On single-walk parallelization of the job shop problem solving algorithms

Wojciech Bożejko*

Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland

ARTICLE INFO
Available online 18 November 2011

Keywords:
Job shop problem
Parallel programming
PRAM

ABSTRACT
New parallel objective function determination methods for the job shop scheduling problem are proposed in this paper, considering makespan and the sum of jobs execution times criteria, however, the methods proposed can be applied also to another popular objective functions such as jobs tardiness or flow time. Parallel Random Access Machine (PRAM) model is applied for the theoretical analysis of algorithm efficiency. The methods need a fine-grained parallelization, therefore the approach proposed is especially devoted to parallel computing systems with fast shared memory (e.g. GPGPU, General-Purpose computing on Graphics Processing Units).

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

In this paper we are proposing new parallel objective function determination methods for the job shop scheduling problems. We are considering an algorithm (e.g. metaheuristic: tabu search, scatter search, etc.) which employs a single process to guide the search. The thread performs in a cyclic way (iteratively) two leading tasks:

(A) objective function evaluation for a single solution or a set of solutions,
(B) management, e.g. solution filtering and selection, collection of history, updating.

Part (B) takes statistically 1–3% total iteration time, thus its acceleration is useless. Part (A) can be accelerated in a multi-thread environment in various manners—our aim is to find either cost-optimal method or non-optimal one in terms of cost while offering the shortest running time. It is noteworthy to observe that if Part (B) takes β percentage of the 1-processor algorithm and if it is not parallelizable, the speedup of the parallel algorithm for any number of processors p cannot be greater than 1/β (according to Amdahl's law). In practice, if Part (B) takes 2% of the total execution time, the speedup can achieve at most the value of 50.

There are only a few papers dealing with parallel algorithms for the job shop scheduling problem, which has a relatively simple formulation and which is very interesting from the theoretical and practical points of view—many real manufacturing systems can be modeled just as the job shop, i.e. in construction projects, chemistry, electronics, etc. It is also considered as an indicator of practical efficiency of the new scheduling algorithms (see [7]). Bożejko et al. [2] proposed a single-walk parallelization of the simulated annealing metaheuristic for the job shop problem. Steinböhler et al. [14] described the method of parallel objective function determination in \(O(\log^2 n)\) time on \(O(n^d)\) processors, where \(n\) is the number of all operations. Bożejko [1] considered a method of parallel objective function calculation for the flow shop problem, which constitutes a special case of the job shop problem. Here we are proposing a more efficient version of the algorithm developed by Steinböhler et al., which works in \(O(\log^3 n)\) time on \(O(n^d)\) processors. Besides, we show a cost-optimal parallelization which takes a time \(O(d)\), where \(d\) is the number of layers in the topological sorted graph representing a solution. Finally, we prove that this method has a constant \(O(1)\) time complexity if we know the value of the upper bound of the objective function value.

Algorithms proposed in this paper are placed in proofs of theorems – they are constructive ones. The main result – the algorithm of the objective function value determination – is placed in the proof of the Theorem 3, and its practical aspects are described in Section 6.

2. Parallel computations model

We make a complexity analysis of the objective function determination algorithms for their implementations on Parallel Random Access Machine (PRAM model). A PRAM consists of many cooperating processors, each being a random access machine (RAM), commonly used in theoretical computer science. Each processor can make local calculations, e.g. additions, subtractions, shifts, conditional and unconditional jumps and indirect addressing. All the processors in the PRAM model are synchronized and have access to a shared global memory in constant time \(O(1)\).
There is also no limit on the number of processors in the machine, and any memory cell is uniformly accessible from any processor. The amount of shared memory in the system is not limitable. We make use of two kinds of PRAMs: CREW (Concurrent Read Exclusive Write) where processors can read from the same memory cell concurrently, and EREW (Exclusive Read Exclusive Write) where the concurrency of reading is forbidden. Both models resemble the GPU programming model. We take advantage of the following well-known facts for the PRAM parallel computer model [5]:

**Fact 1.** Sequence of prefix sums \((y_1,y_2,\ldots,y_n)\) of input sequence \((x_1,x_2,\ldots,x_n)\) such that \(y_k = y_{k-1} + x_k = x_1 + x_2 + \cdots + x_k\) for \(k = 2,3,\ldots,n\) where \(y_1 = x_1\) can be calculated in \(O(\log n)\) time on the EREW PRAM machine with \(O(n/\log n)\) processors.

