ELSEVIER

Discrete Applied Mathematics 94 (1999) 35–50

DISCRETE APPLIED MATHEMATICS

Scheduling a divisible task in a two-dimensional toroidal mesh

Jacek Błażewicz[a,*,1], Maciej Drozdowski[a], Frédéric Guinand[b], Denis Trystram[c]

[a] *Institute of Computing Science, Poznań University of Technology, ul. Piotrowo 3a, 60-965 Poznań, Poland*

[b] *Laboratoire d'Informatique du Havre, Universite du Havre, BP 540, 25, Rue Philippe Lebon, 76058 Le Havre Cedex, France*

[c] *Laboratoire de Modélisation et de Calcul, Institut Fourier, BP 53X, 100, Rue des Mathématiques, 38041 Grenoble Cedex 9, France*

Received 22 November 1996; revised 18 September 1997; accepted 8 May 1998

Abstract

In this paper, a problem of scheduling an arbitrarily divisible task is considered. Taking into account both communication delays and computation time we propose a scheduling method which minimizes total execution time. We focus on two dimensional processor networks assuming a circuit-switching routing mechanism. The scheduling method uses a scattering scheme proposed in Peters and Syska (IEEE Trans. Parallel Distributed Systems 7(3) (1996) 246–255) to distribute parts of the task to processors in a minimum time. We show how to model and solve this problem with a set of algebraic equations. A solution of the latter allows one to analyze the performance of the network depending on various actual parameters of the task and the parallel machine. Though the method is defined for a particular architecture and scattering scheme it can be generalized to analyze other architectures of parallel computer systems.

Keywords: Scheduling; Load distribution; Performance modeling; Communication delays; Divisible task; Broadcasting; Square 2D-torus

1. Introduction

Nowadays, any improvement in the performance of a single processor requires much effort and costs a lot of money. Hence, parallel processing is in the focus of researchers attention as one of the most promising ways to further increase the speed of computers.

* Corresponding author. Tel.: (4861)878 2375 or (4861)879 0790; fax: (4861)877 1525; *E-mail addresses:* blazewic@sol.put.poznan.pl (Błażewicz)

[1] Partially supported by the KBN grant and project CRIT 2.

0166-218X/99/$ - see front matter
PII: S0166-218X(99)00013-X

The advantage of parallel processing can only be taken if the computations are scheduled in an efficient way. Both theoretical and practical considerations of the scheduling problem in parallel systems require assumptions on the model of parallel processing. This includes a model of the computer system, and a model of the program.

In recent years a great variety of models for parallel computations have been proposed (see [1,5,9,16] as examples). In some papers the relation between scheduling and allocation was considered [11]. Few models, however, have taken into account constraints imposed by the architecture of a computer system (e.g. topology of the interconnection network, routing mechanism, commutation mode, etc.) and by the nature of the computational task. On the other hand, the model of a program can vary from a single task to a directed acyclic task graph, where each task can be a single or a multiprocessor task, divisible or indivisible, preemptable or not.

A model of the parallel machine is defined by the number of processors, interconnection network (its topology), the number of ports per processor and a method for message routing [10]. One important element of the computer model is assumption on allowing processors (or not) to simultaneously compute and communicate (overlap of communications by computations assumption). In this work we consider square 2D-torus of processors with circuit-switching or packet-switching method of message routing. For these methods, the communication delays are virtually independent of the covered distance. Moreover, each processor is capable of performing four communications simultaneously (4-port assumption) [14]. In such architecture we use a scattering scheme based on the broadcasting algorithm proposed in [15]. We assume that the computation is a *divisible* task, i.e. it can be divided into parts of arbitrary sizes solved independently in parallel. As an example of a divisible task consider searching for a record in a database file. The database file can be divided with granularity of a record. Hence, for big databases this granularity is quite fine and the division can be almost arbitrary. The parts of the database can be searched for a pattern independently of each other. Similar observations can be made in the case of searching for a pattern in text, audio, graphical files. Also data filtering [7] and processing big measurement data files, modeling behavior of a big number of particles, some problems of linear algebra (e.g. matrix vector product) [2] or the computer graphics can be considered as divisible tasks. We want to find a distribution of the computations among the processors such that both communications and computations finish in the shortest time.

The presented method can be generalized to a wider class of parallel computers and computation models including, for example, other interconnection networks, other numbers of ports working in parallel, processors and links with different speeds (cf. [19]), non-divisible tasks, etc. This gives rise to a methodology of dealing with a wide class of computers. It can be also used to analyze the performance of a computer system.

The rest of the paper is organized as follows. In the next section we introduce basic definitions. In Section 3 the problem of scheduling a divisible task on a square 2D-torus is considered and a procedure for minimizing schedule length is proposed.

Section 4 gives results of performance evaluation for the architecture and scattering method considered in Section 3. In Section 5 generalizations of the method are discussed.

2. Architecture and task models

Architecture of a parallel computer system is a crucial component for its efficiency and for scheduling of the computations. We start with considering architectural aspects of the parallel system to show a relation of our problem with other possible cases. We call by a *node* of the network the set of basic elements: a processor, local memory, and communication interface.

