Available online at www.sciencedirect.com
SCIENCE DIRECT®

J. Parallel Distrib. Comput. 64 (2004) 960–973

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Performance limits of divisible load processing in systems with limited communication buffers

Maciej Drozdowski[a,*] and Paweł Wolniewicz[b]

[a] *Institute of Computing Science, Poznań University of Technology, Piotrowo 3A, 60-965, Poznań, Poland*
[b] *Poznań Supercomputing and Networking Center, Noskowskiego 10, 61-704, Poznań, Poland*

Received 15 April 2002; revised 5 April 2004

Abstract

In this work, we study influence of limited size of communication buffer on the efficiency of divisible loads processing. Divisible loads are computations which can be divided into parts of arbitrary sizes, and these parts can be processed in parallel. To finish processing in the shortest possible time an optimum distribution of the load must be calculated. The method of determining load distribution must take into account not only computing speed, but also interconnection system topology, communication medium speed and startup time. In this work, we include one more parameter: communication buffer size. We propose a general method of studying the influence of the communication buffer size on the interaction between the communication and computations. Three archetypal interconnection topologies are examined: stars, ordinary trees, and binomial trees. The results of modeling the performance of parallel systems show that the influence of communication buffer size is indirect and qualitative in nature. Buffer size affects the performance by causing message fragmentation, or changing load balance among the processors. We analyze performance of several communication algorithms and their interaction with the computations. The simulations show that these classic algorithms are limited.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Parallel processing; Scheduling and load balancing; Performance evaluation; Communication delays; Divisible load; Communication buffer; Information storage and retrieval

1. Introduction

Divisible load model has been introduced in [9], where a linear array of intelligent sensors was considered. The problem was to find optimum balance between advantages of distributed computations on the measurement data and the costs of communication. Later on, divisible load model has been extended in various directions: new interconnection topologies, sophisticated communication methods, memory limitations at the processor side, monetary costs of computation were studied. Despite its ability to analyze intricate details of distributed computer systems, divisible load model remained computationally tractable. Not only was it successful in theoretically analyzing distributed computer systems, but its predictions have also been confirmed in real parallel applications [7,8,11]. Thus, divisible load theory is a new versatile paradigm of distributed computing. Surveys on divisible load processing can be found in [4,5,10,16].

Divisible loads are computations that can be divided into parts and processed independently in parallel. The sizes of the parts can be adjusted to the communication and computation parameters of the system. This means that the granularity of the computations is fine, and is not influencing the load size selection. There are no dependencies among the grains of computations because it is possible to process them in parallel. This model applies, for example, to processing measurement data (e.g. SETI@home), data mining: searching databases, text, audio, and video files, also to some applications of linear algebra, number theory (e.g. Mersenne project), simulation, combinatorial optimization [7,8,11]. Divisible load theory can be applied in the analysis of

*Corresponding author. Fax: +4861-8771525.
E-mail address: maciej.drozdowski@cs.put.poznan.pl (M. Drozdowski).

distributed storage systems such as video on demand systems [3]. The distribution of the information can be optimally geared to the speeds of the communication network, and transfer rates of the storage devices.

In the divisible load theory it is generally assumed that initially volume V of the load resides in one processor P_0 called *originator*. Originator scatters the load to m processors of a distributed computer network. The communication delay inevitably appears. Transferring x units of the load (e.g. bytes) lasts $S + xC$ units of time (e.g. seconds). A processor in the network, on the receipt of the load, intercepts some part of the received load and starts processing it (computing). The rest of the load is sent to other, still inactive processors which are not directly accessible from the originator. Computing x units of the load lasts xA units of time. The problem is to adjust the sizes of the parts sent to the processors such that processing time $C_{\max}$ is the shortest possible. An intermediate conceptual layer between this general scenario and the communication hardware is the communication (scattering) algorithm. We study scattering algorithms dedicated to three message routing topologies: a star, an ordinary tree, a binomial tree.

The purpose of this paper is to examine the impact of communication buffer size D on the performance of divisible load processing. We propose a general method of studying the interaction between the communication and computing subsystems under limited communication buffer sizes. To our best knowledge, it is the first attempt of this kind in the divisible load theory. Buffer size can be considered as an equivalent of maximum transmission unit (MTU) used in TCP/IP protocol. As the buffer size is D, the transferred load must be split into units of length D (e.g. bytes). Sending each transmission unit incurs delay of one startup time which may affect performance of parallel processing. One of the goals of this study is to propose a method of adjusting the communication buffer size.

Organization of the paper is the following. In Section 2 communication system architecture and load scattering algorithms are presented. Section 3 introduces the solution method. The results of modeling and their discussion are given in Section 4, and Section 5, respectively. In the appendix, we summarize the notation.

2. System architecture

We assume that the computer system is homogeneous and synchronization of the events can be accomplished. Startup time S, communication rate C, and processing rate A are the same for the whole system.

The originator does not compute, and its purpose is to communicate. This postulate does not limit generality of our study. If the originator computes, then it can be represented as an additional processor. The processors can communicate and compute simultaneously. This is the case when processors are equipped with some communication hardware. This assumption has been relaxed in the divisible load literature [4] and does not limit generality of the method.

For the simplicity of mathematical modeling and conciseness of presentation we assume that the time of returning the results of computations to the originator can be neglected. In general, returning of the results can be incorporated in the divisible load model (cf. applications in [1,7,8,11]).

Loads greater than D cannot be sent in one message. Communication buffers are filled and messages are cyclically sent to their destinations. Hence, the load is distributed in n *stages*. Each stage is a repetition of the same communication pattern. We will denote the minimum number of stages by $n_{\min}$. We assume that there are no processor memory limitations and arbitrary load may accumulate over the course of processing. Instead of studying scattering algorithms for a multitude of interconnection topologies, we consider three fundamental structures of the scattering and broadcasting algorithms: a star, an ordinary tree, and a binomial tree. These abstract scattering structures can be embedded in buses, meshes, hypercubes, multistage interconnections [6,10,12,13,15].

2.1. Star

The originator is located in the center of the star. All the messages are routed from the originator to the processors, or from the processors to the originator. Only one message can be sent or received by the originator at a time. This kind of communications on the bus can be considered as equivalent to star interconnection. Hence, the star topology can represent a network of workstations, master–slave, or client–server systems. Star interconnection applies also to the networks in which the originator is able to directly access each slave processor. The intermediate communication nodes can be represented as an additional communication delay. Therefore, star can be called *direct* communication topology.

2.2. Ordinary tree

We consider regular balanced tree in which nodes have out-degree p. p is also the number of processor ports that can be used simultaneously to activate other processors. The set of processors in equal distance from the originator, will be called *a layer*. Processors in the same layer perform the same actions, communicate and compute synchronously. If a processor receives some load to relay, it divides it into p equal parts and retransmits them to the next layer processors. Let h

denote the height of the tree. The ordinary tree has $m = \frac{p^{h+1}-1}{p-1}$ processors, when $p > 1$, and $m = h + 1$ for $p = 1$.

In the ordinary tree topology we distinguish two additional cases depending on the ability of the nodes to deal with more than one message simultaneously. If the intermediate node can handle only one message at a time, then we will call this 1-*buffer* case. If it is possible to overlap sending one message with receiving another message, then this situation will be called 2-*buffer* case.

2.3. Binomial tree

Binomial tree has been introduced in [15] as a broadcasting structure, and as a scattering structure in [6,10]. Binomial tree (cf. Fig. 1a) is a tree in which nodes have out-degree p. Each processor (node) at level $0, \dots, i-1$ activates p new processors at level i, for $i = 1, \dots, h$. The set of processors in the same level of the binomial tree will be called a *layer*. Binomial tree takes advantage of the communication delay structure typical of circuit switching and wormhole routing. For these two switching methods communication delay does not depend significantly on the distance covered by the message. Therefore, it is advantageous to send the load to processors in physically large distance from the originator first, and then to redistribute the load locally in a smaller sub-network. Note that a processor in layer i receives load to be redistributed among its descendants in layers $i+1, \dots, h$. Examples of embedding binomial trees in the meshes are shown in Fig. 1b. The path pattern is repeated recursively in sub-meshes of decreasing size.

We assume that processors in the same layer work synchronously. Processors are able to divide the received message into equal parts and simultaneously redistribute the parts to its p ports. The number of processors in a binomial tree with layers $0, \dots, h$ is $m = (p+1)^h$. There are $p(p+1)^{i-1}$ processors in layer $1 \leqslant i \leqslant h$. Since all layers $0, \dots, l$ work synchronously to activate layer $l+1$, there is no room in the communication algorithm for simultaneous distribution of the load for some other layer. Consequently, we do not consider 1- and 2-buffer cases here because overlapping in time distribution of the loads to different layers is not possible.