According to the above statement we can assume that the sum of \(n\) values can be calculated in \(O(\log n)\) time on \(O(n/\log n)\) – processors EREW PRAMs.

**Fact 2.** The minimal and the maximal value of input sequence \((x_1,x_2,\ldots,x_n)\) can be determined in \(O(\log n)\) time on the EREW PRAM machine with \(O(n/\log n)\) processors.

If we do not have enough large number of processors, we can use the following fact to keep the same cost [5]:

**Fact 3.** If the algorithm \(A\) works on \(p\) – processors PRAM in \(t\) time, then for every \(p' < p\) there exists an algorithm \(A'\) for the same problem which works on \(p'\) – processors PRAM in \(O(p't)\) time.

The speedup and cost of a parallel algorithm as compared to a sequential algorithm are two commonly used criteria to evaluate parallel algorithms. Let us consider a problem \(\Phi\) and a parallel algorithm \(A_{par}\). Let us define \(T_{\text{par}}(p)\) – time of calculations of the algorithm \(A_{par}\) which is necessary to solve the problem \(\Phi\) on the machine using \(p\) processors. Let \(T_{\text{seq}}\) be a time of calculations of the best (the fastest) known sequential algorithm \(A_{seq}\) which solves the same problem \(\Phi\) on the sequential machine with the processor identical to processors of the parallel machine. We define the speedup \(S_{\text{par}}(p) = T_{\text{seq}}(p)/T_{\text{par}}(p)\). The cost \(C_{\text{par}}(p)\) of solving a problem by using an algorithm \(A_{par}\) in a \(p\) – processors parallel machine is defined as \(C_{\text{par}}(p) = \alpha \cdot T_{\text{par}}(p)\). This cost is the aggregated time that the processors require for solving the problem.

For sequential algorithms the problem solving time by the fastest known algorithm using one processor constitutes also its cost. We can state that a parallel algorithm is cost-optimal if its executing cost in a parallel system is linearly proportional to the execution time of the fastest known sequential algorithm on one processor.

### 3. The job shop problem

Let us consider a set of jobs \(J = \{1,2,\ldots,n\}\), a set of machines \(M = \{1,2,\ldots,m\}\) and a set of operations \(O = \{1,2,\ldots,o\}\). The set \(O\) is decomposed into subsets connected with jobs. A job \(j\) consists of a sequence of \(o_j\) operations indexed consecutively by \((l_{j-1}+1,l_{j-1}+2,\ldots,l_j)\) which have to be executed in this order, where \(l_j = \sum_{i=1}^{j-1} o_i\) is the total number of operations of the first \(j\) jobs, \(j = 1,2,\ldots,n\), \(l_0 = 0\), \(\sum_{j=1}^{n} o_j = o\). An operation \(i\) has to be executed on machine \(v_i \in M\) without any idleness in time \(p_i > 0\), \(i \in O\). Each machine can execute at most one operation at a time. A feasible solution constitutes a vector of times of the operation execution beginning \(S = (S_1,S_2,\ldots,S_n)\) such that the following constraints are fulfilled:

\[
S_{j-1} + o_j \leq S_j, \quad I = 1,2,\ldots,n.
\]

\[
S_j + p_j \leq S_{j+1}, \quad I = 1,2,\ldots,n.
\]

\[
(S_j + p_j \leq S_{j+1}) \quad \text{or} \quad (S_j + p_j \leq S_j), \quad I \neq 0, \quad v_I = v_j, \quad i \neq j.
\]

Certainly, \(C_j = S_j + p_j\). An appropriate criterion function has to be added to the above constraints. The most frequent are the following two criteria: minimization of the makespan and minimization of the sum of job finishing times. From the formulation of the problem we have \(C_j = C_I, \quad I \neq j\).