The computer system can be characterized by the following features:

- *Interconnection network.* We can distinguish several types of interconnection networks [12,17,18]: Point-to-point interconnection networks such as tori and hypercubes, multistage interconnection networks such as in IBM SP1/SP2 [20], machines based on buses, and shared memory machines.
- *The number of ports per node.* In a point-to-point interconnection network each node has several links. If for sending or receiving messages each node of the network can use only Δ of its links at a time, the communications are called Δ-*port* [12].
- *Message routing method.* Among various routing modes *store-and-forward*, *circuit-switched* and *packet-switched* can be distinguished. The packet-switched modes can be further divided into *wormhole*, *virtual-cut-through* and *buffered wormhole* [13,14,18]. We briefly describe them in the following. In store-and-forward mode the communication delay is $T_{\mathrm{com}} = d(\alpha + L\tau)$ where α corresponds to the startup (message packing, routing decision, circuit setting-up), τ represent the transmission rate (inverse of the bandwidth), L is a message length, and d is the distance between the sender and the receiver (the number of intermediate links). In the circuit-switched and packet-switched modes the delay is $T_{\mathrm{com}} = \alpha + d\delta + L\tau$ where δ represents the time needed to commute a switch (or a packet). Since $\delta \ll \alpha$ (ratio close to 0.1% or 1%), $d\delta$ can be neglected and the communication delay can be considered as independent of the distance.
- Overlap of communications by computations. Each node of the network can be equipped (or not) with devices dedicated to sending and receiving of the messages such that it can simultaneously perform computation and communication.
- When all the nodes in the network have processors with the same speed we say that processors are *identical*. In the opposite case (processors have different speeds), we say that processors are *uniform*.

The model of the machine considered in this paper is characterized by: square 2D-torus topology of the interconnection network, 4-port communications, circuit- or packet-switched communication mode, overlapping of communications by computations, identical processors.

Similarly to the machine model, there exists great variety of task features. For the purposes of this work we assume the following task characteristics:

- *Precedence constraints.* Tasks can be independent or linked by some precedence relation (e.g. represented by directed acyclic graphs). Moreover, tasks can also communicate without fixed precedences (this feature is represented by an undirected task graph).
- *Preemption.* In some computations, it is possible to interrupt the execution, and resume it later without additional cost.
- *Divisibility.* One can consider tasks which can be divided into arbitrarily small parts solved independently in parallel. It is also possible to consider some granularity for the division of the tasks. Hence, tasks can be divisible, with a limited divisibility or indivisible.

In Section 3 we assume a single, divisible task with no preemption. One can also consider as equivalent a set of identical independent divisible tasks that are initially located in the same node.

3. Scheduling a divisible task on a 2D-torus

In this section we propose a scheduling method based on the Peters and Syska broadcasting algorithm [15] and on the observation that all the processors have to finish their computation at the same moment. As the task is arbitrarily divisible, we have to find a distribution of the load among the processors according to the communication delays and the number of activated processors such that the task is completed in the minimum time.

First, we recall the Peters–Syska's broadcasting algorithm for a 2D-torus [15]. For broadcasting only, i.e. with time unit-long communications and without considering computations, it is optimal (in the number of steps) for a square 2D-torus with 5^{2k} processors and circuit-switched commutation mode. The idea of this algorithm is to send the load far away such that each processor can reach four new processors in each phase of sending data. The algorithm is composed of a series of *steps* consisting in two communication *phases*. During the first phase, the message is sent in a *knight* move (chess game): $f = 5^{k-1}$ links away from the sender in one direction and then $2 \times f$ links to the left (see Fig. 1). The second phase consists in *cross* move, sending data f links away. After the two phases one step is completed, and size f of communication distance is reduced to $f/5$ in the next step. Then, all active processors become sources for further broadcasting, and the procedure is repeated recursively in the submeshes with 25-times less nodes. Due to the symmetry of the communication pattern messages going beyond the limits of a submesh are received by the nodes of the neighbouring submeshes. For a 2D-torus with 5^{2k} processors k steps and $2k$ phases are needed to activate all processors. In Fig. 1 two steps of the broadcasting scheme are presented. For the reasons of the figure legibility the second step is presented in the upper left

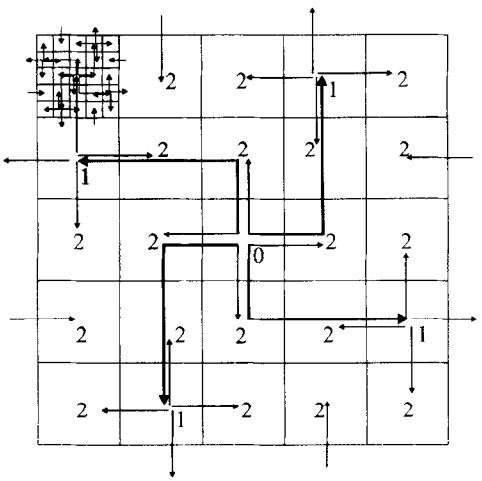

Fig. 1. Data distribution pattern in Peters–Syska broadcasting. Nodes are labeled according to the order of activation.