2.4. Layer activation order

In trees it is possible to activate the layers in the order of the growing distance from the originator. This method will be called the *nearest layer first* (*NLF*). It is also possible to activate the processors in the inverted order. We will call the second method the *largest layer first* (*LLF*). It has been shown in [12] that LLF is the optimal activation sequence for binomial trees when there are no memory limitations in the computer system.

Note, that using LLF instead of NLF order, 2-buffers instead of 1-buffer, binomial trees instead of ordinary trees are examples of optimizations that can be implemented in the communication algorithms. It will be demonstrated that their impact is limited when it comes to the interactions with the computations, and limited communication buffer.

3. Mathematical models

In this section, we formulate the problems of finding optimum distribution of the load as linear programs. Linear programming (LP) is a widely used tool for

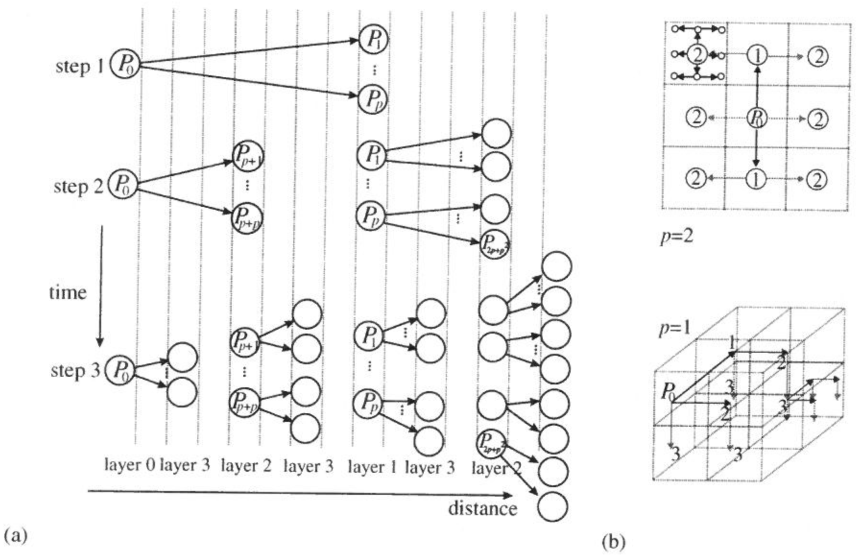


Fig. 1. Binomial tree structure. (a) The idea ($p = 2$), (b) examples of embedding.

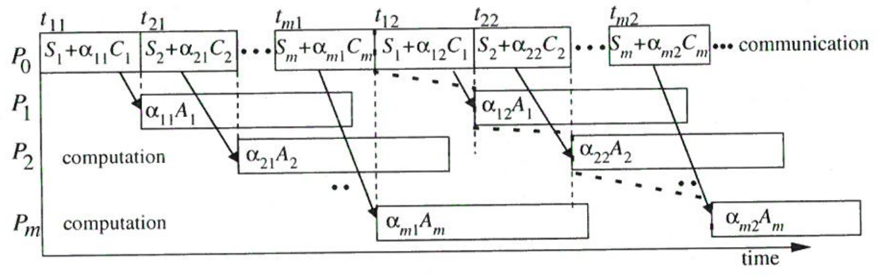


Fig. 2. Communications and computations in a star interconnection.

modeling problems in science and engineering (see e.g. [14]).

3.1. Star

Minimum number of stages is $n_{\min} = \lceil \frac{V}{Dm} \rceil$. A Gantt chart depicting communications and computations in a star network is presented in Fig. 2. The following extension of the standard notation is used in this section:

$\alpha_{ik} \geqslant 0$ is the amount of the load sent to processor P_i in stage k,

$t_{ik} \geqslant 0$ is the start time of the communication to processor P_i in stage k.

The problem of determining optimum distribution of the load in the star, for a given number of stages $n \geqslant n_{\min}$, can be formulated as a linear program

LP LoadDirect
minimize $C_{\max}$
subject to

$$t_{(i+1)k} \geqslant t_{ik} + C\alpha_{ik} + S, \quad i = 1, \dots, m-1, \quad k = 1, \dots, n, \tag{1}$$

$$t_{1(k+1)} \geqslant t_{mk} + C\alpha_{mk} + S, \quad k = 1, \dots, n-1, \tag{2}$$

$$t_{ik} + S + C\alpha_{ik} + A \sum_{l=k}^{n} \alpha_{il} \leqslant C_{\max}, \quad i = 1, \dots, m, \quad k = 1, \dots, n, \tag{3}$$

$$\alpha_{ik} \leqslant D, \quad i = 1, \dots, m, \; k = 1, \dots, n, \tag{4}$$

$$\sum_{i=1}^{h} \sum_{k=1}^{n} \alpha_{ik} = V. \tag{5}$$

In linear program **LoadDirect** inequalities (1) ensure that communications of the same stage do not overlap. By (2) the succeeding stages do not overlap. Inequalities (3) guarantee that computations finish before the end of the schedule. By (4) the communication buffers do not overflow. Eq. (5) guarantees that all the load is processed.

3.2. Ordinary tree

Since the layers work synchronously, we do not have to analyze processors separately. We will denote by

$\alpha_{ik} \geqslant 0$ is the amount of the load sent to each processor of layer i in stage k,

$t_{ikl} \geqslant 0$ is the start time of sending the load to processors in destination layer i from the intermediate node(s) in layer l in stage k, $i = 1, \dots, h$, $k = 1, \dots, n$, $l = 0, \dots, i-1$.

Let us analyze the number of stages. The load for the deeper layers is transferred from the originator to a processor in layer 1, via a communication buffer of size D. Successors of this processor receive at most D units of load altogether in a single installment. Hence, the number of stages is $n \geqslant n_{\min} = \lceil \frac{V}{Dph} \rceil$. Note that a single link from layer l feeds p^{i-l-1} processors in layer i.

3.2.1. NLF

1-buffer: The communications and computation for this case are presented in Fig. 3a. Linear program for the problem is

LP TreeNLF-1
minimize $C_{\max}$
subject to

$$t_{ik(l+1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \dots, h, \quad k = 1, \dots, n, \quad l = 0, \dots, i-2, \tag{6}$$

$$t_{(i+1)k(l-1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \dots, h-1, \quad k = 1, \dots, n, \; l = 1, \dots, i-1, \tag{7}$$

$$t_{i(k+1)(i-1)} \geqslant t_{hki} + Cp^{h-i-1}\alpha_{hk} + S, \quad i = 1, \dots, h-1, \quad k = 1, \dots, n-1, \tag{8}$$

$$t_{2k0} \geqslant t_{1k0} + C\alpha_{1k} + S, \quad k = 1, \dots, n, \tag{9}$$

$$t_{ik(i-1)} + C\alpha_{ik} + S + A \sum_{l=k}^{n} \alpha_{il} \leqslant C_{\max}, \quad i = 1, \dots, h, \quad k = 1, \dots, n, \tag{10}$$

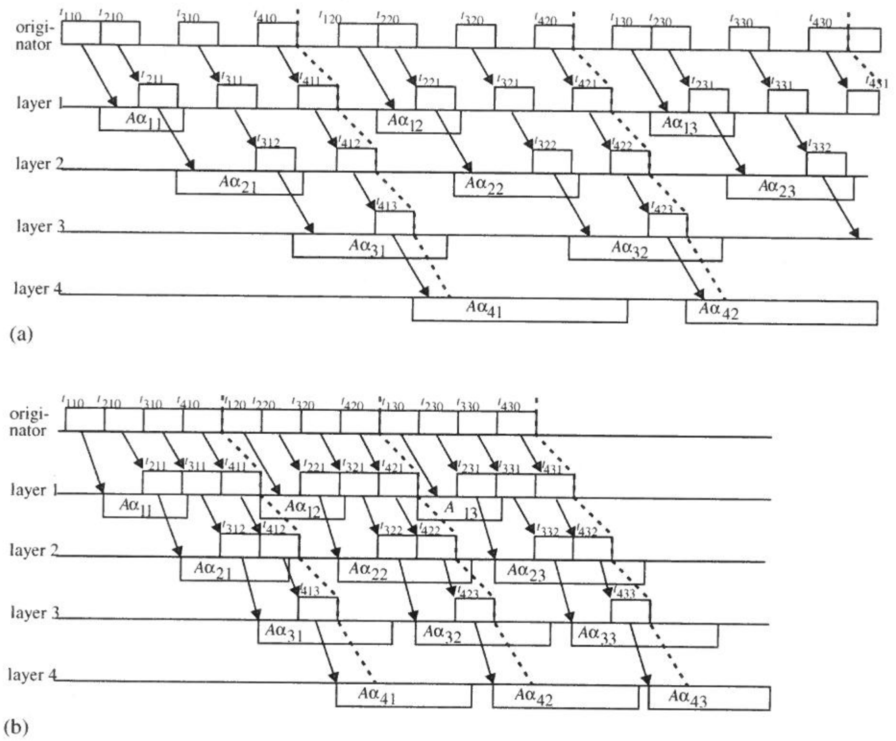


Fig. 3. Communication and computation in an ordinary tree, NLF. (a) 1-buffer. (b) 2-buffer.

