Computers & Operations Research 39 (2012) 2258–2264

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

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*).

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 $1/\beta$ (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. Steinhöfel et al. [14] described the method of parallel objective function determination in $O(\log^2 o)$ time on $O(o^3)$ processors, where o 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 Steinhöfel et al., which works in $O(\log^2 o)$ time on $O(o^3/\log o)$ 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 $O(1)$.

* Tel.: +48 713202961.
E-mail address: wojciech.bozejko@pwr.wroc.pl

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.11.009

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* $O(pt/p')$ *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 Φ and a parallel algorithm A_{par}. Let us define $T_{A_{par}}(p)$ – time of calculations of the algorithm A_{par}, which is necessary to solve the problem Φ on the machine using p processors. Let $T_{A_{seq}}$ be a time of calculations of the best (the fastest) known sequential algorithm A_{seq} which solves the same problem Φ on the sequential machine with the processor identical to processors of the parallel machine. We define the *speedup* $S_{A_{par}}(p)=T_{A_{seq}}/T_{A_{par}}(p)$. The *cost* $C_{A_{par}}(p)$ of solving a problem by using an algorithm A_{par} in a p – processors parallel machine is defined as $C_{A_{par}}(p)=p\cdot T_{A_{par}}(p)$. 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 $\mathcal{J}=\{1,2,\ldots,n\}$, a set of machines $M=\{1,2,\ldots,m\}$ and a set of operations $\mathcal{O}=\{1,2,\ldots,o\}$. The set $\mathcal{O}$ 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 $l_j=\sum_{i=1}^{j}o_i$ is the total number of operations of the first j jobs, $j=1,2,\ldots,n$, $l_0=0$, $\sum_{i=1}^{n}o_i=o$. An operation i has to be executed on machine $v_i\in M$ without any idleness in time $p_i>0$, $i\in\mathcal{O}$. Each machine can execute at most one operation at a time. A feasible solution constitutes a vector of times of the operation execution beginning $\mathcal{S}=(S_1,S_2,\ldots,S_o)$ such that the following constraints are fulfilled:

$$S_{l_{j-1}+1}\geq 0,\quad j=1,2,\ldots,n, \tag{1}$$

$$S_i+p_i\leq S_{i+1},\quad i=l_{j-1}+1,\quad l_{j-1}+2,\ldots,l_j-1,\quad j=1,2,\ldots,n, \tag{2}$$

$$(S_i+p_i\leq S_j)\quad\text{or}\quad(S_j+p_j\leq S_i),\quad i,j\in O,\quad v_i=v_j,\quad i\neq j. \tag{3}$$

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 $\mathcal{C}_j\equiv C_{l_j}$, $j\in\mathcal{J}$.

The first criterion, the time of finishing all the jobs:

$$C_{\max}(S)=\max_{1\leq j\leq n}C_{l_j}, \tag{4}$$

corresponds to the problem denoted as $J\|C_{\max}$ in the literature. The second criterion, the sum of job finishing times:

$$C(S)=\sum_{j=1}^{n}C_{l_j}, \tag{5}$$

corresponds to the problem denoted as $J\|\sum C_i$ in the literature. In fact, we can distinguish a wider class of problems, with the objective function form of $f_{\max}$ and $\sum f_i$, where $f_{\max}=\max_{1\leq i\leq n}f_i(C_i)$ and $\sum f_i=\sum_{i=1}^{n}f_i(C_i)$, 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 metaheuristics 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 $G=(O,U\cup V)$. 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:

$$U=\bigcup_{j=1}^{n}\bigcup_{i=l_{j-1}+1}^{l_j-1}\{(i,i+1)\} \tag{6}$$

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

$$V=\bigcup_{i,j\in O,i\neq j,v_i=v_j}\{(i,j),(j,i)\}. \tag{7}$$

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 $W\subset V$ consisting of exclusively directed arcs, at most one from each pair

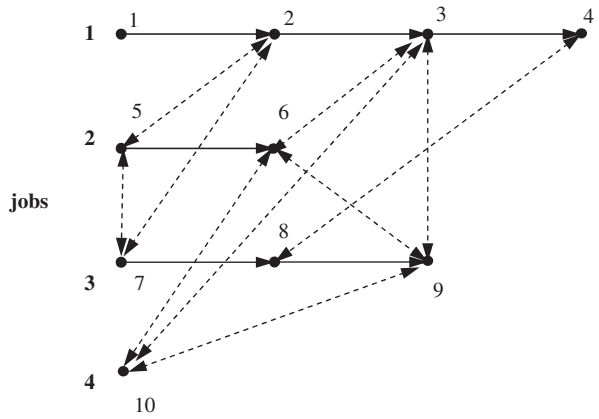

Fig. 1. An example of disjunctive graph for the job shop problem.

