ELSEVIER Parallel Computing 25 (1999) 87–98

PARALLEL COMPUTING

Divisible task scheduling – Concept and verification [1]

Jacek Błażewicz *, Maciej Drozdowski [2], Mariusz Markiewicz [3]

Institute of Computing Science, Poznań University of Technology, ul. Piotrowo 3A, 60-965 Poznań, Poland

Received 12 November 1996

Abstract

In this work the idea of a divisible task is presented. The divisible task is a computation which can be divided with arbitrary granularity into independent parts solved in parallel by distributed computers. A simple model of a communication delay and a computation time is adopted, based on which various computer architectures and communication methods are analyzed. We review the ways of applying the divisible task concept in the case of a linear array, star, bus, hypercube, and mesh of processors. Then, the results of an empirical justification of the analysis are presented. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Deterministic scheduling; Divisible tasks; Distributed processing; Communication delays

1. Introduction

Recent developments in network technology and distributed processing have become an important element of high performance computing. The reduction of the application (task) execution time by exploiting parallelism and distribution of the computation is becoming a common practice. Programming environments like PVM, MPI, Linda, Express etc. enabled load sharing in distributed processing. Still, there is no mature method achieving an efficient use of the computing power in a wide range of circumstances. The progress in VLSI technology results in fast

* Corresponding author. Tel.: +48 61 8782-375; fax: +48 61 8771-525; e-mail: blazewic@poznlv.tup.edu.pl.

[1] This research has been partially supported by a KBN grant and project CRIT2.

[2] E-mail: maciej_d@sol.put.poznan.pl.

[3] E-mail: Mariusz.Markiewicz@cs.put.poznan.pl.

processors which must communicate with slow memories and even slower I/O devices (such as the network). Hence, the bandwidth of communication channels is a scarce resource. Numerous approaches have been proposed to achieve minimal computation time in distributed systems. However, many of them concentrate on selected elements of the computation process while neglecting others. In this work we present a simple model capable of dealing with a wide range of the computer architectures and communication aspects.

We analyse the problem of scheduling tasks which can be divided into parts of arbitrary size. Such parts can be processed separately and independently by a distributed computer system. Tasks with the above features will be called *divisible tasks*. The idea of divisible task was first analysed in the context of chains of intelligent sensors [1]. Then, it was extended to analyse various computer architectures such as stars and buses [2,3], meshes [4], hypercubes [5]. The proposed model of a computation process includes both the time of distribution to the remote sites as well as the processing time. Many important applications can be regarded as divisible tasks. For example, searching for a record in a database with thousands of records can be done in parallel by several machines. The processing elements can work individually, because not much communication is required between them. The database file can be divided into parts with record size granularity which is quite fine compared to the total volume of the database. A similar situation takes place while searching for a pattern in a text, audio, graphic etc. file. Furthermore, sorting, filtering and processing of big measurement data volumes can be modelled in this way. Some problems from linear algebra [6], modelling molecular dynamics, solving partial differential equations which involve loops processing can be included in divisible task paradigm as well.

We will outline now the model of a computation process. A distributed computer consists of m processing elements (PEs). Each of them has a processor, local memory and is able to communicate in the network. Initially, the whole volume V of data to be processed (i.e. a task) resides on the first PE called *originator* (e.g. I/O processor). The originator processes locally α_1 units of data, and sends the rest (i.e. $V - \alpha_1$) to its neighbour(s) for remote processing. PE i intercepts for local processing α_i units of data and sends the rest of the received load to its still idle neighbour(s). The intercepted part of the load is processed with rate A_i (expressed in time units per data unit e.g. seconds per byte). The transfer of x data units over link j lasts $S_j + xC_j$, where S_j is a startup time required to initiate communication, and C_j is the communication rate (e.g. in sec/byte). The problem we consider is finding a distribution of the load (i.e. $\alpha_1, \dots, \alpha_m$) such that the completion time (of the task) is minimal.

