic moves + 1 OTIS move per iteration. Assuming the storing of initial data points, this algorithm requires $2(N^{0.5} - 1)$ electronic moves + 1 OTIS move. The parallel algorithms for both the Durand–Kerner and Ehrlich methods, proposed in this paper, are different from [20] in the following respects: (i) Data initialization and mapping of computations are dissimilar in nature. (ii) The algorithms have lower AT cost as each of them requires ($N^{0.75} + 0.5N^{0.25} - 1$) electronic moves + $2(N^{0.5} - 1)$ OTIS moves per iteration using only N processors, in contrast to N^2 processors as needed by the algorithms in [20]. (iii) The algorithms are implemented on OTIS-2D torus in which each group is a 2D mesh with wraparound connections; whereas the algorithms in [20] are mapped on OTIS-Mesh in which wraparound connections are absent for each group.

The paper is organized as follows. In Sect. 2, we describe the computational model, i.e., OTIS-2D torus on which the proposed algorithms are mapped. The parallel algorithms for the Durand–Kerner and Ehrlich methods are presented in Sects. 3 and 4 respectively, followed by the conclusion in Sect. 5.

2 Topology of OTIS-2D torus

In an OTIS-2D torus network, $N = n^4$ processors are divided into n^2 groups to form a two-dimensional lattice. Each group is basically a two-dimensional torus (2D mesh with wraparound connections) with n rows and n columns. Let us denote the processor placed in the (k, l)th position of the (i, j)th group by $PE_{(i,j,k,l)}$ for $0 \leq i, j, k, l \leq n-1$. Then, within each group, two processors placed at (x, y) and (x', y') are connected if and only if $x' = (x \pm 1) \bmod n$ and $y' = y$ or $y' = (y \pm 1) \bmod n$ and $x' = x$. For group communication, the processor $PE_{(i,j,k,l)}$ is connected to $PE_{(k,l,i,j)}$ via an optical link. We assume that all the links are bidirectional. As an example the OTIS-2D torus is shown in Fig. 1 for $n = 3$. In this figure, the indices for each group are shown below it in boldface and the processor indices are shown adjacent to it. It possesses several topological properties that can be exploited in efficient mapping of parallel algorithms.

We differentiate the electronic and optical links as follows: (i) optical links have larger bandwidth than electronic links and (ii) transfer times including latency are different along optical and electronic links [39]. As the channel capacity, transmission property and mechanism of these two links are different; we keep a separate count for data movement on these links. We represent data movement on electronic link by electronic move and that on optical link by OTIS move for analyzing the time complexity of our proposed algorithms. These two moves actually provide the total communication latency required by the algorithms. We also count the number of different primitive mathematical operations per iteration for each of the algorithm to add for a better understandability of our proposed algorithms.

3 Parallel Durand–Kerner algorithm

Let $P_N(x) = a_0x^N + a_1x^{N-1} + a_2x^{N-2} + \cdots + a_{N-1}x + a_N$ be an N $(= n^4)$-degree polynomial, where the coefficients a_i, $1 \leq i \leq N$, are assumed to be real. Without loss of generality, it is also assumed that $P_N(x)$ is normalized, i.e. $a_0 = 1$ and the roots of this polynomial are simple. Then the iterative scheme for the Durand–Kerner method [15] is as follows:

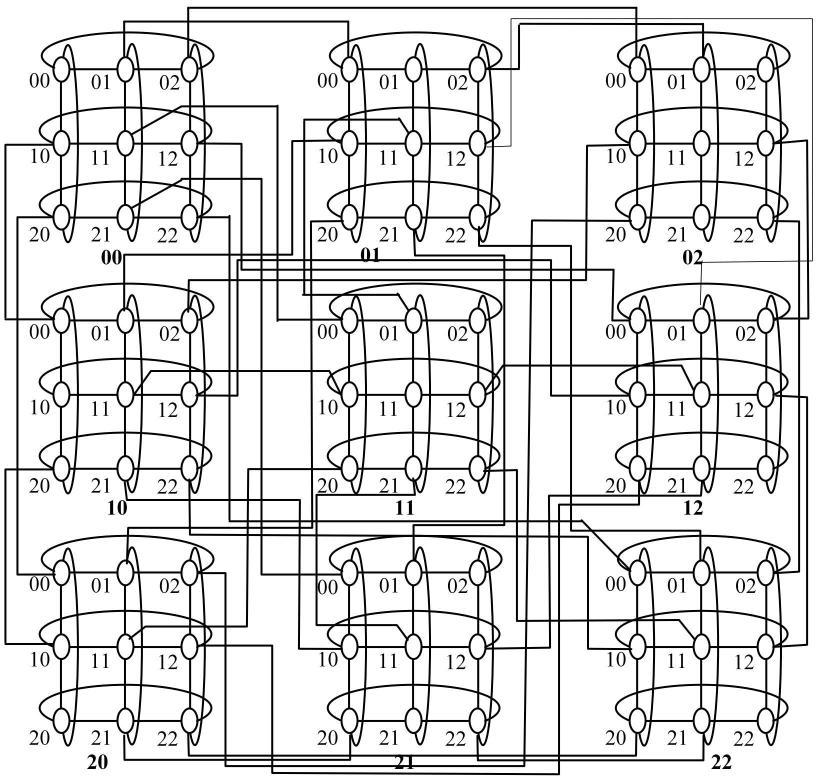

Fig. 1 OTIS-2D torus interconnection network

$$x_i^{(k+1)} = x_i^k - \frac{P_N(x_i^k)}{\prod_{\substack{j=1\\ j\neq i}}^{N}(x_i^k - x_j^k)}, \quad i = 1, 2, \dots, N \tag{3.1}$$

where x_i^k denotes the kth approximation of the root. Following relation (3.1), the next approximations for $N = 9$ are as follows:

$$x_1' = x_1 - \frac{P_9(x_1)}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)\cdots(x_1 - x_9)}$$

$$x_2' = x_2 - \frac{P_9(x_2)}{(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)\cdots(x_2 - x_9)}$$

$$x_3' = x_3 - \frac{P_9(x_3)}{(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)\cdots(x_3 - x_9)}$$

$$\vdots$$

$$x_9' = x_9 - \frac{P_9(x_9)}{(x_9 - x_1)(x_9 - x_2)(x_9 - x_3)\cdots(x_9 - x_8)}.$$

The method is locally convergent and has a quadratic rate of convergence [15]. The initial approximations can be calculated by Guggenheimer procedure [18]. For the sake of simplicity, we denote $P_N(x_i^k)$ by P_i^k.

