Proceedings of the Mardi Gras '94 Conference

TOWARD TERAFLOP COMPUTING AND NEW GRAND CHALLENGE APPLICATIONS

February 10 - 12, 1994 Louisiana State University

Editors

RAJIV K. KALIA AND PRIYA VASHISHTA
Louisiana State University

NOVA SCIENCE PUBLISHERS, INC.

Toward Teraflop Computing
and New Grand
Challenge Applications

ISBN-1-56072-247-9

SOLVING LINEAR RECURRENCE SYSTEMS ON PARALLEL COMPUTERS

Marcin Paprzycki[1] Przemyslaw Stpiczyński[2]

[1] Department of Mathematics and Computer Science, University of Texas of the Permian Basin, Odessa, TX 79762, e-mail: paprzycki_m@gusher.pb.utexas.edu
[2] Numerical Analysis Department, Marie Curie-Skłodowska University, Pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin, POLAND, e-mail: przem@golem.umcs.lublin.pl

Abstract. Two divide and conquer algorithms for the solution of a linear recurrence system of order m of n equations are introduced. The arithmetical complexity of the proposed algorithms is discussed. It is shown that for large n the optimal number of processors is $O(\sqrt{n})$ and that for the optimal number of processors the time complexity of the proposed algorithms is $O(\sqrt{n})$. It is also shown that for fixed n and fixed number of processors p the theoretical speed-up decreases as the order of the system m increases. The results of experiments on a 20-processor Sequent are presented.

Keywords. Linear recurrence systems, parallel algorithms, divide-and-conquer, shared-memory multiprocessors, speedup.

1. INTRODUCTION

The final phase of several numerical algorithms reduces to the solution of a linear recurrence system of order m for n equations. Many parallel algorithms for solving this problem have been proposed [1–4, 6–9, 11, 12]. In [12] two medium grain efficient divide and conquer parallel algórithms were presented and their theoretical implementation on a linear array of processors with message passing facilities was discussed. It has been recently shown that both algorithms have very good numerical properties [13].

The major purpose of this paper is to show that these algorithms can be efficiently implemented on shared memory MIMD multiprocessors. Section 2 contains a short description of the proposed algorithms. Their arithmetical complexity functions are presented and discussed in Section 3. In Section 4 the results of the experiments on a 20-processor Sequent Symmetry are presented and analyzed.

2. PARALLEL ALGORITHMS

Let us define a linear recurrence system of order m for n equations, where $m \le n$:

$$x_k = \begin{cases} 0 & \text{if } k \le 0, \\ f_k + \sum_{j=k-m}^{k-1} a_{kj} x_j & \text{if } 1 \le k \le n. \end{cases} \tag{1}$$

Computation of values x_k reduces to the system of linear equations

$$(I - A)\mathbf{x} = \mathbf{f} \qquad \text{where } I, A \in R^{n \times n} \text{ and } \mathbf{x}, \mathbf{f} \in R^n, \tag{2}$$

where $\mathbf{x} = (x_1, \ldots, x_n)^T$, $\mathbf{f} = (f_1, \ldots, f_n)^T$ and $A = (a_{ik})$ with $a_{ik} = 0$ for $i \le k$ or $i - k > m$.

Without loss of generality we can assume that n is divisible by p (where 'p' is the number of processors) and $q = n/p > m$. The system (2) can be written in the following block form

$$\begin{pmatrix} L_1 & & & & \\ U_2 & L_2 & & & \\ & U_3 & L_3 & & \\ & & \ddots & \ddots & \\ & & & U_p & L_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{p-1} \\ \mathbf{f}_p \end{pmatrix}, \tag{3}$$

where $L_j \in R^{q \times q}$, $\mathbf{x}_j, \mathbf{f}_j \in R^q$ for $j = 1, \ldots, p$ and $U_j \in R^{q \times q}$ for $j = 2, \ldots, p$. Vectors $\mathbf{x}_j$ satisfy the following recurrence relation

$$\begin{cases} \mathbf{x}_1 = L_1^{-1} \mathbf{f}_1, \\ \mathbf{x}_j = L_j^{-1} \mathbf{f}_j - L_j^{-1} U_j \mathbf{x}_{j-1} & \text{for } j = 2, \ldots, p. \end{cases} \tag{4}$$