$$p^{i-1}\alpha_{ik} \leqslant D, \quad i = 1, \ldots, h, \ k = 1, \ldots, n, \tag{11}$$

$$\sum_{i=1}^{h} \sum_{k=1}^{n} p^i \alpha_{ik} = V. \tag{12}$$

In the linear program **TreeNLF-1** inequalities (6) guarantee that relaying a message can start only if the message is fully received first. By (7) the communication to the next layer may not start unless the relaying of the previous message is finished and the next buffer is ready to be reused. Inequalities (8) ensure that the communications of the consecutive stages do not overlap. Constraints (9) follow from the fact that the message to the second layer can be sent immediately after the message to layer 1, because the message to layer 1 is not further relayed. Inequalities (10) ensure that all the load received by the processors is processed before the end of the schedule at $C_{\max}$. By (11) messages fit into the communication buffers, and by (12) all the load is processed.

2-*buffers*: When the number of buffers is sufficient, two consecutive communications can be performed simultaneously. However, the buffers are reused in the third following communication. If the load received in the first message has not been relayed, then the third communication using the same buffer must wait. This restriction prevents accumulation of the load in the intermediate layers. The Gantt chart for this case is presented in Fig. 3b. In the following linear program constraints (10)–(12) are also used, but we do not reproduce them to avoid repetition.

LP **TreeNLF-2**
minimize $C_{\max}$
subject to

$$t_{ik(l+1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \ldots, h,$$
$$k = 1, \ldots, n, \ l = 0, \ldots, i-2, \tag{13}$$

$$t_{(i+1)kl} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \ldots, h-1,$$
$$k = 1, \ldots, n, \ l = 1, \ldots, i-1, \tag{14}$$

$$t_{(i+2)k(l-1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = l+1, \ldots, h-2,$$
$$k = 1, \ldots, n, \ l = 1, \ldots, h-2, \tag{15}$$

$$t_{1(k+1)0} \geqslant t_{(h-1)k1} + Cp^{h-2}\alpha_{(h-1)k} + S,$$
$$k = 1, \ldots, n-1, \tag{16}$$

$$t_{2(k+1)0} \geqslant t_{hk1} + Cp^{h-1}\alpha_{hk} + S, \quad k = 1, \ldots, n-1, \tag{17}$$

$$t_{2(k+1)1} \geqslant t_{hk2} + Cp^{h-2}\alpha_{hk} + S, \quad k = 1, \ldots, n-1, \tag{18}$$

$$t_{i(k+1)(i-1)} \geqslant t_{hk(i-1)} + Cp^{h-i}\alpha_{hk} + S, \quad i = 1, \ldots, h,$$
$$k = 1, \ldots, n-1. \tag{19}$$

Inequalities (13) guarantee that retransmission of the load to the deeper layers may start only after fully receiving the message. By (14) the communications on

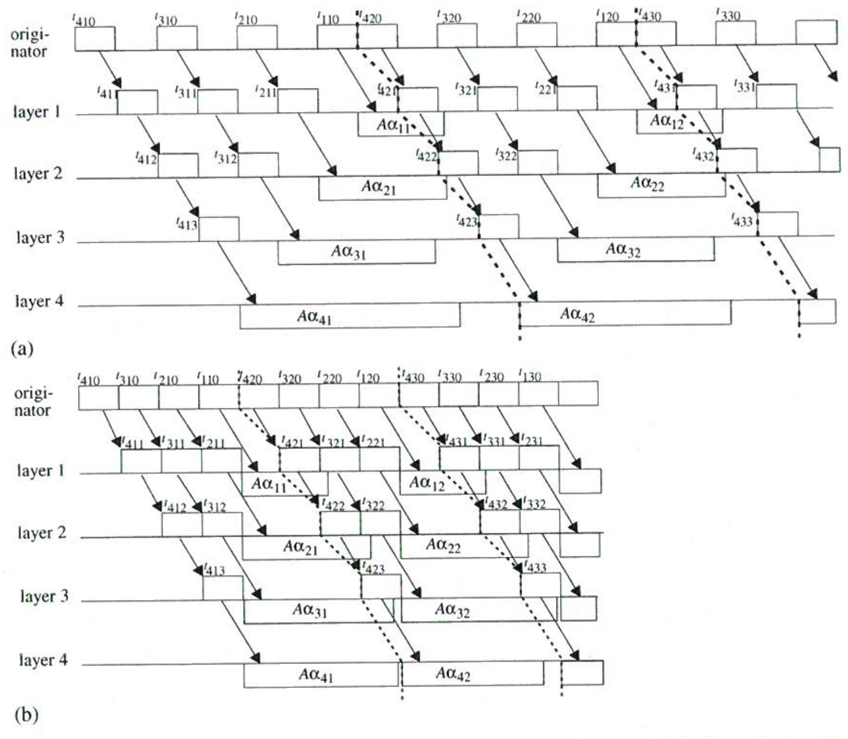


Fig. 4. Communication and computation in an ordinary tree, LLF. (a) 1 buffer, (b) 2-buffer.

the same link do not overlap. Inequalities (15)–(18) ensure that sending a new portion of the load does not start before the buffer at the receiver is available. By (19) the succeeding stages do not collide.

3.2.2. LLF

1-*buffer*: The communication and computation Gantt chart for this case is presented in Fig. 4a. In the following linear program, we skip repetition of constraints (10)–(12).

LP **TreeLLF-1**
minimize $C_{\max}$
subject to

$$t_{ik(l+1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \dots, h, \quad k = 1, \dots, n, \; l = 0, \dots, i-2, \tag{20}$$

$$t_{(i-1)k(l-1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 2, \dots, h, \quad k = 1, \dots, n, \; l = 1, \dots, i-1, \tag{21}$$

$$t_{h(k+1)i} \geqslant t_{(i+1)ki} + C\alpha_{(i+1)k} + S, \quad i = 0, \dots, h-1, \quad k = 1, \dots, n-1. \tag{22}$$

In the above formulation inequalities (20) guarantee that before the relaying the message is fully received first. By inequalities (21) a buffer is not used by two messages simultaneously. Messages from the consecutive stages do not overlap by inequalities (22).

2-*buffers*: The communication and computation Gantt chart for LLF communication strategy and two buffers in an ordinary tree is presented in Fig. 4b. Linear program for the problem is as follows (constraints (10)–(12) are not repeated):

LP **TreeLLF-2**
minimize $C_{\max}$
subject to

$$t_{ik(l+1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 1, \dots, h, \quad k = 1, \dots, n, \; l = 0, \dots, i-2, \tag{23}$$

$$t_{(i-1)kl} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{ik} + S, \quad i = 2, \dots, h, \quad k = 1, \dots, n, \; l = 0, \dots, i-1, \tag{24}$$

$$t_{(i-2)k(l-1)} \geqslant t_{ikl} + Cp^{i-l-1}\alpha_{hk} + S, \quad i = l+2, \dots, h, \quad k = 1, \dots, n, \; l = 1, \dots, h-2, \tag{25}$$

$$t_{h(k+1)0} \geqslant t_{2k1} + Cp\alpha_{2k} + S, \quad k = 1, \dots, n-1, \tag{26}$$

$$t_{h(k+1)i} \geqslant t_{(i+1)ki} + C\alpha_{(i+1)k} + S, \quad i = 1, \dots, h-1, \quad k = 1, \dots, n-1. \tag{27}$$

In the above linear program **TreeLLF-2** inequalities (23) guarantee that the load is first completely received, only than can it be further relayed. By inequalities (24) messages sent by layer l to layers i, and $i-1$, do not overlap. By (25), and (26) no more than two buffers are used in each communication switch. Inequalities (27) ensure that messages from the consecutive stages do not overlap.