In this work we present the most important known results for the problem of divisible task scheduling. To keep the presentation short we examine in detail only simple cases to give the idea of applying the divisible task method. Then, we show how to adjust the model to include more complex situations. Later, an empirical verification of the model is presented. The work has two parts. In Section 2 we review how divisible task concept was applied in different computer architectures. This section is organised on the base of considered interconnection networks. In Section 3 we present results of verifying divisible task concept in a transputer network.

2. Application of divisible task concept

2.1. Linear processor array

In a linear processor array PE i is connected with PE $i+1$ by link i ($i = 1, \ldots, m-1$). PE 1 processes locally α_1 units of data and sends $(V - \alpha_1)$ units to PE 2 which lasts $R_1 + C_1(V - \alpha_1)$. PE 2 processes α_2 units locally and sends to PE 3 $(V - \alpha_1 - \alpha_2)$ data units, etc. Assume that each PE has a communication coprocessor. Thus, immediately after the reception of the load the considered PE starts both processing its share and sending data to its idle neighbour. We assume that the division into the locally processed part and the part sent further is instantaneous. Unless stated otherwise, we assume for the rest of the paper that no results are returned to the originator, or that the data return time can be neglected. We will demonstrate that this restriction can be easily relaxed. When no results are returned an important observation can be made. Namely, all the processors must finish their work at the same moment of time [1]. Otherwise, the processor working longer could be off-loaded and the processor finishing earlier could have received more work which would result in a shorter schedule. This meaningful observation can be applied also in the case of other networks, and other communication methods when no results are returned.

Using the above observation we conclude that communication from PE i to PE $i+1$ lasts as long as the difference between the computing time on PE i and computing on PE $i+1$ (cf. Fig. 1). Thus, a set of linear equations can be formulated from which α_i ($i = 1, \ldots, m$) can be found [1,7,8]:

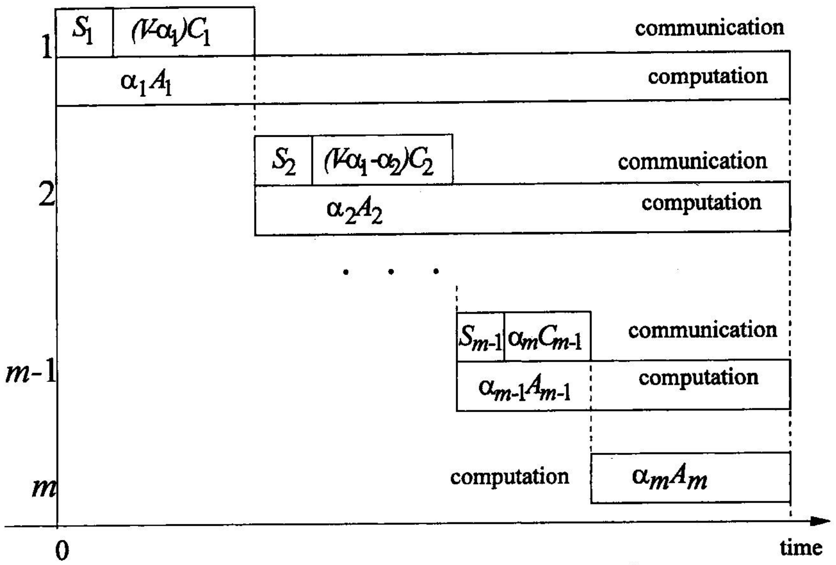

Fig. 1. Data distribution pattern in a linear array of processors.

$$\begin{aligned}
&\alpha_1 A_1 = S_1 + (\alpha_2 + \cdots + \alpha_m)C_1 + \alpha_2 A_2\\
&\ldots\\
&\alpha_i A_i = S_i + (\alpha_{i+1} + \cdots + \alpha_m)C_i + \alpha_{i+1} A_{i+1}\\
&\ldots\\
&\alpha_{m-1} A_{m-1} = S_{m-1} + \alpha_m (C_{m-1} + A_m)\\
&V = \alpha_1 + \cdots + \alpha_m.
\end{aligned} \tag{1}$$