The first criterion, the time of finishing all the jobs:\n
\[
C_{\text{max}}(S) = \max \{C_j\}, \quad I \neq n
\]

corresponds to the problem denoted as \(J|\text{Cmax}|\) in the literature. The second criterion, the sum of job finishing times:

\[
C(S) = \sum_{j=1}^{n} C_j
\]

corresponds to the problem denoted as \(J|\sum C_j|\) in the literature. In fact, we can distinguish a wider class of problems, with the objective function form of \(C_{\text{max}}\) and \(\sum C_j\), where \(C_{\text{max}} = \max \{c_{i,j}\} \quad \text{and} \quad \sum_j f_i(C_j) = \sum_j f_i(C_{j-1}) + f_i(C_j)\), for any non-decreasing functions \(f_i\) (such as a flow time, sum of completion times, tardiness, makespan, etc.).

Both problems, with makespan and with the sum of job finishing times, are strongly NP-hard and although they are similarly modeled, the second one is found to be harder because of the lack of some specific properties (so-called block properties, see [11]) used in optimization of execution time of solution algorithms.

Because of NP-hardness of the problem heuristics and meta-heuristics are recommended as ‘the most reasonable’ solution methods. The majority of these methods refer to the makespan minimization (e.g., [9,13,8,12,3]).

#### 3.1. Models and properties

The most commonly used models of job shop scheduling problems are based on the disjunctive or the combinatorial approaches. Both these models are presented in this section.

##### 3.1.1. Disjunctive model

The disjunctive model (see [11]) is most commonly used. However, it is very unpractical from the point of view of efficiency (and computational complexity). It is based on the notion of disjunctive graph \(G = (V,U \cup V)\). This graph has a set of vertices \(O\) which represent operations, a set of so-called conjunctive arcs (i.e. directed) which show technological order of operation execution:

\[
U = \bigcup_{j=1}^{n} l_j \bigcup_{i=1}^{l_j-1} ((i,i+1))
\]

and the set of disjunctive arcs (non-directed) which show possible schedule of operations execution on each machine:

\[
V = \bigcup_{i=1}^{n} ((i,j),(j,i))
\]

A sample disjunctive graph is presented on Fig. 1 (numbers near vertices are operation numbers, jobs are placed in rows and connected by solid arrows; disjunctive arcs are drawn as broken lines). Disjunctive arcs \(((i,j),(j,i))\) are, in fact, pairs of directed arcs with inverted directions which connect vertices \(i\) and \(j\). A vertex \(i \in O\) has a weight \(p_i\) which equals the time of execution of operation \(O_i\). Arcs have the weight zero. A choice of exactly one arc from the set \(((i,j),(j,i))\) corresponds to determining a schedule of operations execution—‘I before \(j\)’ or ‘\(j\) before \(i\)’. A subset of vertices \(W \subset V\) consisting of exclusively directed arcs, at most one from each pair.
from the value of the objective function for a given representation $G$. Permutations of elements of $i$ can be decomposed into subsets of operations executed on a machine. A set of operations defines a precedence relation of jobs execution on the same machine, generates a disjunctive graph model (many disjunctive graphs may represent one solution, not necessary feasible (i.e. it can include cycles). A complete representation, defining a precedence relation of jobs execution on the same machine, generates a disjunctive graph for the job shop problem, which is in position $G$. An example of disjunctive graph for the job shop problem. Fig. 1. An example of disjunctive graph for the job shop problem.

3.1.2. Combinatorial model

In the case of many applications a combinatorial representation of a solution is better than a disjunctive model for the job shop problem. The presented model follows that of Nowicki and Smutnicki [11]. It is void of redundancy, characteristic for the disjunctive graph model (many disjunctive graphs may represent the same solution of the job shop problem). A set of operations $O$ can be decomposed into subsets of operations executed on a single, determined machine $k \in M$, $M_k = \{i \in O : v_i = k\}$ and let $m_k = |M_k|$. The schedule of operations execution on a machine $k$ is determined by a permutation $\pi_k = (\pi_k(1), \pi_k(2), \ldots, \pi_k(m_k))$ of elements of the set $M_k$, $k \in M$, where $\pi_k(i)$ means such an element from $M_k$ which is in position $i$ in $\pi_k$. Let $\Phi_k(M_k)$ be a set of all permutations of elements of $M_k$. A schedule of operations execution on all machines is defined as $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, where $\pi \in \Phi_1 \times \Phi_2 \times \ldots \times \Phi_m$. For a schedule $\pi$ we create a directed graph (digraph) $G(\pi) = (O \cup E(\pi))$ with a set of vertices $O$ and a set of arcs $E(\pi)$, where $U$ is a set of constant arcs representing the technological order of operations execution inside a job, and a set of arcs representing an order of operations execution on machines is defined as $E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k-1} \{(\pi_k(i), \pi_k(i+1))\}$. Each vertex $i \in O$ has the weight $p_i$, each arc has the weight zero. A schedule $\pi$ is feasible if the graph $G(\pi)$ does not include a cycle. For a given $\pi$, terms of operations beginning can be determined in $O(\sigma)$ time from the recurrent formula:

$$S_i = \max\{S_i + p_i, S_k + p_k\}, \quad j \in O,$$

where an operation $i$ is a direct technological predecessor of the operation $j \in O$ and an operation $k$ is a directed machine predecessor of the operation $j \in O$ for a fixed $\pi$. We assume $S_i = 0$ for these operations $j$ which have not any technological or machine predecessors.

An example of the graph $G(\pi)$ is given in Fig. 3 for the same data, as a disjunctive graph from Fig. 1 – it is visible that the $G(\pi)$ has a more transparent structure and is void of redundance connected with superfluous arcs of the disjunctive representation. For a given feasible schedule $\pi$ the process of determining the objective function value requires the time $O(\sigma)$, which is thus shorter than for the disjunctive representation.

4. Sequential determination of the objective function in $O(\sigma)$ time

Taking into consideration the constraints (1)–(3) presented in Section 3, it is possible to determine the time moments of job completion $C_j$, $j \in O$ and job beginning $S_j$, $j \in O$ in $O(\sigma)$ time on the sequential machine using the recurrent formula:

$$S_i = \max\{S_i + p_i, S_k + p_k\}, \quad j \in O,$$

where an operation $i$ is a direct technological predecessor of the operation $j \in O$ and an operation $k$ is a directed machine predecessor of the operation $j \in O$. The determination procedure of $S_j, j \in O$ from the recurrent formula (10) should be initiated by an assignment $S_j = 0$ for those operations $j$ which do not possess any technological or machine predecessors. Next, in each iteration an
It is easy to observe that the order of determining the size \( S_j \) connected with an operation \( j \) for the graph \( G(\pi) \) is a maximal (in sense of the number of vertices) path in the graph \( G(\pi) \). Now, we define layers of the graph collecting vertices (i.e. operations) for which we can calculate \( S_j \) in parallel, as we have calculated starting times for all machine and technological predecessors of operations in the layer (see Fig. 4).

Definition 1. The layer of the graph \( G(\pi) \) is a maximal (in sense of the number of vertices) subsequence of the sequence of vertices ordered by the topological sort algorithm, such that there are no arcs between vertices of this subsequence. We will need this definition in the next paragraph.

5. Parallel determining of the objective function

Methods based on matrix multiplication: We propose an original method using \( O(n^3) \)/log \( o \))-processor CREW PRAM with the computational complexity \( O(\log o) \). This algorithm is \( O(\log o) \) times more efficient than the algorithm proposed in the paper of Steinhiel et al. [14] and it can be used not only for \( Jl/J \max \) problem, but also for \( Jl/J \Sigma f_i \) problems, as well as for \( Jl/J \Sigma f_i \) as it was introduced in Section 3.

Theorem 1. For a fixed feasible operations order \( \pi \) for the \( Jl/J \Sigma f_i \) or \( Jl/J \Sigma f_i \) problem, the value of the objective function can be determined in \( O(\log o) \) time on \( O(n^3) \)/log \( o \))-processor CREW PRAMs.

Proof. For the graph \( G(\pi) = (V, E, \pi) \) defined in Section 3 for a job shop problem we introduce the matrix of distances \( A = (a_{u,v}) \) with the size \( o \times o \), where \( a_{u,v} \) is the length of the longest path between vertices \( u \) and \( v \). We initialize values \( a_{u,v} \) in the following way:

\[
a_{u,v} = \begin{cases} 
  p_0 & \text{if } (u,v) \in U^* \cup E(\pi), \\
  0 & \text{if } (u,v) \notin U^* \cup E(\pi). 
\end{cases}
\]

The matrix \( A \) will be used for the calculation of the longest paths in the graph \( G(\pi) \). The initial values of the matrix \( A \) can be determined in \( O(1) \) time using \( O(n^2) \) processors, because this requires \( o^2 \) independent assignment instructions, each one for every pair \( (u,v) \), \( u,v = 1, 2, \ldots, o \).