Now, we present the parallel version of the Durand–Kerner algorithm as follows. It can be observed that $\prod_{\substack{j=1\\ j\neq i}}^{N}(x_i^k - x_j^k)$, $1 \leq i \leq N$, is the principal computation of the Durand–Kerner method, which is the main target of our algorithms to parallelize. The idea is as follows. With initial data values, we compute the partial result in each group concurrently. We then rotate the data values row-wise and column-wise within each group to update the partial results. Next, we rotate each block (group) column-wise for further updating of the partial results. This method is repeated until we obtain the final result. We assume that every processor $PE_{(i,j,k,l)}$ has five local registers, i.e. $A_{(i,j,k,l)}$, $B_{(i,j,k,l)}$, $C_{(i,j,k,l)}$, $D_{(i,j,k,l)}$ and $E_{(i,j,k,l)}$. To formulate the parallel Durand–Kerner algorithm, first we give three procedures, namely, Column_Rotation, Group_Row_Rotation and Group_Column_Rotation, which are called from the main algorithm. The procedure Column_Rotation rotates the contents of the registers within each column of all the groups in parallel. The procedure Group_Row_Rotation rotates the contents of all the processors from one group to another row-wise in clockwise direction, whereas the procedure Group_Column_Rotation rotates from one group to another column-wise in downward direction. We assume here that every processor has the capability of rotating the contents of the registers B, C and D concurrently using electronic links. In our proposed algorithm, we use the notations '$\leftarrow$' and ':=' for data movement within group and assignment statement, respectively.

```
Procedure Column_Rotation ($B, C, D$)
{
  $\forall i, j, k, l,\ 0 \leq i, j, k, l \leq n-1$ do in parallel
    {
      $B_{(i,j,(k+1) \bmod n,l)} \leftarrow B_{(i,j,k,l)}$
      $C_{(i,j,(k+1) \bmod n,l)} \leftarrow C_{(i,j,k,l)}$
      $D_{(i,j,(k+1) \bmod n,l)} \leftarrow D_{(i,j,k,l)}$
    }
}

Procedure Group_Row_Rotation ($B, C, D$)
{
  $\forall i, j, k, l,\ 0 \leq i, j, k, l \leq n-1$ do in parallel
    {
      Perform one OTIS move on $B_{(i,j,k,l)}, C_{(i,j,k,l)}, D_{(i,j,k,l)}$
      $B_{(i,j,k,(l+1) \bmod n)} \leftarrow B_{(i,j,k,l)}$
      $C_{(i,j,k,(l+1) \bmod n)} \leftarrow C_{(i,j,k,l)}$
      $D_{(i,j,k,(l+1) \bmod n)} \leftarrow D_{(i,j,k,l)}$
      Perform one OTIS move on $B_{(i,j,k,l)}, C_{(i,j,k,l)}, D_{(i,j,k,l)}$
    }
}
```

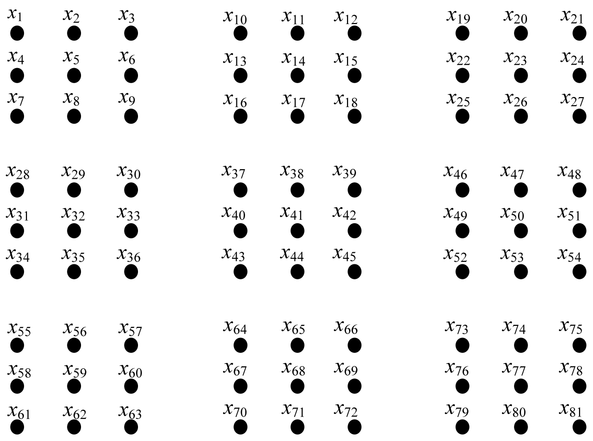
Fig. 2 Initial data storing

```
Procedure $Group\_Column\_Rotation\ (B, C, D)$
{
  $\forall i, j, k, l,\ 0 \le i, j, k, l \le n-1$ do in parallel
    {
      Perform one OTIS move on $B_{(i,j,k,l)}$, $C_{(i,j,k,l)}$, $D_{(i,j,k,l)}$
      $B_{(i,j,(k+1) \bmod n,l)} \leftarrow B_{(i,j,k,l)}$
      $C_{(i,j,(k+1) \bmod n,l)} \leftarrow C_{(i,j,k,l)}$
      $D_{(i,j,(k+1) \bmod n,l)} \leftarrow D_{(i,j,k,l)}$
      Perform one OTIS move on $B_{(i,j,k,l)}$, $C_{(i,j,k,l)}$, $D_{(i,j,k,l)}$
    }
}
```

Algorithm Parallel-DK:

Data initialization: We assume here that the data elements $x_{n^3 i+n^2 j+nk+l+1}$ are initially stored in the registers $A_{(i,j,k,l)}$ for $0 \le i, j, k, l \le n-1$. In other words, they are stored in a row major order within each group and also in the same order block-to-block as shown in Fig. 2 for $n = 3$.

Step 1: $\forall i, j, k, l,\ 0 \le i, j, k, l \le n-1$ *do in parallel*

$$B_{(i,j,k,l)} := A_{(i,j,k,l)}; \quad C_{(i,j,k,l)} := A_{(i,j,k,l)};$$
$$D_{(i,j,k,l)} := A_{(i,j,k,l)}; \quad E_{(i,j,k,l)} := 1$$

Step 2: /* Rotate the contents of $C_{(i,j,k,l)}$ (clockwise) and $D_{(i,j,k,l)}$ (anticlockwise) and subtract it from $A_{(i,j,k,l)}$, multiply by $E_{(i,j,k,l)}$ and store it in $E_{(i,j,k,l)}$ */

```
for $t = 1$ to $\left\lfloor \frac{n}{2} \right\rfloor$ do
  {
    $C_{(i,j,k,(l+1) \bmod n)} \leftarrow C_{(i,j,k,l)}$
    $D_{(i,j,k,(l-1) \bmod n)} \leftarrow D_{(i,j,k,l)}$
    $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - C_{(i,j,k,l)})(A_{(i,j,k,l)} - D_{(i,j,k,l)})$
  }
```