If a feasible solution exists (note that only $\alpha_i \geqslant 0$ is reasonable), equation set Eq. (1) can be solved in O(m) time. When no feasible solution exists the originator and some closest processors are able to process the whole load before the last processor receives any data. In such a case the maximum usable set of processors can be found in O(m log m) time by testing Eq. (1). Using the obtained values of α_i $(i = 1, \ldots, m)$ parameters such as speedup $S_m = VA_1/\alpha_1 A_1 = V/\alpha_1$ and utilization $U_m = S_m/m = V/m\alpha_1$ can be found. On the base of speedup and utilization the performance of architecture and communication method can be evaluated [9–11].

Now, consider the case without communication coprocessors [1,8]. In this case simultaneous communication and computation is not possible. Hence, PE i can process its share of the load after sending the load to PE i + 1, in the interval when PE i + 1 communicates to PE i + 2 and processes α_{i+1}. Eq. (1) is now as follows:

$$\alpha_i A_i = S_{i+2} + (\alpha_{i+2} + \cdots + \alpha_m)C_i + \alpha_{i+1} A_{i+1} \quad \text{for } i = 1, \ldots, m-2, \tag{2}$$

$$\alpha_{m-1} A_{m-1} = \alpha_m A_m.$$

The speedup here is $S_m = VA_1/(\alpha_1 A_1 + (V - \alpha_1)C_1 + S_1)$.

Let us analyse the case where some results are returned and each PE has communication coprocessor [1,7]. In such a situation PE i processes its part of the load in the interval in which PE i + 1 receives the load (for PEs: $i + 1, \ldots, m$), processes α_{i+1}, and returns results (from PEs: $i + 1, \ldots, m$). Thus, Eq. (1) is modified to (for $i = 1, \ldots, m - 1$):

$$\alpha_i A_i = S_i + (\alpha_{i+1} + \cdots + \alpha_m)C_i + \alpha_{i+1} A_{i+1} + S_i + f(\alpha_{i+1} + \cdots + \alpha_m)C_i, \tag{3}$$

where $f(x)$ is the amount of results obtained for x units of data. In the simple cases $f(x) = \beta_1$ (decision problems e.g. search for a pattern), or $f(x) = \beta_2 x$, (e.g. sorting, processing measurement data etc.), where $\beta_1, \beta_2 \in \mathbb{R}^+$.

This methodology can be extended to deal with other communication methods such as circuit switched [12,13] or pipelined [13,14] load distribution.

2.2. Star and bus networks

A star interconnection can be an attractive model for master–slave computations because the parameters of a communication link can represent not only the physical layer but also all the software layers. In star network PE 1 is connected with PE i over link i. No other links exist. For the star interconnection all the load resides initially on PE 1 in the center of the star. For the bus the load initially resides in PE 1. Due to the fact that for the star only PE 1 has connection with other PEs and that for the bus there is only one bus which cannot be used simultaneously, both interconnections can be analysed in the same way and the communication pattern is the same: PE 1 sends data to PE 2 over link 2, then to PE 3 over link 3, etc. Since no results are returned the Gantt chart of communication and computation looks as in Fig. 2. Thus, α_i $(i = 1, \ldots, m)$ can be found from equations [2,3,7]:

$$\begin{aligned}
&\alpha_1 A_1 = S_2 + \alpha_2(C_2 + A_2) \\
&\ldots \\
&\alpha_i A_i = S_{i+1} + \alpha_{i+1}(C_{i+1} + A_{i+1}) \\
&\ldots \\
&\alpha_{m-1}A_{m-1} = S_m + \alpha_m(C_m + A_m) \\
&V = \alpha_1 + \cdots + \alpha_m.
\end{aligned} \tag{4}$$

Solution of equation set (4) can be found in O(m) time provided a feasible solution exists. When a feasible solution does not exist for m and the sequence of communications is fixed the maximum usable set of processors can be found in O(m log m) time. In the case of bus interconnection equations (4) can be slightly modified by assuming that $\forall_i(S_i = R, C_i = C)$ because the bus interconnection is the same for all PEs.

