Int. J. Appl. Math. Comput. Sci., 2003, Vol. 13, No. 2, 205–214

AN EFFICIENCY ANALYSIS OF THE PARALLEL MULTITRANSPUTER IMPLEMENTATION OF TWO-LEVEL OPTIMIZATION ALGORITHMS

JAN SADECKI*

* Department of Electrical Engineering and Automatic Control
Technical University of Opole, ul. Sosnkowskiego 31, 45–233 Opole, Poland
e-mail: `jsad@po.opole.pl`

The paper presents an approach to improve the efficiency of some two-level optimization algorithms by their implementation in parallel MIMD multiprocessor systems. Diagonal decomposition dynamic programming and parametric optimization methods are considered, and some concepts of their parallelization are discussed. Results regarding the implementation of computations in a parallel multitransputer system are presented. For the analysed problems, the obtained values of speedup are close to the theoretical maximum values.

Keywords: parallel computations, multitransputer systems, parallel optimization algorithms, two-level optimization methods

1. Introduction

A number of problems associated with the efficiency of the application of the dynamic programming method to dynamic optimization problems in parallel multitransputer systems are presented in (Sadecki, 1996; 2001). Unfortunately, solving such problems, in particular for large dimensions of state and control vectors, can be very time-consuming even when parallel systems are implemented (Malinowski and Sadecki, 1990). The efficiency of the parallel implementation of such computations depends, in general, on the processor performance and communication bandwidth of the parallel system. The value of the communication bandwidth is especially important for problems for which an exchange of data between each pair of the processors is required, especially for systems containing very large numbers of processors. Highly parallel computers contain thousands of processors (Baker, 2000; Dongarra, 2002; Van der Steen and Dongarra, 2001). If, for a given problem size, communication will eventually dominate computation as the number of processors is increased, then the speedup cannot scale with large numbers of processors without introducing additional levels of parallelism (Eldred and Hart, 1998).

A broad class of optimization algorithms is often referred to as two-level methods (Findeisen, 1974), which can be easily implemented as parallel computations using one or more levels of parallelism. Decomposition is not a new idea. It is well known as a method which permits to reduce computational requirements arising, e.g., in the control of large dynamic systems (Birge and Rosa, 1995; Findeisen *et al.*, 1980; Karbowski and Niewiadomska-Szynkiewicz, 2001; Titli *et al.*, 1978). The implementation of two-level methods in parallel multiprocessor systems allows us to solve problems whose sizes made them previously unsolvable (Averick and More, 1994; Birge and Rosa, 1995), and can lead to very good results in terms of the speedup. These results can be sometimes significantly better than those obtained for currently available parallel computers, especially when a large number of processors are used (Dongarra, 2002). The analysis presented in this paper is directed at two-level algorithms, in particular related to dynamic programming methods (Bellman, 1957). Apart from the basic version of the dynamic programming method, there is a wide group of its modifications characterized by much better properties with respect to both memory and time consumption requirements (Larson, 1968; Sadecki, 1987). One of such methods is e.g. the diagonal decomposition dynamic programming (DDDP) method (Collins, 1970).

In this paper parallel implementations of this method and a two-level algorithm to control a water resource system are presented. Parallel computations were implemented in the multitransputer SUPER NODE 1000 system (Interi, 1991; Occam 2, 1988; TAN, 1989; Wysocki and Kwolek, 1994). Transputer systems permit to achieve a very good speedup of computations, especially for the analysed two-level algorithms. As will be shown, similar results can be obtained for currently available integrated highly parallel computers characterised by a good performance of processor elements and a comparatively good communication bandwidth. Some effective solutions are presented to improve the efficiency of parallel implementations of two-level algorithms.

2. Two-Level Optimization Problem

Computational burdens encountered when solving complex control problems of multidimensional processes can be essentially reduced by decomposing them into a number of subproblems and by solving a set of subtasks associated with some coordination task (Findeisen, 1974; Findeisen *et al.*, 1980; Titli *et al.*, 1978).

The following optimization problem is considered:

$$\max_{\boldsymbol{u}\in U} Q(\boldsymbol{u}). \tag{1}$$

It is assumed that the problem (1) can be rewritten in the following form:

$$\max_{\boldsymbol{u}\in U} Q(\boldsymbol{u}) = \max_{\boldsymbol{u}^1\in U^1(\boldsymbol{v}),\dots,\boldsymbol{u}^P\in U^P(\boldsymbol{v}),\boldsymbol{v}\in V} Q\big[Q_1(\boldsymbol{u}^1,\boldsymbol{v}), Q_2(\boldsymbol{u}^2,\boldsymbol{v}),\dots,Q_P(\boldsymbol{u}^P,\boldsymbol{v})\big], \tag{2}$$

where $\boldsymbol{v}$ denotes the vector of coordination variables, $\boldsymbol{u}^i$'s are some subsets of the control vector $\boldsymbol{u}$ and Q is a function of the elements $Q_1, Q_2, \dots, Q_P$, whereas Q_i depends only on $\boldsymbol{u}^i$ and $\boldsymbol{v}$. Moreover, it is assumed that the constraint relations concerning the vector $\boldsymbol{u}$ can be formulated in such a manner that P separate groups of constraints could arise in $\boldsymbol{u}^1, \boldsymbol{u}^2, \dots, \boldsymbol{u}^P$ dependent on the vector $\boldsymbol{v}$ and subject to some constraints on the vector $\boldsymbol{v}$ itself.

A two-level solution of the problem (1), making use of a decomposition determined by the relation (2), is possible when $\boldsymbol{u}^i$'s are disjoint subsets of $\boldsymbol{u}$ and when it is possible to perform a separate optimization with respect to each $\boldsymbol{u}^i$, i.e., when the performance criterion Q is either in additive or multiplicative (with the condition $Q_i(\boldsymbol{u}^i,\boldsymbol{v}) \geq 0,\ i = 1,2,\dots,P$) or mixed forms (the latter including both additive and multiplicative parts).

