Divisible load scheduling in systems with limited memory

M.Drozdowski[1], P.Wolniewicz[2]

Abstract

In this work we consider scheduling divisible loads on a distributed computing system with limited available memory. The communication delays and heterogeneity of the system are taken into account. The problem studied consists in finding such a distribution of the load that the communication and computation time is the shortest possible. A new robust method is proposed to solve the problem of finding optimal distribution of computations on star network, and networks in which binomial trees can be embedded (meshes, hypercubes, multistage interconnections). We demonstrate that in many cases memory limitations do not restrict efficiency of parallel processing as much as computation and communication speeds.

Keywords: divisible load theory, scheduling, performance evaluation, communication delays, memory constraints.

1 Introduction

In this work we consider scheduling *divisible loads*. Divisible computation can be divided into several independent parts of arbitrary size. Due to the independence the parts can be processed by a set of distributed computers. An example of processing a divisible load is searching in a database. The database file can be divided with granularity of one record, and can be distributed between several computers for processing. Usually the database file is large, and the records are small. Thus, the assumption on arbitrary divisibility of the work is satisfactorily fulfilled. Other applications of divisible load model can be found in processing big measurement data files, image

[1]Institute of Computing Science, Poznań University of Technology, ul.Piotrowo 3A, 60-965 Poznań, Poland. This research was partially supported by KBN grant. Corresponding author.

[2]Poznań Supercomputing and Networking Center, ul.Noskowskiego 10, 61-794 Poznań, Poland.

and signal processing, molecular dynamics simulations, linear algebra, parallel metaheuristics [2, 7]. Communication delays are unavoidable element of distributed computations. We assume that sending x units of data over link j takes $S_j + xC_j$ units of time, where S_j is a communication startup time (expressed e.g. in seconds), and C_j is transfer rate (reciprocal of speed, expressed e.g. in seconds per byte). We also assume that processing x units of load on processor P_i $(i = 0, \ldots, m)$ takes xA_i time units, where A_i is processing rate (expressed e.g. in seconds per byte). The load is distributed from processor P_0 called *originator* to a set of parallel processors. The problem consists in finding distribution of the load, i.e. amounts α_i of work to be send to processor P_i $(i = 0, \ldots, m)$ such that communication and computation time is the shortest possible. The whole load consists of V units of work. These assumptions resulted in a tractable model of distributed computations. Even analytical solutions (i.e. closed form solutions expressing α_i's) were found [2, 6] for many computer architectures. Let us observe that this model can represent not only communications and computations, but also transportation systems and factories. For example, an ore mine can be the originator and ore processing plants can play the role of processors.

It was demonstrated that divisible load model can be used in predicting performance of real computations [4, 7]. However, in clusters of workstations it appeared [7] that the linear dependence of processing time on the size of work is satisfied only if the computations are restricted to the core memory (RAM). Larger work chunks imply using virtual memory. When virtual memory is used, the dependence of processing time on the amount of data becomes more complex. Also the processing speed of the computers is lower. Hence, for efficiency reasons it is preferable to avoid using virtual memory and restrict the load to limited amount B_i of core memory available at processor P_i. We assume that the critical restriction on the size of memory is put during the computation phase. It can be the case of problems where small data sets are unpacked or big data structures arise in computation from small amount of input data. Therefore, the size of communicated message is not limited otherwise than by the memory capacity of the receiver.

The problem of scheduling divisible loads in a star (referred to as a single-level tree) with limited memory was studied in [13]. The communication startup times were not taken into account in [13]. A fast algorithm called Incremental Balancing Strategy (IBS) has been proposed. In IBS processors are loaded incrementally. In each increment distribution of the load is found for processors with available memory according to the standard divisible

load theory methods [2] without taking the memory constraints into account. Then, the distribution of the load is scaled proportionately such that at least one buffer is filled completely. The remaining available buffer capacities are memory sizes in the next increment. This process is continued until distributing all the load. It has not been proved that IBS algorithm derives optimal distribution of the load. Also the problem of finding the optimum activation sequence in the case of memory limitations has been discussed in [13]. It has been demonstrated that the rule for optimum processor activation sequence proposed in [2] does not work in the case with limited memory.

In this work we propose a new method of finding solutions with guaranteed optimality for the problem of scheduling divisible loads in networks of processors with limited memory and communication startup times. The method introduces mathematical programming to the realm of divisible load theory. We analyze two network types: star and binomial tree, in Section 2, and Section 3, respectively. The implications of memory limitations for the performance are studied. The problem of finding the optimum sequence of activating processors with limited memory in a star network is also solved. For the convenience of the reader the notation used in this paper is summarized in the appendix.

2 Star

In this section we analyze a star network of processors. The star network can be viewed as a tree with a root and a set of leaves. The originator is located in the root of the tree. The processors are located in the leaves. This topology conveniently represents computations on a bus network or master-slave model of parallel computations in a cluster of workstations [7]. We assume that the originator both communicates and computes, and that simultaneous computation and communication is possible. For the simplicity of mathematical models it is assumed that the time of returning the results is negligible. It was demonstrated [2, 5, 11] that this assumption does not limit generality of the considerations, and returning of the results can be easily included in mathematical models. In the star network originator sends chunk α_i of load to processor P_i. Immediately after receiving its load P_i starts computing, while the originator immediately starts the communication with the next processor. Let C_{max} denote the length of the schedule. The process of communication and computation is presented in Fig.1.

insert Fig.1 here

2.1 The linear program method

Here we assume that the sequence of sending the load to the processors is $P_1, \ldots, P_m$, and is fixed. Our problem can be formulated as a linear program LP1:

LP1:
minimize C_{max}
subject to:

$$\alpha_i A_i + \sum_{j=1}^{i} (S_j + \alpha_j C_j) \leq C_{max} \quad i = 0, \ldots, m \tag{1}$$

$$\alpha_i \leq B_i \quad i = 0, \ldots, m \tag{2}$$

$$\sum_{i=0}^{m} \alpha_i = V \tag{3}$$

$$\alpha_i \geq 0 \quad i = 0, \ldots, m \tag{4}$$

Let us explain the above formulae. We are to minimize schedule length C_{max} by finding values of variables α_i, C_{max} such that: by equations (1) each processor completes not later than at C_{max}, by equations (2) no processor is assigned more load than the size of its memory, according to equation (3) all the load fractions add up to the total load V. In the equation (1) for $i = 0$ we have $\sum_{j=1}^{0}(S_j + \alpha_j C_j) = 0$. Linear programming is a method of modeling and solving engineering problems [9, 14, 15]. LP1 has $m + 2$ variables and $3m + 4$ constraints. The solution of LP1 is a point in $m + 2$-dimensional space. Constraints (1), ..., (4) restrict the area of admissible solutions to a convex polytope. It is known that the optimum solution is located in one of the polytope corners. Unfortunately, the location of the optimum depends on the problem instance and no closed-form expression of α_i seems possible. The linear programs can be solved in polynomial time, e.g. in $O(m^{3.5}L)$ time [12, 14] using the interior point methods, where L is the length of the string encoding all the parameters (A_i, C_i, S_i, B_i, V) of LP1. Linear programs can be solved by many public domain and licensed codes. All linear programming formulations in this paper were solved by `lp_solve` ver. 2.0 [1], a public domain linear programming code. Our method is more time-consuming but it is also more robust than the algorithm proposed in [13]. Consider Example 3 from [13].

Example. $m = 4$ (i.e. we have originator and 4 additional processors). Processing rates are: $A_0 = 1, A_1 = 5, A_2 = 4, A_3 = 3, A_4 = 2$. Available memory

sizes are: $B_0 = 10, B_1 = 20, B_2 = 45, B_3 = 15, B_4 = 30$. Communication rates are $C_1 = 4, C_2 = 3, C_3 = 2, C_4 = 1$. All startup times are $S_i = 0$, for $i = 1, \ldots, 4$. V=100. By solving LP1 we obtain:

processor	B_i	α_i	comm. completion	computation completion
P_0	10	10	0	10
P_1	20	15	60	135
P_2	45	30	150	270
P_3	15	15	180	225
P_4	30	30	210	270

This schedule has $C_{max} = 270$, and is shorter than the one found by IBS algorithm in [13]. This is so because the optimality of LP formulation is guaranteed, whereas IBS is a fast heuristic. The length of the schedule is determined by the completion of computations on processors P_2 and P_4. P_1, P_2 memory is not fully utilized. Note that in the interval [10,210] P_0 is not computing but only communicating. □

2.2 Performance modeling

Now, we will discuss the influence of memory size on the performance of star networks.