3.3. *Binomial tree*

Before going into the further details let us analyze the duration of the communication from the originator to layer i in some stage k. In binomial trees nodes receive load once and then redistribute it to the deeper layers of a tree. Therefore, each communication must comprise load not only for the node, but also for the successors in a binomial tree. There are $p(p+1)^{i-1}$ processors in layer $i \geqslant 1$. The originator sends load to layer i in i steps. First, the originator sends over each of its p communication links $p(p+1)^{i-2}\alpha_{ik}$ load units to layer 1. The remaining load $p(p+1)^{i-2}\alpha_{ik}$ will be sent to layer i via direct successors of the originator in layers $2, \dots, i$ (cf. Fig. 1a). Analogously, each processor in layer $j \leqslant i-2$ sends load to layer i in $i-j$ steps. First, $p(p+1)^{i-j-2}\alpha_{ik}$ units of load are sent to layer $j+1$ over each of p ports. The remaining $p(p+1)^{i-j-2}\alpha_{ik}$ load units are sent from layer j to layer i via j's direct binomial tree successors in layers $j+2, \dots, i$. In the last ith communication step, $(p+1)^{i-1}$ processors in layers $0, \dots, i-1$ send α_{ik} load units over $p(p+1)^{i-1}$ ports to each processor in layer i. Note that all layers communicate synchronously, and the same amounts of load are sent from active layers to the next activated layer. Total communication time is equal to $Si + C\alpha_{ik}(1 + p\sum_{j=0}^{i-2}(p+1)^{i-j-2}) = Si + C\alpha_{ik}(p+1)^{i-1}$. We will use this closed-form summation result in the following formulations.

Let us consider the number of stages. The originator can send at most D units of load to each of its p neighbors in layer 1. As it was said the originator in the first step sends $p(p+1)^{i-2}\alpha_{ik} \leq D$ load units to its descendants in layer 1. Hence $\alpha_{ik} \leqslant \frac{D}{p(p+1)^{i-2}}$. The total number of processors in layer i is $p(p+1)^{i-1}$. Therefore, total load transferred to layer $i > 1$ in a single stage is at most $\frac{Dp(p+1)^{i-1}}{p(p+1)^{i-2}} = (p+1)D$. Note that this quantity does not depend on i. For layer 1 the load is at most pD. As we have computing layers $1, \dots, h$, total load that can be distributed in one stage is $(h(p+1)-1)D$. Thus, the number of necessary iterations is $n \geqslant \frac{V}{D(h(p+1)-1)}$.

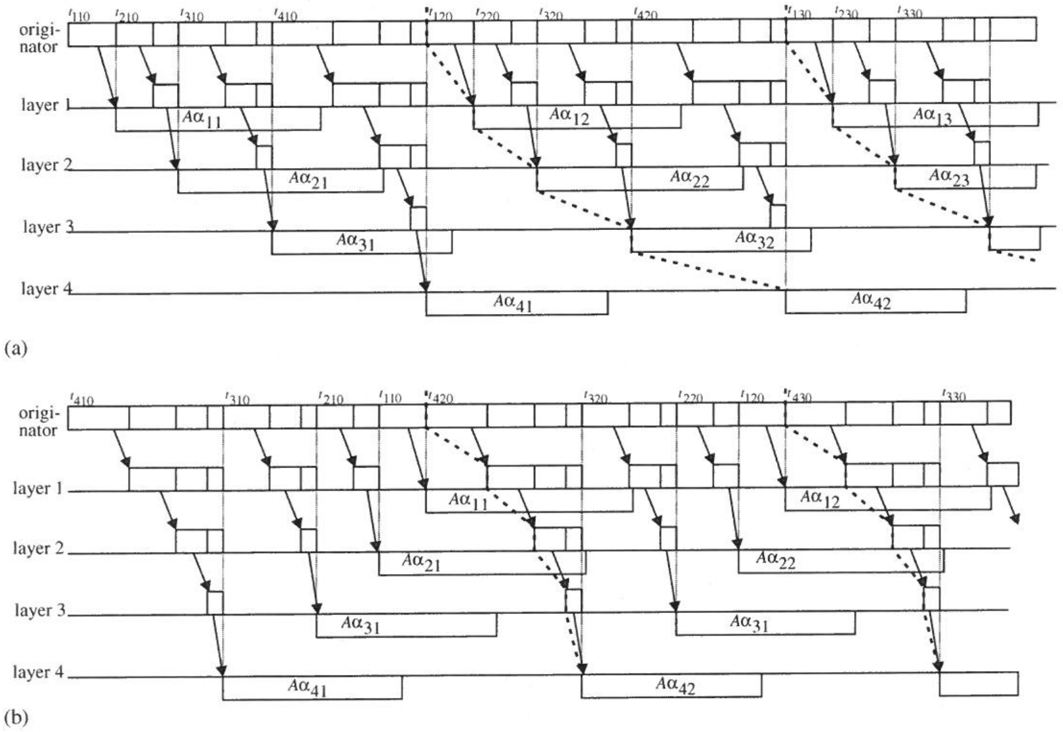
As it was noted the layers work synchronously. For this reason it is not needed to introduce to the linear programs constraints expressing the fact that the message must be fully received first and only then can it be relayed. Hence, only variable t_{ik0} is needed in the linear programs. The remaining variables t_{ikl} for $l = 1, \dots, i-1$ will not be used.

3.3.1. *NLF*

The communication and computation diagram for NLF activation strategy is presented in Fig. 5a. Linear program formulation is as follows:

LP BinomialTreeNLF
minimize $C_{\max}$
subject to

$$t_{(i+1)k0} \geqslant t_{ik0} + C(p+1)^{i-1}\alpha_{ik} + Si, \quad i = 1, \dots, h-1, \quad k = 1, \dots, n, \tag{28}$$



origi-nator
layer 1
layer 2
layer 3
layer 4
t_{110} t_{210} t_{310} t_{410} t_{120} t_{220} t_{320} t_{420} t_{130} t_{230} t_{330}
$A\alpha_{11}$ $A\alpha_{12}$ $A\alpha_{13}$
$A\alpha_{21}$ $A\alpha_{22}$ $A\alpha_{23}$
$A\alpha_{31}$ $A\alpha_{32}$
$A\alpha_{41}$ $A\alpha_{42}$
(a)
t_{410} t_{310} t_{210} t_{110} t_{420} t_{320} t_{220} t_{120} t_{430} t_{330}
$A\alpha_{11}$ $A\alpha_{12}$
$A\alpha_{21}$ $A\alpha_{22}$
$A\alpha_{31}$ $A\alpha_{31}$
$A\alpha_{41}$ $A\alpha_{42}$
(b)

Fig. 5. Communication and computation in a binomial tree. (a) NLF strategy, (b) LLF strategy.

$$t_{1(k+1)0} \geqslant t_{hk0} + C(p+1)^{h-1}\alpha_{hk} + Sh, \quad k = 1,\dots,n-1, \tag{29}$$

$$t_{ik0} + C(p+1)^{i-1}\alpha_{ik} + Si + A\sum_{l=k}^{n}\alpha_{il} \leqslant C_{\max}, \quad i = 1,\dots,h,\ k = 1,\dots,n, \tag{30}$$

$$\alpha_{1k} \leqslant D, \quad k = 1,\dots,n, \tag{31}$$

$$p(p+1)^{i-2}\alpha_{ik} \leqslant D, \quad i = 2,\dots,h,\ k = 1,\dots,n, \tag{32}$$

$$\sum_{i=1}^{h}\sum_{k=1}^{n} p(p+1)^{i-1}\alpha_{ik} = V. \tag{33}$$

Inequalities (28) ensure that messages sent to the consecutive layers do not overlap. By (29) the messages from the consecutive stages do not overlap. The remaining constraints are analogous to the ordinary tree case.

3.3.2. LLF

The communication and computation diagram for LLF activation order is presented in Fig. 5b. The linear program in this case is as follows (we skip repeating constraints (30)–(33)):

LP **BinomialTreeLLF**
minimize $C_{\max}$
subject to

$$t_{(i-1)k0} \geqslant t_{ik0} + C(p+1)^{i-1}\alpha_{ik} + Si, \quad i = 2,\dots,h,\ k = 1,\dots,n, \tag{34}$$

$$t_{h(k+1)0} \geqslant t_{1k0} + C\alpha_{1k} + S, \quad k = 1,\dots,n-1. \tag{35}$$

By (34) scattering to the consecutive destination layers do not overlap. Inequalities (35) ensure that the succeeding stages do not overlap.

The LP formulations presented in this section assumed that all stages and processors are really needed. It is not always true. When some number of processors, or stages, is sufficient to process the whole load, then some processors may receive no load. Using excessive processors introduce additional startup times, but does not increase processing speed. In such situations, superfluous stages and processors were removed successively until all load assignments were positive. The LP formulations were adjusted accordingly.

4. Performance modeling

We present results obtained by means of `lp_solve` [2], a public domain linear program solver. Over 2600 instances of LP problems were solved. The biggest successfully solved instance had over 50 000 variables and 156 000 constraints. Since the space of possible parameter values is enormous, we restricted the study to the combinations that seem typical of applications.