The solution of the problem (2) is then reduced to solving the local tasks of the form

$$\max_{\boldsymbol{u}^i\in U^i(\boldsymbol{v})} Q_i(\boldsymbol{u}^i,\boldsymbol{v}) = \hat{Q}_i(\boldsymbol{v}), \quad i = 1,2,\dots,P, \tag{3}$$

and the coordination task

$$\max_{\boldsymbol{v}\in V} Q\big[Q_1(\boldsymbol{v}), Q_2(\boldsymbol{v}),\dots,Q_P(\boldsymbol{v})\big]. \tag{4}$$

On the other hand, during the solution of the successive local tasks, the values of $\hat{\boldsymbol{u}}^i$, $Q_i(\hat{\boldsymbol{u}}^i,\boldsymbol{v})$ and, if necessary, those of the gradient $\nabla Q_i(\hat{\boldsymbol{u}}^i,\boldsymbol{v}),\quad i = 1,2,\dots,P$, are computed and then utilized for solving the coordination task whose purpose is to determine the successive approximations for the value of $\boldsymbol{v}$. The gradient $\nabla Q_i(\hat{\boldsymbol{u}}^i,\boldsymbol{v})$, $i = 1,2,\dots,P$ is computed when gradient algorithms are used for solving the coordination task. A graphical scheme of the above algorithm is presented in Fig. 1.

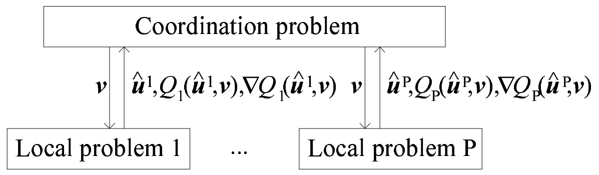


Fig. 1. Structure of a two-level algorithm.

Assuming a set of all operations required to be carried out for solving one local task as the least portion of a task which can be performed by a processor, the discussed two-level algorithm can be implemented in a simple manner parallelly in a multiprocessor system (Sadecki, 1987). Notice that the Master-Slave structure, being natural for methods of this type, is less effective for transputer systems with a large number of processors in particular (Sadecki, 2001).

Furthermore, in distributed memory systems, each of the tasks specified in a given algorithm should be solved in a parallel way, i.e., in the case of the methods considered the coordination algorithm also ought to be solved parallelly. A general diagram of such an algorithm is presented in Fig. 2. The vertical communication corresponds in principle to a data exchange between two algorithms implemented by the same processor, namely, between the algorithm implementing a local task and a fragment of the coordination algorithm implemented parallelly, allocated to this processor. On the other hand, the horizontal communication corresponds to the one between particular processors, including the exchange of data necessary for the correct implementation of the coordination algorithm.

A decrease in the computation time resulting from a parallel realization of the two-level optimization problem is rather evident. But decomposition can lead to a decrease in the communication requirements, too, especially for the each-to-each communication problem (bidirectional data

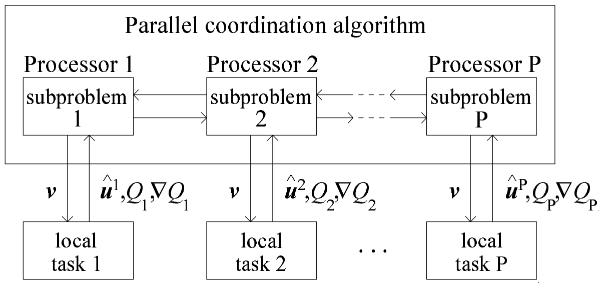


Fig. 2. Diagram of a parallel two-level algorithm.

transmission between each pair of processors). For example, the number of required communication tasks L_s for the last problem is determined as

$$L_s = P(P-1),$$

where P denotes the number of processors.

If we divide all processors into L groups, each of them containing approximately P/L processors, the total number of communication tasks required to perform the same exchange of data is determined as follows:

$$L_d = \frac{P}{L}\left(\frac{P}{L}-1\right)L + L(L-1) + \left(\frac{P}{L}-1\right)L,$$

where $(P/L)(P/L-1)L$ denotes the number of required communication tasks in all groups of processors, $L(L-1)$ denotes the number of communication tasks between all groups, and $(P/L-1)L$ denotes the total number of communication tasks permitting to send data obtained from other groups to all processors in each group.

Figure 3 presents the values of the factor $S_d = L_s/L_d$ as a function of the number of processor groups L for a given value of P. This graph shows that the communication requirements can be significantly decreased as a consequence of the realised decomposition.

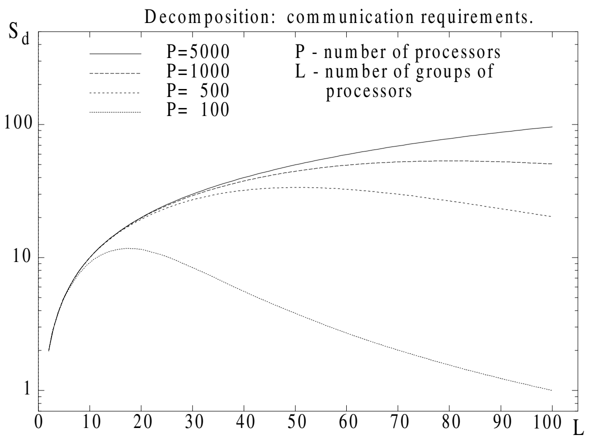


Fig. 3. Communication requirements for the two-level parallel each-to-each communication.

3. Diagonal Decomposition Method

The DDDP method, being a direct two-level optimization method with a particularly simple coordination strategy, constitutes one of numerous versions of algorithms used to improve the properties of dynamic programming. It can, however, be employed for a comparatively narrow class of optimization problems (Collins, 1970; Collins and Law, 1970; Larson, 1968; Sadecki, 1987; 1992; Sadecki and Galewicz, 1991).