We modeled a system of initiator and 9 identical processors with $A_i = A$=1E-6, connected by identical communication links with startup $S_i = S = 0.001$, $C_i = C$=1E-6, and $B_i = B$. The sizes of available memory were equal on all processors and the originator. A feasible solution of LP1 may not exist when the sum of buffer capacities is smaller than V. When a feasible solution existed we recorded the best solution for one of the number of processors from the range $1, \ldots, 10$ (including the originator). The results of modeling are collected in Fig.2. On the horizontal axis we have size of the problem V, on the vertical axis we have schedule length C_{max}. Plots for memory sizes from $B = 10$ to B =1E9 are presented. As it can be verified with $B = 10$ we can solve problems with size up to $V = 100$ on ten processors. Two more lines denoted "sat" and "inf" are depicted in Fig.2. Line "sat" represents a system with total memory sizes exactly equal to V. This means that $B = \frac{V}{m+1}$ and memory buffers are *saturated*. Schedule length in a saturated system is $C^{sat}_{max} = \sum_{j=1}^{m}(S + \frac{V}{m+1}C) + \frac{V}{m+1}A$. Line "inf" represents schedule length C^{inf}_{max} in a system with *unlimited memory*. In this case memory size is

big enough to hold any loads and we can calculate the distribution of the load according to the classical divisible load theory methods [2, 5]. The plots of processing times for particular memory sizes are located between lines "inf" and "sat". As it can be seen line "sat" approaches line "inf" at $V \approx$1E4. For bigger volumes the two lines form a kind of *tunnel* in which schedule length for the particular memory size must fit. The width of this tunnel shows influence of memory limitation on the schedule length because its upper line represents the system which has just as much memory as needed to hold the load, while the lower line represents a system which has unlimited available memory. From the results presented in Fig.2 it can be concluded that for big problems (where V is big), memory limitations are not as important as the communication and computation speeds. insert Fig.2 here

In order to demonstrate the influence of memory size constraints on C_{max} we collected in Fig.3 values of "inf" and "sat" cases for various processing rates A. A system with A =1E-9 has the fastest, while the system with $A = 1$ has the slowest processors. The plots for $A = 1$ are so close that they are drawn one on another. Also the lines for the saturated systems with A =1E-9 and A =1E-6 are so close that they are indistinguishable. For small A schedule length in saturated system is dominated by communications which last $\sum_{j=1}^{m}(S + \frac{VC}{m+1})$. Therefore, the lines for A =1E-9 and A =1E-6 in saturated system are very close. The increase of A results in lines "inf" going up in the Fig.3. The "sat" lines must be located above "inf" lines. As A increases the difference between "inf" and "sat" decreases such that eventually for $A = 1$ they are indistinguishable. It can be observed that for computationally intensive applications which have big A, and big volumes V the difference between "sat" and "inf" cases is small and schedule length is dominated by communication and processing speeds. insert Fig.3 here

Now we will calculate width of the tunnel, i.e. the ratio of schedule lengths in the saturated and unlimited memory cases, on a set of identical processors for big problem sizes. In the following we denote by $\rho = \frac{C}{A}$, and $\sigma = \frac{S}{A}$.

Lemma 1 *In the star interconnection* $\lim_{V\to\infty,\rho\to 0} \frac{C_{max}^{inf}}{C_{max}^{sat}} = 1$.

Proof. In the saturated case $C_{max}^{sat} = \sum_{i=1}^{m}(S + \frac{VC}{m+1}) + \frac{AV}{m+1} = mS + \frac{V(mC+A)}{m+1}$. It was proved [2] that when results are not returned, and memory is unlimited, all processors must stop computing in the same moment of time. From this observation we can infer that time $A_i\alpha_i$ of computing on

processor P_i activated earlier is equal to time $S_{i+1} + \alpha_{i+1}(A_{i+1} + C_{i+1})$ of sending the load to processor P_{i+1} activated later and computing on P_{i+1}. As we consider identical processors we have: $A\alpha_i = S + \alpha_{i+1}(A + C)$. α_i can be expressed as a linear function of α_m: $\alpha_{m-i} = \alpha_m(1+\rho)^i + \frac{\sigma}{\rho}((1+\rho)^i - 1)$ for $i = 1, \ldots, m$. All α_is must add up to V. Therefore, $V = \sum_{i=0}^{m} \alpha_i = \alpha_m \frac{(1+\rho)^{m+1}-1}{\rho} + \frac{\sigma}{\rho^2}((1+\rho)^{m+1} - 1 - \rho - m\rho)$. From which α_m, and $C_{max}^{inf} = A\alpha_0$ can be calculated: $C_{max}^{inf} = \frac{A[V - \frac{\sigma}{\rho^2}((1+\rho)^{m+1}-1-\rho-m\rho)]\rho(1+\rho)^m}{(1+\rho)^{m+1}-1} + \frac{A\sigma}{\rho}((1+\rho)^m - 1)$. Finally, we have the desired ratio for big V:

$$\lim_{V\to\infty} \frac{C_{max}^{inf}}{C_{max}^{sat}} = \frac{(m+1)\rho(\rho+1)^m}{(m\rho+1)((\rho+1)^{m+1}-1)}. \tag{5}$$

For computation intensive applications $A \gg C$ and $\rho \to 0$. After applying de l'Hospital rule we obtain $\lim_{V\to\infty,\rho\to 0} \frac{C_{max}^{inf}}{C_{max}^{sat}} = \lim_{\rho\to 0} \frac{(m+1)\rho(\rho+1)^m}{(m\rho+1)((\rho+1)^{m+1}-1)} \stackrel{H}{=} \lim_{\rho\to 0} \frac{(m+1)[(\rho+1)^m + m\rho(\rho+1)^{m-1}]}{m((\rho+1)^{m+1}-1)+(m\rho+1)(m+1)(\rho+1)^m} = 1$. □

We conclude that in the case of big sizes V and computationally intensive applications executed on a set of identical processors, memory limitations are not as restrictive, for the schedule length, as computation and communication speeds. This observation is confirmed by Fig.2, and Fig.3. On the other hand it should not be forgotten that this result applies to homogeneous computing systems. In heterogeneous systems, the difference between C_{max}^{inf} and C_{max}^{sat} can be arbitrarily big. For example, when a fast processor has small memory buffer and a slow processor has a large buffer then the equivalent speed of the system is dominated by the slow processor in the "sat" case. Furthermore, parameters A, C, S may change with the amount of the assigned load [7].

From equation (5) a width of the tunnel for fixed ρ, and m tending to infinity can be derived:

Lemma 2 *In the star interconnection*

$$\lim_{V\to\infty, m\to\infty} \frac{C_{max}^{inf}}{C_{max}^{sat}} = \frac{e^{\frac{1}{k}}}{(e^{\frac{1}{k}} - 1)(k+1)}.$$

where $k = \frac{A}{C(m+1)}$.

Proof. The above formula can be derived from (5) when assuming that $\frac{C}{A} = \rho = \frac{1}{k(m+1)}$, and after observing that $\lim_{x\to\infty}(1 + \frac{1}{x})^x = e$. □

We finish this section with an observation on the way of activating the processors in the solutions of LP1. Activation of the processors is ruled by two effects: memory limitations and schedule length minimization. When memory size on one processor is small then more processors must be used, though it is not as efficient as it would be in unlimited memory case. On the other hand, when computation times are short in relation to communication times then it is advantageous to use few processors. Hence, for A =1E-6 less machines were used for some given volume V than for A =1E-3.

2.3 Optimal activation sequence in star

In this section we address the problem of finding the optimal sequence of activating processors in a star network when memory buffers have limited sizes and communication delays include startup times. In the preceding discussion it was assumed that the sequence of activating processors is fixed. Here we relax this restriction and allow for selecting the best sequence of activating processors. This problem was raised in [13].