Table 1

Phase number	Nodes computing before sending	Number of activated nodes	Nodes computing after sending
1	1	4	$1+4$
2	5	4×5	25
3	25	4×25	125
...	...	...	...
i	5^{i-1}	$4 \times 5^{i-1}$	5^i
...	...	...	...
$2k$	5^{2k-1}	$4 \times 5^{2k-1}$	5^{2k}

square only. Table 1 gives the numbers of computing and activated processors for the consecutive phases.

We will call by a *layer* a set of processors activated in the same phase of scattering. According to the distribution method, all the load is located initially in one node called *originator*. Since the communication and computation can overlap, the originator is never idle. In the first "knight" phase of scattering the originator sends some part of the data to four other nodes. Some part of the data volume remains in the originator in order to be distributed in the subsequent phases. On a receipt of the load the newly accessed processors can start computations and sending data in the second "cross" phase. Thus, the received data is divided into a part solved locally, and parts sent in the subsequent phases of scattering. Simultaneously, the originator sends data to four processors in an analogous cross phase. Note that each processor receives data only once, but sends messages several times. Therefore, each data transfer comprises load not only for the immediately accessed processor but also for its descendants in scattering. In the following phases of scattering communications of active processors are

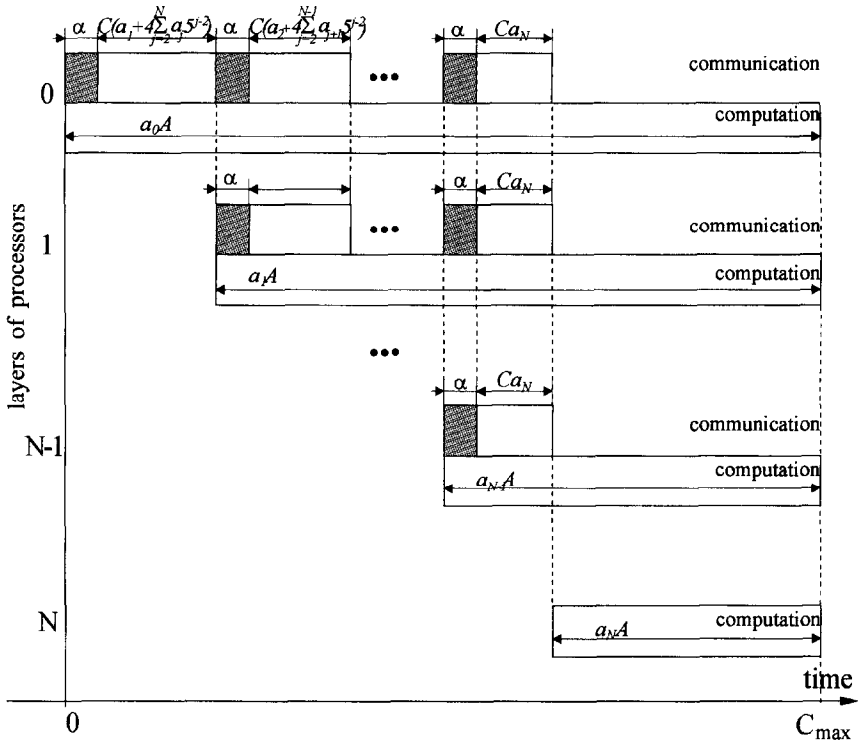

Fig. 2. Communication–computation diagram for Peters – Syska scattering with $2k$ phases.

the same which means that all active processors synchronously perform either "cross" or "knight" phase and the same volumes of data are sent. This process is repeated until the last layer of processors. The communication–computation diagram in Fig. 2 illustrates the process of distributing the data (the notations are explained below).

For simplicity of the presentation we assume that no results are returned. Then, in the optimal schedule computations on all processors of all the layers must finish simultaneously. Otherwise, it would be possible to reduce the computation time by shifting some load from processors finishing later to processors finishing earlier. We assume also that computation time on processors depends linearly on the amount of processed data. This assumption applies, for example, in the case of distributed searching for a pattern in text, audio, graphical files or computing simple statistical parameters of big data volumes etc.

The following notations will be used in the paper:

- τ – the transfer rate (e.g. in seconds per byte, seconds per floating point number etc.),
- α – the communication startup time,
- σ – the computation speed of the processors (e.g. in bytes per second, instructions per second, flops, etc.)

- $N = 2k$ – the number of scattering phases,
- V – the total volume of data to be processed (the load in other words),
- $A = V/\sigma$ – processing time on one processor,
- $C = V\tau$ – transmission time of the whole load over a link without the startup,
- a_i – the share of the load processed by a processor activated in stage i ($i = 0, \ldots, N$).
- $C_{\max}$ – the length of the schedule which is the time from the beginning of data distribution till the moment when the last processor stops.