	P_{00}	P_{01}	P_{02}	P_{10}	P_{11}	P_{12}	P_{20}	P_{21}	P_{22}
G_{00}	x^{1}_{1-3} $x_{1}x_{2}x_{3}$	x^{2}_{1-3} $x_{1}x_{2}x_{3}$	x^{3}_{1-3} $x_{1}x_{2}x_{3}$	x^{10}_{10-12} $x_{10}x_{11}x_{12}$	x^{11}_{10-12} $x_{10}x_{11}x_{12}$	x^{12}_{10-12} $x_{10}x_{11}x_{12}$	x^{19}_{19-21} $x_{19}x_{20}x_{21}$	x^{20}_{19-21} $x_{19}x_{20}x_{21}$	x^{21}_{19-21} $x_{19}x_{20}x_{21}$
G_{01}	x^{4}_{4-6} $x_{4}x_{5}x_{6}$	x^{5}_{4-6} $x_{4}x_{5}x_{6}$	x^{6}_{4-6} $x_{4}x_{5}x_{6}$	x^{13}_{13-15} $x_{13}x_{14}x_{15}$	x^{14}_{13-15} $x_{13}x_{14}x_{15}$	x^{15}_{13-15} $x_{13}x_{14}x_{15}$	x^{22}_{22-24} $x_{22}x_{23}x_{24}$	x^{23}_{22-24} $x_{22}x_{23}x_{24}$	x^{24}_{22-24} $x_{22}x_{23}x_{24}$
G_{02}	x^{7}_{7-9} $x_{7}x_{8}x_{9}$	x^{8}_{7-9} $x_{7}x_{8}x_{9}$	x^{9}_{1-9} $x_{7}x_{8}x_{9}$	x^{16}_{16-18} $x_{16}x_{17}x_{18}$	x^{17}_{16-18} $x_{16}x_{17}x_{18}$	x^{18}_{16-18} $x_{16}x_{17}x_{18}$	x^{25}_{25-27} $x_{25}x_{26}x_{27}$	x^{26}_{25-27} $x_{25}x_{26}x_{27}$	x^{27}_{25-27} $x_{25}x_{26}x_{27}$
G_{10}	x^{28}_{28-30} $x_{28}x_{29}x_{30}$	x^{29}_{28-30} $x_{28}x_{29}x_{30}$	x^{30}_{28-30} $x_{28}x_{29}x_{30}$	x^{37}_{37-39} $x_{37}x_{38}x_{38}$	x^{38}_{37-39} $x_{37}x_{38}x_{39}$	x^{39}_{37-38} $x_{37}x_{38}x_{39}$	x^{46}_{46-48} $x_{46}x_{47}x_{48}$	x^{47}_{46-48} $x_{46}x_{47}x_{48}$	x^{48}_{46-48} $x_{46}x_{47}x_{48}$
G_{11}	x^{31}_{31-33} $x_{31}x_{32}x_{33}$	x^{32}_{31-33} $x_{31}x_{32}x_{33}$	x^{33}_{31-33} $x_{31}x_{32}x_{33}$	x^{40}_{40-42} $x_{40}x_{41}x_{42}$	x^{41}_{40-42} $x_{40}x_{41}x_{42}$	x^{42}_{40-42} $x_{40}x_{41}x_{42}$	x^{49}_{49-51} $x_{49}x_{50}x_{51}$	x^{50}_{49-51} $x_{49}x_{50}x_{51}$	x^{51}_{49-51} $x_{49}x_{50}x_{51}$
G_{12}	x^{34}_{34-36} $x_{34}x_{35}x_{36}$	x^{35}_{34-36} $x_{34}x_{35}x_{36}$	x^{36}_{34-36} $x_{34}x_{35}x_{36}$	x^{43}_{43-45} $x_{43}x_{44}x_{45}$	x^{44}_{43-45} $x_{43}x_{44}x_{45}$	x^{45}_{43-45} $x_{43}x_{44}x_{45}$	x^{52}_{52-54} $x_{52}x_{53}x_{54}$	x^{53}_{52-54} $x_{52}x_{53}x_{54}$	x^{54}_{52-54} $x_{52}x_{53}x_{54}$
G_{20}	x^{55}_{55-57} $x_{55}x_{56}x_{57}$	x^{56}_{55-57} $x_{55}x_{56}x_{57}$	x^{57}_{55-57} $x_{55}x_{56}x_{57}$	x^{64}_{64-66} $x_{64}x_{65}x_{66}$	x^{65}_{64-66} $x_{64}x_{65}x_{66}$	x^{66}_{64-66} $x_{64}x_{65}x_{66}$	x^{73}_{73-75} $x_{73}x_{74}x_{75}$	x^{74}_{73-75} $x_{73}x_{74}x_{75}$	x^{75}_{73-75} $x_{73}x_{74}x_{75}$
G_{21}	x^{58}_{58-60} $x_{58}x_{59}x_{60}$	x^{59}_{58-60} $x_{58}x_{59}x_{60}$	x^{60}_{58-60} $x_{58}x_{59}x_{60}$	x^{67}_{67-69} $x_{67}x_{68}x_{69}$	x^{68}_{67-69} $x_{67}x_{68}x_{69}$	x^{69}_{67-69} $x_{67}x_{68}x_{69}$	x^{76}_{76-78} $x_{76}x_{77}x_{78}$	x^{77}_{76-78} $x_{76}x_{77}x_{78}$	x^{78}_{76-78} $x_{76}x_{77}x_{78}$
G_{22}	x^{61}_{61-63} $x_{61}x_{62}x_{63}$	x^{62}_{61-63} $x_{61}x_{62}x_{63}$	x^{63}_{61-63} $x_{61}x_{62}x_{63}$	x^{70}_{70-72} $x_{70}x_{71}x_{72}$	x^{71}_{70-72} $x_{70}x_{71}x_{72}$	x^{72}_{70-72} $x_{70}x_{71}x_{72}$	x^{79}_{79-81} $x_{79}x_{80}x_{81}$	x^{80}_{79-81} $x_{70}x_{79}x_{81}$	x^{81}_{79-81} $x_{79}x_{80}x_{81}$