Let us denote by a binary variable x_{ij}, for $i, j = 1, \ldots, m$, the order of activating the processors. $x_{ij} = 1$ denotes that P_j is activated on ith position in the sequence. $x_{ij} = 0$ denotes the opposite situation. The problem of optimally activating the processors and distributing the load can be formulated as a mixed nonlinear programming problem:

MNP1:
minimize C_{max}
subject to:

$$C_{max} \geq \alpha_0 A_0 \tag{6}$$

$$C_{max} \geq \sum_{k=1}^{i} \sum_{j=1}^{m} x_{kj}(\alpha_j C_j + S_j) + \sum_{j=1}^{m} x_{ij} \alpha_j A_j \qquad i = 1, \ldots, m \tag{7}$$

$$1 \geq \sum_{i=1}^{m} x_{ij} \qquad j = 1, \ldots, m \tag{8}$$

$$1 \geq \sum_{j=1}^{m} x_{ij} \qquad i = 1, \ldots, m \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, m \tag{10}$$

$$V = \alpha_0 + \sum_{j=1}^{m}\sum_{i=1}^{m} x_{ij}\alpha_j \tag{11}$$

$$B_j \geq \alpha_j \geq 0 \qquad j = 0, \ldots, m \tag{12}$$

The above MNP1 formulation is a mixed problem because we have both binary variables x_{ij}, and continuous variables α_i, C_{max}. MNP1 is nonlinear because in equations (7), (11) we have multiplication of the variables. Equations (6) and (7) demand that all processors finish computing before C_{max}. In equations (7) term $\sum_{k=1}^{i}\sum_{j=1}^{m} x_{kj}(\alpha_j C_j + S_j)$ is the time of sending to the processor activated as i-th in the sequence, and $\sum_{j=1}^{m} x_{ij}A_j\alpha_j$ is the computation time of the i-th processor in the sequence. Equations (8)-(10) guarantee that the sequence of activating the processors is correct: each PE is activated at most once by (8), each position in the activation sequence is occupied by at most one processor by (9). Due to weak inequalities some processors may remain idle. Equation (11) guarantees processing of the whole load. Observe that some machines may be missing in the activation sequence, and $x_{ij} = 0$ for $i, j = 1, \ldots, m$ is a valid solution for constraints (8)-(10). Yet, it would not be a valid solution because appropriate communication time would not appear in equations (7). In order to prevent such a situation term $\sum_{i=1}^{m} x_{ij}\alpha_j$ in equation (11) guarantees that only the chunks sent to the processors are counted as really processed. Equations (12), guarantee that the load can be feasibly assigned to the processors. Let us apply the above formulation to solve Example 3 from [13].

Example. We have the same data as in the previous example: $m = 4, V = 100$, $A_0 = 1, A_1 = 5, A_2 = 4, A_3 = 3, A_4 = 2$, $B_0 = 10, B_1 = 20, B_2 = 45, B_3 = 15, B_4 = 30$, $C_1 = 4, C_2 = 3, C_3 = 2, C_4 = 1$, $S_i = 0$, for $i = 1, \ldots, 4$. Ms.Excel ver.7.0 managed to obtain the following solution to MNP1:

processor order	B_i	α_i	communication completion	computation completion
P_0	10	10	0	10
P_2	45	35.2941	105.8824	247.0588
P_4	30	30	135.8824	195.8824
P_1	20	12.3529	185.2941	247.0588
P_3	15	12.3529	210	247.0588

The sequence of activating the processors, according to the solver we used, is P_2, P_4, P_1, P_3. Schedule length is $C_{max} = 247.0588$, and it is better than

the one found in [13]. The reasons for this were given earlier. For the same instance with $V = 50$ (also considered in [13]) the following solution was obtained for MNP1:

processor order	B_i	α_i	communication completion	computation completion
P_0	10	10	0	10
P_4	30	24.277	24.277	72.832
P_3	15	9.711	43.699	72.832
P_2	45	4.162	56.185	72.832
P_1	20	1.850	63.584	72.832

Thus, the sequence found is P_4, P_3, P_2, P_1, and $C_{max} = 72.832$. □

The computational complexity of the general purpose mixed nonlinear solvers applied to MNP1 is high. These codes are capable of solving hard computational problems such as traveling salesman problem, quadratic assignment problem, and even more involved ones. It has been shown [8] that the problem of scheduling divisible loads in a star network with limited processor memory buffers and communication startup times is computationally hard (strictly **NP**-hard). According to the current state of knowledge [10] only algorithms with computational complexity growing exponentially with the size of the problem are known for this kind of problems. Thus, the codes finding optimal solutions of MNP1 have the worst-case execution time growing exponentially e.g. with the number of binary variables x_{ij}. As exponential functions increase explosively with the value of the argument, exponential-time algorithms are in practice restricted to small instances of the solved problem. This leaves space for heuristic methods which find good solution fast, and this is the advantage of IBS strategy proposed in [13].

3 Binomial trees

In this section we consider computer interconnections allowing for embedding a *binomial tree*. This is the case of hypercubes [11], meshes [3, 5, 11], multistage interconnections [5], and chains [5] with wormhole routing or circuit-switched routing. By a *binomial tree* of *degree* p we mean a graph which is acyclic and *each* node on levels $0, \ldots, i$ has p successors on level $i + 1$. Processors are the nodes of the binomial tree, and p is the number of processor ports which can communicate simultaneously. In Fig.4 the main idea of

scattering in the binomial tree is illustrated: First the load is sent very far, and in the subsequent steps it is redistributed in smaller area of the network. It is assumed that all processors are identical and are able to communicate and compute simultaneously. Also the communication links are identical. All processors on level i are activated simultaneously in i-th iteration of load scattering algorithm and will be called a *layer* i. We assume that the number of layers is h, and 0 is the layer of the originator. A great advantage of binomial trees is that the number of active processors grows exponentially with the number of layers. In a binomial tree of degree p layer $i > 0$ has $p(1+p)^{i-1}$ processors.

insert Fig.4 here.

Two different ways of activating the layers have been proposed: The natural order of the layers called NEAREST LAYER FIRST (NLF) [3], and the order of decreasing number of processors in the layer called LARGEST LAYER FIRST (LLF) [11]. Fig.5 and Fig.6 show diagrams of communication and computation for strategies NLF, LLF, respectively. In NLF (cf.Fig.5) layers receive the load for themselves and for their descendants in the binomial tree. Immediately after receiving the load processors start processing their share of the load, while the rest is sent to the following layers. Thus, processors start computing in the order of the layer number. In LLF strategy (cf. Fig.6) the layers start computing in the order $h, h-1, \ldots, 1$. To activate some layer i the intermediate layers $0, \ldots, i-1$ transfer the load to layer i, but do not compute. In the two following sections we study NLF and LLF strategies.

insert Fig.5 and Fig.6 here

3.1 Nearest Layer First

The problem of determining optimal distribution of load V in binomial tree of degree p under NLF strategy can be solved by the following linear program:

LP NLF:
minimize C_{max}
subject to:

$$\alpha_i A + \sum_{j=1}^{i}(S + C\alpha_j + Cp\sum_{k=j+1}^{h}(p+1)^{k-j-1}\alpha_k) \leq C_{max} \quad i = 0, \ldots, h \quad (13)$$

$$\alpha_0 + p\sum_{i=1}^{h}(1+p)^{i-1}\alpha_i = V \quad (14)$$

$$B \geq \alpha_i \geq 0 \quad i = 0, \ldots, h \quad (15)$$

In LP NLF α_i, for $(i = 0, \ldots, h)$, are variables denoting the amount of load assigned to each processor in layer i. In equations (13) term $\sum_{j=1}^{i}(S + C\alpha_j + Cp\sum_{k=j+1}^{h}(p+1)^{k-j-1}\alpha_k)$ is the communication time spent until activating layer i. Note that layers $1, \ldots, i$ receive load not only for themselves but also the load to be redistributed to layers $i+1, \ldots, h$. Equations (13) ensure that all layers stop computing before the end of the schedule C_{max}. By equation (14) all the load is processed, and by (15) assignments of the load can be accommodated in the memory buffers of the processors. LP NLF is formulated with the assumption that all h layers are working. However, it may happen that fewer layers will process all the load. In such a case some layers are not assigned any load, but still communication startup time appears in inequalities (13). This case is easy to recognize: some layers receive 0 load, and decreasing h reduces C_{max}. Hence, less layers should be used. By binary search over the admissible numbers of layers the appropriate value of h can be found.

Now, we will study performance of NLF algorithm in a binomial tree of degree 4, and 7 layers ($m = 78125$ processors). This tree can be embedded in a 2-dimensional toroidal mesh [3]. We modeled a system with $A = C =$1E-6, $S =$1E-3, and memory sizes from $B = 10$ to $B=$1E9. The schedule lengths C_{max} vs. size of the problem is depicted in Fig.7. Line "inf" represents a system with unlimited memory. Line "sat" represents a system with total memory size equal V. Thus, in saturated case each processor has memory buffer of size $B = \frac{V}{(p+1)^h}$. Schedule lengths for "sat" and "inf" cases are very close to each other in the case of big volumes V. As it was in the case of star topology, the two lines form a tunnel in which plots for particular memory sizes are located. In Fig.8 only "sat" and "inf" cases are depicted for various processing rates A. The behavior is similar to the star topology: For big load volumes V the two lines are parallel. As A increases (e.g. because the application is computationally intensive) the "inf" line moves up until it overlaps with line "sat". Now we are going to calculate the relative width of the tunnel for big V and A.

insert Fig.7 and Fig.8 here

Lemma 3 *Under NLF strategy in binomial tree* $\lim_{V\to\infty,\rho\to 0} \frac{C_{max}^{inf}}{C_{max}^{sat}} = 1$.