Now, we will show how to find the distribution of the load (i.e. a_i's) such that all processors finish at the same time. Observe that computation of a_i part of the load lasts $a_i A$ units of time. Each part v transferred to a certain receiver must include data for all its descendants. Hence, a processor in layer i must receive part $v = a_i + 4a_{i+1} + 20a_{i+2} + \cdots + 4 \times 5^{N-i-1}a_N$. A transfer of this part over one link lasts $\alpha + vC$. Furthermore, computation on a transmitter must last as long as transmission to the receiver and computing on the receiver (cf. Fig. 2). Finally, all the load must be processed. The equation system from which we find a distribution of the load for a 2D-grid with 5^{2k} processors is the following :

$$
\begin{aligned}
a_{N-1}A &= \alpha + a_N(C + A),\\
a_{N-2}A &= \alpha + C(a_{N-1} + 4a_N) + Aa_{N-1},\\
a_{N-3}A &= \alpha + C(a_{N-2} + 4a_{N-1} + 20a_N) + Aa_{N-2},\\
a_{N-4}A &= \alpha + C(a_{N-3} + 4a_{N-2} + 20a_{N-1} + 100a_N) + Aa_{N-3},\\
a_{N-5}A &= \alpha + C(a_{N-4} + 4a_{N-3} + 20a_{N-2} + 100a_{N-1} + 500a_N) + Aa_{N-4},\\
&\vdots\\
a_{N-i}A &= \alpha + C\Big(a_{N-i+1} + 4\sum_{j=2}^{i} a_{N-i+j}5^{j-2}\Big) + Aa_{N-i+1},\\
&\vdots\\
a_0A &= \alpha + C\Big(a_1 + 4\sum_{j=2}^{N} a_j 5^{j-2}\Big) + Aa_1 = C_{\max},\\
1 &= a_0 + 4\sum_{i=1}^{N} 5^{i-1}a_i.
\end{aligned}
\tag{1}
$$

Now, we describe how to solve the above set of equations in linear time. In the above equations each variable a_{N-i} depends on $a_{N-i+1}, a_{N-i+2}, \ldots, a_N$ (for $i = 1, \ldots, N$). We can represent a_{N-1} as a function of a_N. Since $a_{N-i+1}, \ldots, a_{N-1}$ ($i = 2, \ldots, N$) are linear functions of a_N and a_{N-i} is a linear function of $a_{N-i+1}, \ldots, a_{N-1}$, then also a_{N-i} is a linear function of a_N. Thus, $a_i = k_i a_N + l_i$, for $i = 0, \ldots, N-1$, represents a_i as a function of a_N. The k_i, l_i coefficients can be calculated from the following recursive equations:

$$k_N = 1, \quad l_N = 0, \quad k_{N-1} = 1 + \frac{C}{A}, \quad l_{N-1} = \frac{\alpha}{A}, \tag{2}$$

$$k_{N-i} = k_{N-i+1} + \frac{C}{A}(k_{N-i+1} + 4\sum_{j=2}^{i} 5^{j-2} k_{N-i+j}), \quad i = 2, \ldots, N,$$

$$l_{N-i} = \frac{\alpha}{A} + l_{N-i+1} + \frac{C}{A}(l_{N-i+1} + 4\sum_{j=2}^{i} 5^{j-2} l_{N-i+j}), \quad i = 2, \ldots, N.$$

Finally, a_N can be found from the last equation of set (1):

$$a_0 + 4\sum_{i=1}^{N} 5^{i-1} a_i = k_0 a_N + l_0 + 4\sum_{i=1}^{N} 5^{i-1}(a_N k_i + l_i) = 1,$$

from which we obtain

$$a_N = \frac{1 - l_0 - 4\sum_{i=1}^{N} 5^{i-1} l_i}{k_0 + 4\sum_{i=1}^{N} 5^{i-1} k_i}. \tag{3}$$

The remaining a_i coefficients can be found from Eqs. (2).

We will concentrate now on the issues of feasibility and optimality of the above approach.

We see that for Eq. (1) a feasible solution sometimes may not exist. This is because in Eq. (3) a negative value of a_N is possible. We do, however, allow only positive values of a_i. The nonexistence of a feasible solution is a result of the fact that sending data to the required number of processors lasts longer than computing the whole load on a smaller number of processors, which requires less phases in the scattering. The biggest number of phases for which a feasible solution exists can be found by binary search over k in $O(\log k)$ trials, where a trial consists in solving Eq. (1). Since (1) can be solved in $O(k)$ time the optimal distribution of the load can be found in $O(k \log k)$ time.

The following observation and lemma describe the properties of the optimal schedules. Observation 1 applies in the general case, Lemma 1 applies in the considered scattering method.