Yet, in general the complexity of establishing the optimal sequence of communications is unknown. For simple cases ($S_i = 0$ or identical communication links) such a sequence can be found in polynomial time. Surprisingly, when there is no startup time ($S_i = 0$) then the optimal order of activating PEs is the order of decreasing link speed and does not depend on the PEs' speeds. When there are no communication coprocessors, Eq. (4) should have as the first equation [2,3]: $A_1\alpha_1 = A_m\alpha_m$. This approach can be extended to trees [3,9,15] (no startup time), pipelined data distribution [14], variable link and/or PE speed [16], many tasks

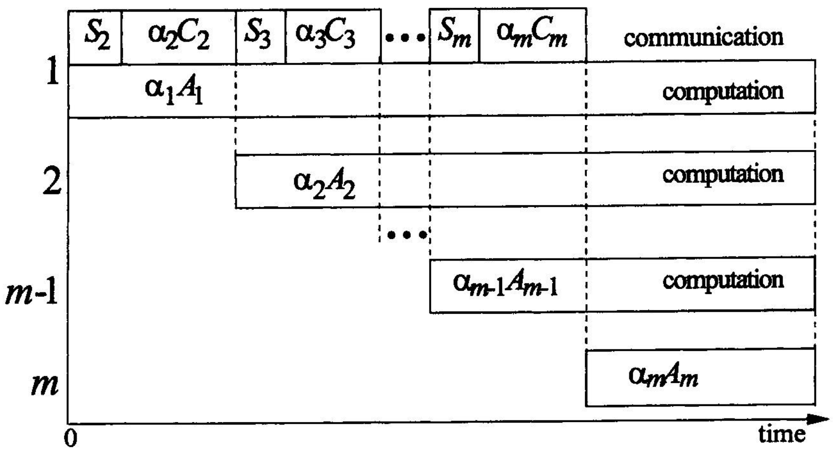

Fig. 2. Data distribution pattern in a star network.

served in FIFO order [17]. However, for many buses and non-zero startup time this problem is strongly NP-hard [7].

2.3. Regular mesh

In this section we demonstrate how to a find distribution of the load in a two-dimensional square mesh of processors (cf. Fig. 3). The solution of the problem is tightly connected with a data scattering method. For the store-and-forward communication mode it has been analysed in [4]. For the circuit-switched communication mode an efficient scattering method based on broadcasting algorithm from [18] can be applied. In the scattering method we assume that the mesh has dimensions $5^d \times 5^d$, where $d = \log_{25} m \in Z^+$. Moreover, it is assumed that the opposite end PEs are connected, i.e. the mesh is a torus. The scattering algorithm repetitively applies two moves (cf. Fig. 3) from each activated PE: a 'knight' move (like in the chess game), and a 'cross' move. The distance covered decreases in the consecutive pairs of moves. The advantage of this method is that in each move the number activated PEs grows four times, which is maximum possible. Thus, after i moves 5^i PEs are active. In the following we assume that all PEs and communication links are identical, with parameters A,C, and S, respectively. By a *layer* we mean the set of PEs activated in the same move. Each PE of layer i computes α_i units of data. The distribution of the load can be found from the following equations [19]:

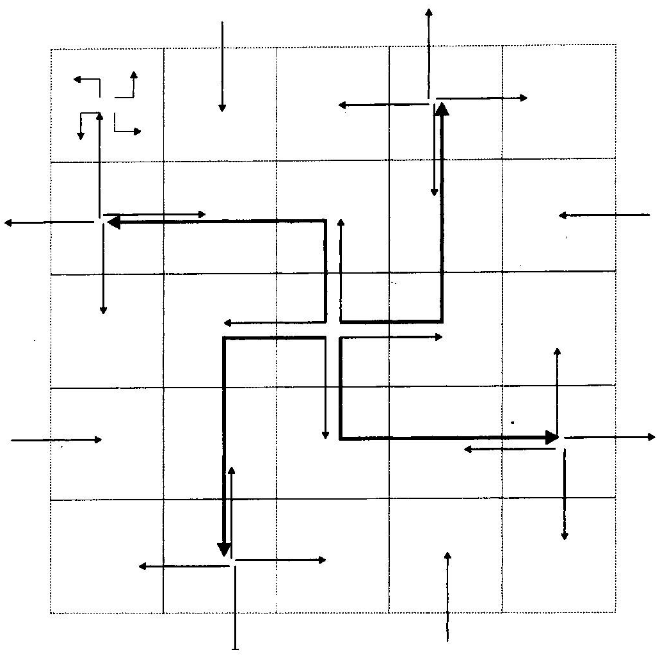

Fig. 3. Scattering in two-dimensional mesh using the method from Ref. [17].

$$\alpha_{d-i}A = S + C\left(\alpha_{d-i+1} + 4\sum_{j=2}^{i}\alpha_{d-i+j}5^{j-2}\right) + A\alpha_{d-i+1} \quad i = 1,\ldots,d, \tag{5}$$

$$V = \alpha_0 + 4\sum_{i=1}^{d}5^{i-1}\alpha_i. \tag{6}$$

As previously, using all layers may be infeasible when computing on the few first layers lasts shorter than activating the last layer. The maximum number of layers can be found by binary search in O(log m log log m) time. Again, this method gives rise to performance evaluation of the architecture with particular scattering algorithm [4,19]. This approach can be extended to three-dimensional meshes [13] and even arbitrary dimension meshes basing on broadcasting method from [20].

2.4. Hypercube

PEs of a dimension $d = \log_2 m$ hypercube can be labelled using binary numbers in such a way that the connected PEs have labels exactly one bit different. Again, we assume that PEs and communication links are identical and have communication coprocessors. Here we present a simple communication method based on store-and-forward routing. Initially originator holds all the load. It intercepts α_0 units of data for local processing. The rest is sent in equal parts to its d neighbours establishing layer 1. Each PE of layer 1 intercepts α_1 units of data and sends the rest of the obtained load to its $d-1$ still idle neighbours. These neighbours constitute layer 2. This distributing to the nearest neighbour is continued until activating the single processor in layer d. Note, that there are $\binom{d}{i}$ PEs in layer i which receive data from i neighbours and send the unused load to $d-i$ neighbours. Thus, distribution of the load can be found from the equations [5]:

$$\alpha_i A = S + \frac{\left(V - \alpha_0 - \cdots - \alpha_i\binom{d}{i}\right)C}{\binom{d}{i}(d-i)} + \alpha_{i+1}A. \tag{7}$$

$$V = \sum_{i=0}^{d}\binom{d}{i}\alpha_i. \tag{8}$$

From the above equation set the maximum number of usable layers and distribution of the load among them can be found in O(d log d) time. This approach can be extended to other data distribution patterns [13].

3. The experiments

3.1. The testbed

In the following paragraphs we present results of a practical verification of the divisible task concept in a star architecture. The basic star model (cf. e.g. [3])

assumed that the results are returned in the inverted order of sending data. Here, we also examined the case when the results are returned in the same order as the data were sent (cf. Fig. 4). For simplicity of implementation and experimentation we assumed that overlapping computation and communication on the same PE is not possible. In such a situation the time of processing on PE i and returning results from this processor must be equal to the time of sending to PE $i + 1$ and processing on PE $i + 1$. Hence, the basic equation set (4) must be modified as follows:

$$\begin{aligned} &\alpha_i A_i + S_i + f(\alpha_i)C_i = S_{i+1} + \alpha_{i+1}(C_{i+1} + A_{i+1}) \quad i = 1, \ldots, m-1 \\ &V = \alpha_1 + \alpha_2 + \cdots + \alpha_m \\ &\alpha_1, \alpha_2, \ldots, \alpha_m \geqslant 0. \end{aligned} \tag{9}$$

where $f(x)$ is the amount of results returned for x units of data.