Proof. We will give a formula for the ratio of schedule length C_{max}^{sat} in the saturated case and C_{max}^{inf} in the unlimited memory case. In the saturated case all processors are assigned the same load equal to the buffers

size $B = \alpha_i = \frac{V}{(p+1)^h}$, for $i = 0, \ldots, h$. C^{sat}_{max} is determined by the duration of all communications plus processing on layer h (cf. Fig.5). Thus, $C^{sat}_{max} = hS + \frac{CV}{m}\sum_{j=1}^{h}(1 + p\sum_{k=j+1}^{h}(p+1)^{k-j-1}) + \frac{AV}{m}$, where $m = (p+1)^h$ is total number of processors. This formula can be reduced to $C^{sat}_{max} = hS + \frac{VA}{m} + \frac{VC((p+1)^h-1)}{mp}$. The formula expressing C^{inf}_{max} has been given in [11]: $C^{inf}_{max} = A(V + \frac{\sigma}{p+\rho})\frac{p(p+\rho+1)^{-h}+\rho}{p+\rho} + \frac{\sigma(hp-1)}{p+\rho}$. Hence,

$$\frac{C^{inf}_{max}}{C^{sat}_{max}} = \frac{A(V + \frac{\sigma}{p+\rho})\frac{p(p+\rho+1)^{-h}+\rho}{p+\rho} + \frac{\sigma(hp-1)}{p+\rho}}{hS + \frac{VA}{m} + \frac{VC((p+1)^h-1)}{mp}} \tag{16}$$

It can be verified that $\lim_{V\to\infty,\rho\to 0}\frac{C^{inf}_{max}}{C^{sat}_{max}} = \lim_{\rho\to 0}\frac{\frac{p(p+\rho+1)^{-h}+\rho}{p+\rho}}{\frac{1}{m}+\frac{\rho((p+1)^h-1)}{mp}} = \frac{\frac{p(p+1)^{-h}}{p}}{\frac{1}{(p+1)^h}} = 1$. $\square$.

Thus, in binomial trees spanned in homogeneous computer networks, under NLF strategy, when size V of the problem is big, and the problem is computationally intensive ($\rho \to 0$), the influence of the limited memory is insignificant.

3.2 Largest Layer First

In this section we consider a different strategy of activating the layers. According to LLF strategy h is the first layer, and 1 is the last layer activated. We will give a linear program solving the problem of distributing the load optimally in binomial tree under LLF. Then, we compare the results of modeling performance of systems with LLF and NLF scattering methods.

Before formulating a linear program for LLF strategy consider duration of the communication from the originator to layer i. There are $p(p+1)^{i-1}$ processors in layer i. First the originator sends over each of its p communication links $p(p+1)^{i-2}\alpha_i$ load units to layer 1. The remaining load $p(p+1)^{i-2}\alpha_i$ will be sent to layer i via direct successors of the originator in layers $2, \ldots, i$ (cf. Fig.6). Each processor in layer $j < i-1$ sends $p(p+1)^{i-j-2}\alpha_i$ units of data to layer $j+1$. The remaining $p(p+1)^{i-j-2}\alpha_i$ is sent from layer j to layer i via j's direct binomial tree successors in layers $j+1, \ldots, i$. Finally, layers $0, \ldots, i-1$ send α_i load units to 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 $Si + C\alpha_i(1 + p\sum_{j=0}^{i-2}(p+1)^{i-j-2}) = Si + C\alpha_i(p+1)^{i-1}$. The problem can be solved by a linear program:

LP LLF:
minimize C_{max}
subject to:

$$\alpha_i A + \sum_{j=i}^{h}(Sj + C(p+1)^{j-1}\alpha_j) \leq C_{max} \quad i = 0, \ldots, h \tag{17}$$

$$\alpha_0 + p\sum_{i=1}^{h}(1+p)^{i-1}\alpha_i = \quad V \tag{18}$$

$$B \geq \alpha_i \geq \quad 0 \quad i = 0, \ldots, h \tag{19}$$

In LP LLF equations (17) guarantee that all processors finish computing before the end of the schedule. By equation (18) all the load is processed, and by equations (19) all processors are able to accommodate the assigned load. It may happen that the assumed number of layers h is too big. Reduction of h results in shorter schedule. Yet, the problem becomes more involved because we send to the larger layer first. A solution of LP LLF may activate layers non-continuously. Some layers may receive load for processing, while the remaining layers would still contribute startup time S in inequalities (17), though they receive nothing. We observed that in the solutions of LP LLF layers with higher index (i.e. with more processors) are assigned some load first in consecutive manner (without gaps). Thus, for given h it is possible to check LP LLF only with the last layers $h, \ldots, h-j$. The best number of utilized layers can be found by binary search over the range of h. In the worst case this procedure must be repeated for various values of h. Hence, the total number of calls to LP LLF needed to find optimum distribution of the load is $O(h \log h)$, where $h = \log_{p+1} m$, and m is the number of available processors. In the following we prove that this strategy leads to optimal solutions because it is alwys profitable to activate layer $i+1$ (with more processors) before layer i.

Lemma 4 *Let C_{max}^i denote schedule length for some volume V assigned to layer i but not to layer $i+1$, and C_{max}^{i+1}, when V is assigned to $i+1$, but not to i. Then, $C_{max}^i > C_{max}^{i+1}$.*

Proof. Let us calculate length of the schedule when layer i is used to process V, but layer $i+1$ is not exploited. Layer i has $p(p+1)^{i-1}$ processors.

Thus, $C^i_{max} = S + C(p+1)^{i-1}\frac{V}{p(p+1)^{i-1}} + \frac{AV}{p(p+1)^{i-1}} = S + \frac{CV}{p} + \frac{AV}{p(p+1)^{i-1}}$. Analogously, $C^{i+1}_{max} = S + \frac{CV}{p} + \frac{AV}{p(p+1)^i}$. Hence, $C^i_{max} > C^{i+1}_{max}$ for $i > 0$. $\Box$.

By the above lemma it is profitable to activate the layers consecutively from the layer with more processors to the layer with less processors (without gaps in between).

We studied the performance of a computer network with embedded binomial tree under LLF strategy. In order to find the shortest processing time over various orders of activating layers we used the result of Lemma 4, and increased the number of active layers from the last one to the first. The solution with the smallest schedule length was selected. In general, the behavior of C_{max} under changing V, B, A is very similar to the case of NLF behavior. Schedule lengths in the saturated system and in the system with unlimited memory is presented in Fig.9. Also here a tunnel between "inf" and "sat" cases can be observed. In the following lemma we will show that for big volumes and computation-intensive applications the relative difference is very small.

insert Fig.9 here

Lemma 5 *Under LLF strategy in binomial tree* $\lim_{V\to\infty,\rho\to 0} \frac{C^{inf}_{max}}{C^{sat}_{max}} = 1$.

Proof. Schedule length in the saturated case is $C^{sat}_{max} = \sum_{j=1}^{h}(Sj + \frac{VC}{m}(p+1)^{j-1}) + \frac{AV}{m} = S(h+1)h/2 + \frac{V}{m}(C\frac{m-1}{p} + A)$, where $m = (p+1)^h$ is the total number of processors. The formula for C^{inf}_{max} has been given in [11]:

$$C^{inf}_{max} = \frac{AV}{M} + \frac{A\sigma p}{\rho M}\sum_{j=1}^{h}\frac{c_{\pi(j)}-1}{P^{\pi}_j}(\sum_{i=1}^{j}(h-i+1)P^{\pi}_{i-1}),$$

where: $M = 1 + \frac{p}{\rho}(1 - \frac{1}{P^{\pi}_h})$, $c_{\pi(j)} = 1 + \rho(p+1)^{h-j}, c_{\pi(0)} = 1$, and $P^{\pi}_j = \prod_{i=0}^{j} c_{\pi(i)}$. Thus in LLF strategy,

$$\frac{C^{inf}_{max}}{C^{sat}_{max}} = \frac{\frac{AV}{M} + \frac{A\sigma p}{\rho M}\sum_{j=1}^{h}\frac{c_{\pi(j)}-1}{P^{\pi}_j}(\sum_{i=1}^{j}(h-i+1)P^{\pi}_{i-1})}{S(h+1)h/2 + \frac{V}{m}(C\frac{m-1}{p} + A)} \qquad (20)$$

When the volume of load is big and the application is computationally intensive, we have:

$$\lim_{V\to\infty,\rho\to 0}\frac{C^{inf}_{max}}{C^{sat}_{max}} = \lim_{V\to\infty,\rho\to 0}\frac{\frac{VA}{1+\frac{p}{\rho}(1-\frac{1}{P^{\pi}_h})}}{\frac{VA}{m}(\rho\frac{m-1}{p}+1)} =$$

$$\lim_{\rho\to 0}\frac{m}{(1+\frac{p}{\rho}(1-\frac{1}{P_h^\pi}))(\rho\frac{m-1}{p}+1)}=\lim_{\rho\to 0}\frac{m}{1+\frac{p}{\rho}(1-\frac{1}{P_h^\pi})}\stackrel{H}{=}1$$

since $\lim_{\rho\to 0}P_h^\pi=1$, we applied de l'Hospital rule and obtained:
$\lim_{\rho\to 0}\frac{p(1-\frac{1}{P_h^\pi})}{\rho}\stackrel{H}{=}\lim_{\rho\to 0}\frac{p(\sum_{i=0}^{h-1}(p+1)^i+2\rho((p+1)^3+...+3\rho^2((p+1)^6+...)}{(P_h^\pi)^2}=m-1.$ □

A similar conclusion can be drawn as in the star interconnection and under NLF strategy. In binomial trees spanned in homogeneous computer networks, under LLF strategy, when size V of the problem is big, and the problem is computationally intensive ($\rho\to 0$), then the relative influence of the limited memory on the processing time is negligible.