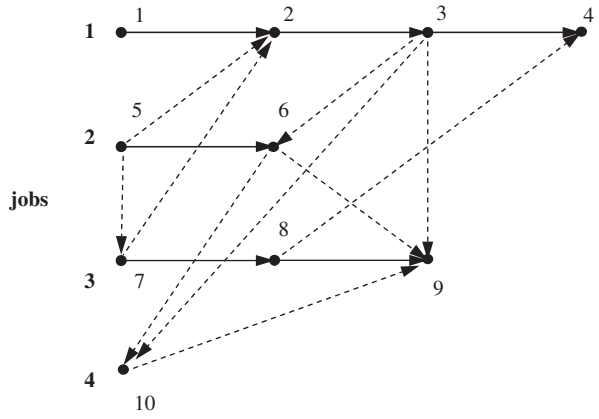

Fig. 2. An example of the graph $G(W)$ for the job shop problem.

$\{(i,j),(j,i)\}$, we call a *representation* of disjunctive arcs. Such a representation is complete if all the disjunctive arcs have determined direction. A complete representation, defining a precedence relation of jobs execution on the same machine, generates one solution, not necessary feasible (i.e. it can include cycles). A feasible solution is generated by a complete representation W such that the graph $G(W)=(O,U\cup W)$ is acyclic (see Fig. 2). For a feasible schedule values S_i of the vector of operations execution starting times $\mathcal{S}=(S_1,S_2,\ldots,S_o)$ can be determined as a length of the longest path entering to the vertex i (without p_i). As the graph $G(W)$ includes o vertices and $O(o^2)$ arcs, determining the value of the objective function for a given representation W takes $O(o^2)$ time by using Bellman algorithm of paths in graphs determination.

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 redundance, characteristic for the disjunctive graph model (many disjunctive graphs may represent the same solution of the job shop problem). A set of operations $\mathcal{O}$ can be decomposed into subsets of operations executed on a single, determined machine $k\in M$, $M_k=\{i\in\mathcal{O}:v_i=k\}$ 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 π_k. Let $\Phi_n(M_k)$ be a set of all permutations of elements of M_k. A schedule of operations

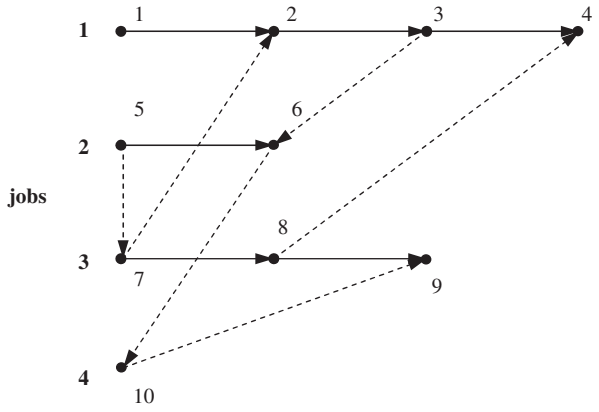

Fig. 3. An example of the $G(\pi)$ graph of combinatorial model for the job shop problem.

execution on all machines is defined as $\pi=(\pi_1,\pi_2,\ldots,\pi_m)$, where $\pi\in\Phi_n$, $\Phi_n=\Phi_n(M_1)\times\Phi_n(M_2)\times\cdots\times\Phi_n(M_m)$. For a schedule π we create a directed graph (digraph) $G(\pi)=(\mathcal{O},U\cup E(\pi))$ with a set of vertices $\mathcal{O}$ and a set of arcs $U\cup E(\pi)$), 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

$$E(\pi)=\bigcup_{k=1}^{m}\bigcup_{i=1}^{m_k-1}\{(\pi_k(i),\pi_k(i+1))\}. \tag{8}$$

Each vertex $i\in\mathcal{O}$ has the weight p_i, each arc has the weight zero. A schedule π is feasible if the graph $G(\pi)$ does not include a cycle. For a given π, terms of operations beginning can be determined in $O(o)$ time from the recurrent formula:

$$S_j=\max\{S_i+p_i,S_k+p_k\},\quad j\in\mathcal{O}, \tag{9}$$

where an operation i is a direct technological predecessor of the operation $j\in\mathcal{O}$ and an operation k is a directed machine predecessor of the operation $j\in\mathcal{O}$ for a fixed π. We assume $S_j=0$ 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 π the process of determining the objective function value requires the time $O(o)$, which is thus shorter than for the disjunctive representation.