4.1. Star

We modeled a system with $m = 10$ processors, $C = 1\text{E-6}$, $A = 1\text{E-3}$, and $S = 1\text{E-3}$ (these values can be for example: bandwidth 1 Mbyte/s, processing rate 1 kbyte/s, startup time 1 ms). Parameters V, D, were variable.

Let us start this section with considering the distribution patterns. In most of the cases the sizes of load chunks sent to the processors grow slowly both with the processor number and the stage number. It is because processors are activated early when the initial chunks are small. In the last stage the chunk sizes decrease in order to achieve simultaneous completion of the computations on all processors. This facilitates perfect load balance. The changes of the data chunks are demonstrated in Fig. 6. Processor numbers are aligned along the horizontal axis, sizes of the chunks are presented on the vertical axis. Note that no load is sent to P_{10} in the last stage.

We observed in our simulations that when $n > n_{\min}$, the limited buffer size manifests only when D is very small and messages are excessively fragmented. In order to better expose the influence of the limited communication buffer size we analyzed the case of the minimum possible number of scattering stages, i.e. $n_{\min} = \lceil \frac{V}{Dm} \rceil$. The results are shown in Fig. 7. For $D = \infty$ minimum number of stages is $n = 1$, and the minimum number of processors is $m = 1$. The lower bound $LB = S + \frac{VA}{m}$, representing ideal circumstances of

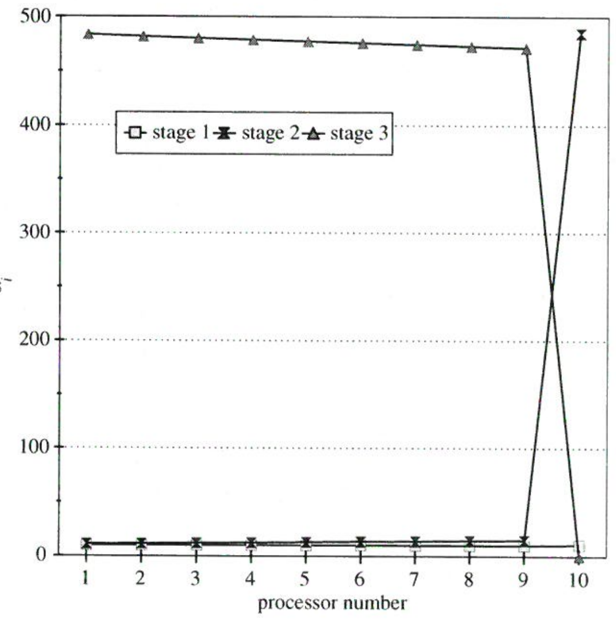


Fig. 6. Example load chunks sizes in different stages for star ($m = 10, A = 1\text{E}-3, C = 1\text{E-6}, D = 1\text{E4}, S = 1\text{E-3}, V = 5\text{E3}$).

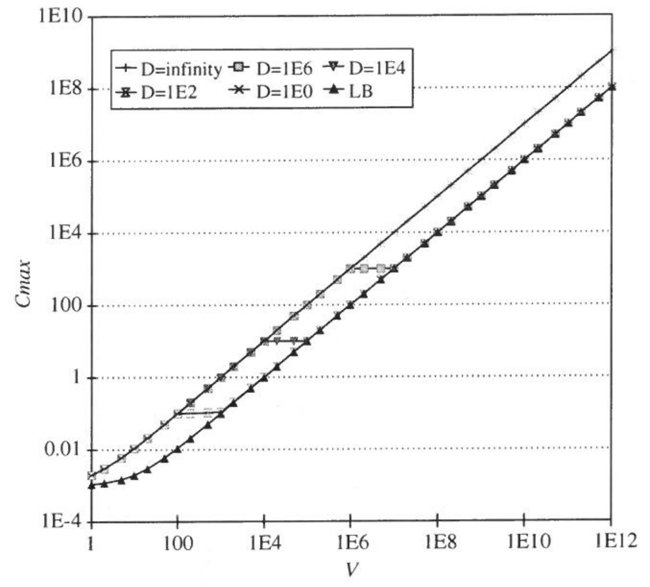


Fig. 7. $C_{\max}$ vs. V for various D and $n_{\min}$ in a star (A = 1E-3, C = 1E-6, S = 1E-3).

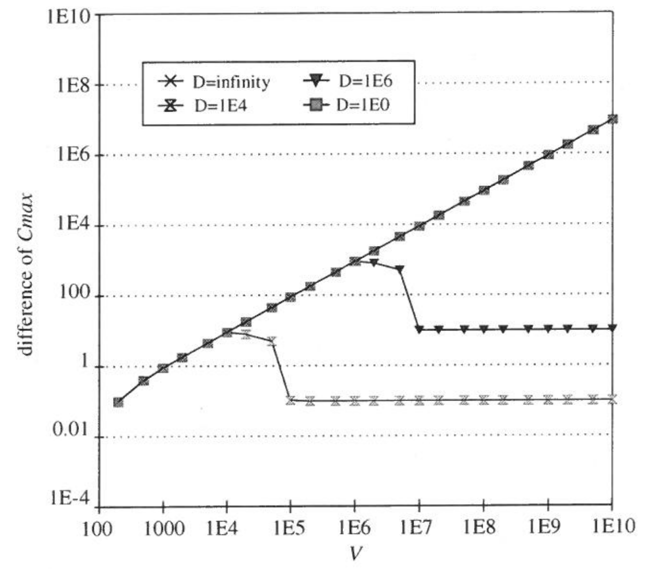


Fig. 8. Differences between $C_{\max}$ for various D in a star, for $n_{\min}$. The processing time for D = 1E2 is the reference (A = 1E-3, C = 1E-6, S = 1E-3).

processing the load, is added to show existing potential for reducing the processing time. In the ideal case at least one startup time S must elapse before any processor starts computing. The computing phase may not last shorter than $\frac{VA}{m}$ which is the case of ideal load balance. In Fig. 7 lines for $D = \infty$ and $D = 1$ overlap by the coincidence of values A, C, S. The lines for other communication buffer sizes follow the line of infinite buffer when $V \leqslant D$, because only one processor is activated. When V exceeds D more than one message must be sent, and it is profitable to activate additional processors. Additional processing power compensates for the growing V, and $C_{\max}$ does not increase significantly for $V \in [D, mD]$. If $V > Dm$ all processors are activated, the communications overlap computations, therefore processing time is similar to the lower bound.

Fig. 7 does not show that the differences between the processing times in absolute terms can be arbitrarily big. This is demonstrated in Fig. 8 showing the difference between the processing time for D = 1E2, and other buffer sizes, for $n_{\min}$. There is no difference for $V <$ 1E2 because only one message is sent. The difference grows with V until D, and levels off for $V > mD$. For $V > mD$ the lines are parallel because the duration of the computing part is the same. Only the communication time in the first stage is different due to different communication buffers sizes and different message lengths.

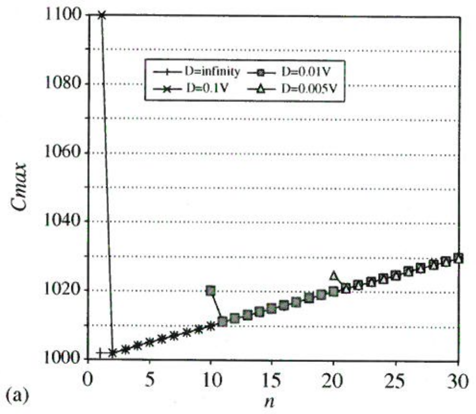
On the basis of the above charts one may think that the communication buffer should be small, yet, big enough to avoid excessive message fragmentation. Still, there is one more way in which communication buffer may influence processing time. Consider an example: $m = 3, A = C = 1, S = 0, V = 3$. When $D = 1$, only one stage is needed ($n = 1$). The processors receive $\alpha_{11} = \alpha_{21} = \alpha_{31} = D = 1$ units of load. The communication phase lasts 3 units of time, the computations on the last activated processor complete 4 units of time after the communication start. On the other hand, when $D \geqslant 1.5$ a different load distribution is possible: $\alpha_{11}' = 1.5, \alpha_{21}' = 1, \alpha_{31}' = 0.5$, processors stop computing at time 3.5. Thus, too small communication buffer may cause imbalance of the computing completion times.

In Fig. 9 dependence of processing time on D, and n for fixed V is shown. D is expressed as a fraction of V. $C_{\max}$ decreases with growing n. The rate of the decrease is fast initially, but later the returns from increasing n are diminishing. After exceeding a certain limit, $C_{\max}$ grows linearly with n. This is an effect of startup time S appearing with each communication. S is added even if the size of the load chunk is 0. This linear increase of $C_{\max}$ can be considered an inaccuracy of the model because it means that we still send the assumed number nm of messages though some of them contain no load.