In our modeling of LLF strategy we observed several interesting facts:

- It was shown in [11] that LLF strategy is optimal in a system with unlimited memory. In the saturated system it is not, because LLF has greater number of communication startups than NLF. This communication overhead is not compensated for by a better distribution of the load and shorter computation time.

- In the earlier publications on divisible load theory [3, 11] systems with unlimited memory were considered (i.e. case 'inf'). LP formulations had more restricted form and e.g. inequality (17) had form of equation. As a result in LLF strategy, when volume V is small and available memory is not restricted, only few layers can be activated (even if we have many layers) to satisfy the classical version of LP LLF. Thus, small increase of V may be satisfactory to activate more layers and in this way reduce schedule length. This is demonstrated in the example presented below. Consequently, with V increasing C_{max}^{inf} may decrease. This is evident in Fig.9 where lines for 'inf' case and A = 1E-3, and $A=1$ are not smooth for small V.

- The above irregular behavior was not observed in the LP NLF model.

- We observed that for $A\approx C$ only the last layer was populated. When $A\gg C$ the layers closer to the originator were populated more often.

- None of NLF, LLF strategies dominates the other in all cases. However, for big volumes and LLF shorter schedules were obtained.

Example. Consider a system with $h=2, p=4, A$ = 1E-3, C = 1E-6, S =1E-3, $V=20$. In the system with unlimited memory [11] equations

describing distribution of the load have positive solution only for one layer (5 processors altogether). Schedule length in this case is $C_{max}^{sat} \approx 0.0048$. However, when $V = 24$ all 25 processors can be activated, and $C_{max}^{sat} \approx 0.003$. Using LP LLF, and only the last layer we obtain $C_{max} \approx 0.0029$ in the first, and $C_{max} \approx 0.0031$, in the second case. □

4 Conclusions

In this paper we analyzed divisible load distribution in systems with limited memory. Interconnection topologies of a star, and binomial tree under two different distribution strategies were studied. It appeared that in homogeneous systems and big computationally intensive applications mainly the processor and communication speeds limit performance of the systems. The problem of optimal order of activating the processors in the heterogeneous star has been addressed.

In our discussion we assumed that only the size of the receiver memory is restricting distribution of the load. The communication system is not limiting the size of the message. This may not be the case in practice. Therefore, a system with limited communication system capacity can be a subject of further work.

References

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

[2] Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos CA (1996)

[3] J.Błażewicz, M.Drozdowski, F.Guinand, D.Trystram, Scheduling a divisible task in a 2-dimensional mesh, *Discrete Applied Mathematics* 94(1-3), 1999 (June), 35-50.

[4] Błażewicz, J., Drozdowski, M., Markiewicz, M.: Divisible task scheduling - concept and verification. Parallel Computing **25** (1999) 87–98

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

[6] M.Drozdowski, W.Głazek, Scheduling divisible loads in a three-dimensional mesh of processors, *Parallel Computing* 25 (1999) 381-404.

[7] 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, LNCS 1900*, Springer-Verlag, (2000), 311-319

[8] M.Drozdowski, P.Wolniewicz, On the complexity of divisible job scheduling with limited memory buffers, Technical Report RA-001/2001, Institute of Computing Science, Poznań University of Technology, 2001.

[9] R.Fourer, Linear Programming Frequently Asked Questions, 2000, `http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html`,

[10] M.R.Garey, D.S.Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[11] W.Głazek, *Algorithms for Scheduling Divisible Tasks on Parallel Machines*, 1998, Ph.D. Thesis, Dept. of Electronics, Telecommunication and Computer Science, Technical University of Gdańsk, Poland.

[12] J.Gondzio, T.Terlaky, A computational view of interior-point methods for linear programming, in: Beasley (ed.), *Advances in Linear and Integer Programming*, Oxford University Press, Oxford 1996, 107-146.

[13] X.Li, V.Bharadvaj, C.C.Ko, Optimal divisible task scheduling on single-level tree networks with buffer constraints, *IEEE Trans. on Aerospace and Electronic Systems* 36, No. 4, 2000, 1298-1308.

[14] I.J.Lustig, R.E.Marsten, D.F.Shanno, Interior point methods for linear programming: Computational state of the art, *ORSA J. on Computing* 6, No. 1, 1994, 1-14.

[15] G.L.Nemhauser, L.A.Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.

Appendix: Notation summary

A - processing rate in a homogeneous system,
A_i - processing rate of processor P_i in heterogeneous system,
B_i - size of memory buffer of processor P_i in heterogeneous system,
C - transfer rate in a homogeneous system,
C_j - transfer rate of link j in heterogeneous system,
C_{max} - schedule length,
C_{max}^{inf} - schedule length in a system with unlimited memory,
C_{max}^{sat} - schedule length in a system with total memory size equal to V,
h - number of layers in a binomial tree
m - number of processors,
p - degree of a node in a binomial tree,
$\rho = \frac{C}{A}$ - defined only for homogeneous systems,
$\sigma = \frac{S}{A}$ - defined only for homogeneous systems,
S - communication startup time in a homogeneous system,
S_j - communication startup time for link j in heterogeneous system,
V - total size of the load.

Figure captions

Fig. 1 Communication and computation in a star.

Fig. 2 Schedule length for a star network for various problem and memory sizes.

Fig. 3 Schedule length for a star network with unlimited and saturated memory for various processing rates.

Fig. 4 The idea of scattering in a binomial tree. $p = 2$.

Fig. 5 Communication and computation in a binomial tree under NLF strategy.

Fig. 6 Communication and computation in a binomial tree under LLF strategy.

Fig. 7 Schedule length in a binomial tree under NLF strategy for various problem and memory sizes.

Fig. 8 Schedule length in a binomial tree under NLF strategy with unlimited and saturated memory

Fig. 9 Schedule length in a binomial tree under LLF strategy with unlimited and saturated memory

Fig.1

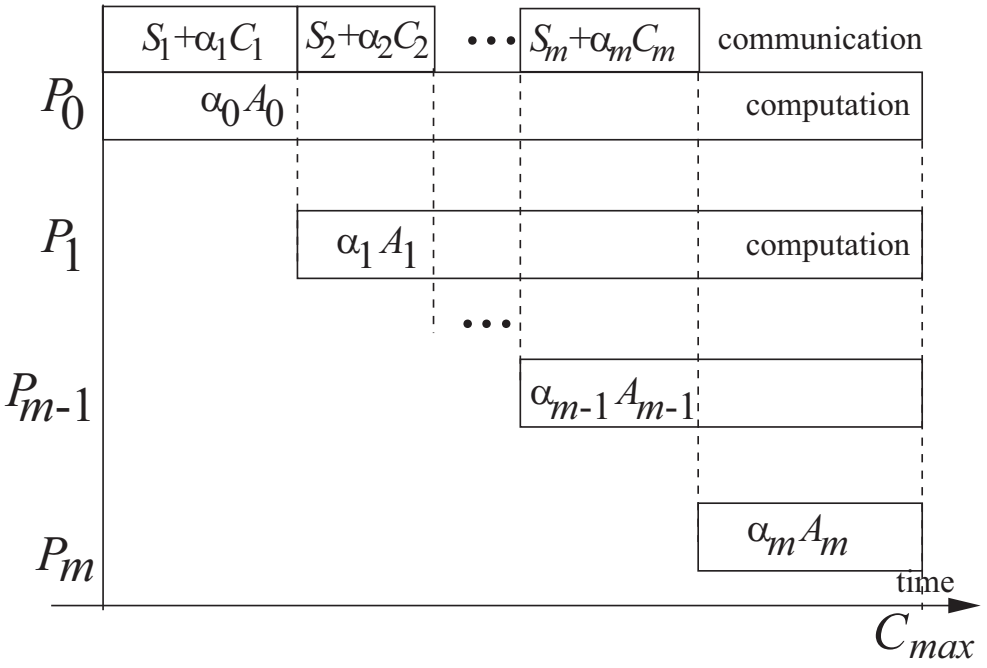

$S_1+\alpha_1 C_1$
$S_2+\alpha_2 C_2$
$S_m+\alpha_m C_m$
communication
P_0
$\alpha_0 A_0$
computation
P_1
$\alpha_1 A_1$
computation
P_{m-1}
$\alpha_{m-1} A_{m-1}$
P_m
$\alpha_m A_m$
time
C_{max}