Consider the general optimization problem described by the relations (5)–(7). A control process is given and it is described by the system of state equations

$$\boldsymbol{x}(k+1) = \boldsymbol{f}\big[\boldsymbol{x}(k), \boldsymbol{u}(k), k\big], \quad k = 0,1,\dots,K-1, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{5}$$

where $\boldsymbol{x}$ is the n-dimensional state vector $(\boldsymbol{x} \in \mathbb{R}^n)$, $\boldsymbol{u}$ denotes the m-dimensional control vector $(\boldsymbol{u} \in \mathbb{R}^m)$, and $\boldsymbol{f}$ stands for an n-dimensional vector function. The performance index is defined in the form of the functional

$$J\big[\boldsymbol{x}, \boldsymbol{u}\big] = \sum_{k=0}^{K-1} l\big[\boldsymbol{x}(k), \boldsymbol{u}(k), k\big] + \Psi\big[\boldsymbol{x}(K), K\big], \tag{6}$$

where l is a scalar cost function and Ψ denotes a scalar terminal cost function.

Moreover, some restrictions are imposed on the state and control variables, which can be generally formulated as the following relations:

$$\begin{aligned} &\boldsymbol{x}(k) \in \Omega_x[k], && \Omega_x \subset \mathbb{R}^n, \quad k = 0,1,\dots,K, \\ &\boldsymbol{u}(k) \in \Omega_u\big[x(k), k\big], && \Omega_u \subset \mathbb{R}^m, \quad k = 0,1,\dots,K-1. \end{aligned} \tag{7}$$

The optimization task consists in finding a control vector $\boldsymbol{u}(k)$, $k = 0,1,\dots,K-1$ such that if (5) and (7) are satisfied, it minimizes the performance index (6).

It is assumed that Eqn. (5) will be linear in both the state variables $\boldsymbol{x}$ and the control ones $\boldsymbol{u}$:

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k), \quad k = 0,1,\dots,K-1, \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0, \end{aligned} \tag{8}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times n$ matrices and, additionally, $\boldsymbol{B}$ is a diagonal matrix.

Furthermore, it is assumed that the functions $l[\boldsymbol{x}, \boldsymbol{u}, k]$ and $\Psi[\boldsymbol{x}(K), K]$, occurring in the performance index (6), are separated with respect to all the components of the vectors $\boldsymbol{x}$ and $\boldsymbol{u}$, i.e., they can be written in the form

$$l[\boldsymbol{x}, \boldsymbol{u}, k] = \sum_{i=1}^{n} \big\{h_i\big[x_i(k)\big] + q_i[u_i(k)]\big\}, \quad k = 0,1,\dots,K-1, \tag{9}$$

$$\Psi\big[\boldsymbol{x}(K), K\big] = \sum_{i=1}^{n} p_i\big[x_i(K)\big]. \tag{10}$$

Owing to the above assumptions, the only quantity which reflects the relationship between particular components of the vector $\boldsymbol{x}$ is the matrix $\boldsymbol{A}$.

The basic idea of the diagonal decomposition method consists in decomposing the matrix $\boldsymbol{A}$ into two matrices as follows:

$$\boldsymbol{A} = \boldsymbol{Q} + \boldsymbol{D}, \tag{11}$$

where the matrix $\boldsymbol{Q}$ comprises the diagonal elements of the matrix $\boldsymbol{A}$, whereas the matrix $\boldsymbol{D}$ contains all the other elements of this matrix. Assuming further that the initial value is known for the trajectory $\boldsymbol{x}^{(0)}(k),\ k = 0, 1, \dots, K$, Eqn. (8) can be written in the form

$$\boldsymbol{x}(k+1) = \boldsymbol{Q}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) + \boldsymbol{D}\boldsymbol{x}^{(0)}(k), \qquad k = 0, 1, \dots, K-1, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0. \tag{12}$$

As a result of the decomposition of the matrix $\boldsymbol{A}$, n independent one-dimensional problems are to be solved instead of an n-dimensional problem. The state equation and the performance criterion for such local tasks assume the following forms:

$$x_i(k+1) = a_{ii}x_i(k) + b_{ii}u_i(k) + \boldsymbol{d}_i\boldsymbol{x}^{(0)}(k), \tag{13}$$

and

$$\min_{u_i \in \Omega_u} J_i[x_i, u_i] = \sum_{k=1}^{K-1} \big\{ h_i\big[x_i(k)\big] + q_i\big[u_i(k)\big] \big\} + p_i\big[x_i(K)\big], \tag{14}$$

respectively, where $x_i(0) = x_{0i},\ i = 1, 2, \dots, n,\ k = 0, 1, \dots, K-1$. Here a_{ii} and b_{ii} are the diagonal elements of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, and $\boldsymbol{d}_i$ stands for the i-th row of the matrix $\boldsymbol{D}$.

Assuming the initial value of $\boldsymbol{x}^{(0)}(k),\ k = 0, 1, \dots, K$, each of these problems is solved in succession, achieving a new solution of $\boldsymbol{x}^{(1)}(k),\ k = 0, 1, \dots, K$. The obtained solution is substituted for $\boldsymbol{x}^{(0)}(k),\ k = 0, 1, \dots, K$ and the computing process is repeated until the required accuracy of the solution is achieved. The advantages resulting from the application of the dynamic programming method are rather obvious. If in the basic dynamic programming method the computational requirements increase exponentially with the increment in the dimension of the state vector, then in the diagonal decomposition method this increase will be approximately of a linear character.