Observation 1 *If the computer system 1 computes a bigger load than any other computer system computes in the same time, then system 1 computes a given load in shorter time than the time in which any other system computes the same load.*

Proof Observe, that for a computer system with maximal load computed in a given time, the function of the computed load vs. time has always greater values than for any other system. We can conclude that the inverse function (time as a function of the load) for the considered system has never a greater value than for any other system. Hence, a computer system which computes the biggest load in any given time also computes a given load in the minimal time. □

Lemma 1. *It is always advantageous for the schedule length minimization to use as many layers as possible.*

Proof Consider two identical nodes which are working in the same amount of time. The first node receives and processes locally a_1 units of data. The second node, in the same time receives data, computes locally a_2 units of the load and sends $4a_3$ units to four neighbors (neighbors in the sense of the scattering algorithm) for remote processing. Since working times for both processing elements are the same we have

$$\alpha + a_1(C + A) = \alpha + C(a_2 + 4a_3) + a_2A$$

from which we get

$$a_1 = \frac{C(a_2 + 4a_3) + Aa_2}{C + A}.$$

In order to compare the amounts of data computed by the first node and the second node together with its neighbors we calculate

$$a_1 - (a_2 + 4a_3) = \frac{C(a_2 + 4a_3) + Aa_2}{C + A} - (a_2 + 4a_3) = -\frac{4Aa_3}{C + A}.$$

The above value is negative when sending data to further layers is possible and a_3 is positive. We conclude that the second system computes more in the same time. Hence, according to Observation 1 it also computes the given load in the minimal time. This argument can be extended recursively to many layers. In this case a_3 would represent the load computed in deeper layers. □

4. Performance analysis

In this section we use the previously introduced method to analyze the influence of the architecture and task parameters on the performance.

In Figs. 3–6 we illustrate the influence of the startup time α, transfer rate τ and the size V of the computational task on the execution time in 2D-torus consisting of 25, 125, 625 and 3125 processors. Each processor can perform $\sigma = 10^6$ elementary operations per second. Along the vertical axis we have execution time, along the horizontal axis we have the size of the task. Fig. 3 represents the case of $\alpha=0$, $\tau=2\times10^{-6}$ s, Fig. 4 the case of $\alpha=0$, $\tau=10^{-12}$ s, and Fig. 5 the case of $\alpha=5\times10^{-4}$ s, $\tau=10^{-12}$ s. For big transfer rate τ the difference between execution on 25, 125, 625 or 3125 processors is very small and the curves in Fig. 3 overlap. We conclude that in this case the first stages of data distribution dominate in the execution time and the communication network becomes a bottle-neck in computation. For zero startup time all processors can take part in computation (cf. Fig. 4). In Fig. 5 we can see that the startup influences the number of processors which can take part in computation. For small sizes of the computational task and a big startup only small numbers of processors can take part in computations. The bigger the computational task is, the smaller the influence of the startup. It follows that the function of the execution time vs. the load becomes more linear.

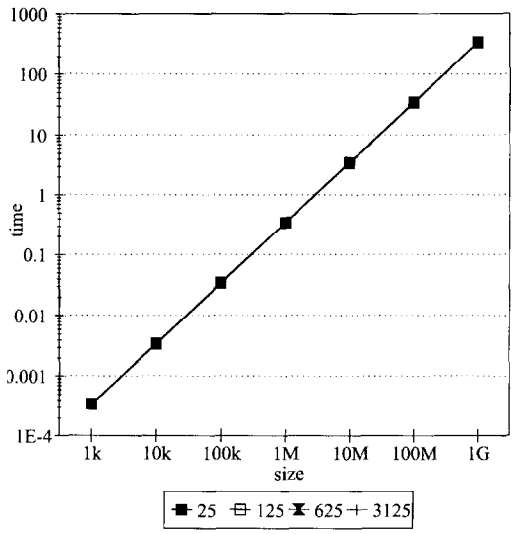

Fig. 3. Execution time vs. size of the task, $\alpha = 0, \tau = 2 \times 10^{-6}$ s.

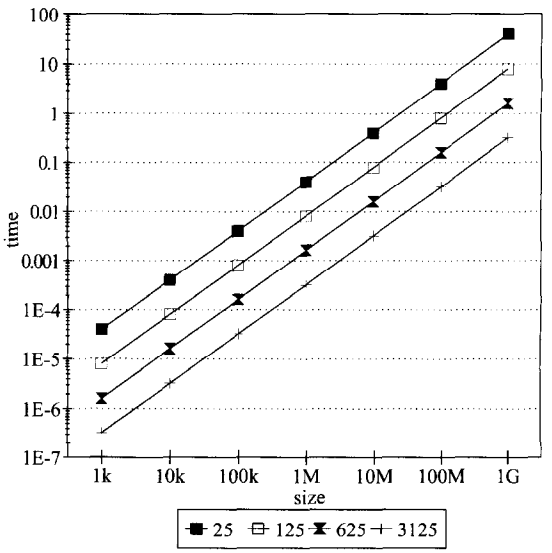

Fig. 4. Execution time vs. size of the task, $\alpha = 0, \tau = 10^{-12}$ s.

In Fig. 6 we present the same function for $\sigma = 10^6$, and $\alpha = 4.85 \times 10^{-6}$ s, $\tau = 3.3 \times 10^{-9}$ s – good values achievable at the current stage of technology (CRAY T3D). It can be observed that in the linear part of the curves, the increase in the number of used processors returns a diminishing reduction of the schedule length (compare Fig. 4). This means that the speedup levels off.