Fig.2

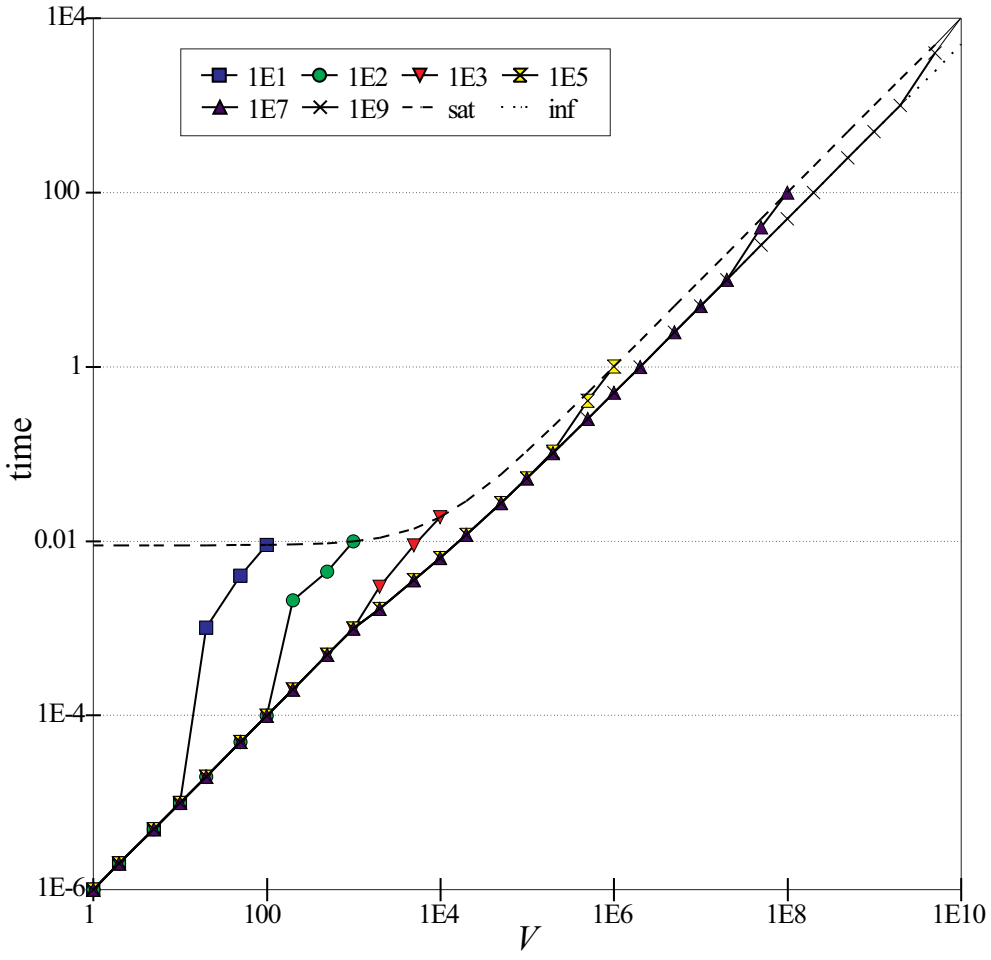

1E4
100
1
0.01
1E-4
1E-6
time
1E1
1E2
1E3
1E5
1E7
1E9
sat
inf
1
100
1E4
1E6
1E8
1E10
V

Fig.3

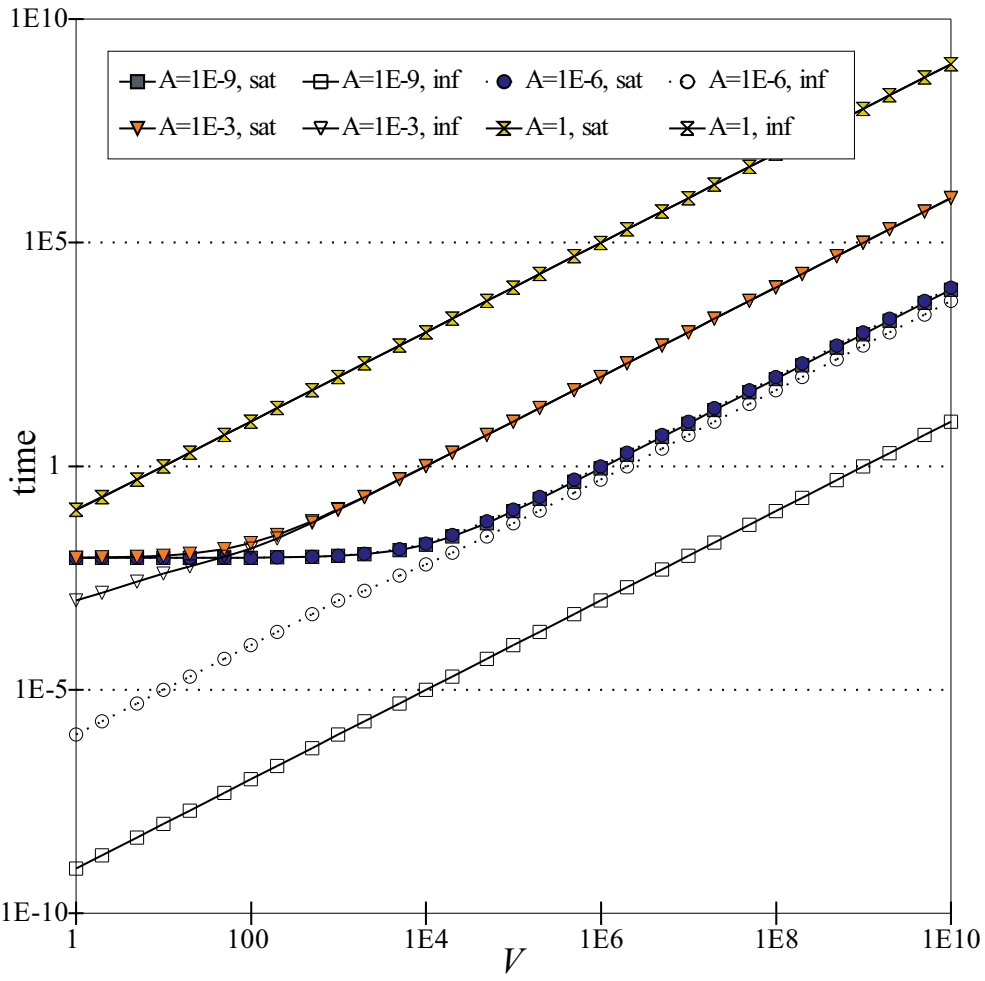

A=1E-9, sat
A=1E-9, inf
A=1E-6, sat
A=1E-6, inf
A=1E-3, sat
A=1E-3, inf
A=1, sat
A=1, inf
1E10
1E5
1
1E-5
1E-10
time
1
100
1E4
1E6
1E8
1E10
V