The above method has been practically applied in a T805 transputer network depicted in Fig. 5(a). As it can be verified in Fig. 5(a) the underlying topology is not

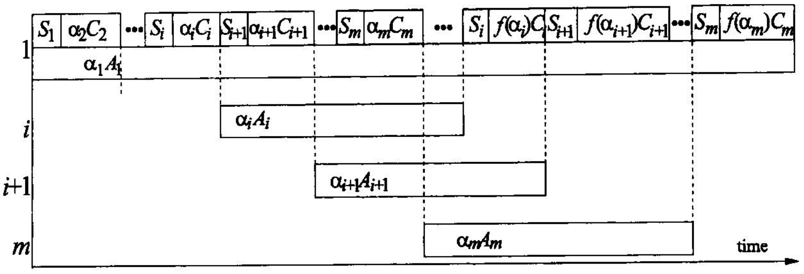

Fig. 4. Communications and computations in a star. The sequences of data distribution and collection of the results are the same.

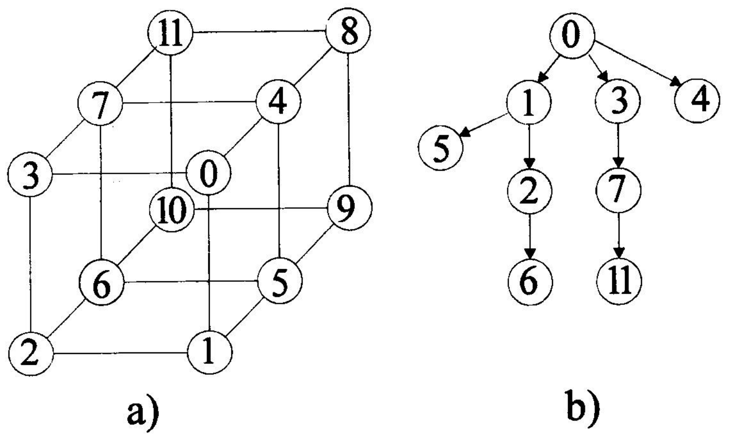

Fig. 5. The transputer testbed: (a) topology; (b) data distribution paths in an experiment with eight processors.

a star. Thus, by a star we mean here a logical interconnection observed in scattering. The communication algorithm is based on the wormhole routing. The considered application was a search for a pattern in a text file. In all experiments the returned results fit in one 1000-byte packet. Hence, $f(\alpha_i) = 1000$. Parameters A_i were measured for each PE as an average of 100 tests consisting in searching in 300 000-byte file. Parameters C_i, S_i were calculated using linear regression from a set of transmission time measurements where the originator (labelled 0) sent to PE i messages of size $1, \ldots, 100$ packets (which is in the range $1000, \ldots, 101\,584$ of data bytes with step 1016 bytes).

3.2. The results

The first experiment considered only a pair of processors: the originator plus the PE labelled 11. It consisted in transferring and processing 300 000 bytes of data. The difference between execution time measured experimentally and calculated was below 0.5%. In the next experiment we used the three PEs labelled 6, 9, 11, respectively. For three PEs the interconnection can be considered as a star. Fig. 6 presents an absolute value of relative difference between the expected and measured execution time. Every point represents an average of 100 experiments. As it can be seen in Fig. 6 the difference decreases fast and for $V \geqslant 40\,000$ it is smaller than 10% while for