Let U_j^k denote the kth column of the matrix U_j. Since $U_j^k = 0$ for $j = 2, \ldots, p$ and $k = 1, \ldots, q - m$ we obtain the following formula for Algorithm 1:

$$\begin{cases} \mathbf{x}_1 = L_1^{-1} \mathbf{f}_1, \\ \mathbf{x}_j = \mathbf{s}_j - \sum_{k=0}^{m-1} x_{(j-1)q-k} \mathbf{y}_j^k & \text{for } j = 2, \ldots, p, \end{cases} \tag{5}$$

where $L_j \mathbf{s}_j = \mathbf{f}_j$ and $L_j \mathbf{y}_j^k = U_j^{q-k}$ for $k = 0, \ldots, m - 1$.

To develop the second algorithm let us observe that each matrix U_j can be written as

$$U_j = -\sum_{k=1}^{m} \sum_{l=k}^{m} \beta_j^{kl} \mathbf{e}_k \mathbf{e}_{q-m+l}^T, \tag{6}$$

where $\beta_j^{kl} = a_{(j-1)q+k,(j-1)q-m+l}$ and $\mathbf{e}_k$ denotes the kth unit vector of R^q. Substituting into (5) we obtain the following formula for Algorithm 2 (for more detailed description of both algorithms see [12])

$$\begin{cases} \mathbf{x}_1 = L_1^{-1} \mathbf{f}_1, \\ \mathbf{x}_j = \mathbf{s}_j + \sum_{k=1}^{m} \alpha_j^k \mathbf{y}_j^k & \text{for } j = 2, \ldots, p, \end{cases} \tag{7}$$

where $\alpha_j^k = \sum_{l=k}^{m} \beta_j^{kl} \mathbf{e}_{q-m+l}^T \mathbf{x}_{j-1}$, $L_j \mathbf{y}_j^k = \mathbf{e}_k$ for $k = 1, \ldots, m$, and $L_j \mathbf{z}_j = \mathbf{f}_j$.

Methods (5), (7) are examples of *divide-and-conquer* algorithms. First, several linear recurrence systems of order m for q equations are to be solved separately (in parallel). Second, a set of values needs to be calculated in ~~each~~ block and communicated to the next block effectively decoupling the original system. This is the sequential part of all three algorithms. Finally, the solution to the original problem is calculated independently in each block (in parallel).

3. COMPLEXITY OF THE ALGORITHMS

Let us first modify for the shared memory computer the arithmetical complexity formulas for (1) formally developed in [12]. Let τ_a denote the time of the execution of the basic arithmetic operation *multiply and add*. The complexity of a sequential algorithm for solving (1) can be expressed by the following formula

$$T_1(n) = m\left(n - \tfrac{m+1}{2}\right)\tau_a. \tag{8}$$

The complexity of Algorithm 1 is represented by:

$$T_{p,1}(n) = m\left[mp + (m+2)\tfrac{n}{p} - \tfrac{1}{2}(m^2 + 6m + 1)\right]\tau_a; \tag{9}$$

whereas the complexity of Algorithm 2 is defined by:

$$T_{p,2}(n) = m\left[(\tfrac{3}{2}m + \tfrac{1}{2})p + (m+2)\tfrac{n}{p} - (m^2 + \tfrac{9}{2}m + \tfrac{1}{2})\right]\tau_a, \tag{10}$$

A number of observations can be made about the above complexity functions.

1. The arithmetical complexity functions for the parallel algorithms have the same overall structure as the functions for other divide and conquer algorithms (see for instance [5, 10]).

2. Algorithm 2 gives better time complexity results for a small number of processors.

3. For small m and large n the optimal number of processors for both parallel algorithms is of order $O(\sqrt{n})$.

4. For small m, large n and an optimal number of processors $p_{opt} = O(\sqrt{n})$ both parallel algorithms perform in time $O(\sqrt{n})$.