Fig. 3 Contents of registers E, B, C and D after Step 2

Illustration 1 After this step, $E_{(i,j,k,l)}$ contains the partial product $(x_q - x_r) \times (x_q - x_{r+1})(x_q - x_{r+2}) \cdots (x_q - x_{r+(n-3)})(x_q - x_{r+(n-2)})(x_q - x_{r+(n-1)})$ where $q \neq r$, $q = n^3 i + n^2 j + nk + l + 1$, $r = n^3 i + n^2 j + nk + 1$ and $0 \leq i, j, k, l \leq n - 1$. This step is illustrated in Fig. 3, in which the contents of E registers is shown as x^{β}_{a-b} denoting $(x_\beta - x_a)(x_\beta - x_{a+1}) \cdots (x_\beta - x_{b-1})(x_\beta - x_b)$ for $\beta \notin \{a, a+1, \ldots, b\}$ and $a < b$. The contents of the $B_{(i,j,k,l)}, C_{(i,j,k,l)}$ and $D_{(i,j,k,l)}$ registers are also shown from left to right below the contents of $E_{(i,j,k,l)}$ registers. Note that we do not use any separator between the contents of the above three registers due to lack of space.

Step 3: /* Rotate $B_{(i,j,k,l)}$, $C_{(i,j,k,l)}$, $D_{(i,j,k,l)}$ within each group column-wise in downward direction, then subtract from $A_{(i,j,k,l)}$ and multiply by $E_{(i,j,k,l)}$ */

```
for $s = 1$ to $n - 1$ do
  {
    Call Column_Rotation $(B, C, D)$
    $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
     $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
  }
```

Table 1 Contents of register E after Step 3

Processor→ Group↓	P_{00}	P_{01}	P_{02}	P_{10}	P_{11}	P_{12}	P_{20}	P_{21}	P_{22}
G_{00}	x^{1}_{1-9}	x^{2}_{1-9}	x^{3}_{1-9}	x^{4}_{1-9}	x^{5}_{1-9}	x^{6}_{1-9}	x^{7}_{1-9}	x^{8}_{1-9}	x^{9}_{1-9}
G_{01}	x^{10}_{10-18}	x^{11}_{10-18}	x^{12}_{10-18}	x^{13}_{10-18}	x^{14}_{10-18}	x^{15}_{10-18}	x^{16}_{10-18}	x^{17}_{10-18}	x^{18}_{10-18}
G_{02}	x^{19}_{19-27}	x^{20}_{19-27}	x^{21}_{19-27}	x^{22}_{19-27}	x^{23}_{19-27}	x^{24}_{19-27}	x^{25}_{19-27}	x^{26}_{19-27}	x^{27}_{19-27}
G_{10}	x^{28}_{28-36}	x^{29}_{28-36}	x^{30}_{28-36}	x^{31}_{28-36}	x^{32}_{28-36}	x^{33}_{28-36}	x^{34}_{28-36}	x^{35}_{28-36}	x^{36}_{28-36}
G_{11}	x^{37}_{37-45}	x^{38}_{37-45}	x^{39}_{37-45}	x^{40}_{37-45}	x^{41}_{37-45}	x^{42}_{37-45}	x^{43}_{37-45}	x^{44}_{37-45}	x^{45}_{37-45}
G_{12}	x^{46}_{46-54}	x^{47}_{46-54}	x^{48}_{46-54}	x^{49}_{46-54}	x^{50}_{46-54}	x^{51}_{46-54}	x^{52}_{46-54}	x^{53}_{46-54}	x^{54}_{46-54}
G_{20}	x^{55}_{55-63}	x^{56}_{55-63}	x^{57}_{55-63}	x^{58}_{55-63}	x^{59}_{55-63}	x^{60}_{55-63}	x^{61}_{55-63}	x^{62}_{55-63}	x^{63}_{55-63}
G_{21}	x^{64}_{64-72}	x^{65}_{64-72}	x^{66}_{64-72}	x^{67}_{64-72}	x^{68}_{64-72}	x^{69}_{64-72}	x^{70}_{64-72}	x^{71}_{64-72}	x^{72}_{64-72}
G_{22}	x^{73}_{73-81}	x^{74}_{73-81}	x^{75}_{73-81}	x^{76}_{73-81}	x^{77}_{73-81}	x^{78}_{73-81}	x^{79}_{73-81}	x^{80}_{73-81}	x^{81}_{73-81}

Illustration 2 After Step 3, register $E_{(i,j,k,l)}$ contains $(x_q - x_r)(x_q - x_{r+1}) \times (x_q - x_{r+2}) \cdots (x_q - x_{r+(n^2-2)})(x_q - x_{r+(n^2-1)})$ for $q = n^3 i + n^2 j + nk + l + 1$ and $r = n^3 i + n^2 j + 1, 0 \le i, j, k, l \le n-1$. The contents of $E_{(i,j,k,l)}$ register only are shown in Table 1.

```
Step 4: for $t = 1$ to $n-1$ do
        {
        for $q = 1$ to $n-1$ do
           {
             Call Group_Row_Rotation $(B, C, D)$
             $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
             $\times(A_{(i,j,k,l)} - D_{(i,j,k,l)})$
             for $s = 1$ to $n-1$
               {
                 Call Column_Rotation $(B, C, D)$
                 $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
                 $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
               }
           }
        Call Group_Column_Rotation $(B, C, D)$
        $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
        $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
```