From the above figures we draw a conclusion that there are some optimal communication buffer size D^*, and number of stages n which on one hand, prevent excessive message fragmentation, and the other hand, balance the load well. We discuss this in Section 5.

4.2. Ordinary and binomial trees

Relations specific for trees are shown in this section, because dependence of $C_{\max}$ on D, n are similar to the ones for the star. Unless specified differently, we

(a) 

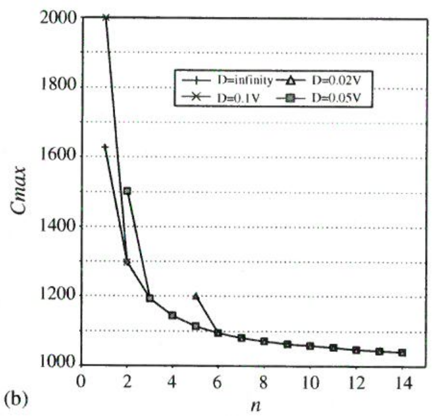
(b) 

Fig. 9. Dependence of $C_{\max}$ on D, n. (a) ($m = 10, A = 1$E-4, $C = 1$E-4, $S = 1$E-1, $V = 1$E7), (b) ($m = 10, A = 1$E-3, $C = 1$E-4, $S = 1$E-2, $V = 1$E7).

assumed $p = 2, h = 10, A = 1$E-3, $C = 1$E-6, $S = 1$E-3 in all the following simulations.

First let us analyze the sizes of the load chunks assigned to the processors of the consecutive layers. Let us remind that the load sent to the deeper layers of both ordinary, and binomial trees is split each time the load is relayed (cf. inequalities (11), (32)). The message sent from the originator to the first layer has its size limited to D. Thus, the sizes of messages sent to layer i are at most $\frac{D}{p^{i-1}}$ in the ordinary trees, and $\frac{D}{p(p+1)^{i-2}}$ in the binomial trees. The exponential reduction of the load chunks restricts usability of the deep trees, especially when the load grains come into play. The optimum distribution of the load among the layers does not expose a fixed regularity because processing in trees resemble a heterogeneous star. The layers are like heterogeneous processors connected to the originator via heterogeneous communication links. In the heterogeneous star the optimum selection of the processors to be used and the activation order have combinatorial nature [10]. However, some common patterns have been observed. The initial layers with few processors often received no load. It appears to be advantageous to activate deeper layers which have more processors while omitting the initial layers. The size of the chunks sent to the deep layers is restricted by the communication buffer size D used for the communication between the originator and the first layer. Therefore, to exploit the processors of the deep layers to the full extent, the maximum load was sent, i.e. $\frac{D}{p^{i-1}}$ in the ordinary trees, and $\frac{D}{p(p+1)^{i-2}}$ in the binomial trees. The selection of the used layers, and the order of activating them remain open problems which have combinatorial nature.

The complex nature of the optimum layer activation order is exposed in one more way. A bigger load may be processed in shorter time than a smaller load when NLF activation order is applied, and h, n are minimum possible. This is illustrated in Fig. 10 showing dependence of the processing time on the size of the problem for binomial tree with NLF activation order and

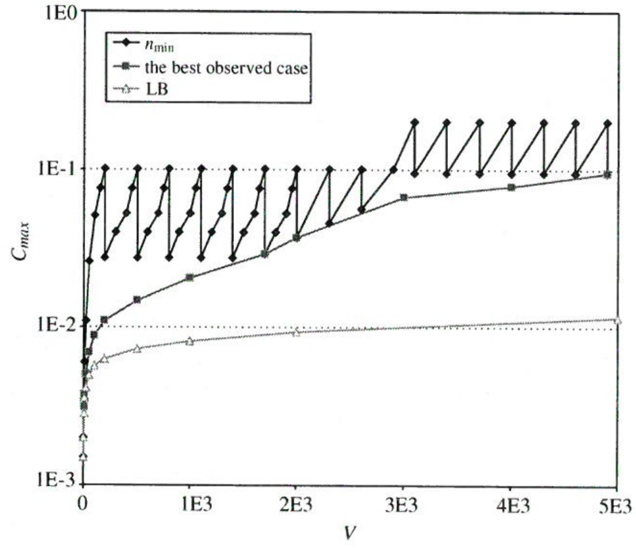


Fig. 10. Processing time in a binomial tree for small loads (NLF, $D = 1$E2, $A = 1$E-3, $C = 1$E-6, $S = 1$E-3).

$D = 1$E2. It can be seen in Fig. 10 that for $n_{\min}$ processing time may decrease with increasing V. The explanation for this counterintuitive behavior is that for the given V only a certain number of layers can be activated within the limited span of communication time. Adding more layers is not productive because there is no work for them. On the other hand, adding a little more load allows for activating a new layer which has at least the same number of processors as all the preceding layers (because of NLF activation order). This allows for shifting most of the load to the deep layer, and thus reduces the processing time. The medium line of the best observed case illustrates the possible benefit of increasing the number of stages. However, there is a technical difficulty in finding optimum distributions for big n caused by the numerical instability of LP solver. The third line (LB) shows potential gains from using a different strategy which bases on selecting one deep layer for computations while using the other layers for communications only. The alternative processing strategy was constructed as follows. At least one layer must be activated. Suppose only layer i is activated, and the load is sent in equal chunks in all stages. The number of processors in layer i is $p(p+1)^{i-1}$. The processing time is $C_{\max}(n) = n(iS + \frac{V}{np(p+1)^{i-1}}C(p+1)^{i-1}) + A\frac{V}{np(p+1)^{i-1}}$, which is a function of n. The first derivative of $C_{\max}$ over n is $iS - \frac{AV}{p(p+1)^{i-1}n^2}$. $C_{\max}$ has minimum for $n^* = \sqrt{\frac{AV}{iSp(p+1)^{i-1}}}$. The lower bound was selected as minimum $C_{\max}(n^*)$ over layers $i = 1, \dots, h$. This strategy is effective as far as processing time is considered, but average utilization of the processing resources may be unsatisfactory. It can be concluded that processing time depends on the selection of the activated layers, and the activation order. To avoid arbitrary decisions in selecting the set of used layers, we assumed $n_{\min}$ in the following discussion.

The dependence of $C_{\max}$ on V for various D, binomial tree, LLF activation order is shown in Fig. 11. These dependencies for binomial and ordinary trees, NLF and LLF activation orders are very similar. In the case of $D = \infty$ only one message is needed to send all the load. Hence, only p processors in layer 1 are activated. Processing time for $D = 1E0$ is bigger than for $D = \infty$. This means that communication buffer is too small, message fragmentation is excessive, communication time dominates and is even longer than processing the whole load on one layer. For $D \in [1E2, 1E6]$ the changes of the processing time are similar to the case of the star topology (cf. Fig. 7). When $V < pD$ only one layer of processors is activated. When $V \in [pD, (h(p+1)-1)D]$ the growing V is compensated for by activating additional layers. Note the uneven changes in this interval resulting from the changes of the number of layers that can be effectively exploited. When $V > (h(p+1)-1)D$ processing times approach the same line. This results from the fact that processing time in the first layer sets the time span of a single stage. In other words communications and computing in the deeper layers is hidden in computing in the first layer. The computing time in the first layer is DA, when maximum size of the buffer is utilized. The number of stages is $n \geq \frac{V}{D(h(p+1)-1)}$. Combining these two formulae we get processing time $nDA \approx \frac{VA}{(h(p+1)-1)}$ which is the asymptote approached by the lines for $D \in [1E2, 1E6]$. This situation could have been avoided provided that the first layers were not computing, but only relaying the load. The lowest line (LB) represents an alternative processing strategy described in the preceding paragraph. Big difference between LB strategy and other lines demonstrates that a non-classic approach to the communication-computation interaction may be profitable.

