FILOMAT (Niš) 9:2 (1995), 273–284
Filomat '94, Niš, October 22-24, 1994
Geometry. Computer Sciences

ASYNCHRONOUS METHODS FOR SIMULTANEOUS DETERMINATION OF POLYNOMIAL ROOTS

S. Tričković, M. Trajković and M. Petković

ABSTRACT. *In this paper we present the implementation of simultaneous method for the determination of polynomial roots on a distributed memory multicomputer. The total cost of such a parallelization per iteration is the sum of a computation time and a communication time needed for a total exchange of the data at each iteration step. In order to decrease the communication time, an asynchronous implementation is considered. The computation of the root approximations is still shared among processors but the updating is performed using only nearest neighbor communications. The price to be paid to decrease this time consists in reducing the order of convergence of asynchronous methods. A general theorem which consider the lower bound of the order of convergence is given. Also, the computational efficiency of the synchronous and asynchronous versions are studied in the case of hypercube topology.*

1. Introduction

Mathematical models in scientific engineering including digital signal processing or automatic control reduce to the problem of finding roots of polynomials with degree 100 and higher [**15,16**]. In these cases the parallel processing becomes of great interest to speed up the determination of roots.

In practice, all methods for finding polynomial roots can be divided (although not strictly) in three classes: analytic, geometric and algebraic. Parallel implementation of geometric or algebraic methods often requires fine grain parallelism (see, e.g. [**2,17**]). On the other side, the multicomputers are rather composed of a network of processors with distributed memory which assumes their coarse grain parallelism. For this reason, we are interested here in analytic methods and their application on a distributed MIMD

1991 *Mathematics Subject Classification.* 65H05, 65W05.

machine. Moreover, we will restrict our study to iteration methods for the simultaneous calculation of all roots of a polynomial. As it is well known, these methods run in several identical versions so that they are very suitably for the implementation on parallel computers (see, e.g., [6,7,8,9,11,12,13]). All n roots are found simultaneously, n versions of the same algorithm can be run on a Multiple Instruction/Multiple Data (MIMD) parallel computer consisting of k ($\leq n$) processors. The main advantage of parallel implementation is that a great deal of computation can be done simultaneously. The details concerning an application of simultaneous methods on parallel computers, including an analysis of total running time of a parallel iteration, the determination of the optimal number of processors as well as experimentations, may be found in [6,7,8,9].

Many of the simultaneous methods can be written in the form

$$\mathbf{z}^{(m+1)} = F(\mathbf{z}^{(m)}), \tag{1}$$

where F is an operator in $\mathbb{C}^n$, $\mathbf{z}^{(m)} = (z_1^{(m)}, ..., z_n^{(m)})$ is a vector of approximations to the roots $\zeta_1, ..., \zeta_n$ of a given polynomial P of degree n with any initial vector $\mathbf{z}^{(0)}$. In this paper we will always assume that the initial vector $\mathbf{z}^{(0)}$ is chosen so that all $z_i^{(m)}$'s tend to the ζ_i's $(i = 1, ..., n)$. For the construction and a detailed study of simultaneous methods see the book [18].

As we have noted, we are concerned with the distributed memory multicomputers. Such parallel computers are modeled by a connected graph. The vertices of this graph are the processors and its edges are the communication links. The exchange of data between two nodes which are not directly connected must pass through different other nodes. Hence, the communication strategy has a great influence to the efficiency of the applied method. Our aim is to establish such a strategy which will decrease the communication time and, at the same time, preserve the computational efficiency of the implemented method.

In practice, the implementation of simultaneous methods on parallel computers is usually performed by three standard network topologies: rings, torus and hypercubes. In short, the model is as follows: k is the number of processors connected through a regular graph of diameter D and degree d. The exchange cost of length L messages between two neighbor processors is the sum of a start up β_c and a propagation time proportional to the message length $L\tau_c$, that is $T_{\text{One-to-one}} = \beta_c + L\tau_c$. Furthermore, one assumes that a processor can communicate simultaneously with all its neighbors (link-bound model), and that the links are full duplex. The arithmetic cost is modeled by the computation time τ_a, where τ_a is usually the mean of a

floating point addition and the floating point multiplication. Typical values of τ_c, τ_a and β_c for several types of multiprocessors can be found in the paper [4]. We emphasize that in our analysis the times for computing the starting points and checking the stopping criteria will be neglected. Accordingly, the computational cost of algorithms is the sum of a computation time with a communication time.