The problem of determining objective function value for the \( Jl/J \Sigma f_i \) or \( Jl/J \Sigma f_i \) job shop problems requires finding the lengths of the longest paths from the vertex 0 \( \in U^* \) to vertices \( 1, 2, \ldots, n \) (which corresponds to determination of the following values of job execution finishing times: \( C_1, C_2, \ldots, C_n \), where \( n \) defines the number of jobs, as it was defined in Section 3. To determine the length of paths, it is enough to execute \( \log o \) parallel steps because in each step \( k = 1, 2, \ldots, \log o \) the algorithm described below updates the lengths of the longest paths between vertices with the distance (in the sense of the number of vertices) of at most 1, 2, 4, 8, \ldots, \( 2^{\log o} \) after having executed the \( \log o \) steps the matrix \( A \) possesses information about the length of paths between vertices with the distance (in the sense of the number of vertices) of \( 2^{\log o} \), that is, between all the vertices, because the number of vertices on the longest (in the sense of the number of vertices) path in the graph \( G(\pi) \) must not be greater than \( o \) (\( G(\pi) \) is an acyclic digraph). For technical needs of the algorithm, an additional three-dimensional table \( T = [t_{u,w,v}] \) of the size \( o \times o \times o \) is defined. It is used for a transitive closure calculation of \( G(\pi) \).

The algorithm requires execution of the following identical steps \( \log o \) times:

1. updating \( t_{u,w,v} \) for all pairs \( (u,v) \) due to the formula \( t_{u,w,v} = a_{u,w} + a_{w,v} \).
2. updating \( a_{u,v} \) for all pairs \( (u,v) \) on the basis of the equation \( a_{u,v} = \max\{a_{u,v}, a_{w,v} + t_{u,w,v}\} \).

Step 1 executed on \( o^3 \) processors can take the time \( O(1) \). On \( o^3/\log o \) processors the calculations have to be made \( \log o \) times, so the computational complexity of this step is \( O(\log o) \).

Step 2 consists in determining a maximum of \( o+1 \) values, which can be done on \( O(o/\log o) \) processors in \( O(\log o) \) time. As such a maximum should be determined for \( o^2 \) pairs \( (u,v) \) and these calculations are independent and have to be repeated \( \log o \) times, therefore, using \( p = O(o^3/\log o) \) processors, the whole algorithm has a computational complexity:

\[
T_{par}(p) = O(\log o) / O(\log^2 o).
\]

Finally, for the \( Jl/J \Sigma f_i \) problem, all the \( f_i(C_j) \), where \( C_j = a_{0,j} \), should be summarized. These values can be taken from table A. Summation takes the time \( O(\log n) \) using \( O(n/\log n) \)-processor CREW PRAMs keeping computational complexity \( O(\log^2 o) \) and the number of processors \( O(o^3/\log o) \) for the whole method described because the number of jobs \( n \) is smaller or equals the number of operations \( o \).

Similarly, for the \( Jl/J \Sigma f_i \) problem, it is necessary to determine maximum of all values \( f_i(C_j) \), \( j \in \mathbb{J} \). This step has also computational complexity \( O(\log n) \) using \( O(n/\log n) \)-processor CREW PRAM machine keeping computational complexity \( O(\log^2 o) \) and the number of processors \( O(o^3/\log o) \) for the entire method. \( \square \)

Table 1 presents times of \( C_{max} \) calculations due to the matrix multiplication based method from Theorem 1. The 32-processor nVidia GeForce 9500 GT card (GPU) with CUDA support was used for the calculation. The maximum for each pair \( (u,v) \) of vertices was calculated in \( O(o) \) time using a single processor, because the number of processes \( o^3/\log o \) was too big for the hardware used in the experiment. Therefore, the whole parallel algorithm has the scaled computational complexity \( O(o \log^2 o) \) instead of \( O(\log^2 o) \).
The complexity of the whole method to \( O(n^3) \) steps (compare Theorem 1) which decreases computational complexity from \( O(n^3) \) to \( O(n \log n) \). The conjecture refers to the upper bound of the optimal solution, so the algorithm has to work on solutions close to optimal, i.e. as an element of a metaheuristics.