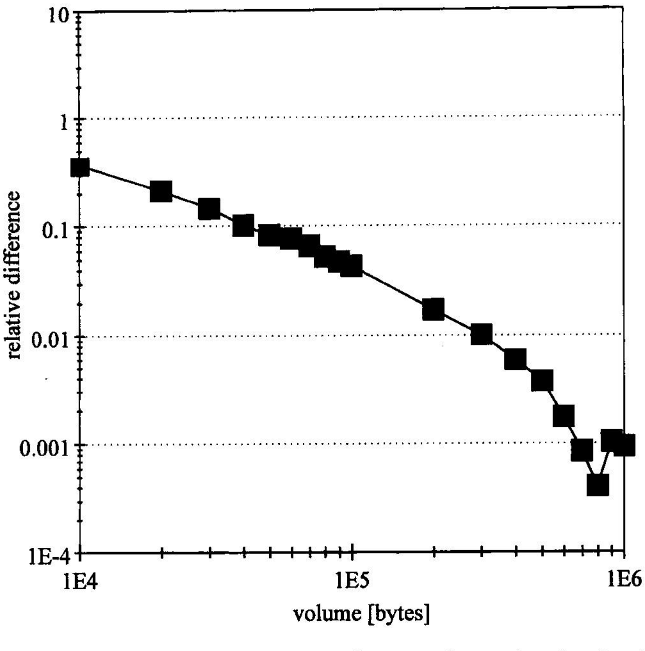

Fig. 6. A relative difference between the expected and measured execution time for three processors.

$V \geqslant 400\,000$ it is below 1%. In the following experiment we tried to use eight processors. Yet, it turned out that the construction of the routing table caused that PEs were simultaneously computing and processing. This resulted in approximately 25% difference between the measurement and the expectation. Such a big discrepancy was caused by the fact that parameters A_i no longer reflected the speed of processing because on routing PEs the routing process competed for processing power with the application. Analogously, parameters C_i, S_i were no longer valid. We changed the data distribution sequence according to the routing table such that the routing process is not activated together with the application process. The topology of data distribution paths is depicted in Fig. 5(b). We activated the PEs in the following order: 6, 2, 5, 1, 11, 7, 3, 4. As in [3] the results were returned in the inverted order of sending the data. In this way, we avoided simultaneous routing and processing by PEs. In Fig. 7 we present the difference between the expected and measured execution time. As it can be verified the difference is in the range [−1.5%–1.5%]. We conclude that the practical verification proved viability of the divisible task concept.

4. Conclusions

In this work we presented the idea of a divisible task. A crucial element related to it is the data distribution method which hides the interconnection and

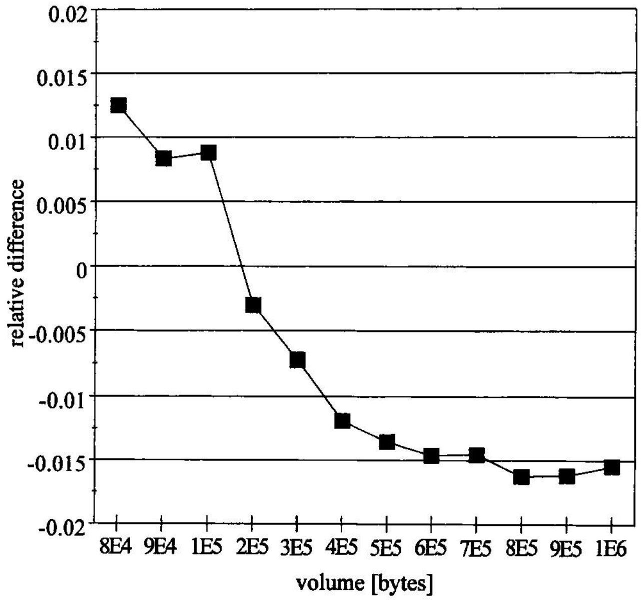

Fig. 7. A relative difference between the expected and measured execution time for eight processors.

other architectural issues. The divisible task concept provides a simple methodology for modelling computations in distributed computer systems. Hence, it can be used to evaluate performance of the computers and their communication systems. The experiments confirm applicability of this concept in real-life situations. Further research may include more sophisticated communication methods as well as other applications of the divisible task concept in distributed computations.