The investigation based on the parameters τ_c, τ_a and β_c shows that the network topology has a great influence on the global cost of parallel methods. It has been shown in [7] that, among three mentioned standard topology, the hypercube is the best topology and the relation

$$T_{\text{total}}^{\text{hypercube}} \leq T_{\text{total}}^{\text{torus}} \leq T_{\text{total}}^{\text{ring}}$$

holds. Another advantage of the hypercube topology appears when full parallelism (the degree of a polynomial is equal to the number of processors) is available. Namely, the communication time of an iteration grows linearly with the degree on a ring of processors, with the square root of the degree for a torus, but only logarithmically on a hypercube.

2. Implementation of synchronous methods

Before demonstrating the strategy which decreases the communication time, we present the implementation of so-called **synchronous parallel methods** (like (1)) [8,9]. The term **"synchronous"** does not refer here to the control mode of the multicomputer, but refer to the structure of the algorithm. Actually, considering the iteration formula (1), the next approximate vector $\mathbf{z}^{(m+1)}$ is calculated using the most recent components of $\mathbf{z}^{(m)}$.

In the parallelization of the parallel algorithm (1) we assume that the number of processors k ($\leq n$) is given in advance. The starting vector $\mathbf{z}^{(0)}$ is computed by all the processors $P_1, \ldots, P_k$ using some suitable search procedure (see, e.g., [5,10,14]). Furthermore, each step of the algorithm consists in sharing the computation of n improved approximations $z_1^{(m)}, \ldots, z_n^{(m)}$ among the processors and in updating their data $\mathbf{z}^{(m)}$ through a broadcast procedure (shorter $BCAST(\mathbf{z}^{(m)})$). As in [7], let $I_1, \ldots, I_k$ be disjunctive partitions of the set $\{1, \ldots, n\}$ where $\cup I_j = \{1, \ldots, n\}$. To obtain good load balancing between the processors, the index sets $I_1, \ldots, I_k$ are chosen so that the number of their components $\omega(I_j)$ $(j = 1, \ldots, k)$ is determined as $\omega(I_j) \leq \left[\frac{n}{k}\right]$. At the m–th iteration step the processor P_j $(j = 1, ..., k)$ computes $z_i^{(m)}$ for all $i \in I_j$ by the iteration formula (1) and then it transmits

these values to all other processors using a broadcast procedure (referred to as $BCAST(\mathbf{z}^{(m)})$. The program terminates when some stopping criterion (referred to as $STOP(\mathbf{z}^{(m)})$) is fulfilled, for instance, if

$$\max_{1\le i\le n} \left|P\left(z_i^{(m)}\right)\right| < \delta$$

for a given sufficiently small δ. According to the previous we give a program in pseudocode for a parallel implementation of a simultaneous method (1) (following [7]):

Program SYNCHRONOUS SIMULTANEOUS METHOD
begin
  for all $j = 1, \ldots, k$ **do** determination of the starting approximations $\mathbf{z}^{(0)}$;
  $m := 0$
  do
    for all $j = 1, \ldots, k$ **do in parallel**
    begin
(*) Compute $z_i^{(m+1)} := F_i(\mathbf{z}^{(m)})$, $i \in I_j$
(**) Communication: $BCAST\left(\mathbf{z}^{(m+1)}\right)$;
    end
    $m := m + 1$
  until $STOP\left(\mathbf{z}^{(m)}\right)$ holds true;
  OUTPUT $\mathbf{z}^{(m)}$
end

As it was presented in [18, Ch. 6], the number of basic arithmetic operations for a wide class of simultaneous iteration methods can be given in the form

$$N(n) = \alpha n^2 + \beta n + \gamma,$$

where n is the polynomial degree and α, β and γ are integers. Dealing with sufficiently large n we can take approximately that $N(n) = \alpha n^2 + o(n^2)$. Following [8], the total time for the presented implementation of the synchronous method can be expressed as the sum of the computation time $N(n)\tau_a/k$ and the communication time $D\beta_c + O(n/d)\tau_c$, that is

$$T_{\text{syn}} = \left(\frac{\alpha n^2 + o(n^2)}{k}\right)\tau_a + D\beta_c + O\left(\frac{n}{d}\right)\tau_c. \tag{2}$$

From (2) we see that the communication time cannot be neglected; moreover, it has a great influence on the total execution time and appears to be a major shortcoming of this parallelization of the simultaneous methods.