4. Parallel Implementation of Computations

The diagonal decomposition method can be easily implemented in a parallel multiprocessor system. The coordination algorithm can be only reduced to carrying out some communication procedures, consisting in the exchange between processors of the successive approximation for the trajectory $x_i^{(l)}(k),\ k = 0, 1, \dots, K,\ i = 1, 2, \dots, n$, where l denotes the number of the iteration. As the smallest portion of a task which can be performed by one processor it is assumed to consider a set of all the operations that should be performed in order to solve one local task (13) and (14). Thus, the parallel algorithm can be formulated as follows:

Parallel diagonal decomposition algorithm:

(i) Each processor solves one (when $n = P$) or several (when $n > P$) local tasks, where P denotes the number of processors used.

(ii) Each processor $P_i,\ i = 1, 2, \dots, P$ sends the values of $x_i^{(l)}(k),\ k = 0, 1, \dots, K$ calculated by itself (l signifies the iteration number) to all the other processors when $\boldsymbol{A}$ is a dense matrix, or only to some of them when $\boldsymbol{A}$ is a band matrix (cf. Fig. 4).

(iii) Steps (i) and (ii) are repeated until the required accuracy of the solution is achieved.

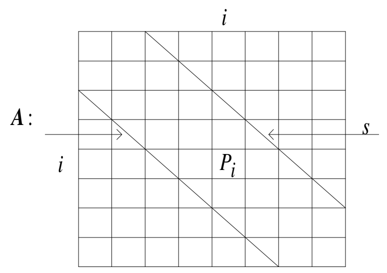


Fig. 4. Structure of the matrix $\boldsymbol{A}$.

Information referring to communication, namely, to the issue with which processors each of them should communicate (Step (ii)) can be determined on the basis of the structure of the matrix $\boldsymbol{A}$ and the number of processors P used for computations.

If the matrix $\boldsymbol{A}$ is a band one with the bandwidth equal to s, the number of processors used for computations will be $P = 8$ (cf. Fig. 4) and, in general, the i-th processor will make computations for the i-th group of local tasks, then at the stage of communication it will have to exchange data only with other four processors, namely, with those which perform computations for the local tasks denoted by the indices $i-2,\ i-1,\ i+1,\ i+2$.

The efficiency analysis of the parallel implementation of the diagonal decomposition method has been carried out on the basis of the following optimization problem:

Example 1. A control process is described by the system of state equations:

$$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k), \quad k = 0, 1, \dots, K-1, \tag{15}$$

where $\boldsymbol{A}$ is the matrix with a dominant main diagonal and consists of the elements $a_{ij},\ i,j = 1,2,\dots,n$ determined in the interval $[0,1]$. Furthermore, it is assumed that $b_{ii} = 1$ and $x_i(0) = 2,\ i = 1,2,\dots,n$. We wish to determine control $\boldsymbol{u}$ which minimizes the value of the performance criterion

$$\min_{\boldsymbol{u}\in\Omega_u} J[\boldsymbol{x},\boldsymbol{u}] = \sum_{k=0}^{K-1}\sum_{i=1}^{n}\big[x_i^2(k)+u_i^2(k)\big]. \tag{16}$$

Constraints are specified for both state and control variables in the form

$$x_i(k)\in[-2,2],\quad u_i(k)\in[-1,1],$$
$$i=1,2,\dots,n,\quad k=0,1,\dots,K. \tag{17}$$

Each of the local tasks, obtained as a result of the decomposition of the problem (16) and (17), is solved making use of the conventional dynamic programming algorithm. Hence for these tasks, both the state variables, $x_i,\ i = 1,2,\dots,n$, and the control ones, $u_i,\ i = 1,2,\dots,n$, are digitized. The numbers of discrete levels for these variables are denoted by N and M, respectively.

The analysis associated with the assessment of the efficiency of the discussed parallel algorithms is carried out based on the speedup (Brochard, 1989; Sadecki, 2001):

$$S(P) = \frac{T(1)}{T(P)}, \tag{18}$$

where P denotes the number of processor units employed in computations, $T(1)$ is the implementation time of the sequential algorithm on one processor, which is made parallel, and $T(P)$ denotes the implementation time of the parallel algorithm considered.

The results of the computations ilustrating the obtained values of the speedup are presented in Figs. 5–7. Figure 5 refers to solving the above problem for $n = 18$ with the use of $P = 2,3,\dots,9$ transputers and for $N = M = K = 10$.

Figure 6 deals with two different dimensions of the state vector, $n = 50$ and $n = 100$, for several values of P, as well as for several variants of the values for $N,\ M,\ K$. The results presented in both the figures are obtained for a P-transputer system of the linear chain structure (Sadecki, 2001). One can observe very clearly a negative effect of non-uniformity in the distribution of tasks upon the computing speedup, appearing for the values of P and n for which n is not divisible by P.

On the other hand, in the cases in which the processor load is identical, very good values of the speedup are obtained. In fact, they differ very little from the theoretical maximum ones. On the other hand, such a good uniformity of the processor load on the level of the local tasks is

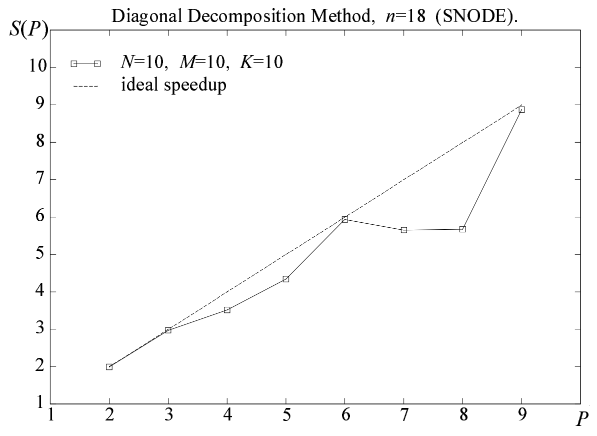


Fig. 5. Parallel diagonal decomposition algorithm: $S = S(P),\ P = 1,2,\dots,9$.