Table 2 Results for $t = 1$ and $q = 2$ in Step 4

Processor→ Group↓	P_{00}	P_{01}	P_{02}	P_{10}	P_{11}	P_{12}	P_{20}	P_{21}	P_{22}
G_{00}	x^{1}_{1-27}	x^{2}_{1-27}	x^{3}_{1-27}	x^{4}_{1-27}	x^{5}_{1-27}	x^{6}_{1-27}	x^{7}_{1-27}	x^{8}_{1-27}	x^{9}_{1-27}
G_{01}	x^{10}_{1-27}	x^{11}_{1-27}	x^{12}_{1-27}	x^{13}_{1-27}	x^{14}_{1-27}	x^{15}_{1-27}	x^{16}_{1-27}	x^{17}_{1-27}	x^{18}_{1-27}
G_{02}	x^{19}_{1-27}	x^{20}_{1-27}	x^{21}_{1-27}	x^{22}_{1-27}	x^{23}_{1-27}	x^{24}_{1-27}	x^{25}_{1-27}	x^{26}_{1-27}	x^{27}_{1-27}
G_{10}	x^{28}_{28-54}	x^{29}_{28-54}	x^{30}_{28-54}	x^{31}_{28-54}	x^{32}_{28-54}	x^{33}_{28-54}	x^{34}_{28-54}	x^{35}_{28-54}	x^{36}_{28-54}
G_{11}	x^{37}_{28-54}	x^{38}_{28-54}	x^{39}_{28-54}	x^{40}_{28-54}	x^{41}_{28-54}	x^{42}_{28-54}	x^{43}_{28-54}	x^{44}_{28-54}	x^{45}_{28-54}
G_{12}	x^{46}_{28-54}	x^{47}_{28-54}	x^{48}_{28-54}	x^{49}_{28-54}	x^{50}_{28-54}	x^{51}_{28-54}	x^{52}_{28-54}	x^{53}_{28-54}	x^{54}_{28-54}
G_{20}	x^{55}_{55-81}	x^{56}_{55-81}	x^{57}_{55-81}	x^{58}_{55-81}	x^{59}_{55-81}	x^{60}_{55-81}	x^{61}_{55-81}	x^{62}_{55-81}	x^{63}_{55-81}
G_{21}	x^{64}_{55-81}	x^{65}_{55-81}	x^{66}_{55-81}	x^{67}_{55-81}	x^{68}_{55-81}	x^{69}_{55-81}	x^{70}_{55-81}	x^{71}_{55-81}	x^{72}_{55-81}
G_{22}	x^{73}_{55-81}	x^{74}_{55-81}	x^{75}_{55-81}	x^{76}_{55-81}	x^{77}_{55-81}	x^{78}_{55-81}	x^{79}_{55-81}	x^{80}_{55-81}	x^{81}_{55-81}

```
    for $s = 1$ to $n - 1$
      {
        Call Column_Rotation $(B, C, D)$
        $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
        $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
      }
    }
```

Illustration 3 The contents of $E_{(i,j,k,l)}$ for $t = 1$ and $q = 2$ in Step 4 is $(x_q - x_r) \times (x_q - x_{r+1})(x_q - x_{r+2}) \cdots (x_q - x_{r+(n^3-2)})(x_q - x_{r+(n^3-1)})$, $q = n^3 i + n^2 j + nk + l + 1$ and $r = n^3 i + 1, 0 \le i, j, k, l \le n - 1$, as shown in Table 2.

```
Step 5: /* This step performs the computation over the last row blocks */
  for $q = 1$ to $n - 1$ do
    {
      Call Group_Row_Rotation $(B, C, D)$
      $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
      $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
      for $s = 1$ to $n - 1$
        {
          Call Column_Rotation $(B, C, D)$
          $E_{(i,j,k,l)} := E_{(i,j,k,l)}(A_{(i,j,k,l)} - B_{(i,j,k,l)})(A_{(i,j,k,l)} - C_{(i,j,k,l)})$
          $\times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$
        }
    }
```

Table 3 Contents of registers E after Step 5

Processor→ Group↓	P_{00}	P_{01}	P_{02}	P_{10}	P_{11}	P_{12}	P_{20}	P_{21}	P_{22}
G_{00}	x^{1}_{1-81}	x^{2}_{1-81}	x^{3}_{1-81}	x^{4}_{1-81}	x^{5}_{1-81}	x^{6}_{1-81}	x^{7}_{1-81}	x^{8}_{1-81}	x^{9}_{1-81}
G_{01}	x^{10}_{1-81}	x^{11}_{1-81}	x^{12}_{1-81}	x^{13}_{1-81}	x^{14}_{1-81}	x^{15}_{1-81}	x^{16}_{1-81}	x^{17}_{1-81}	x^{18}_{1-81}
G_{02}	x^{19}_{1-81}	x^{20}_{1-81}	x^{21}_{1-81}	x^{22}_{1-81}	x^{23}_{1-81}	x^{24}_{1-81}	x^{25}_{1-81}	x^{26}_{1-81}	x^{27}_{1-81}
G_{10}	x^{28}_{1-81}	x^{29}_{1-81}	x^{30}_{1-81}	x^{31}_{1-81}	x^{32}_{1-81}	x^{33}_{1-81}	x^{34}_{1-81}	x^{35}_{1-81}	x^{36}_{1-81}
G_{11}	x^{37}_{1-81}	x^{38}_{1-81}	x^{39}_{1-81}	x^{40}_{1-81}	x^{41}_{1-81}	x^{42}_{1-81}	x^{43}_{1-81}	x^{44}_{1-81}	x^{45}_{1-81}
G_{12}	x^{46}_{1-81}	x^{47}_{1-81}	x^{48}_{1-81}	x^{49}_{1-81}	x^{50}_{1-81}	x^{51}_{1-81}	x^{52}_{1-81}	x^{53}_{1-81}	x^{54}_{1-81}
G_{20}	x^{55}_{1-81}	x^{56}_{1-81}	x^{57}_{1-81}	x^{58}_{1-81}	x^{59}_{1-81}	x^{60}_{1-81}	x^{61}_{1-81}	x^{62}_{1-81}	x^{63}_{1-81}
G_{21}	x^{64}_{1-81}	x^{65}_{1-81}	x^{66}_{1-81}	x^{67}_{1-81}	x^{68}_{1-81}	x^{69}_{1-81}	x^{70}_{1-81}	x^{71}_{1-81}	x^{72}_{1-81}
G_{22}	x^{73}_{1-81}	x^{74}_{1-81}	x^{75}_{1-81}	x^{76}_{1-81}	x^{77}_{1-81}	x^{78}_{1-81}	x^{79}_{1-81}	x^{80}_{1-81}	x^{81}_{1-81}

Illustration 4 After this step, $E_{(i,j,k,l)}$ contains the final product $\prod_{\substack{r \neq q \\ r=1}}^{N} (x_q - x_r)$, *for all* $i, j, k, l,\ q = n^3 i + n^2 j + nk + l + 1, 1 \leq r \leq N, 0 \leq i, j, k, l \leq n-1$ which is shown in Table 3.