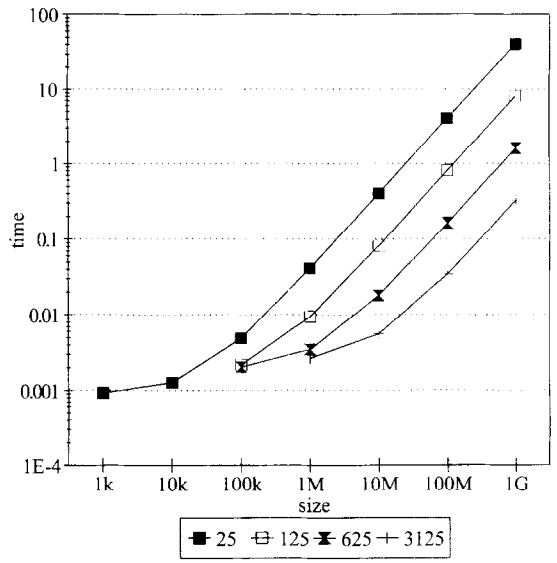

Fig. 5. Execution time vs. size of the task, $\alpha = 5 \times 10^{-4}$ s, $\tau = 10^{-12}$ s.

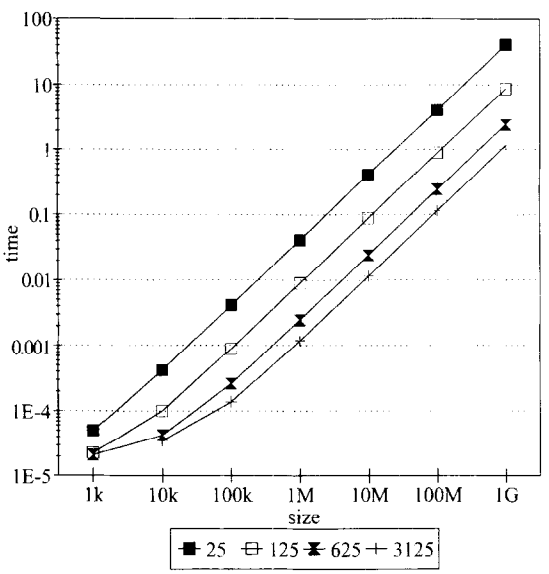

Fig. 6. Execution time vs. size of the task, $\alpha = 4.85 \times 10^{-6}$ s, $\tau = 3.3 \times 10^{-9}$ s.

Fig. 7 depicts the speedup for different sizes of the computational task ($\alpha = 10^{-4}$ s, $\tau = 10^{-7}$ s, $\sigma = 10^{6}$). The fact that the size of the task may contribute to the achievable speedup is often ignored. For small computational tasks only small numbers of processors can take part in computation and only a limited speedup can be sustained. The bigger the task, the greater speedup can be achieved. For big sizes speedup does

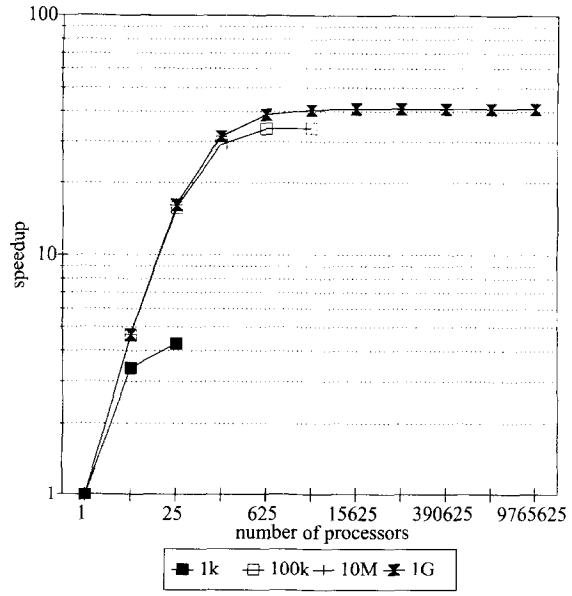

Fig. 7. Speedup for different sizes of the computational task.

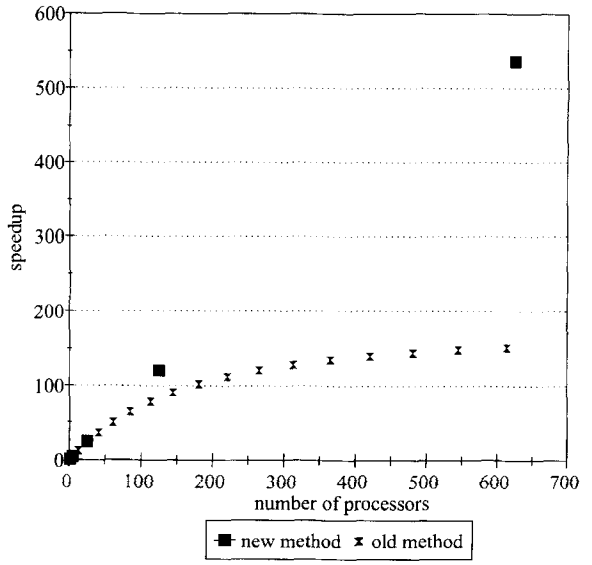

Fig. 8. Speedup with Peters–Syska scattering vs. scattering proposed in [3].

not differ very much. We conclude that for big tasks, the size does not influence the processing speed.