5. For fixed p and m the increase in n does not increase the speedup.

6. For fixed n and p the increase of m decreases the speedup.

The last two properties are illustrated on Figure 1 which represents the predicted speedup for various values of m, optimal and fixed p and for $n = 1000, \ldots, 10,000$. It should be noted that for $n = 10,000$ the optimal number of processors $p_{opt} \approx 100$.

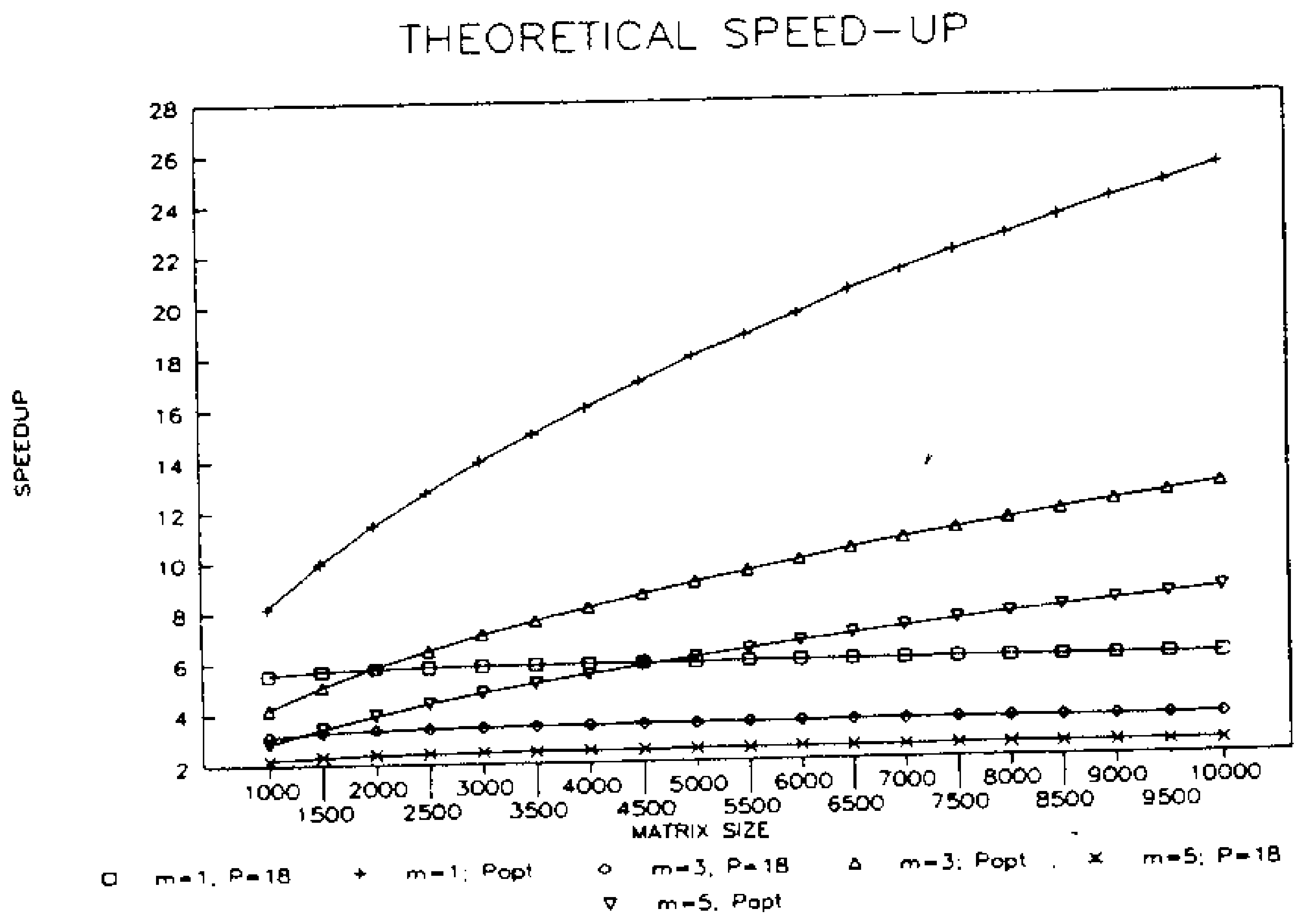

Figure 1: Effects of changes in p, m and n on the theoretical speedup.