Step 6: $\forall i, j, k, l, 0 \leq i, j, k, l \leq n-1$ *do in parallel*

$$A_{(i,j,k,l)} := A_{(i,j,k,l)} - \frac{P_N(x_{n^3 i + n^2 j + nk + l + 1})}{E_{(i,j,k,l)}}$$

Step 7: Stop

At the end of the algorithm, the improved approximations are stored in the registers $A_{(i,j,k,l)}$, $0 \leq i, j, k, l \leq n-1$. For the test of convergence, we can use the Adams method. The convergence test on the individual approximate roots can be performed in parallel. If the desired accuracy is achieved, the computation will be stopped; otherwise, all the steps of the algorithm Parallel-DK are repeated. As the same steps are repeated, this can be implemented in a pipeline fashion to reduce the overall run time for multiple iterations.

Time complexity and AT cost Step 2 requires $n/2$ electronic moves. Step 3 requires $n-1$ electronic moves. Step 4 is computed in $n^2(n-1)$electronic moves + $2n(n-1)$ OTIS moves. Step 5 requires $n(n-1)$ electronic moves + $2(n-1)$ OTIS moves, and each of the Steps 1 and 6 requires constant time. As $N = n^4$, the above algorithm requires a total of $(N^{0.75} + 0.5N^{0.25} - 1)$ electronic moves + $2(N^{0.5} - 1)$ OTIS moves using N processors. Similarly we calculate the total number of mathematical operations. It requires $3N^{0.75} + N^{0.25} - 2$ subtractions, $3N^{0.75} + N^{0.25} - 3$ multiplications and one division in each iteration. Therefore the AT cost of the algorithm is $O(N^{1.75})$. Note that the AT cost of the parallel algorithm presented in [20] is $O(N^{2.5})$ as it uses N^2 processors and requires $2(N^{0.5} - 1)$ electronic moves + 1 OTIS move. Hence, the above algorithm has a lower AT cost than the parallel algorithm [20].

The correctness of the algorithm can easily be seen from the illustrations and figures after respective steps.

Remark It can be noted that the above algorithm does not account for input time of the initial data values as it assumes that data values are already stored in the processors. However, if we assume that only the top processors of each group have the input/output ports, then they can be fed through these processors for each block in a pipeline fashion in $(N^{0.25}-1)$ electronic moves. The same approach has been adopted for storing the initial data for mesh sort, as described in [2]. In this case, the algorithm requires a total of $(N^{0.75}+1.5N^{0.25}-2)$ electronic moves and $2(N^{0.5}-1)$ OTIS moves. However, this does not affect the asymptotic running time of the above algorithm. This can be compared with $6(N^{0.5}-1)$ electronic moves $+$ 2 OTIS moves for row–column mapping and $4(N^{0.5}-1)$ electronic moves $+$ 1 OTIS move for group mapping using N^2 processors as required by the parallel algorithm reported in [20].

The above algorithm is scalable and hence valid for any value of N. Let us consider a more realistic situation where only p $(p<N)$ processors are available to find the roots of N-degree polynomial. For the sake of simplicity, we assume that $N=kp$, where k is a positive integer. Following the data partitioning method for obvious reason, we first divide the whole input data set into $k=N/p$ groups: $\{x_1,x_2,\dots,x_p\}$, $\{x_{p+1},x_{p+2},\dots,x_{2p}\},\dots,\{x_{(k-1)p+1},xx_{(k-1)p+2},\dots,x_{kp}\}$. For each iteration, we need to process all the k input data subsets to obtain the final product term respective to that iteration. This can be illustrated as follows. The algorithm is run for first data subset $\{x_1,x_2,\dots,x_p\}$ to yield the partial product term for the first iteration $\prod_{\substack{j=1\\ j\neq i}}^{p}(x_i^k-x_j^k)$, $1\le i\le p$, and is temporarily stored in the register of each processor. Then for the same iteration, second data subset, i.e. $\{x_{p+1},x_{p+2},\dots,x_{2p}\}$, is taken up and the Steps 1 to 5 are followed to find the second partial product term $\prod_{j=p+1}^{2p}(x_i^k-x_j^k)$, $1\le i\le p$, for the first iteration. The product of this partial product term and the previous one becomes the current partial product term. Similarly, all the successive data subsets are processed to obtain the final product term for the first iteration as $\prod_{\substack{j=1\\ j\neq i}}^{N}(x_i^k-x_j^k)$, $1\le i\le N$. All the above explained steps are repeated for all the iterations to finally obtain the root of an N-degree polynomial. This requires $O(kN^{0.75})$ time and $O(N^{0.25})$ buffers per processor.

4 Parallel Ehrlich algorithm

In this section, we describe another zero finding algorithm, i.e. Ehrlich algorithm. The $(k+1)$th iteration of the method is as follows:

$$x_i^{k+1}=x_i^k-\frac{\alpha_i^k}{1-\alpha_i^k\beta_i^k},\quad i=1,2,\dots,N \tag{4.1}$$

where $P_N'(x_i^k)=\frac{dP_N(x)}{dx}|_{X=X_i^k}$, $\alpha_i^k=\frac{P_N(x_i^k)}{P_N'(x_i^k)}$ and $\beta_i^k=\sum_{\substack{j=1\\ j\neq i}}^{N}\frac{1}{(x_i^k-x_j^k)}$.

We now present the parallel version of the Ehrlich algorithm on OTIS-2D torus for $N = n^4$ data points using n^4 processors. It can be noted from (4.1) that $\beta_i^k = \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{1}{(x_i^k - x_j^k)}$ is the principal computation, which is similar to that of Durand–Kerner iterative scheme. Therefore, this can be similarly parallelized as the Durand–Kerner method. We write Parallel Ehrlich algorithm specifying the steps that are the same as Parallel-DK and the changes made of some steps as follows.