In Fig. 8 we compare the speedup for the scattering method used in this paper with the method proposed in [4]. In [4] the scattering consists in sending data to the nearest

idle nodes. Thus, the nodes activated in the same stage of scattering form a rectangular layer (or wave front). In the analysis of performance in [4] a simplifying assumption was adapted. It was assumed that all processors in a layer are equivalent. In fact, a node on the edge of the rectangle can be accessed from two predecessors while the node in the corner can be accessed only from one processor in the preceding layer. Furthermore, $\alpha = 0$ was assumed. Fig. 8 depicts the speedup for the network with $\tau = 10^{-8}$ s, $\alpha = 0$, $\sigma = 10^6$. As can be seen, Peters–Syska scattering performs better than the previous scattering scheme.

5. Extensions to the model

The extensions concern two elements of this work: extensions of the task model and extensions of the machine model.

The presented scattering method is valid for square 2D-torus with 5^{2k} processors. However, it is possible to generalize the tiling of the square 2D-torus to other sizes as shown in [6]. Using other scattering algorithms coming from the field of communication optimization is also possible. It seems that for any point-to-point interconnection network, if we knew the scattering algorithm and the number of the processors activated in a scattering phase our method could be applied, e.g. for kD-grids, hypercubes [3], multistage networks.

It is possible to consider uniform processors instead of identical ones. The difference in Eq. (1) is that the computation part of the equation would be weighted by the speed of the respective processor. In this case it would be necessary to write for each transmitter – receiver pair an equation stating that computing time on the transmitter is equal to the sum of the transmission time to the receiver and the computing time on the latter processor.

It is possible to assume 1-port communication instead of simultaneous 4-port communication. The communications to four nodes would be done sequentially rather than in parallel. Then, the number of equations would be multiplied by four (an equation for each of the four receivers) for each transmitter, due to the fact that the communications would be performed in an asynchronous manner. Thus, in each phase of scattering the number of equations would grow four times in each phase, and $\sum_{i=1}^{2k} 4^i$ equations would be necessary for a 2D-torus with 5^{2k} processors. A similar observation can be made for 2-port communication. However, there exists another scattering scheme, based on the notion of a minimum broadcast graph [8], which can be applied for this case.

It is not always possible to overlap the time needed to pack the message with the communication startup. Moreover, the receiver needs some time to unpack the message. These times can be included in α parameter in Eq. (1).

Though we considered circuit- or packet-switched routing, the same method can be applied for store-and-forward routing. In such a case the communication part in Eq. (1)

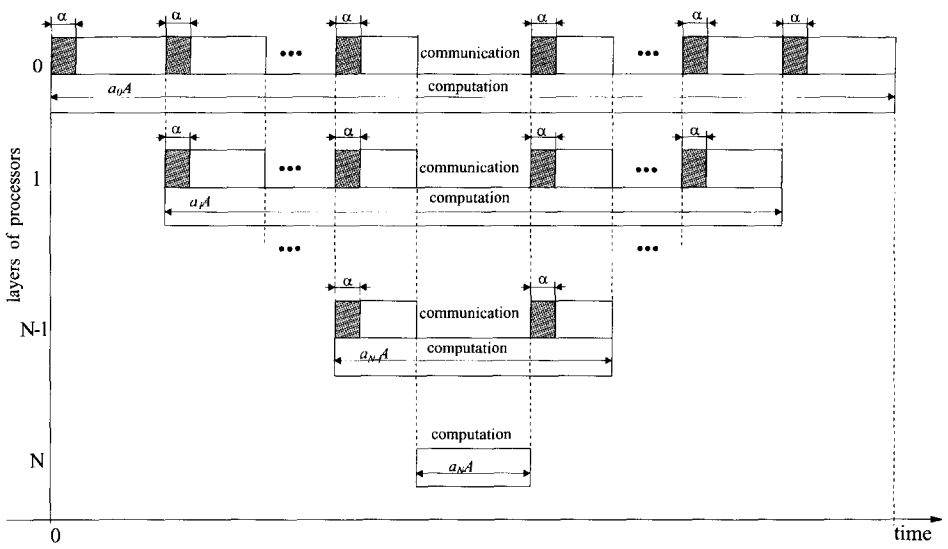

Fig. 9. Communication–computation diagram for the case of results returning.

would be multiplied by the number of links passed on the way from one layer to another.

The divisibility of a task is a strong assumption which may be unjustified in practice. If we consider tasks that are not arbitrarily divisible, the above method, without any change, calculates a lower bound on the schedule length. Usually the processed data has some granularity, e.g. a byte, a floating point number, a column in a matrix. In such a case we may still use the same method and round the calculated a_i up to the nearest unit of granularity. This may result in the deviation of the processing time from the value obtained from (1). The value of the deviation, however, can be bounded from above. Note, that when a_i is rounded up, the transferred volume increases at most by one unit, the communication time increases at most by τ and the computation time increases by $1/\sigma$. Since we have at most $2k$ successive communication phases before the computation on any processor (cf. Fig. 2) the deviation in the processing time is at most $2k\tau + 1/\sigma$.