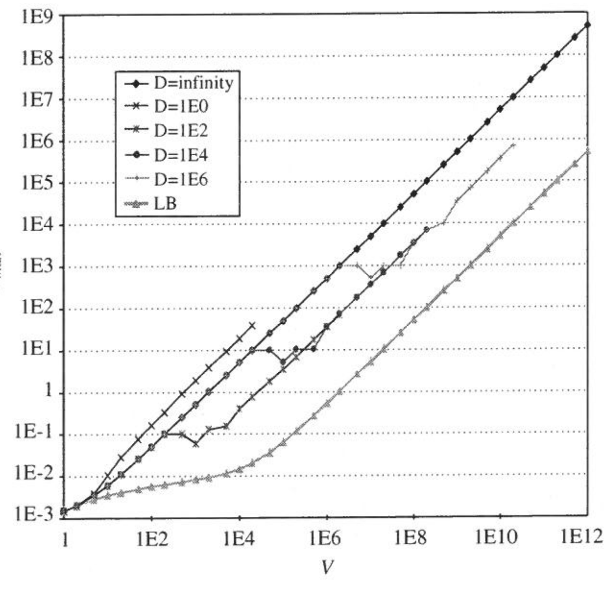
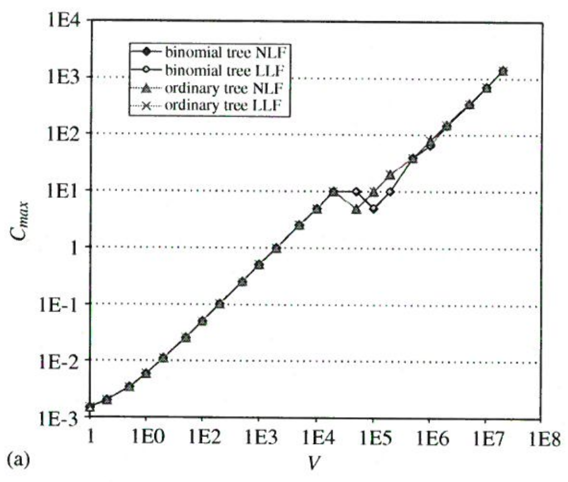


Fig. 11. Processing time in a binomial tree vs. V for various D (LLF, $n_{\min}$).

In the following discussion, we compare, in the sense of processing times, NLF with LLF layer activation order, 1-buffer with 2-buffer case. Let us observe that comparing the binomial trees with the ordinary trees is not easy because different numbers of processors are activated in these structures. Therefore, we compared a binomial tree with $h = 5$ which has $m = 243$ processors with an ordinary tree which has $h = 7$ and $m = 255$ processors. Hence, the difference in processing power is less than 5%. In Fig. 12a the dependencies of $C_{\max}$ on V for 1-buffer binomial and ordinary trees for $D = 1E4$ are shown. This relation for 1-buffer case and other values of D, also for 2-buffers and $D > 1E0$ are very similar in the form. In Fig. 12a the differences between all the cases appear only if $V \in [pD, D(h(p+1)-1)]$. The explanation is that for $V < pD$ only one layer is activated in all cases. For $V > D(h(p+1)-1)$ the whole processing time is dominated by computations on the first layer. The deeper layers receive inadequate load, as mentioned in the preceding paragraph. It can be seen that there are problem sizes where the ordinary tree dominates, and problem sizes where the binomial tree dominates. Though the binomial tree has smaller total number of processors, it is hard to claim that it is better than the ordinary tree because in neither case has the computational capacity been fully exploited. In Fig. 12b the same relationship for $D = 1E0$ and two buffers is shown. For $V > D(h(p+1)-1)$, i.e. when $n > 1$, processing time for ordinary tree is approximately 50% of the time for binomial tree. This situation is caused by different times that elapse between the stages. Though communication times are shorter in binomial trees, the communications of the consecutive stages may not



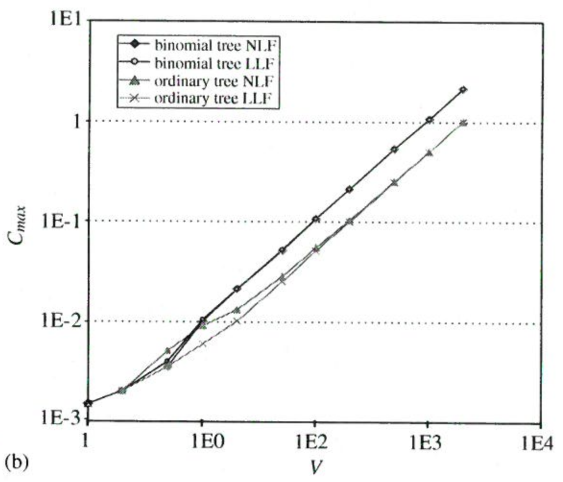
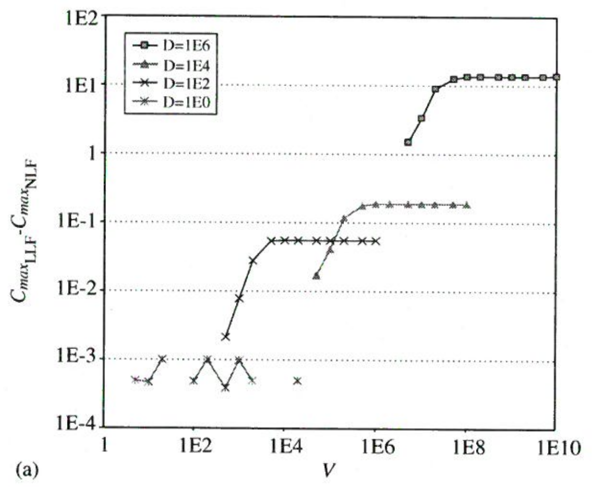


Fig. 12. Processing time in binomial trees and ordinary trees vs. V ($n_{\min}$) (a) 1-buffer $D = 1E4$, (b) 2-buffers $D = 1E0$.



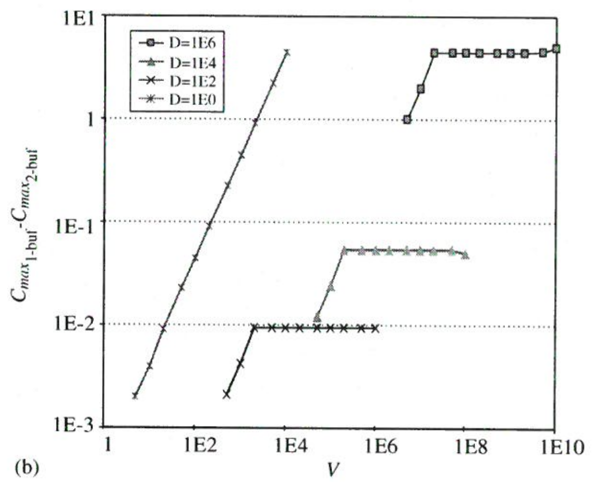


Fig. 13. Reduction of schedule length as an effect of optimizations in trees. (a) Binomial trees NLF vs. LLF, (b) ordinary trees 1-buffer vs. 2-buffer.

overlap. On the contrary, in the ordinary trees such an overlap is possible.

The difference between LLF and NLF layer activation modes for the binomial trees is shown in Fig. 13a. NLF activation order is faster than LLF (with the exception of several cases for $D = 1$) because of the following phenomena: as the deeper layers are underutilized, their communications and computations can be hidden in the interval of computing on the first layer. Processing time is determined by the length of the first message to the first layer, and computations on the first layer. The computations on the first layer last the same time in LLF, and NLF. Therefore the time until starting the computations on the first layer makes the difference in the schedule length. In LLF the first layer is activated as the last one, and the bigger D is the worse LLF is. This situation completely reverses the domination of the LLF activation order for networks with the unlimited communication buffers, as shown in [12]. For ordinary trees the situation is very similar.

The difference in processing time between 1- and 2-buffer ordinary trees with LLF activation sequence is shown in Fig. 13b. As it can be seen for $D>1$ the difference stabilizes. The explanation is the same as for the previous figure. The difference comes from the time until activating the first layer. Two buffers allow for faster activation of the first layer in LLF. For $D = 1$ the difference steadily grows with V because messages are short and the load of a certain layer is computed within the interval of communications with the other layers. Thus, mainly communication time matters in the whole processing time. Communication with 2 buffers are faster. Hence 2-buffer case domination grows with growing V. In NLF activation case situation is similar for $D = 1$. For $D>1$ no difference between 1- and 2-buffer cases has been observed. Since computations on layer 1 determine total processing time, and layer 1 is activated first in NLF, advantages of shorter communication to other layers in 2-buffer way have no influence on the processing time.

5. Discussion

The influence of the limited communication buffer size manifests in several ways. When the communication buffer is too small messages are excessively fragmented and processing time is dominated by communication time. Insufficient communication buffer may cause load imbalance. On the other hand, also big buffers may cause imbalance when minimum possible number of stages is used. Thus, even for big buffers it is reasonable to implement limited sizes of the messages to activate computations quickly, or to balance the load. There is a direct relation between the communication buffer size and the number of stages. It is generally advantageous to have many scattering phases because all processors are activated, and load is balanced better.