Algorithm parallel-E:

Step 1: The same as Step 1 of Parallel-DK
Step 2: The same as Step 2 of Parallel-DK, except $E_{(i,j,k,l)} := E_{(i,j,k,l)} \times (A_{(i,j,k,l)} - C_{(i,j,k,l)}) \times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$ is replaced by

$$E_{(i,j,k,l)} := E_{(i,j,k,l)} + \frac{1}{(A_{(i,j,k,l)} - C_{(i,j,k,l)})} + \frac{1}{(A_{(i,j,k,l)} - D_{(i,j,k,l)})}$$

Step 3: Through Step 5: The same, except

$$E_{(i,j,k,l)} := E_{(i,j,k,l)} \times (A_{(i,j,k,l)} - C_{(i,j,k,l)}) \times (A_{(i,j,k,l)} - B_{(i,j,k,l)}) \\ \times (A_{(i,j,k,l)} - D_{(i,j,k,l)})$$

is replaced by

$$E_{(i,j,k,l)} := E_{(i,j,k,l)} + \frac{1}{A_{(i,j,k,l)} - B_{(i,j,k,l)}} + \frac{1}{A_{(i,j,k,l)} - C_{(i,j,k,l)}} \\ + \frac{1}{A_{(i,j,k,l)} - D_{(i,j,k,l)}}$$

Step 6: $\forall i, j, k, l,\ 0 \le i, j, k, l \le n - 1$

$$B_{(i,j,k,l)} := P_N(x_{n^3 i + n^2 j + nk + l + 1})$$

$$C_{(i,j,k,l)} := P'_N(x_{n^3 i + n^2 j + nk + l + 1})$$

$$B_{(i,j,k,l)} := \frac{B_{(i,j,k,l)}}{C_{(i,j,k,l)}}$$

$$A_{(i,j,k,l)} := A_{(i,j,k,l)} - \frac{B_{(i,j,k,l)}}{1 - (B_{(i,j,k,l)} \times E_{(i,j,k,l)})}$$

Step 7: Stop

The time complexity for parallel Ehrlich algorithm for each iteration can be given as $N^{0.75} + 0.5N^{0.25} - 1$ electronic moves + $2(N^{0.5} - 1)$ OTIS moves. It also needs $3N^{0.75} + N^{0.25} - 1$ subtractions, $3N^{0.75} + N^{0.25} - 3$ additions, $3N^{0.75} + N^{0.25} - 1$ divisions and one multiplications in each iteration. The scalability of the above algorithm can be developed similarly as that of Parallel-DK algorithm.

5 Conclusion and discussion

Two parallel algorithms have been presented for finding the roots of an N-degree polynomial equation. The algorithms are mapped on an OTIS-2D torus using N processors. Both the algorithms following the Durand–Kerner and Ehrlich iterative schemes have been shown to run in $(N^{0.75} + 0.5N^{0.25} - 1)$ electronic moves + $2(N^{0.5} - 1)$ OTIS moves per iteration. The algorithm based on Durand–Kerner method requires $3N^{0.75} + N^{0.25} - 2$ subtractions, $3N^{0.75} + N^{0.25} - 3$ multiplications and one division in each iteration for finding the roots of N-degree polynomial. The second algorithm, based on Ehrlich method, needs $3N^{0.75} + N^{0.25} - 1$ subtractions, $3N^{0.75} + N^{0.25} - 3$ additions, $3N^{0.75} + N^{0.25} - 1$ divisions and one multiplications in each iteration. Obviously, in each iteration, the parallel Durand–Kerner algorithm is faster than the parallel Ehrlich algorithm. However, as the Ehrlich method has faster rate of convergence (cubic) than that (quadratic) of the Durand–Kerner method, the total number of iterations needed for obtaining polynomial roots by the Ehrlich would be less. The algorithms have been shown to have a lower AT cost than the parallel algorithms as described in [20] on OTIS-Mesh network. The scalability of the algorithms is also discussed.

Note that the parallel algorithms proposed in this paper, can be easily modified to design similar parallel algorithms for Lagrange and Hermite interpolation [19] (see Appendix) due to their similarity with the Durand–Kerner method.

Appendix

Given a set of tabulated values $y_1, y_2, \ldots, y_N$ of a function $y = f(x)$ at some discrete points $x_1, x_2, \ldots, x_N$, the N-point Lagrange and Hermite interpolation formulae are as follows [19].

Lagrange formula:

$$f(x) = \pi(x) \sum_{i=1}^{N} \frac{y_i}{(x - x_i)\pi'(x_i)}$$

where $\pi(x) = \prod_{i=1}^{N}(x - x_i)$, $\pi'(x_i) = \prod_{j=1, j\neq i}^{N}(x_i - x_j)$ for $i = 1, 2, \ldots, N$.

Hermite formula:

$$f(x) = \sum_{i=1}^{N} h_i(x) f(x_i) + \sum_{i=1}^{N} \bar{h}_i(x) f'(x_i)$$

where $h_i(x) = [1 - 2L_i'(x_i)(x - x_i)][L_i(x_i)]^2$, $\bar{h}_i(x) = (x - x_i)[L_i(x_i)]^2$,

$$L_i(x) = \frac{\pi(x)}{\pi'(x_i)} \quad \mathit{for}\ i = 1, 2, \ldots, N,$$

and $f'(x_i)$ and $L_i'(x_i)$ denote the derivative values of $f(x)$ and $L(x)$ respectively at the point $x = x_i$.