Methods based on partitioning into layers: The main problem in obtaining a good speedup value of the methods mentioned above is the fact that a computational complexity of the sequential method of determining makespan value for the job shop problem is \( O(n) \). It is, however, difficult to parallelize it because of its recurrent nature. Now we show another approach of determining objective function value, which is more time-consuming, but cost-optimal. First, we need to determine the number of layers \( d \) of the graph \( G(\pi) \). A sample of layer determination for the conjunctive graph from Fig. 3 is shown in Fig. 4.

Theorem 3. For a fixed feasible operations order \( \pi \) for the \( J/C_{\text{max}} \) problem, the number of layers from Definition 1 of the graph \( G(\pi) \) can be calculated in \( O(\log n) \) time on the CREW PRAMs with \( O(n^3/\log n) \) processors.

Proof. Here we use the graph \( G(\pi) \) with an additional 0 vertex. Let \( B = \{b_j\} \) be an incidence matrix for the graph \( G(\pi) \), i.e. \( b_{ij} = 1 \) if there is an arc \( i,j \) in the graph \( G(\pi) \), otherwise \( b_{ij} = 0 \), \( i,j = 1,2,\ldots,n \). The proof is given in three steps.

1. Let us calculate the longest paths (in the sense of the number of vertices) in \( G(\pi) \). We can use the algorithm from the proof of Theorem 1 with the incidence matrix \( B \) instead of the matrix \( A \). We need the time \( O(\log n) \) and CREW PRAMs with \( O(n^3/\log n) \) processors.

2. We sort distances from the vertex 0 to every vertex in an increasing order. Their indexes, after having been sorted, correspond to the topological order of vertices. This takes the time \( O(\log n) \) and CREW PRAMs with \( o + 1 = O(\log n) \) processors, using Cole's parallel merge sort algorithm [4]. We obtain a sequence \( \text{Topo}[i], i = 0,1,2,\ldots,o \).

3. Let us assign each element of the sorted sequence to one processor, without the last one. If the next value of the sequence (distance from 0) \( \text{Topo}[i + 1], i = 0,1,\ldots,o - 1 \) is the same as \( \text{Topo}[i] \) considered by the processor \( i \), we assign \( \text{Topo}[i] = i \). Otherwise, if \( \text{Topo}[i + 1] > \text{Topo}[i] \), this step requires the time \( O(1) \) and processors. Next, we add all values \( \text{Topo}[i], i = 0,1,\ldots,o - 1 \). To make this step we need the time \( O(\log n) \) and CREW PRAMs with \( O(n^3/\log n) \) processors. We get \( d = 1 + \sum_{i=0}^{o-1} \text{Topo}[i] \) because there is an additional layer connected with exactly one vertex 0.

The most time- and processor-consuming is Step 1. We need the time \( O(\log n) \) and the number of processors \( O(n^3/\log n) \) of the CREW PRAMs.

Theorem 4. For a fixed feasible operations order \( \pi \) for the \( J/C_{\text{max}} \) problem, the value of objective function can be determined in \( O(d) \) time on \( O(1/d) \)-processor CREW PRAMs, where \( d \) is the number of layers of the graph \( G(\pi) \).

Proof. Let \( \Gamma_k, k = 1,2,\ldots,d \), be the number of calculations of the operations finishing moment \( C_i, i = 1,2,\ldots,o \) in the \( k \)-th layer. Certainly \( \sum_{k=1}^{d} \Gamma_k = o \). Let \( p \) be the number of processors used. The time of computations in a single layer \( k \) after having divided calculations into \( \lfloor \Gamma_k/p \rfloor \) groups, each group containing (at most) \( p \) elements, is \( \lfloor \Gamma_k/p \rfloor \) (the last group cannot be full). Therefore, the total computation time in all \( d \) layers equals \( \sum_{k=1}^{d} \lfloor \Gamma_k/p \rfloor \leq \sum_{k=1}^{d} \Gamma_k = o \).
Theorem 5. For a fixed feasible operations order \( \pi \) for the \( J/C_{\text{max}} \) problem, the value of objective function can be determined in \( O(\log^2 \alpha) \) time on \( O(\alpha) \)-processor CREW PRAMs, where \( \alpha \) is an upper bound of the objective function value.