3. Asynchronous simultaneous methods

In order to decrease the communicate time the following strategy can be applied [**3,9,13**]: In each iteration, a processor does not have to wait at predetermined points, for example, the end of the total-exchange, for predetermined messages to become available. This type of algorithms is called **asynchronous** by Baudet [**1**]. The term **"asynchronous"** only refers to the fact that, at each step, the local computation is performed using only a part of the global information. An asynchronous algorithm can be modeled as follows:

Assume that the new approximation $z_i^{(m+1)}$ is calculated by a processor P_h, $h \in \{1, ..., k\}$. Evidently, this processor *must know* the value of $z_i^{(m)}$ and $z_i^{(m+1)}$ is calculated by the formula

$$z_i^{(m+1)} = F_i(\mathbf{z}^{(m^*)}), \quad \text{where} \quad \mathbf{z}^{(m^*)} = (z_1^{(m-r(1,m,h))}, \ldots, z_n^{(m-r(n,m,h))}). \tag{3}$$

In (3) $\mathbf{z}^{(m^*)}$ is the vector of the last values z_j known by the processor P_h at step m, represented by $z_j^{(m-r(j,m,h))}$. Here $r(j,m,h)$ is a **delay** depending on j, m and h and indicating that the processor P_h only knows the value of z_j computed at step $m - r(j,m,h)$. In the sequel, the maximum delay will be denoted by r, that is, $r = \max_{j,k,h} r(j,m,h)$.

The presented asynchronous algorithm (3) will run if the following strategy of distribution of the indices is chosen:

1. $z_i^{(m+1)}$ is calculated by only one processor for all $i = 1, ..., n$;
2. $r(i,m,h) = 0$, that is, the processor P_h must know $z_i^{(m)}$, $i \in I_h$.

Hypothesis 1 insures that there is no redundancy in the computation. Hypothesis 2 has already been discussed and it must be satisfied to provide the convergence of the sequence $(z_i^{(m)})$. Besides, in regard to Hypothesis 1, this implies that at step m, if $z_i^{(\mu)}$ $(0 \leq \mu \leq m,\ i \in \{1, \ldots, n\})$ is known by a set of processors, its value is the same for all these processors.

A short analysis given in [**9**] shows that, excepting hypotheses 1 and 2, some additional conditions must be satisfied. Namely, if each processor always updates the same components, then each processor will know the most recent entries of the components which are updated in its neighborhood, but the other components will never be updated. This causes that, for each $h \in \{1, ..., k\}$ and at each iteration step m, there exists j such that the processor P_h knows only the value of $z_j^{(0)}$, that is, $r(j,m,h) = m$. This fact

implies that the convergence is not insured. For this reason the strategy of the implementation of asynchronous methods should provide such indices distribution that I_h be different in each iteration step and the delay $r(j,m,h)$ is bounded above by some $\rho < m$. If $I(h,m)$ denotes the set of indices of the components updated by the processor P_h at step m, then the mentioned conditions can be expressed as follows:

(i) $I(h,m)$ $(h = 1,\dots,k)$ form a partition of $(1,...,n)$ at each step m;

(ii) If $i \in I(h,m)$, then the processor P_h knows $z_i^{(m)}$, $i \in I_h$;

(iii) For each processor P_h and for each component i there exists an integer ρ such that the sets $I(h,m)$ have the property that the number of steps separating two evaluations of this component in the neighborhood of h does not exceed the delay ρ.