It is conceivable to include, in the model presented above, nonlinear dependence of processing time on the amount of load. In such a case one should substitute appropriate dependence of processing time on a_{N-i} for terms Aa_{N-i} in Eq. (1).

In the above discussion we assumed that no results are returned. However, it is not difficult to relax this assumption and include the return of results. In such a case the communication–computation diagram would look like the one in Fig. 9 and in formulation (1) each equation for a receiver-transmitter pair would express the fact that computation on the node of a layer activated earlier lasts as long as sending to, computing on, and receiving results from the deeper layer node.

6. Conclusions

In this paper we considered scheduling a divisible task on a 2D-torus with 5^{2k} processors. A method of finding distribution of the computation under communication delays has been proposed. The performance of the system using the presented way of distributing the computation has been analyzed. A range of possible extensions has been proposed. The considered method seems to be a promising approach in the field of scheduling with the communication delays, a kind of frontier between the domains of scheduling and communication optimization. Future research may include for example analysis of precedence constraints among divisible tasks or pipelining of data distribution.

References

[1] S. Bataineh, T.-Y. Hsiung, T.G. Robertazzi, Closed form solutions for bus and tree networks of processors load sharing a divisible job, IEEE Trans. Comput. 43 (10) (1994) 1184–1196.
[2] J.-Y. Blanc, D. Trystram, Implementation of parallel numerical routines using broadcast communication schemes, in: H.Burkhart (Ed.), Proceedings of the CONPAR 90-VAPP IV, Joint International Conference on Vector and Parallel Processing, Lecture Notes in Computer Science, vol. 457, Springer, Berlin, 1990, pp. 467-478.
[3] J. Błażewicz, M. Drozdowski, Scheduling divisible jobs on hypercubes, Parallel Computing 21 (1995) 1945–1956.
[4] J. Błażewicz, M. Drozdowski, Performance limits of two-dimensional network of load-sharing processors, Found. Comput. Decision Sci. 21 (1) (1996) 3–15.
[5] J. Błażewicz, M. Drozdowski, J. Węglarz, Scheduling multiprocessor tasks – a survey, Microcomput. Appli. 13 (2) (1994) 89–97.
[6] Ch. Calvin, Optimisation du surcoût des communications dans la parallélisation des algorithmes numériques, Ph.D. Thesis, INPG Grenoble France, 1995.
[7] Y.C. Cheng, T.G. Robertazzi, Distributed computation with communication delays, IEEE Trans. Aerospace Electronic Systems 24 (1988) 700–712.
[8] P. Fraigniaud, J.G. Peters, Structured communications in torus networks, Proceedings of the 28th Hawaii International Conference on System Sciences, January 1995, pp. 584–593.
[9] F. Guinand, Ordonnancement avec Communications pour Architectures Multiprocesseurs dans Divers Modèles d'Exécution, Ph.D. Thesis, INPG Grenoble, France, 1995.
[10] K. Hwang, Advanced Computer Architecture – Parallelism, Scalability, Programmability, McGraw-Hill and MIT Press, New York, 1993.
[11] P. Kruger, T.-H. Lai, V.A. Dixit-Radia, Job scheduling is more important than processor allocation for hypercube computers, IEEE Trans. Parallel Distributed Systems 5 (1994) 488–497.
[12] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays – Trees – Hypercubes, Morgan Kaufman Publishers, Inc., San Mateo, CA, 1992.
[13] P. Michallon, Schémas de communications globales dans les réseaux de processeurs : application à la grille torique, Ph.D. Thesis, INPG Grenoble, France, 1994.
[14] L.M. Ni, P.K. McKinley, A survey of wormhole routing techniques in direct networks, Computer 26 (1993) 62–76.
[15] J.G. Peters, M. Syska, Circuit-switched broadcasting in torus networks, IEEE Trans. Parallel Distributed Syst 7 (3) (1996) 246–255.
[16] K. Ramamritham, J.A. Stankovic, W. Zhao, Distributed scheduling of tasks with deadlines and resource requirements, IEEE Trans. Comput. 38 (1989) 1110–1123.
[17] J. de Rumeur, Communications dans les réseaux de processeurs, Masson, Paris, 1994.
[18] Y. Saad, H. Schultz, Data communication in parallel architectures, Parallel Computing 15 (11) (1989) 131–150.

[19] J. Sohn, T.G. Robertazzi, An optimum load sharing strategy for divisible jobs with time varying processor speed and channel speed, Technical Report 706, Dept of Electrical Eng., SUNY at Stony Brook, Stony Brook, New York, January. 1995.

[20] C.B. Stunkel, D.G. Shea, D.G. Grice, P.H. Hochshild, M. Tsao, The SP1 high-performance switch, Proceedings of the Scalable High-Performance Computing Conference, May 23–25, 1994, Knoxville, Tennesse, IEEE Computer Society Press, Silver Spring, MD, pp. 150–157.