Proof. Let us observe that the number of layers \( d \) corresponds to the number of vertices on the longest (in the sense of the number of vertices) path in \( G(\pi) \). This number \( d \) which can be bounded by \( C_{\text{max}}/\min_p (\min_p \text{ is a minimal processing time of all operations}) \), where \( C_{\text{max}} \) is the length (as a sum of weights) of the longest path in \( G(\pi) \). The proof is 'a contrario'.

Let us assume that there is a path with more than \( C_{\text{max}}/\min_p \) vertices. Therefore, its length (as a sum of weights) would be greater than \( C_{\text{max}} \), which is impossible because \( C_{\text{max}} \) is the length of the longest path in \( G(\pi) \). We can use the upper bound of the objective function \( A \) instead of \( C_{\text{max}} \), as well as a minimal \( \min_p \) value which is 1, because \( C_{\text{max}}/\min_p < \alpha/1 \) and we are looking for the upper bound. The other part of the proof is similar to the proof of Theorem 4, as regards the time \( d \leq A \) estimation and the number of processors \( o/d \leq o \).

From the above theorem a surprising conclusion can be drawn, namely: if we can determine the upper bound of the objective function value \( A \), the calculations take constant time \( O(1) \). The trivial upper bound \( A \) of the makespan is the sum of the processing times of all operations. Although the algorithm for determining the \( d \) value has computational complexity \( O(\log^2 \alpha) \) (which makes the efficiency of the layer-based method worse than efficiency of the matrix multiplication approach which is visible in Figs. 5 and 6 comparison), it can be executed only once, at the very beginning. Next, one can calculate only how the \( d \)

\[
\sum_{i=1}^{d} (T_{\text{seq}}(i)/p+1) = o/p + d.
\]

To obtain the time of computations \( O(d) \) we should use \( p = O(\log^2 \alpha) \) processors.

This theorem provides a cost-optimal method of parallel calculation of the objective function value for the job shop problem with the makespan criterion.

Table 2 presents times of \( C_{\text{max}} \) calculations due to the layer-based method from Theorems 3 and 4 tested on the set of Lawrence [10] benchmark instances and run on the 480-processor nVidia GTX480 GPU with CUDA support. We use the following notions: \( T_{\text{seq}} \) – sequential time obtained by using 1 processor of the GPU, \( T_{\text{par}} \) – parallel time with using shared memory for calculations, \( S_1 \) – orthodox speedup obtained with using shared memory based parallel algorithm (not possible for larger instances, shared memory has only 488B), \( T_{\text{GPU}} \) – parallel time with using global memory for calculations, \( S_2 \) – orthodox speedup obtained with using global memory based parallel algorithm. Due to the hardware limitations the number of processors used was set as \( o \) (instead of \( o^2 \log \alpha \) as in Theorem 3), so the computational complexity was scaled from \( O(\log^3 \alpha) \) to \( O(\alpha^2 \log \alpha) \). The computational experiments shown in Fig. 5 confirm the theoretical results.

### Table 2

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>( o )</th>
<th>( T_{\text{seq}} )</th>
<th>( T_{\text{GPU}} )</th>
<th>( S_1 )</th>
<th>( T_{\text{par}} )</th>
<th>( S_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 \times 5</td>
<td>50</td>
<td>0.11</td>
<td>16.88</td>
<td>0.01</td>
<td>9.37</td>
<td></td>
</tr>
<tr>
<td>15 \times 5</td>
<td>75</td>
<td>0.42</td>
<td>27.59</td>
<td>0.03</td>
<td>11.59</td>
<td></td>
</tr>
<tr>
<td>20 \times 5</td>
<td>100</td>
<td>1.22</td>
<td>44.86</td>
<td>0.07</td>
<td>16.05</td>
<td></td>
</tr>
<tr>
<td>15 \times 10</td>
<td>150</td>
<td>6.23</td>
<td>91.39</td>
<td>0.20</td>
<td>31.29</td>
<td></td>
</tr>
<tr>
<td>20 \times 10</td>
<td>200</td>
<td>19.01</td>
<td>41.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 \times 15</td>
<td>225</td>
<td>27.69</td>
<td>43.72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 \times 10</td>
<td>300</td>
<td>79.27</td>
<td>45.83</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>16.94</td>
<td>0.029</td>
<td>45.33</td>
<td>0.404</td>
<td>26.92</td>
</tr>
</tbody>
</table>

\[ d \leq \log \alpha \ |

\[ T_{\text{seq}}(i)/p+1 = o/p + d. \]