In this way, (i) and (ii) imply that the hypotheses 1 and 2 respectively are satisfied, while (iii) provides the convergence with $r = \rho$. If these conditions are fulfilled, then a a program in pseudocode for a parallel implementation of an asynchronous simultaneous method (3) is as follows:

Program ASYNCHRONOUS SIMULTANEOUS METHOD
begin
  for all $j = 1,\dots,k$ **do** determination of the starting approximations $\mathbf{z}^{(0)}$;
  $m := 0$
  do
    for all $j = 1,\dots,k$ **do in parallel**
    begin
(0) Compute $I(h,m)$;
(1) Compute $z_i^{(m+1)} := F_i(\mathbf{z}^{(m)})$, $i \in I(h,m)$
(2) Send z_i, $i \in I(h,m)$, to neighbors;
    end
    $m := m + 1$
(3) **until** $STOP(\mathbf{z}^{(m)})$ holds true;
  OUTPUT $\mathbf{z}^{(m)}$
end

The checking of the stoping criteria (3) is more difficult in the case of the asynchronous implementation since there are the possibility that some processors verify the stop condition but not the other ones. For more details about the detection of termination see [3]. We only note that the step (3) has to include such a strategy which synchronize the processors, namely, the first processors which terminate may signal the end of the execution to the others.

The implementation of an asynchronous method is executed in such a way that, at each iteration step, a processor sends the most recently computed entries to its neighbors only. As in the case of synchronous algorithm, the computation time is again $N(n)\tau_a/k$, but the communication time becomes $\beta_c + O(n/k)\tau_c$ since it corresponds to sending n/k values from each nodes to their neighbors in parallel. Therefore, the total time per one iteration step is

$$T_{\text{asy}} = \Big(\frac{\alpha n^2 + o(n^2)}{k}\Big)\tau_a + \beta_c + O\big(\frac{n}{k}\big)\tau_c. \tag{4}$$

Comparing with (2), we obtain **one** start up instead of D and a propagation time of $O(n/k)$ instead of $O(n/d)$. Thus, the communication cost is decreased by $(D-1)\beta_c + O(n/d - n/k)\tau_c$. However, the order of convergence of the asynchronous method is reduced (see the next section), which could increase the number of iteration steps. If these two (contradictory) features can be balanced in a satisfactory way (by the choice of a suitable network topology, a good strategy for the indices distribution and synchronization of stop test, and an efficient iteration algorithm), then we can hope that the implemented asynchronous algorithm be more efficient that the corresponding synchronous algorithm.

4. R-order of convergence of asynchronous methods

Let $\epsilon_m^{(m)} = z_i^{(m)} - \zeta_i$ be the error of an asynchronous method of the form (3) which generates the sequences $(z_i^{(m)})$ of approximations to the roots $\zeta_1, \ldots, \zeta_n$. For a wide class of iteration methods for the simultaneous determination of polynomial roots (see the book [18]), the following relation can be derived:

$$\epsilon_i^{(m+1)} = \alpha_i\Big(\epsilon_i^{(m)}\Big)^q \sum_{\substack{j=1 \\ j\neq i}}^{n} \beta_{ij}\epsilon_j^{(m-r(j,m,h))} \quad (i = 1, ..., n), \tag{5}$$

where α_i and β_{ij} are complex constants and $q \geq 1$ is integer. Then we have the following assertion:

Theorem 1. *Suppose that a polynomial P has only simple roots $\zeta_1, \ldots, \zeta_n$ and starting approximations $z_1^{(0)}, \ldots, z_n^{(0)}$ are reasonably close to these roots. Further, assume that $r(j,m,h)$ is bounded for all $j = 1, \ldots, n$ and all $h = 1, ..., k$.* Then the asynchronous algorithm (3) for which the relations (5) are

valid is locally convergent with the order of convergence at least η_A, where η_A is the only positive root of the equation

$$\eta^{r+1} - q\eta^r - 1 = 0, \quad r = \max_{j,m,h} r(j,m,h). \tag{6}$$

Proof. Under the conditions of theorem we can find the constant A and B so that $|\alpha_i| \leq A$ and $|\beta_{ij}| \leq B$. If at the iteration step m we define the absolute error e_m by $e_m = ||\mathbf{z}^{(m)} - \zeta||_\infty$, then from (5) we obtain

$$e_{m+1} \leq C\epsilon_m^q \epsilon_{m-r} \quad \text{with } C = (n-1)AB. \tag{7}$$

In regard to the assumed closeness of root approximations we can adopt that $e_0 < 1$. Then from (7) it follows that the sequence (e_m) tends to zero.