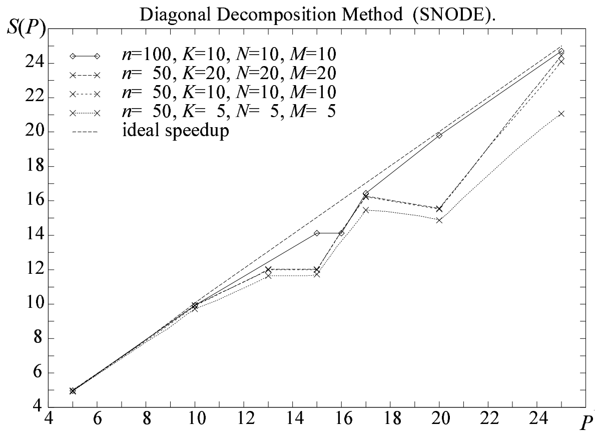


Fig. 6. Parallel diagonal decomposition algorithm: $S = S(P),\ P$ from 5 to 25.

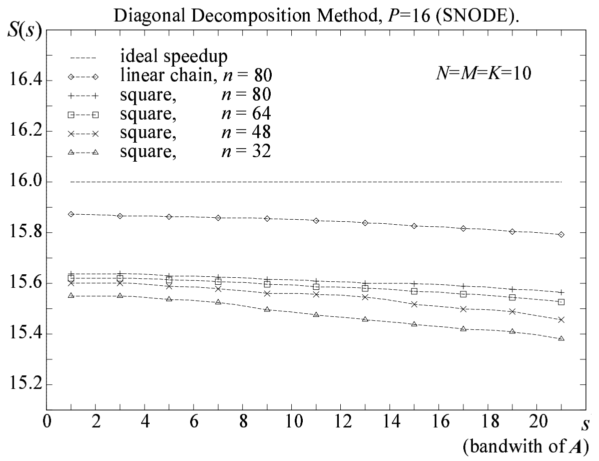


Fig. 7. Influence of the bandwidth of the matrix $\boldsymbol{A}$ on the speedup.

obtained due to the same structure of the local tasks and the same values of $N,\ M,\ K$ for all local tasks.

Figure 7 illustrates the influence of the load associated with the interprocessor communication on the speedup of parallel computations at $P = 16$ and for several values of n in computations made in systems configured as a linear chain and square structures. The load for the method considered depends in principle on the bandwidth of the matrix $\boldsymbol{A}$ denoted by s in Fig. 4. As can be seen from Fig. 7, the effect is very small. This illustrates the power of multitransputer systems, in particular due to effective interprocessor communication algorithms, as well as the high efficiency of the parallel application of the diagonal decomposition method itself.

If a large number of processors are available $(P \gg n)$, then the computing performance can be increased by adding the next level of parallelism concerning the parallel realization of the local tasks.

Transputers were most popular in Europe from the end of 1980s to the mid 1990s. An important question can be formulated as follows: Can these quite good results in terms of the speedup be comparable with the results which can be obtained for currently available highly parallel computers?

As was mentioned above, the efficiency of the parallel implementation of a broad class of algorithms depends, in general, on the processor performance and the communication bandwidth of the parallel system. This efficiency depends approximately on some coefficient α, being the ratio of the processor performance to the communication bandwidth.

A parallel system has a good performance when it consists of processor elements with a good performance and with a comparatively good communication bandwidth. For the analysed transputer system, the value of this coefficient is quite good, i.e., $\alpha = 0.025$. For the current highly parallel computers, the values of this coefficient are approximately of the same magnitude. So, it can be said that a parallel realization of the DDDP method in those systems can lead to comparable results (as regards the speedup) with those obtained for transputers. This is shown in Fig. 8. The plots present the results of a simple theoretical simulation for various systems with different processor performances and communication bandwidths. As can be seen, the results obtained for transputers are comparable with those obtained for systems with high processor performance and high speed communication (Van der Steen and Dongarra, 2001). Of course, this concerns only the speedup values. This factor is often used to compare optimization and create parallel algorithms (Sadecki, 2001).

Figure 9 presents the influence of the processor performance and communication bandwidth on the parallel

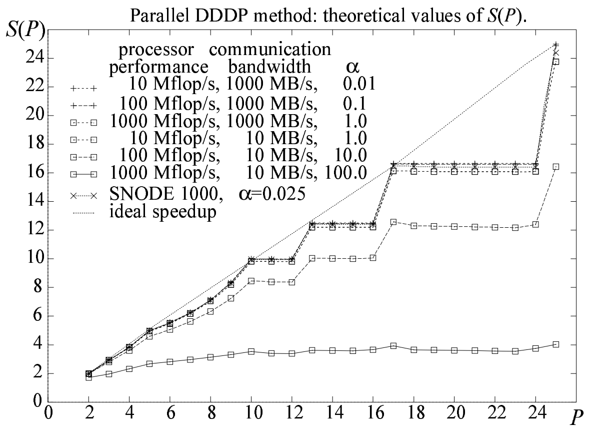


Fig. 8. Influence of the processor performance and communication bandwidth on the speedup.

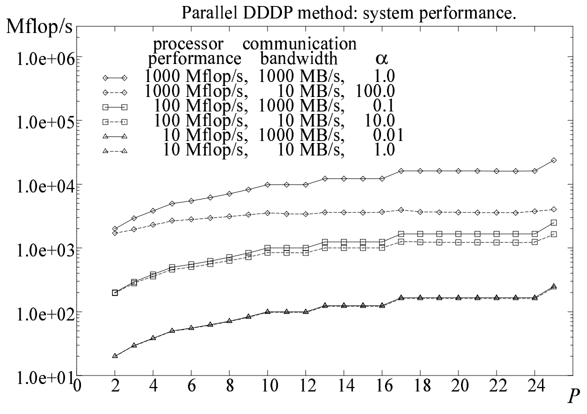
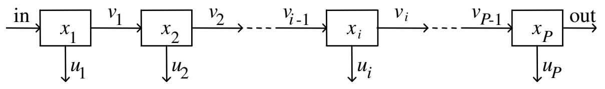


Fig. 9. Influence of the processor performance and communication bandwidth on the parallel system performance.

system performance. The performance guaranteed for parallel systems results from the performances of the processor elements used and the speeds of interprocessor communication.