<table>
<thead>
<tr>
<th>value of ( c )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>( C_2 )</td>
<td>( C_3 )</td>
<td>( C_4 )</td>
<td>...</td>
<td>( C_0 )</td>
<td></td>
</tr>
</tbody>
</table>

which can be obtained using \( O(\alpha) \) processors in \( O(1) \) time (each processor writes its own number \( i \) and \( C_i \)), i.e. for the sample from Fig. 4:

<table>
<thead>
<tr>
<th>value of ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>( C_2 )</td>
<td>( C_3 )</td>
<td>( C_4 )</td>
<td>...</td>
<td>( C_0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Afterwards we obtain the sorted second row, however, there are numbers of corresponding vertices in the first row

<table>
<thead>
<tr>
<th>value of ( i )</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>8</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>10</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>( C_2 )</td>
<td>( C_3 )</td>
<td>( C_4 )</td>
<td>...</td>
<td>( C_0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This two-row table will be called \( \text{Topo}(\pi) \). To avoid concurrent readings, we can multiply this table to the second, identical, table \( \text{Topo}(\pi) \) using \( O(\alpha) \) processors in constant time \( O(1) \). Each processor \( i = 1, 2, \ldots, o-1 \) compares a value \( \text{Topo}(\pi)(i) \) with \( \text{Topo}(\pi)(i+1) \). If \( \text{Topo}(\pi)(i+1) < \text{Topo}(\pi)(i) \), then a processor generates (writes to a variable \( c \) 1, otherwise it generates 0. To determine the number of layers \( d \) it is enough to get the last \( \text{Topo}(\pi) \) value: \( d = \text{Topo}(\pi)(n) \). Because it corresponds to the longest path (understood as the number of vertices) from the vertex 0 (table \( \text{Topo} \) is sorted). In our sample the number of layers \( d = 7 \). For the parallel algorithm to determine the objective function value, it is necessary to know not only what is the index of the first vertex in the layer, but also how many vertices belong to each layer and what are their numbers. Such a table \( \text{first}_i \) of the first vertices in each layer can be created in \( O(1) \) time as follows: each processor which generates 1 (i.e. its \( c = 1 \)) writes to the table \( \text{first}_i \) the value \( \text{first}_i = \text{Topo}(\pi)(i) \). In this way we obtain the \( d \)-elementary table \( \text{first} \) from which, together with the table \( \text{Topo} \), we can get all the information of layers.

However, in general, if we have no knowledge of the number of layers \( d \), we can employ the following estimation.

### 6. Practical aspects of the objective function value determination

Theorem 4 on 480-processor GPU.

\[ d \leq \log_2 \frac{n}{a o} \quad \log_2 a \leq \log_2 \left( \frac{n}{o} \right) \]

The trivial upper bound \( d \) is \( \log_2 \frac{n}{a o} \) \( o \leq \frac{n}{a} \) \( o \leq \frac{n}{a} \) \( \frac{n}{a} \leq \frac{n}{a} \) \( \frac{n}{a} \leq \frac{n}{a} \)

Fig. 6. Execution times comparison of the layers based procedure of \( C_{\text{max}} \) determination for the job shop problem on 480-processor GPU with scaled theoretical complexity.
7. Conclusion

In this paper, there were designed new methods of parallel objective function value calculation for a given job execution sequence in the job shop problem. Considering the computational complexity $O(o)$ for the sequential case, the new parallel methods have been proposed with significantly lower computational complexity $O(\log^2 o)$, $O(\log o \log A)$, $O(d)$ and $O(A)$, where $o$ is the number of operations, $A$ is an upper bound of the objective function value and $d$ is the number of layers created during the work of a topological sort algorithm. The method proposed here can be applied not only with makespan and the sum of jobs criteria, but also with any objective function of the form of $f_{\text{max}}$ and $\sum f_i$, for non-decreasing functions $f_i$.

We are considering the parallelization of the recurrent method of the objective function calculation in this paper. As the future work one can try to design a parallel version of the non-recurrent formula, e.g. for the makespan, which probably can give greater speedups, but for much greater number of processors needed for the calculations.

References