4. RESULTS

The experiments were performed on a 20 processor Sequent Symmetry. The code was implemented using Fortran 77 and Sequent provided parallelisation primitives (DOACROSS). Each result presented here is an average of multiple runs on an empty machine. For programming simplicity it was assumed that n is divisible by p. Since one of the processors is running the operating system and since 19 is not an optimal divisor, the largest number of processors used was 18.

Figure 2 presents the comparison between speedups of the two proposed algorithms for $n = 6930$ (which was the biggest size we were able to fit into the memory for $m = 8$), $m = 1$, 4 and 8, and $p = 2, \ldots, 18$. Speedup of both algorithms was calculated against the same sequential algorithm.

It should be pointed out that the wavy shape of the curves is partially caused by the fact that the experiments were not carried out when the number of processors p was not a divisor of the system size n. As predicted, for increasing m the practical speedup decreases. There is no performance improvement for $p = 2$ as the overhead of the parallel solution is bigger than the gains from parallelisation. As m increases it takes more and more processors to obtain any speedup. It can be also observed that, in all cases, Algorithm 2 gives better performance than Algorithm 1. This is a general result that we have observed in all experiments with large n and for which we do not have a complete explanation. From now on we will report the performance of Algorithm 2 only.

Figure 3 compares the performance of Algorithm 2 for $m = 1$ for various matrix sizes. A couple of observations can be made. First, the system overhead can be observed for $p > 12$ for matrix sizes up to $n = 5040$. Only for the largest system size