The DDDP algorithm is a direct two-level optimization method with a particularly simple coordination strategy. The next example concerns a two-level control problem with a more complex coordination strategy realised by the gradient algorithm. ⧫

Example 2. The discussed two-level method is used for solving the dynamic optimization problem associated with the control of a water resource system. This example concerns the control problem for a system of n reservoirs connected in cascade, where n is the number of reservoirs ($n = P$), and P denotes the number of processors used for computations (cf. Fig. 10).



$$x_i = x_i(k), \quad v_i = v_i(k), \quad k = 0,1,\dots,K,$$
$$u_i = u_i(k), \quad k = 0,1,\dots,K-1.$$

Fig. 10. Diagram of the connection network in the system of P water reservoirs.

The problem considered is described by the system of state equations of the following form:

$$\begin{aligned}
x_1(k+1) &= x_1(k) - u_1(k) - v_1(k) + \text{in},\\
x_2(k+1) &= x_2(k) - u_2(k) - v_2(k) + v_1(k),\\
&\;\vdots\\
x_i(k+1) &= x_i(k) - u_i(k) - v_i(k) + v_{i-1}(k),\\
&\;\vdots\\
x_{P-1}(k+1) &= x_{P-1}(k) - u_{P-1}(k)\\
&\qquad - v_{P-1}(k) + v_{P-2}(k),\\
x_P(k+1) &= x_P(k) - u_P(k) + v_{P-1}(k) - \text{out},
\end{aligned} \tag{19}$$

where $k = 1,2,\dots,K-1$, $x_i(0) = x_{0i}$, $x_i(K) = x_{Ki}$, $i = 1,2,\dots,P$, $x_i \in [0, x_{i_{\max}}]$, $u_i \in [0, u_{i_{\max}}]$, $i = 1,2,\dots,P$, $v_i \in [0, v_{i_{\max}}]$, $i = 1,2,\dots,P-1$.

The control problem consists in computing controls $u_1(k), u_2(k), \dots, u_P(k)$, $k = 0,1,\dots,K-1$, which would ensure the required characteristics of water intakes from P water reservoirs during twenty four hours, determined by the parameters $c_1(k), c_2(k), \dots, c_P(k)$, $k = 0,1,\dots,K-1$.

The performance criterion is of the form

$$\min_{\substack{u_i \in [0, u_{i_{\max}}]\\ i=1,2,\dots,P}} Q(\boldsymbol{x}, \boldsymbol{u}) = \sum_{k=0}^{K} \sum_{i=1}^{P} \left[c_i(k) - u_i(k)\right]^2 + a \sum_{i=1}^{P} \left[x_i(K) - x_{Ki}\right]^2, \tag{20}$$

where a is positive constant weighting factor deciding upon the accuracy of satisfying the terminal condition.

Assuming the flows v_i, $i = 1,2,\dots,P-1$ as coordination variables, we can decompose the above problem to obtain P local tasks (corresponding to the index values equal to $1,2,\dots,P$):

$$\begin{aligned}
x_i(k+1) &= x_i(k) - u_i(k) - v_i(k) + \text{in},\\
&\qquad \text{for} \quad i = 1,\\
x_i(k+1) &= x_i(k) - u_i(k) - v_i(k) + v_{i-1}(k),\\
&\qquad \text{for} \quad 1 < i < P,\\
x_i(k+1) &= x_i(k) - u_i(k) + v_{i-1}(k) - \text{out},\\
&\qquad \text{for} \quad i = P,\\
k &= 0,1,\dots,K-1,
\end{aligned} \tag{21}$$

$$\min_{u_i \in [0, u_{\max}]} Q_i(x_i, u_i) = \sum_{k=0}^{K} \left[c_i(k) - u_i(k)\right]^2 + a\left[x_i(K) - x_{Ki}\right]^2, \tag{22}$$

and one coordination task of the general form

$$\min_{v_i,\ i=1,2,\dots,P-1} \Big[J_1(v_1) + J_2(v_1, v_2) + \cdots + J_{P-1}(v_{P-2}, v_{P-1}) + J_P(v_{P-1})\Big]. \tag{23}$$

♦

The problem formulated in Example 2 is solved parallelly making use of decomposition and coordination implemented according to the parametric optimization method as described above (Findeisen *et al.*, 1980). The local tasks are solved by the dynamic programming method, whereas the coordination algorithm is implemented based on a simple gradient method including the constraints by means of a penalty function. The numbers of discrete levels for the state variables x_i, $i = 1,2,\dots,P$ and control ones u_i, $i = 1,2,\dots,P$, are denoted by N and M, respectively. The assumption that $n = P$ results from a desire to avoid non-uniformity in the allocation of tasks. In the case of $n > P$, the non-uniformity will be negligible when n is divisible by P. Then each processor would solve not one but two, three or more local tasks. In case n is not divisible by P, a portion of processors would solve one more local task than the remainder of the processors. The effect of such non-uniformity in the distribution of tasks on the global speedup can be essential and it was shown before (Figs. 5 and 6).

Moreover, in the problem considered it is assumed that the water level in the water reservoirs should be contained within some strictly specified boundaries, and that the throughput of both channels $u_1, u_2, \dots, u_P$ and those of $v_1, v_2, \dots, v_{P-1}$ is constrained. For simplicity, the same constraints are assumed for particular components of the vectors $\boldsymbol{x}$, $\boldsymbol{u}$ and $\boldsymbol{v}$, namely,

$$x_i(k) \in [0,15], \quad u_i(k) \in [0,10], \quad x_{0i} = 5, \quad x_{Ki} = 5, \quad i = 1,2,\dots,P,$$

$$v_i(k) \in [0,7], \quad i = 1,2,\dots,P-1.$$

The function $\text{in}(t)$ assumes constant values in particular time intervals, varying from 3 to 5 during twenty four hours.