Let $\epsilon_0 = O(E)$, where $0 < E < 1$ and let the order of convergence of the sequence (e_m) be η_A, that is, $e_{m+1} = O(e_m^{\eta_A})$. Then

$$e_m = \left(E^{\eta_A^m}\right), \quad e_{m-r} = O\left(E^{\eta_A^{m-r}}\right), \quad (r = 0, 1, \ldots, m).$$

From (7) we obtain

$$e_{m+1} = O\left(e_m^q \cdot e_{m-r}\right) = O\left(E^{q\eta_A^m + \eta_A^{m-r}}\right).$$

According to the last relation and the fact that $e_{m+1} = O\left(E^{\eta_A^{m+1}}\right)$, by the comparision of the exponents it follows

$$\eta_A^{r+1} = q\eta_A^r + 1.$$

Hence, $\eta_A > 0$ should satisfy the equation (6). □

Remark 1. Let $y(\eta) = \eta^{r+1} - q\eta^r - 1$. Since $y(q) = -1 < 0$ and $y(q+1) = (q+1)^r - 1 \geq 0$, and taking into account that the equation $y(\eta) = 0$ has the unique positive root, it follows that the order of convergence η_A of the asynchronous method belongs to the interval $(q, q+1]$. Particularly, if $r = 0$ for all $i = 1, ..., n$ and $m = 0, 1, ...$, which means that we have a synchronous method, one obtains the order of this method $\eta_S = q + 1$.

The lower bound of the order of convergence of asynchronous methods as the function of the delay r is given in Table 1. We observe that r has a very strong influence on the value of the convergence rate so that the main problem which has to be solved in the implementation consists of the minimization of the delay r.

	r				
q	0	1	2	3	4
1	2	1.618	1.466	1.380	1.325
2	3	2.414	2.206	2.107	2.056
3	4	3.303	3.104	3.036	3.012

Table 1

The lower bounds of the order of convergence of asynchronous methods in function of the delay r

5. Efficiency of implementation

In this section we will compare the implementation of an asynchronous method and corresponding synchronous method on a hypercube multicomputer. Let N_{syn} and N_{asy} be respectively the number of iteration steps of a synchronous and an asynchronous method. Evidently, the asynchronous method will be more efficient if $N_{\text{asy}}T_{\text{asy}} < N_{\text{syn}}T_{\text{syn}}$. By virtue of (2) and (4) this inequality may be written as follows:

$$N_{\text{asy}}\Big[\Big(\frac{\alpha n^2+o(n^2)}{k}\Big)\tau_a+\beta_c+O\big(\frac{n}{k}\big)\tau_c\Big] < N_{\text{syn}}\Big[\Big(\frac{\alpha n^2+o(n^2)}{k}\Big)\tau_a+D\beta_c+O\big(\frac{n}{d}\big)\tau_c\Big]. \quad (8)$$

Since the relation $\beta_c \gg \tau_c$ generally occurs ([4]) on distributed memory computers, the inequality (8) becomes

$$\frac{N_{\text{asy}}}{N_{\text{syn}}} < \frac{\dfrac{\alpha n^2}{k}+D\dfrac{\beta_c}{\tau_a}}{\dfrac{\alpha n^2}{k}+\dfrac{\beta_c}{\tau_a}}. \quad (9)$$

This inequality will be considered together with the condition

$$\frac{\beta_c}{\tau_a} < \frac{\alpha n^2}{k}. \quad (10)$$

Namely, if $\beta_c/\tau_a > \alpha n^2/k$, then it could be faster to use less processors in the synchronous implementation (see [8]).