In trees we observed that communication buffer size significantly restricted the amounts of load which could be transferred to the deeper layers. The bandwidth of the initial layers was to small to feed deeper layers with load. Consequently, deeper layers completed computations before receiving a new chunk of load, and the processing time was dictated by the first layer. This phenomenon is a great loss of efficiency. Advanced communication methods using, e.g., binomial trees, LLF strategy, or 2-buffer communication nodes, do not outperform their less sophisticated counterparts such as ordinary trees, NLF strategy, 1-buffer nodes. Several remedies can be suggested. It is possible to increase the communication buffer size such that processors receive enough load to keep computing during the whole communication phase of one stage. Still, this solution does not scale well because communication buffers would have to grow exponentially with the number of layers. A different solution is to change the communication algorithm and, e.g., send several messages to the deep layers per each message sent to layer 1. Also this solution has a disadvantage: a startup time elapses with each sent message. It is possible to use the initial layers for communication only, and the deep layers for computations. Potential of such an alternative strategy is shown as LB in Fig. 11.

Let us now consider the optimum size D^* of communication buffer. Due to the many phenomena affecting the schedule length finding a generally optimum solutions is difficult. Nevertheless, some good heuristics are needed. Let us omit the transient states, and let us assume that all messages sent from the originator are of size D^*. It is a reasonable idea to have a communication buffer such that processors keep computing until the next communication stage. In a star it means $AD^* \geqslant m(S + D^*C)$, and $D^* \geqslant \frac{mS}{A-mC}$. Note that $D^* > 0$ only when $A > mC$, and idle times may arise inevitably when mC is too big.

Now we estimate analogous buffer size for trees. In binomial trees processors in layer i receive load $\alpha_{ik} = \frac{D^*}{p(p+1)^{i-2}}$ in some stage k. The communication time from the originator to layer i is $Si + C(p+1)^{i-1}\alpha_{ik}$ (cf. Section 3.3). After substituting α_{ik} and summing over layers $i = 1, \ldots, h$ we get total communication time in one stage $\frac{S(h+1)h}{2} + \frac{CD^*(p+1)h}{p}$. Since layer h receives the least load its computing time $\frac{AD^*}{p(p+1)^{h-2}}$ is the shortest in a tree. The requirement that computing time is at least equal to the communication time can be formulated as $\frac{AD^*}{p(p+1)^{h-2}} \geqslant \frac{S(h+1)h}{2} + \frac{CD^*(p+1)h}{p}$, from which we get $D^* \geqslant \frac{S(h+1)hp(p+1)^{h-2}}{2(A - Ch(p+1)^{h-1})}$. Also here D^* exists only when $A > Ch(p+1)^{h-1}$, i.e. using too many layers h may result in inevitable idle times.

In 2-buffer ordinary tree even and odd communications can overlap each other. Thus communications in one stage last $h(S + CD^*)$. Computations in layer h, which receives the least load, last $\frac{D^*A}{p^{h-1}}$. Hence, we get the requirement $D^* \geqslant \frac{hS}{A/p^{h-1} - Ch}$. In 1-buffer ordinary tree two consecutive messages from the originator to layer 1 and from layer 1 to 2 cannot overlap. The two messages can be sent in time $2S + CD^* + CD^*/p$. Since the messages in the deeper layers are shorter, and can overlap the communications of the first layer, the communication time in one stage lasts approximately $hS + hCD^*(1 + \frac{1}{p})$. The computing time on the last layer is $\frac{D^*A}{p^{h-1}}$. From this we get a requirement $D^* \geqslant \frac{hS}{\frac{A}{p^{h-1}} - hC(1+\frac{1}{p})}$.

Note that the above formulae expressing D^* link structure of the tree (i.e. h,p), communication parameters S, C, and computing rate A. Not for all combinations of these parameters $D^* > 0$ exists.

6. Conclusions

In this paper, we proposed a formal method of analyzing computations in distributed systems with communication buffers of limited size. The method based on divisible load theory is versatile and computationally tractable. It establishes a link between scheduling and communication optimization. The results regarding communication optimization reach beyond just selecting a good communication buffer size. For trees it was observed that classic scattering algorithms do not perform well when the communication buffer size is limited. Two negative effects were observed: underutilization of the deep tree layers, and a communication bottleneck around the originator. Hence there is a demand for different scattering algorithms in trees. An idea of such an algorithm has been proposed.

Acknowledgments

This research was partially supported by a grant of the State Committee for Scientific Research.

Appendix A. Summary of the notation

α_{ik}	amount of load sent to processor P_i in stage k (bytes)
A	computing rate (s/byte)
C	communication rate (s/byte)
$C_{\max}$	schedule length
D	communication buffer size (bytes)
h	number of computing layers in a tree
m	the number of processors
n	number of the stages
$n_{\min}$	minimum feasible number of the stages
p	out-degree of a node in a tree, also number of processor ports working simultaneously
P_0	the originator
$P_1, \dots, P_m$	processors
S	communication startup time (s)
t_{ik}	start of communication to processor P_i in stage k, for star network (s)
t_{ikl}	start of sending load to processors in destination layer i in stage k from the intermediate node(s) in layer l, for ordinary and binomial trees (s)
V	total volume of the load (bytes)

References

[1] G.D. Barlas, Collection-aware optimum sequencing of operations and closed form solutions for the distribution of a divisible load on arbitrary processor trees, IEEE Trans. Parallel Distrib. Systems 9 (1998) 411–429.

[2] M. Berkelaar, lp_solve-Mixed Integer Linear Program solver, ftp://ftp.es.ele.tue.nl/pub/lp_solve, 1995.

[3] V. Bharadwaj, G. Barlas, Access time minimization for distributed multimedia applications, Multimedia Tools Appl. 12 (2000) 235–256.

[4] V. Bharadwaj, D. Ghose, V. Mani, T. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[5] V. Bharadwaj, D. Ghose, T. Robertazzi, Divisible load theory: a new paradigm for load scheduling in distributed systems, Cluster Comput. 6 (2003) 7–17.

[6] J. Błażewicz, M. Drozdowski, F. Guinand, D. Trystram, Scheduling a divisible task in a 2-dimensional mesh, Discrete Appl. Math. 94 (1999) 35–50.

[7] J. Błażewicz, M. Drozdowski, M. Markiewicz, Divisible task scheduling—concept and verification, Parallel Comput. 25 (1999) 87–98.

[8] S.K. Chan, V. Bharadwaj, D. Ghose, Experimental study on large size matrix–vector product computations using divisible load paradigm on distributed bus networks, Technical Report TR-OSSL/BV-DLT/01, Open Source Laboratory, Department of Electrical and Computer Engineering, The National University of Singapore, Singapore, 2000.

[9] Y.-C. Cheng, T.G. Robertazzi, Distributed computation with communication delay, IEEE Trans. Aerospace Electron. Systems 24 (1988) 700–712.

[10] M. Drozdowski, Selected problems of scheduling tasks in multiprocessor computer systems, Series: Monographs, No.321, Poznań University of Technology Press, Poznań, 1997, Also: http://www.cs.put.poznan.pl/~maciejd/txt/h.ps.

[11] M. Drozdowski, P. Wolniewicz, Experiments with scheduling divisible tasks in clusters of workstations, in: A. Bode, T. Ludwig, W. Karl, R. Wismüller (Eds.), Euro-Par 2000, Lecture Notes in Computer Science, Vol. 1900, Springer, Berlin, 2000, pp. 311–319.

[12] W. Głazek, Algorithms for scheduling divisible tasks on parallel machines, Ph.D. Thesis, Department of Electronics, Telecommunication and Computer Science, Technical University of Gdańsk, Poland, 1998.

[13] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[14] G.L. Nemhauser, L.A. Wolsey, Integer and Combinatorial Optimization, Wiley, New York, 1988.

[15] J.G. Peters, M. Syska, Circuit-switched broadcasting in torus networks, IEEE Trans. Parallel Distrib. Systems 7 (1996) 246–255.

[16] T. Robertazzi, Ten reasons to use divisible load theory, IEEE Comput. 36 (2003) 63–68.

Maciej Drozdowski received M.Sc. in control engineering in 1987, and Ph.D. in computer science in 1992. In 1997, he defended his habilitation in computer science. Currently, he is an associate professor at the Institute of Computing Science, Poznan University of Technology. His research interests include design and analysis of algorithms, complexity analysis, combinatorial optimization, scheduling, computer performance evaluation.

Paweł Wolniewicz graduated from Poznan University of Technology and received M.Sc. in computer science in 1997. In 2003 he presented and defended his Ph.D. thesis in computer science at the Institute of Computing Science, Poznan University of Technology. Currently, he works for Poznan Supercomputing and Networking Centre, Poznan, Poland. His research interests include metacomputers, distributed environments and scheduling.