The results obtained as a consequence of the implementation of the above algorithm in a multitransputer system are presented in Figs. 11 and 12. They depict the values of the speedup associated with the implementation of this algorithm on $P = n$ transputers as compared to the implementation time of the same algorithm with the use of one transputer.

Computations were made in the system configured as a linear chain, with several variants of digitization accepted for the dynamic programming method used on a low level of the algorithm. Figure 11 refers to the case when $P = n = 2, 3, 4, \ldots, 10$ transputers are applied in computations. Figure 12 presents the results for a larger number of transputers, i.e., $P = n = 5, 10, 15, \ldots, 45, 50$. In turn, Fig. 13 presents times of parallel solutions of the problem considered for the cases as in Fig. 12. Even a rough analysis of the obtained results shows that very good values of the speedup are acquired, approaching the level of $S = 48$ at $P = 50$.

Almost a linear character of the increment in the obtained diagrams results from the fact that, in the variants considered, the number of the processors used is equal to the dimension of the analysed problem ($P = n$), which ensures a good uniformity in the distribution of tasks allocated to particular processors. Such good results in the range of the obtained speedups show that the idea of parallel computations applied to complex problems can be very effective and can constitute the second important stage of improving their efficiency, in addition to the decomposition and coordination methods used at the first stage. These results can also be a sort of encouragement to conduct further investigations in this direction.

In order to supplement the results presented above, Fig. 14 presents an example of the solution to the problem formulated in Example 2 for $n = 5$. In particular, the solutions for $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$ and $\boldsymbol{v}(t)$ are presented, as well as the values of the factor $\boldsymbol{c}(t)$ for the middle water reservoir.

5. Conclusions

Two-level algorithms presented in the paper have been implemented parallelly making use of dynamic programming methods. To a great extent, these methods have affected speedups. A dynamic programming algorithm enables us to achieve rather high uniformity in the distribution of tasks among particular processors. Furthermore, the dynamic programming method enables us to take account of many types of constraints in a simple

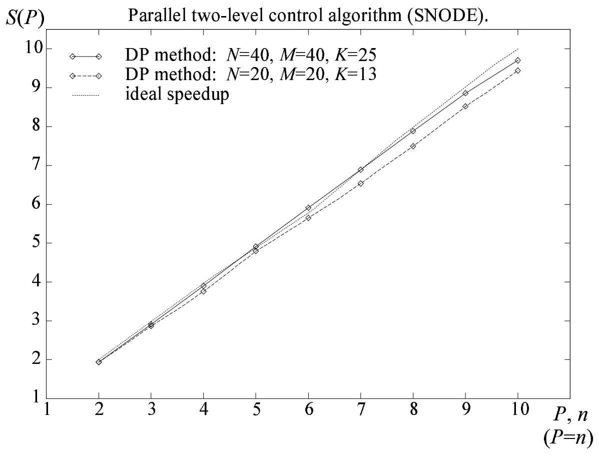


Fig. 11. Two-level control of a water resource system: $S = S(P = n)$, n ranging from 2 to 10.

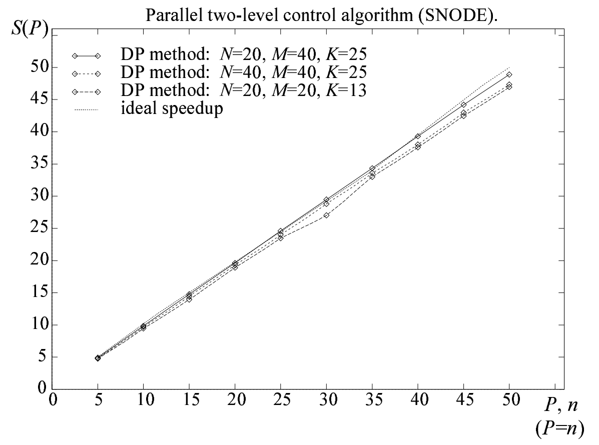


Fig. 12. Two-level control of the water resource system: $S = S(P = n)$, n ranging from 5 to 50.

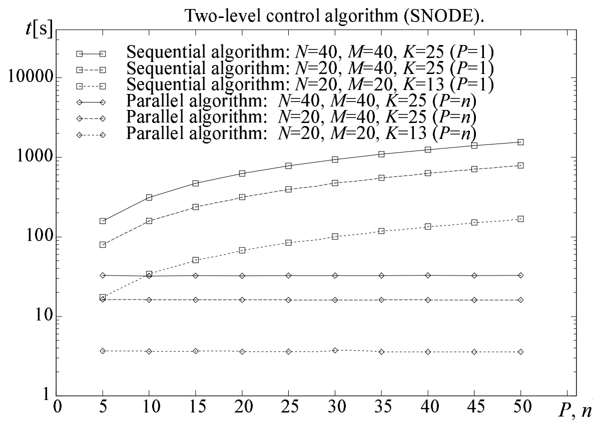


Fig. 13. Two-level control of the water resource system: computation time $t = t(P = n)$, n ranging from 5 to 50.