Eventual validity of the inequality (9) can be suitable verified by a graphical interpretation in the plane $\Big(\dfrac{N_{\text{asy}}}{N_{\text{syn}}}, \dfrac{\beta_c}{\tau_a}\Big)$. A typical graph is displayed in

Fig. 1. The intersection of the curve

$$\frac{\beta_c}{\tau_a} = \frac{\frac{\alpha n^2}{k}\left(1 - \frac{N_{\text{asy}}}{N_{\text{syn}}}\right)}{\frac{N_{\text{asy}}}{N_{\text{syn}}} - D}, \tag{11}$$

and the horizontal bound line $\beta_c/\tau_a = \alpha n^2/k$, which are obtained from (9) and (10) taking the sign "=" instead of "<", gives the area where the asynchronous implementation could be faster (shaded area). Of course, this area will be feasible only if the network topology is such that the ratio β_c/τ_a (dashing line on Fig. 1) is smaller than the bound value $\alpha n^2/k$ (full horizontal line). In fact, the realistic area where the asynchronous algorithm could be faster is bounded by the curve (11) and the realistic parametric ratio $R := \beta_c/\tau_a$ (darker shaded area). This ratio usually belongs to the interval $[10^2, 10^3]$ (see [4]). But, these conditions are not still sufficient. Let V is the **critical ratio** which is given by the absissa of the intersection of the curve (11) and the dashing line, that is (from (9) for $\beta_c/\tau_a = R$),

$$V = \frac{\frac{\alpha n^2}{k} + DR}{\frac{\alpha n^2}{k} + R}.$$

Then the asynchronous implementation will be more efficient only if the ratio $N_{\text{asy}}/N_{\text{syn}}$ can be realized in practice, that is, if $N_{\text{asy}}/N_{\text{syn}} < V$.

We give a short analysis for a theoretical value of the ratio $N_{\text{asy}}/N_{\text{syn}}$ taking into account the accuracy of the initial errors $|z_i^{(0)} - \zeta_i|$, the required accuracy δ and the orders of convergence η_A and η_S of the asynchronous and synchronous methods respectively. Besides, we assume that (complex) roots of tested polynomials are normalized to lie in the unit disk. In that case, a stopping criterion can be given by

$$\max_{1 \le i \le n} |z_i^{(m)} - \zeta_i| < \delta = 10^{-\nu},$$

where m is the iteration index and ν is the number of significant decimal digits at the approximations $z_1^{(m)}, \ldots, z_n^{(m)}$. If $|z_i^{(0)} - \zeta_i| = O(10^{-1})$ and η is the order of convergence of applied simultaneous method, then the (theoretical) number of iteration steps, necessary for obtaining the accuracy δ, can be determined approximately as $m \cong \log \nu / \log \eta$ (following from $10^{-\nu} = 10^{-\eta^m}$). According to this we could expect that the ratio $N_{\text{asy}}/N_{\text{syn}}$ be approximately

$$\frac{N_{\text{asy}}}{N_{\text{syn}}} \cong \frac{\log \eta_S}{\log \eta_A}. \tag{12}$$