Fig.4

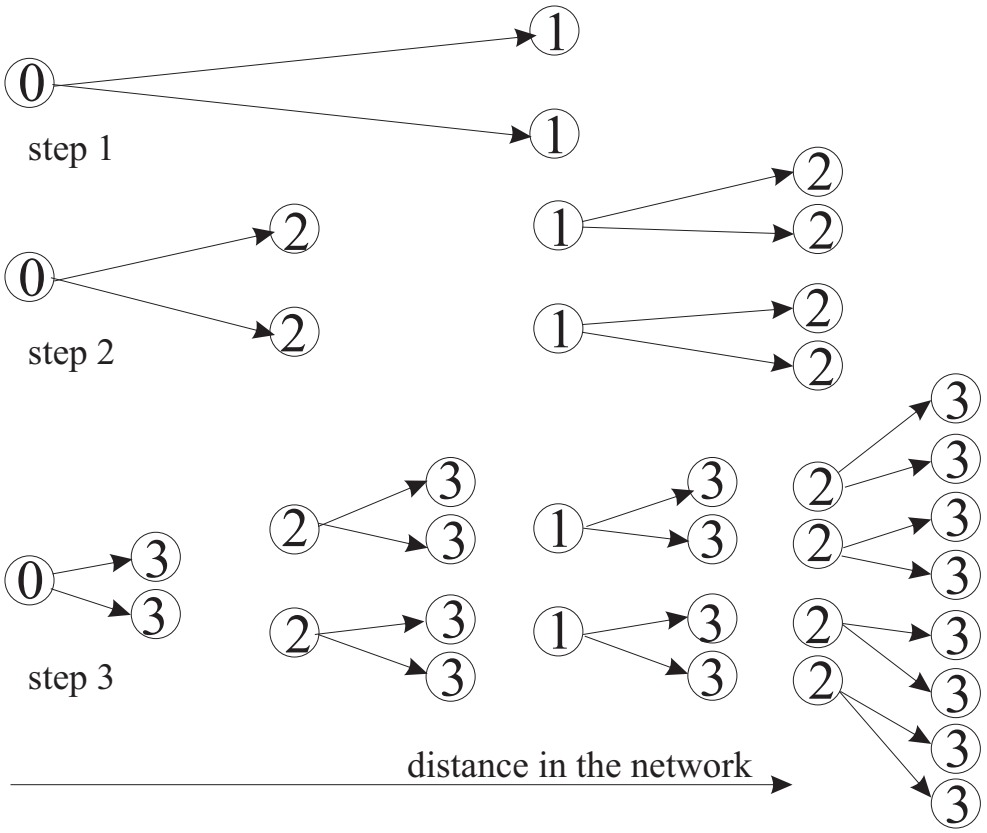

0
1
1
step 1
0
2
2
step 2
1
2
2
1
2
2
0
3
3
step 3
2
3
3
2
3
3
1
3
3
1
3
3
2
3
3
2
3
3
2
3
3
2
3
3
distance in the network

Fig.5

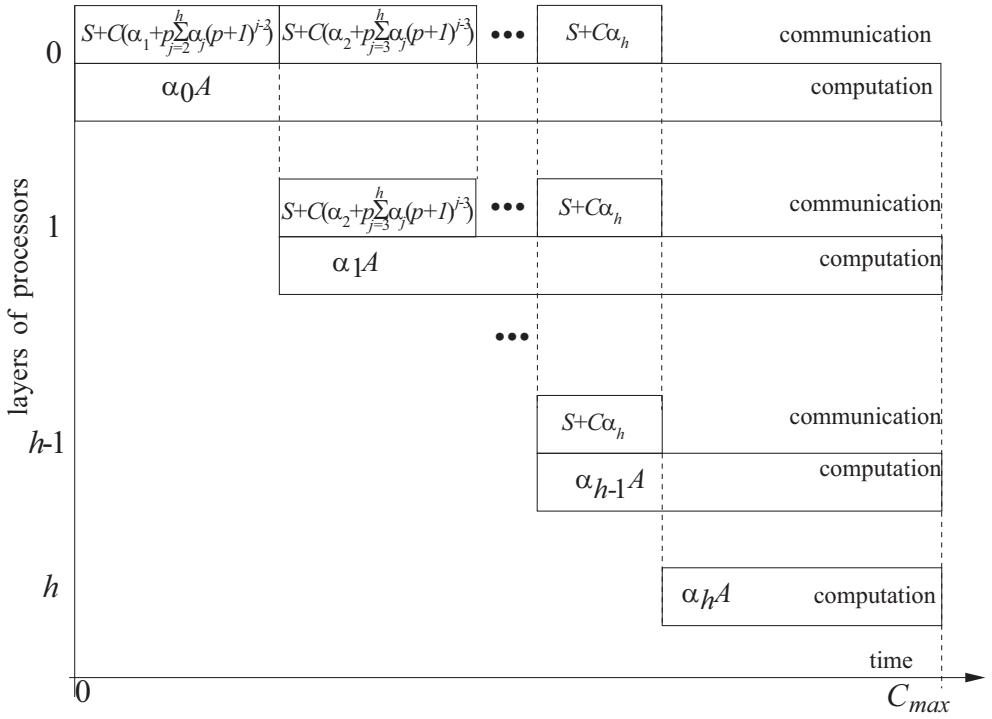

layers of processors
0
1
h-1
h
communication
computation
time
0
C_{max}

Fig.6

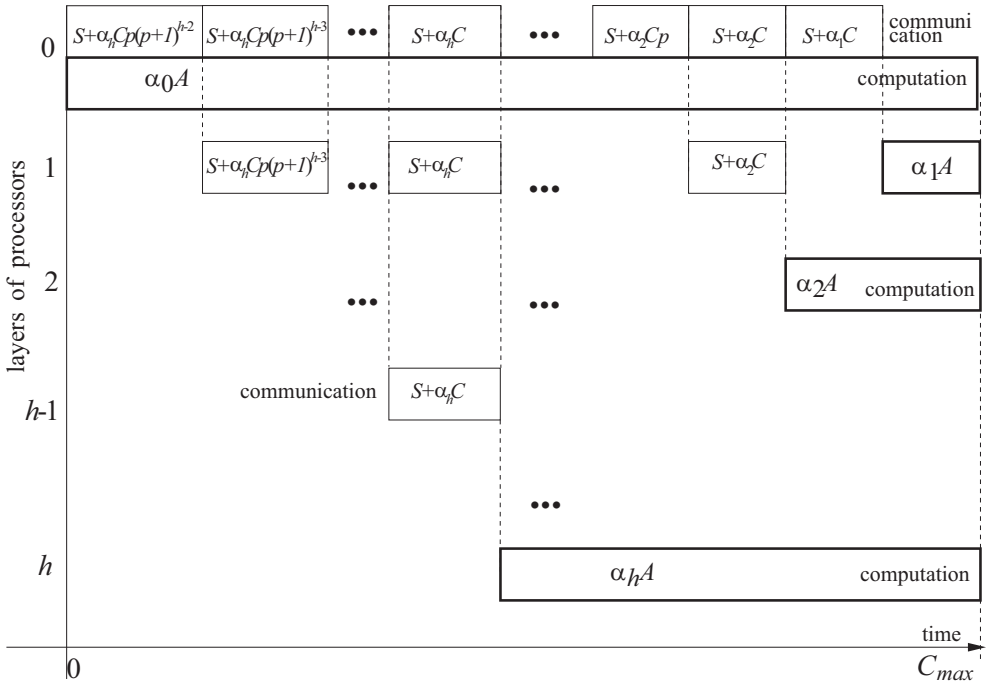

0
1
2
h-1
h
layers of processors
S+α_hCp(p+1)^(h-2)
S+α_hCp(p+1)^(h-3)
S+α_hC
S+α_2Cp
S+α_2C
S+α_1C
communi
cation
α_0A
computation
α_1A
α_2A
communication
α_hA
time
0
C_max