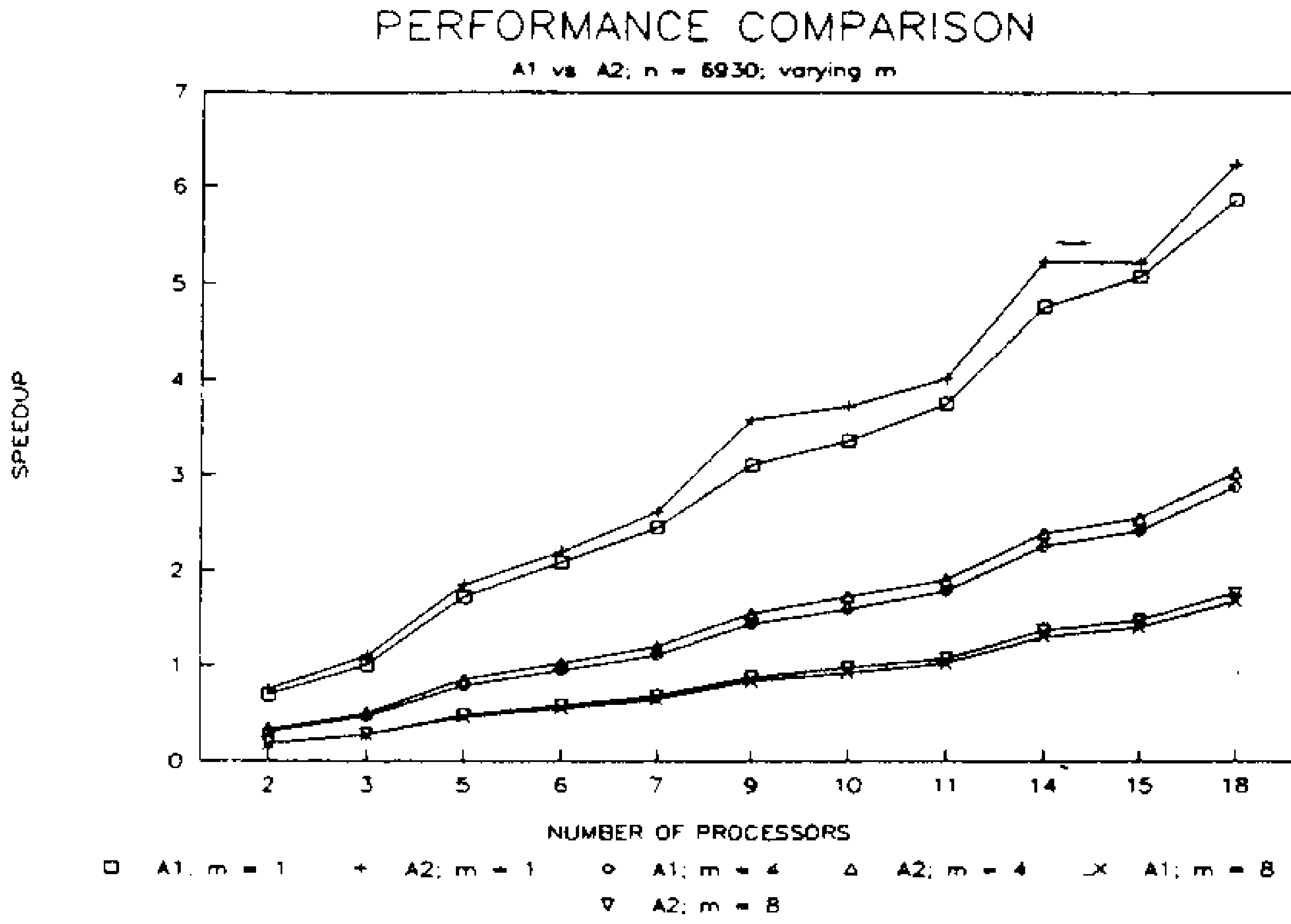

Figure 2: Comparison between the two algorithms for various m.