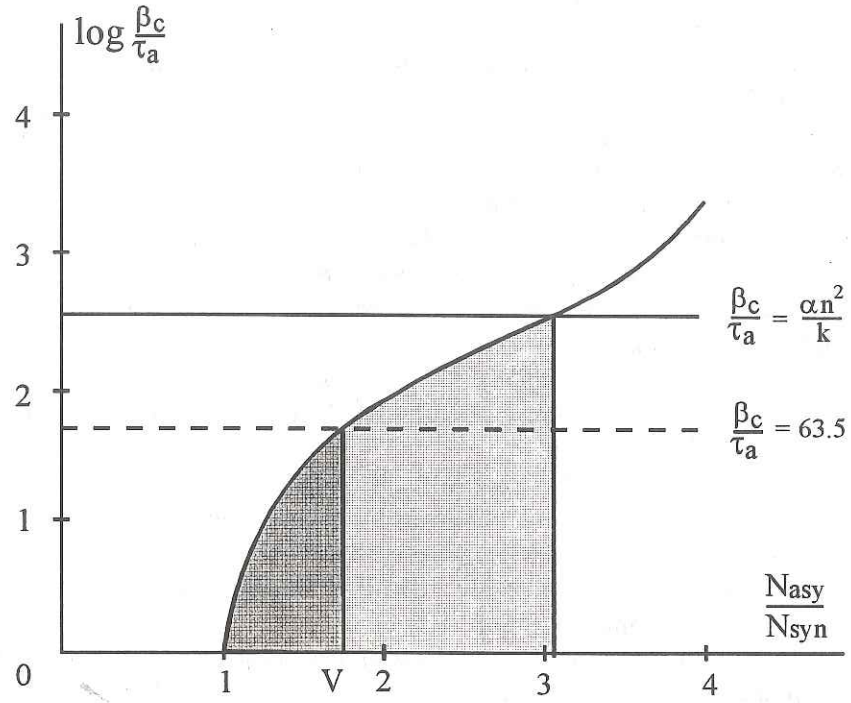

Fig. 1 Dominant area of asynchronous implementation

Finally, from a theoretical point of view, an asynchronous method will be more efficient than the corresponding synchronous method if

$$\log \eta_S / \log \eta_A < V = \frac{\frac{\alpha n^2}{k} + DR}{\frac{\alpha n^2}{k} + R} = 1 + \frac{D-1}{\frac{\alpha n^2}{kR} + 1}.$$

References

[1] G.M. Baudet, *Asynchronous iterative methods for multiprocessors.* J. of ACM **2** (1978), 226-244.

[2] M. Ben-Or, E. Freig, D. Kozen and P. Tiwari, *A fast parallel algorithm for determining all roots of a polynomial with real roots.* Proc. ACM (1989), 340-349.

[3] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and distributed computation - numerical methods.* Prentice-Hall Inc. 1989.

[4] L. Bomans and D. Roose, *Communication benchmarks for the iPSC/2.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 93-104.

[5] D. Braess and K. Hadeler, *Simultaneous inclusion of the zeros of a polynomial.* Numer. Math. **21** (1973) 161-165.

[6] M. Cosnard and P. Fraigniaud, *Asynchronous Durand-Kerner and Aberth polynomial root finding methods on a distributed memory multicomputer.* Parallel Computing **9** (1989) 79-84.

[7] M. Cosnard and P. Fraigniaud, *Finding the roots of a polynomial on an MIMD multicomputer.* Parallel Computing **15** (1990) 75-85.

[8] M. Cosnard and P. Fraigniaud, *Asynchronous polynomial root finding methods.* Research report 90-21, LIP-IMAG, Ecole Normale Supérieure de Lyon, France 1990.

[9] M. Cosnard and P. Fraigniaud, *Analysis of asynchronous polynomial root finding methods on a distributed memory multicomputer.* IEEE Transaction on Parallel and Distributed Systems (to appear).

[10] M.R. Farmer and G. Loizou, *Locating multiple zeros interactively.* Comput. Math. Appl. **11** (1985) 595-603.

[11] P. Fraigniaud, *Performance analysis of broadcasting in hypercubes.* Hypercube and Distributed Computers (Proc. I European Workshop on hypercube and Distributed Computers, eds. F. Andre and J. P. Verjus), North Holland, Amsterdam 1989, pp. 311-328.

[12] T.L. Freeman, *Calculating polynomial zeros on a local memory parallel computer.* Parallel Computing **12** (1989) 351-358.

[13] T.L. Freeman and M.K. Bane, *Asynchronous polynomial zero-finding algorithms.* Parallel Computing **17** (1991) 673-681.

[14] H. Guggenheimer, *Initial approximations in Durand-Kerner's root finding method.* BIT **26** (1986) 537-539.

[15] L.H. Jamieson and T.A. Rice, *A highly parallel algorithms for root extraction.* IEEE Trans. on Comp. **28** (1989), 443-449.

[16] J.L. Nicolas and A. Schinzel, *Localisation des zéros de polynomes intervenant end théorie du signal.* Reserach report, University of Lyon 1, 1988.

[17] V. Pan, *Sequential and parallel complexity of approximate evaluation of polynomial zeros.* Comput. Math. Appls **14** (1987), 591-622.

[18] M.S. Petković, *Iterative methods for simultaneous inclusion of polynomial zeros. Springer-Verlag, Berlin-Heidelberg-New York 1989.*

Faculty of Electronic Engineering, P.O. Box 73, 18 000 Niš