Fig.7

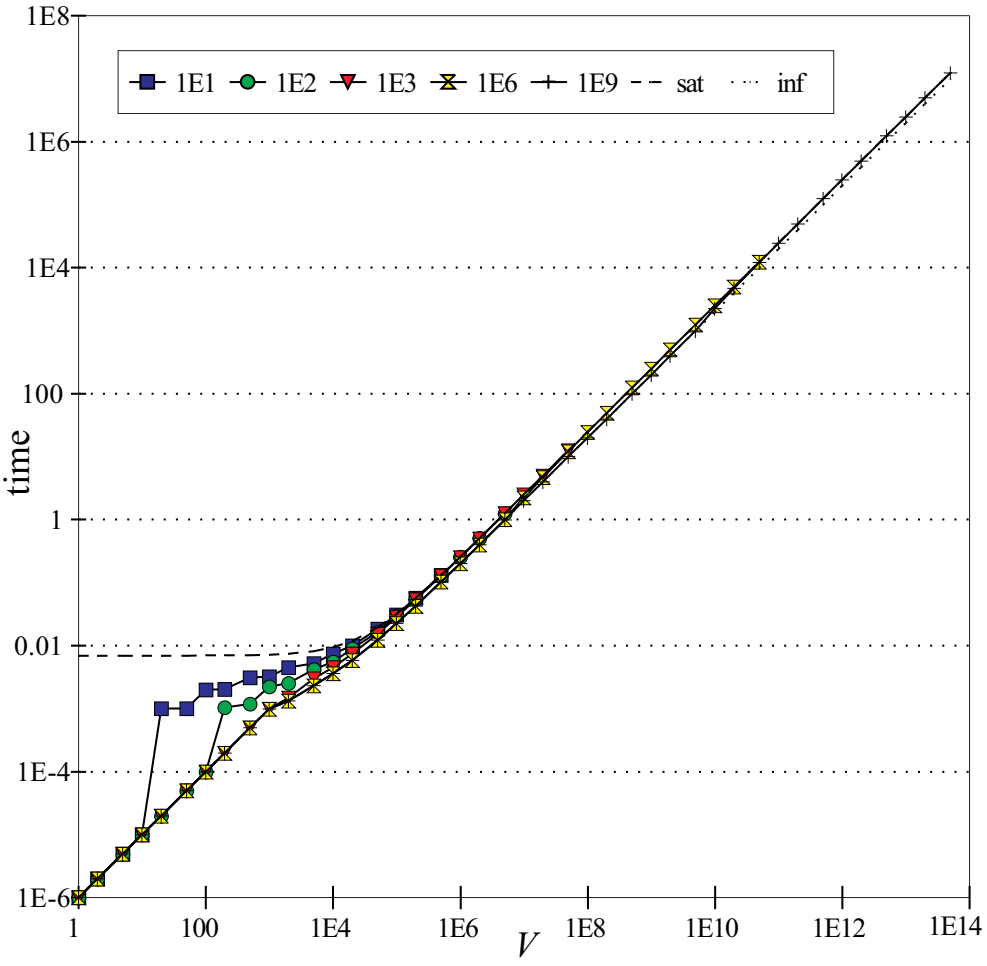

1E8
1E6
1E4
100
1
0.01
1E-4
1E-6
time
1E1
1E2
1E3
1E6
1E9
sat
inf
1
100
1E4
1E6
1E8
1E10
1E12
1E14
V

Fig.8

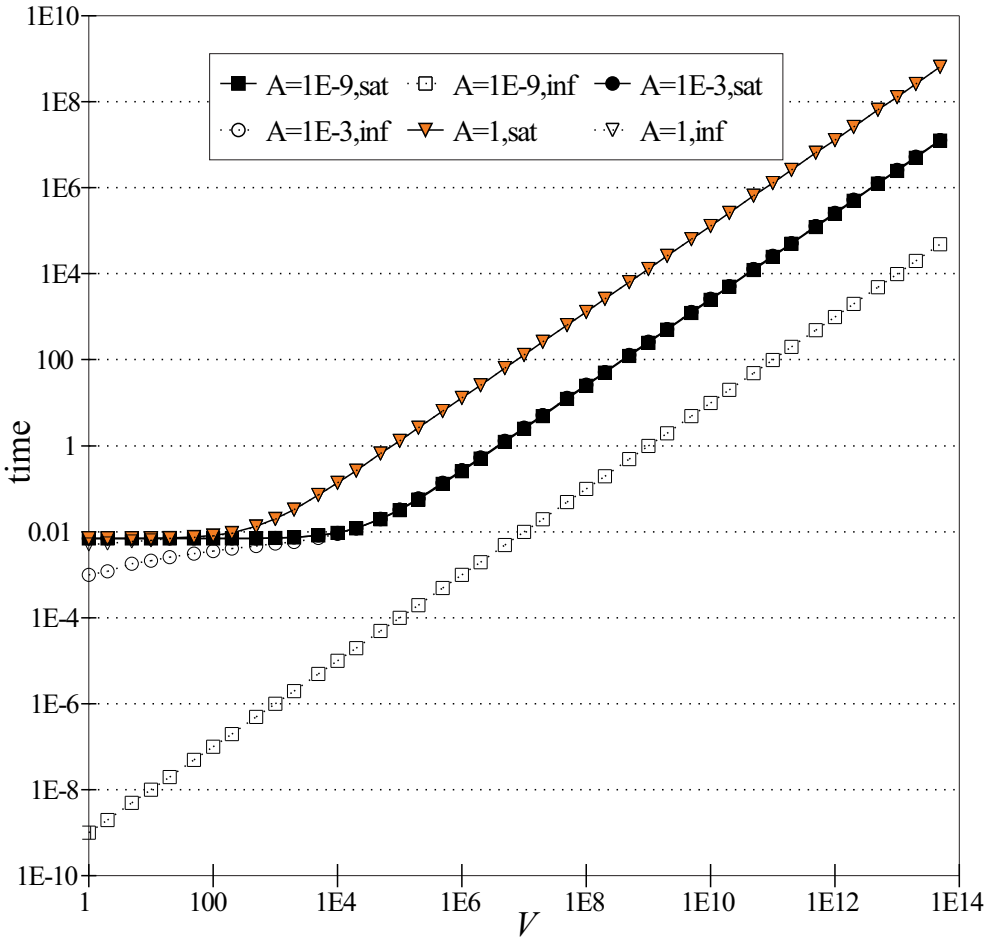
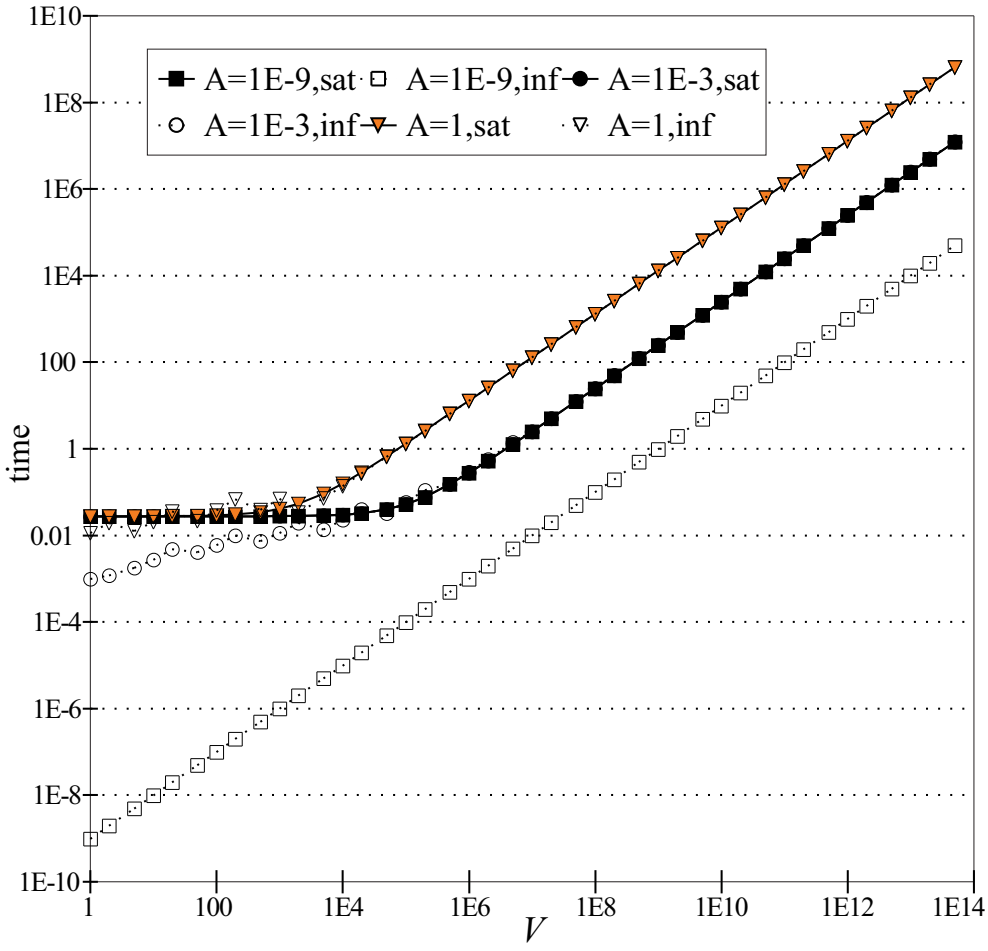

A=1E-9,sat
A=1E-9,inf
A=1E-3,sat
A=1E-3,inf
A=1,sat
A=1,inf
time
V
1E10
1E8
1E6
1E4
100
1
0.01
1E-4
1E-6
1E-8
1E-10
1
100
1E4
1E6
1E8
1E10
1E12
1E14

Fig.9


A=1E-9,sat
A=1E-9,inf
A=1E-3,sat
A=1E-3,inf
A=1,sat
A=1,inf
1E10
1E8
1E6
1E4
100
1
0.01
1E-4
1E-6
1E-8
1E-10
time
1
100
1E4
1E6
1E8
1E10
1E12
1E14
V

Authors biographies

Maciej Drozdowski Received M.Sc. in control engieering 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. Member of IEEE Computer.

PawełWolniewicz Graduated from Poznan University of Technology and received M.Sc. in computer science in 1997. Currently, he works for Poznan Supercomputing and Networking Centre, Poznan, Poland. He is Ph. D. Candidate in departament of computer science Poznan University of Technology. His research interests include metacomputers, distributed environments and scheduling