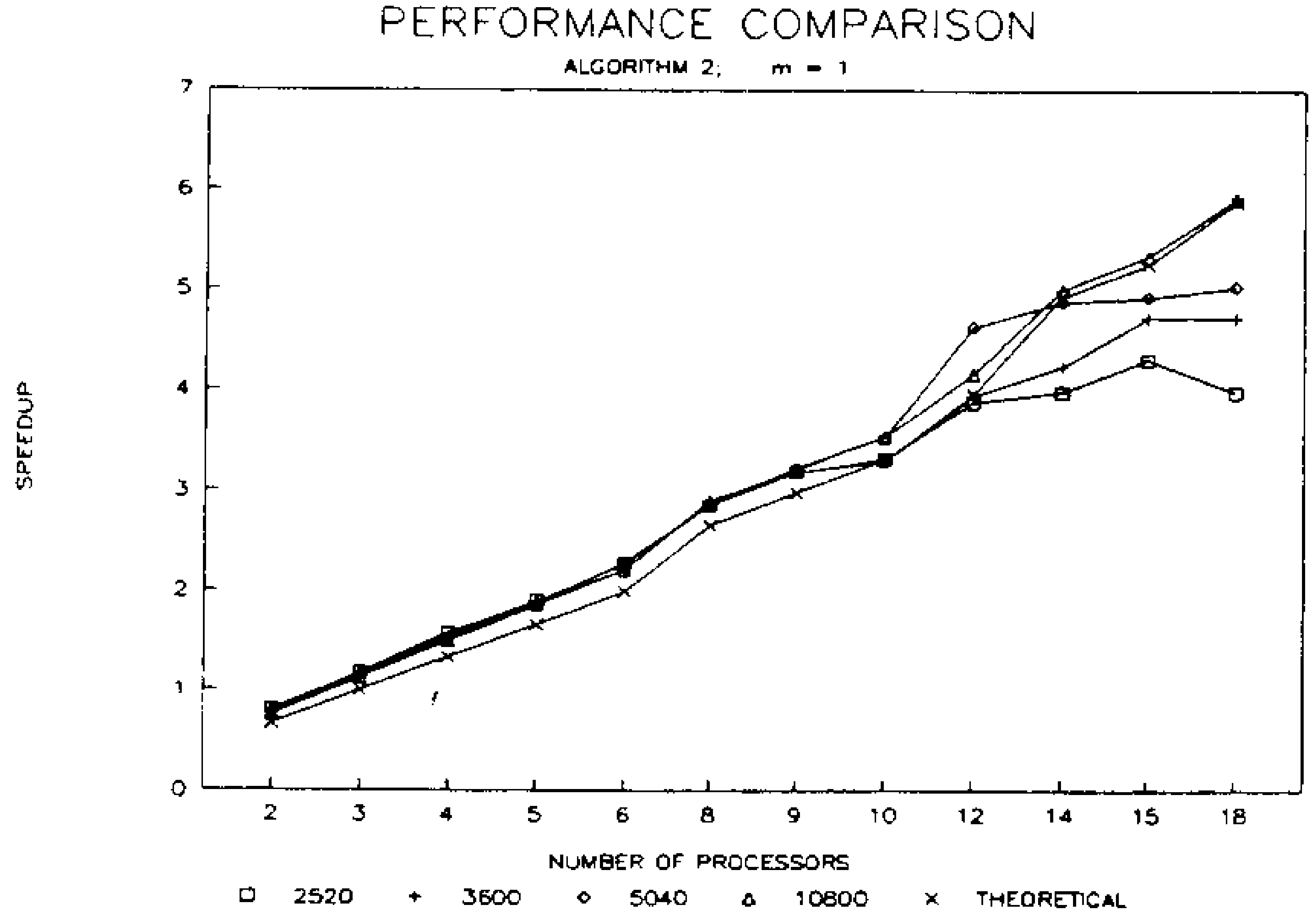

Figure 3: Performance of Algorithm 2 for various n.

the levelling-off effect disappears. Second, as it was predicted theoretically, for a given number of processors the increase in n does not produce speedup increase. Third, the theoretical functions developed in Section 3 provide surprisingly good speedup predictions (the difference between the actual and predicted speedup is smaller than 5%).

ACKNOWLEDGEMENT

Computer time grant from the Department of Computer Science Engineering of the Southern Methodist University is kindly acknowledged.

REFERENCES

1. A. Borodin and I. Munro, *The computational complexity of algebraic and numerical problems*, New York: American Elsevier, 1975.
2. D.A. Carlson, "Solving linear recurrence systems on mesh–connected computers with multiple global buses", *J. Parallel Dist. Comput.* Vol.8, 1990, pp. 89-95.
3. S.-C. Chen, Speedup of iterative programs in multiprocessing systems, University of Illinois at Urbana, 1975.
4. S.-C. Chen and D.J. Kuck, "Time and parallel processor bounds for linear recurrence systems", *IEEE Trans. Comput.* Vol. 24, 1975, pp. 701-717.
5. J. Dongarra and L. Johnsson, "Solving Banded Systems on Parallel Processor", *Parallel Comput.* Vol. 5, 1987, pp. 219-246.
6. D. Heller, "A survey of parallel algorithms in numerical linear algebra", *SIAM Review* Vol. 20, 1978, pp. 740-777.
7. A.C. Greenberg, R.E.Lander, M.S. Paterson and Z. Galil, "Efficient parallel algorithms for linear recurrence computation", *Inf. Proc. Letters* Vol. 15, 1982, pp. 31-35.
8. D.J. Kuck, *Structure of Computers and Computations*, New York: Wiley, 1978.
9. J. Modi, *Parallel Algorithms and Matrix Computations*, Oxford: Oxford University Press, 1988.
10. M. Paprzycki and I. Gladwell, "Solving Almost Block Diagonal Systems on Parallel Computers", *Parallel Comput.* Vol. 17, 1991, pp. 133-153.
11. A.H. Sameh and R.P. Brent, "Solving triangular systems on a parallel computer", *SIAM J. Numer. Anal.* Vol. 14, 1977, pp. 1101-1113.
12. P. Stpiczyński, "Parallel algorithms for solving linear recurrence systems", in: L. Bougé et al. eds., *Lecture Notes in Computer Science* Vol. 634, Berlin: Springer–Verlag, 1992, pp. 343-348.
13. P. Stpiczyński, "Error analysis of two parallel algorithms for solving linear recurrence systems", *Parallel Comput.* Vol. 19, 1993, pp. 917-923.