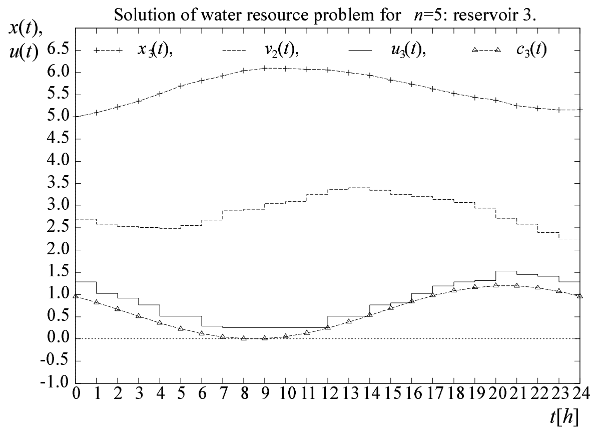


Fig. 14. Solution of the analysed two-level control of the water resource system for $n = 5$ for the middle water reservoir.

way. The applied decomposition method is a key step towards the effective parallelization of complex optimization computations. Two examples of parallel implementations of these methods in a multitransputer system are presented. The efficiency of the parallel two-level algorithms depends on the efficiency of the parallel algorithm used to solve the coordination task, the efficiency of the algorithm applied to solving local tasks, as well as the ability to obtain uniformity in the distribution of tasks on both levels. The presented algorithms enable us to achieve a high efficiency of parallel computations in terms of high speedups, on both levels of optimization computations. In many cases, the obtained values of the speedup are not considerably different from the value of $S(P) = P$, theoretically the best value to be obtained. The discussed models of parallel computations can be easily applied to more than two-level parallel computation. If a large number of processors are available, then parallel computation can be implemented on local levels, too.

Acknowledgements

Computations were partly conducted during the author's stay at the Centre for Mathematical Software Research, the University of Liverpool, which is gratefully acknowledged.

References

Averick B.M. and More J.J. (1994): *Evaluation of large-scale optimization problems on vector and parallel architectures*. — SIAM J. Optim., Vol. 4, No. 4, pp. 708–721.

Bellman R. (1957): *Dynamic Programming*. — Princeton: Princeton Univ. Press.

Baker M. (Ed.) (2000): *Cluster computing white paper*. — University of Portsmouth, UK, available at http://www.dcs.port.ac.uk/ mab/tfcc/WhitePaper.

Birge J.R. and Rosa C.H. (1995): *Parallel decomposition of large-scale stochastic nonlinear programs*. — Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, available at http://users.iems.nwu.edu/∼jrbirge//Public/html /rsann.html.

Brochard L. (1989): *Efficiency of some parallel numerical algorithms on distributed systems*. — Parall. Comput., Vol. 12, No. 1, pp. 21–44.

Collins D.C. (1970): *Reduction of dimensionality in dynamic programming via the method of diagonal decomposition*. — J. Math. Anal. Appl., Vol. 30, No. 1, pp. 223–234.

Collins D.C. and Lew A. (1970): *Dimensional approximation in dynamic programming by structural decomposition*. — J. Math. Anal. Appl., Vol. 30, No. 2, pp. 375–384.

Dongarra J.J. (2002): *Performance of various computers using standard linear equations software*. — Computer Science Department, University of Tennessee, Knoxville, available at http://www.netlib.org/benchmark/performance.ps.

Eldred M.S. and Hart W.E. (1998): *Design and implementation of multilevel parallel optimization on the Intel teraflops*. — Sandia National Laboratories, Albuquerque, available at http://endo.sandia.gov /DAKOTA/papers/MDO98_paper.pdf.

Findeisen W. (1974): *Multilevel Control Systems*. — Warsaw: Polish Scientific Publishers, (in Polish).

Findeisen W., Szymanowski J. and Wierzbicki A. (1980): *Theory and Computational Optimization Methods*. — Warsaw: Polish Scientific Publishers, (in Polish).

Interi G. (1991): *Using the SN1000*. — Liverpool: Liverpool University Press.

Karbowski A. and Niewiadomska-Szynkiewicz E. (Eds.) (2001): *Parallel and Distributed Computations*. — Technical University of Warsaw, Warsaw, (in Polish).

Larson R. (1968): *State Increment Dynamic Programming*. — New York: Elsevier.

Malinowski K. and Sadecki J. (1990): *Parallel implementation of dynamic programming methods in multiprocessor systems of different structures: Analysis of efficiency*. — Archives of Automatic Control and Remote Control, Vol. XXXV, No. 3–4, pp. 119–140, (in Polish).

Occam 2 (1988): *Occam 2, Reference Manual*. — London: Prentice Hall, INMOS Ltd.

Sadecki J. (1987): *Parallel implementation of dynamic programming methods in multiprocessor systems and investigation of their efficiency*. — Ph.D. thesis, Warsaw University of Technology, (in Polish).

Sadecki J. and Galewicz St. (1991): *Parallel computations in real two-processor system: Dynamic programming method*. — Archives of Automatic Control and Robotics, Vol. XXXVI, No. 1, pp. 193–203, (in Polish).

Sadecki J. (1992): *Possibilities of speedup of optimization computations by their implementation in parallel two-processor system: Decomposition algorithms in dynamic programming*. — Sci. Papers of the Higher School of Eng. in Opole, Series: Electrical Eng., No. 35, pp. 5–27, (in Polish).

Sadecki J. (1996): *Parallel optimization algorithms of complex systems and hierarchical control: Parallel distributed memory systems*. — Research project carried out for the State Committee for Scientific Research in Poland, No. 3 P403 02706, Final Report, Higher School of Eng. in Opole, Poland, pp. 142, (in Polish).

Sadecki J. (2001): *Parallel Optimization Algorithms and Investigation of Their Efficiency: Parallel Distributed Memory Systems*. — Series: Studies and Monographs, Technical University of Opole, Opole, Poland, (in Polish).

TAN (1989): *The Transputer Applications Notebook, System and Performance*. — Melksham, Wiltshire: Redwood Press Ltd., INMOS Ltd.

Titli A., Singh M.G. and Hassan M.F. (1978): *Hierarchical optimisation of dynamical systems using multiprocessors*. — Comp. Electric. Eng., Vol. 5, No. 1, pp. 3–14.

Van der Steen A.J and Dongarra J.J. (2001): *Overview of recent supercomputers*. — Available at http://www.top500.org/ORSC/2001/.

Wysocki M. and Kwolek B. (1994): *Parallel Computations and Transputers in Automatic Control Engineering*. — Rzeszów: Technical University Press, (in Polish).

Received: 1 June 2001
Revised: 5 October 2002