References

1. Aberth O (1973) Iteration method for finding zeros of a polynomial simultaneously. Math Comput 27:339–344
2. Akl SG (1985) Parallel sorting algorithm. Academic Press, Orlando
3. Anderson E, Brooks J, Gassl C, Scott S (1997) Performance of the Cray T3E. In: Proc of supercomputing conference, San Jose, California, USA
4. Arabnia HR (1993) In: Atkins S, Wagner AS (eds) A transputer-based reconfigurable parallel system, transputer research and applications (NATUG 6). IOS Press, Vancouver, pp 153–169
5. Arif Wani M, Arabnia HR (2003) Parallel edge-region-based segmentation algorithm targeted at reconfigurable multi-ring network. J Supercomput 25(1):43–63
6. Azad HS (2007) The performance of synchronous parallel polynomial root extraction on a ring multicomputer. Clust Comput 10(2):167–174
7. Ben-Or M, Feig E, Kozzen D, Tiwary P (1968) A fast parallel algorithm for determining all roots of a polynomial with real roots. In: Proc of ACM, pp 340–349
8. Bhandarkar SM, Arabnia HR (1995) The REFINE multiprocessor theoretical properties and algorithms. Parallel Comput 21(11):1783–1806
9. Bini DA, Gemignani L (2004) Inverse power and Durand–Kerner iterations for univariate polynomial root-finding. Comput Math Appl 47:447–459
10. Bose B, Broeg B, Kwon Y, Ashir Y (1995) Lee distance and topological properties of k- ary n-cubes. IEEE Trans Comput 44:1021–1030
11. Cosnard M, Fraigniaud P (1990) Finding the roots of a polynomial on an MIMD multicomputer. Parallel Comput 15:75–85
12. Day K (2002) Topological properties of OTIS-Networks. IEEE Trans Parallel Distr Syst 13:359–366
13. Durand E (1960) Solutions numeriques des equations algebriques, vol I. Mason, Paris
14. Ehrlich LW (1967) A modified Newton method for polynomials. Commun ACM 10:107–108
15. Freeman TL (1989) Calculating polynomial zeros on local memory parallel computer. Parallel Comput 12:351–358
16. Freeman TL, Bane MK (1991) Asynchronous polynomial zero-finding algorithms. Parallel Comput 17:673–681
17. Gemignani L (2007) Structured matrix methods for polynomial root finding. In: Proc of the 2007 Intl symposium on symbolic and algebraic computation, pp 175–180
18. Guggenheimer H (1986) Initial approximation in Durand–Kerner's root finding method. BIT 26:537–539
19. Hildebrand FB (1956) Introduction to numerical analysis. McGraw-Hill, New York
20. Jana PK (2006) Polynomial interpolation and polynomial root finding on OTIS-Mesh. Parallel Comput 32:301–312
21. Jana PK (1999) Finding polynomial zeroes on a Multi-mesh of trees (MMT). In: Proc of the 2nd int conference on information technology, Bhubaneswar, December 20–22, pp 202–206
22. Jana PK, Sinha BP (2006) An improved parallel prefix algorithm on OTIS-Mesh. Parallel Proc Lett 16:429–440
23. Jana PK, Sinha BP (1998) Fast parallel algorithm for Graeffe's root squaring technique. Comput Math Appl 35:71–80
24. Jana PK, Sinha BP, Datta Gupta R (1999) Efficient parallel algorithms for finding polynomial zeroes. In: Proc of the 6th int conference on advance computing, CDAC, Pune University Campus, India, December 14–16, pp 189–196
25. Kalantari B (2008) Polynomial root finding and polynomiography. World Scientific, New Jersey
26. Kerner IO (1966) Ein Gesamtschrittverfahren Zur Berechnung der Nullstetten on polynomen. Numer Math 8:290–294
27. Kessler RE, Schwarzmeier JL (1993) CRAY T3D: A new dimension for Cray research. CompCon, Houston, pp 176-182
28. Marsden G, Marchand P, Harvey P, Esener S (1993) Optical transpose interconnection system architectures. Optic Lett 18(13):1083–1085
29. Mirankar WL (1968) Parallel methods for approximating the roots of a function. IBM Res Dev 297–301
30. Mirankar WL (1971) A survey of parallelism in numerical analysis. SIAM Rev 524–547
31. Noakes M et al (1993) The J-machine multicomputer: an architectural evaluation. In: Proc of the 20th int symposium on computer architecture

32. Rajasekaran S, Sahni S (1998) Randomized routing, selection and sorting on the OTIS-Mesh. IEEE Trans Parallel Distr Syst 9:833–840
33. Rice TA, Jamieson LH (1989) A highly parallel algorithm for root extraction. IEEE Trans Comp 38(3):443–449
34. Sahni S (2001) Models and algorithms for optical and optoelectronic parallel computers. Int J Found Comput Sci 12(13):249–264
35. Sahni S, Wang CF (1998) BPC permutations on the OTIS hypercube optoelctronic computer. Informatica 263–269
36. Schedler GS (1967) Parallel numerical methods for the solution of equations. Commun ACM 286–290
37. Skachek V, Roth RM (2008) Probabilistic algorithm for finding roots of linearized polynomials. Design, codes and cryptography. Kluwer, Norwell
38. Wang CF, Sahni S (2001) Matrix Multiplication on the OTIS-Mesh optoelectronic computer. IEEE Trans Comput 50(7):635–646
39. Winogard S (1972) Parallel iteration methods in complexity of computer communications. Plenum, New York
40. Zane F, Marchand P, Paturi R, Esener S (2000) Scalable network architectures using the optical transpose interconnection system (OTIS). J Parallel Distr Comput 60:521–538
41. Zhanc X, Wan M, Yi Z (2008) A constrained learning algorithm for finding multiple real roots of polynomial. In: Proc of the 2008 intl symposium on computational intelligence and design, pp 38–41
42. Zhu W, Zeng Z, Lin D (2008) An adaptive algorithm finding multiple roots of polynomials. Lect Notes Comput Sci 5262:674–681

Keny T. Lucas obtained B.E. (Electronics and Communication) and M.Tech. (Computer Science) form Birla Institute of Technology, Mesra, India, and is presently pursuing Ph.D. in Computer Science and Engineering from Indian School of Mines University, Dhanbad, India. He has served various industries and institutions for sixteen years. He was a member of the team responsible to deploy large-scale enterprise-wide information system for the client in association with Pricewaterhouse Coopers, Calcutta. He has taught in various UG and PG courses at National Institute of Foundry and Forge Technology, Ranchi and Birla Institute of Technology, Mesra, India. Presently, he is an Assistant Professor in the Department of Information Management at Xavier Institute of Social Service, Ranchi, India. He was involved with IBM university collaborative program and was trained at IBM Research Lab, IIT Delhi campus, India. He also received research support for the Eclipse research project. He has also contributed many research papers, and his research interest includes parallel algorithm, interconnection network, and grid computing.

Prasanta K. Jana was born in Howrah, India on 12 January 1961. He has obtained M.Tech. in Computer Science from University of Calcutta, India, in 1988 and Ph.D. from Jadavpur University, Calcutta, in 2000. Currently he is an Associate Professor in the department of Computer Science and Engineering of Indian School of Mines University, Dhanbad, India. He worked as a visiting scientist at the ACM Unit of Indian Statistical Institute, Kolkata, in 2003. Dr. Jana has contributed 50 research papers in the refereed journals and conference proceedings. He has served as a reviewer of many international journals including IEEE Trans. on Parallel & Distri. Systems, IEEE Comm. Letters, J. of Parallel & Distri. Computing, J. of Pattern Recognition, Intl. J. of Computers and Applications. He has also served as the program committee member of several international conferences. His main research interests include parallel algorithms, optoelectronic parallel computing, interconnection networks, and computational biology. He visited the University of Aizu, Wakamatsu, Japan, in 2003 and Las Vegas, USA, in 2008 for academic purposes.