References

[1] Y.C. Cheng, T.G. Robertazzi, Distributed computation with communication delays, IEEE Transactions on Aerospace and Electronic Systems 24 (6) (1988) 700–712.

[2] V. Bharadwaj, D. Ghose, V. Mani, Optimal sequencing and arrangement in distributed single-level tree networks with communication delays, IEEE Transactions on Parallel and Distributed Systems 5 (9) (1994) 968–976.

[3] Y.C. Cheng, T.G. Robertazzi, Distributed computation for a tree network with communication delays, IEEE Transactions on Aerospace and Electronic Systems 26 (3) (1990) 511–516.

[4] J. Błażewicz, M. Drozdowski, Performance limits of two-dimensional network of load-sharing processors, Foundations of Computing and Decision Sciences 21 (1) (1996) 3–15.

[5] J. Błażewicz, M. Drozdowski, Scheduling divisible jobs on hypercubes, Parallel Computing 21 (1995) 1945–1956.

[6] J.-Y. Blanc, D. Trystram, Implementation of parallel numerical routines using broadcast communication schemes, in: E. Burkhart (Ed.), Lecture Notes in Computer Science 457, Proceedings of CONPAR 90-VAPP IV, Joint International Conference on Vector and Parallel Processing, Springer, Berlin, 1990, pp. 467–478.

[7] J. Błażewicz, M. Drozdowski, Scheduling divisible jobs with communication startup costs, Discrete Applied Mathematics 76 (1–3) (1997) .

[8] V. Mani, D. Ghose, Distributed computation in linear networks: Closed-form solutions, IEEE Transactions on Aerospace and Electronic Systems 30 (2) (1994) 471–483.

[9] S. Bataineh, T.G. Robertazzi, Ultimate performance limits for networks of load sharing processors, Proceedings of the 1992 Conference on Information Sciences and Systems, Princeton, NJ, 1992, pp. 794–799.

[10] D. Ghose, V. Mani, Distributed computation with communication delays: Asymptotic performance analysis, Journal of Parallel and Distributed Computing 23 (1994) 293–305.

[11] T.G. Robertazzi, Processor equivalence for a linear daisy chain of load sharing processors, IEEE Transactions on Aerospace and Electronic Systems 29 (4) (1993) 1216–1221.

[12] V. Bharadwaj, D. Ghose, V. Mani, An efficient load distribution strategy for distributed linear network of processors with communication delays, Computers and Mathematics with Applications 29 (9) (1995) 95–112.

[13] M. Drozdowski, Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems, Poznań University of Technology Press, 1997.

[14] V. Bharadwaj, D. Ghose, V. Mani, Multi-installment techniques in tree networks, IEEE Transactions on Aerospace and Electronic Systems 31 (2) (1995) 555–567.

[15] S. Bataineh, T. Hsiung, T.G. Robertazzi, Closed form solutions for bus and tree networks of processors. Load sharing a divisible job, IEEE Transactions on Computers 43 (10) (1994) 1184–1196.

[16] J. Sohn, T.G. Robertazzi, An optimum load sharing strategy for divisible jobs with time-varying processor speed and channel speed, CEAS Technical Report 706, University of Stony Brook, 1995.

[17] J. Sohn, T.G. Robertazzi, A multi-job load sharing strategy for divisible jobs on bus networks, CEAS Technical Report 697, University of Stony Brook, 1994.

[18] J.G. Peters, M. Syska, Circuit-switched broadcasting in torus Networks, IEEE Transactions on Parallel and Distributed Systems 7 (3) (1996).
[19] J. Błażewicz, M. Drozdowski, F. Guinand, D. Trystram, Scheduling under architecturalm constraints, Technical Report RA-003/95, Institute of Computing Science, Poznań University of Technology, 1995.
[20] J.L. Park, H. Choi, Circuit-switched broadcasting in torus mesh networks, IEEE Transactions on Parallel and Distributed Systems 7 (2) (1996) 184–190.