4. Sequential determination of the objective function in $O(o)$ 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, $j\in\mathcal{O}$ and job beginning S_j, $j\in\mathcal{O}$ in $O(o)$ time on the sequential machine using the recurrent formula:

$$S_j=\max\{S_i+p_i,S_k+p_k\},\quad j\in\mathcal{O}, \tag{10}$$

where an operation i is a direct technological predecessor of the operation $j\in\mathcal{O}$ and an operation k is a directed machine predecessor of the operation $j\in\mathcal{O}$. The determination procedure of S_j,$j\in\mathcal{O}$ 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

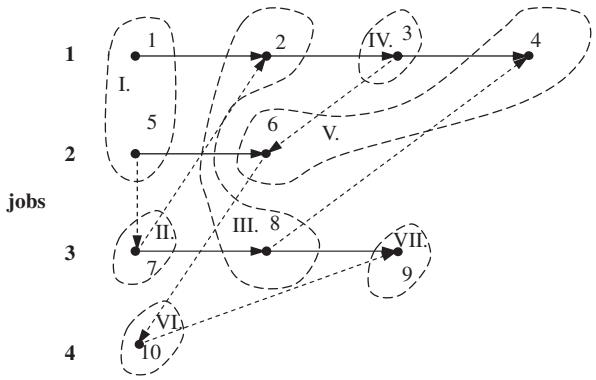

Fig. 4. A sample of $G(\pi)$ graph for the job shop problem with $d=7$ layers.

operation j has to be chosen for which:

1. the execution beginning moment S_j has not been determined yet, and
2. these moments were determined for all its direct technological and machine predecessors; for such an operation j the execution beginning moment can be determined from (10).

It is easy to observe that the order of determining S_j times corresponds to the index of the vertex of the graph $G(\pi)$ connected with an operation j after the topological sorting of this graph. The method mentioned above is in fact a simple sequential topological sort algorithm without indexing of operations (vertices of the graph). If we add to this algorithm an element of indexing vertices, for which we calculate S_j value, we obtain a sequence which is the topological order of vertices of 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 $O(o^3/\log o)$-processor CREW PRAM with the computational complexity $O(\log^2 o)$. This algorithm is $O(\log o)$ times more efficient than the algorithm proposed in the paper of Steinhöfel et al. [14] and it can be used not only for $J\|C_{\max}$, but also for $J\|\sum f_i$ problems, as well as for $J\|f_{\max}$, as it was introduced in Section 3.

Theorem 1. *For a fixed feasible operations order π for the $J\|\sum f_i$ or $J\|f_{\max}$ problem, the value of the objective function can be determined in $O(\log^2 o)$ time on $O(o^3/\log o)$-processor* CREW PRAMs.

Proof. For the graph $G^*(\pi)=(O^*,U^*\cup E)$ 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 initiate values $a_{u,v}$ in the following way:

$$a_{u,v}=\begin{cases} p_u & \text{if } (u,v)\in U^*\cup E(\pi),\\ 0 & \text{if } (u,v)\notin U^*\cup E(\pi). \end{cases} \tag{11}$$

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 $O(o^2)$ 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 $J\|\sum f_i$ or $J\|f_{\max}$ job shop problems requires finding the lengths of the longest paths from the vertex $0\in U^*$ to vertices $l_1,l_2,\ldots,l_n$ (which corresponds to determination of the following values of job execution finishing times: $C_{l_1},C_{l_2},\ldots,C_{l_n}$), 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 $\lceil\log(o)\rceil$ parallel steps because in each step $k=1,2,\ldots,\lceil\log(o)\rceil$ 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 $\lceil\log(o)\rceil$ 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 $2^{\log(o)}=o$, 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 $\lceil\log o\rceil$ times:

1. updating $t_{u,w,v}$ for all triples (u,w,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 $a_{u,v}=\max\{a_{u,v},\max_{1\le w\le o}t_{u,w,v}\}$.

Step 1 executed on o^3 processors can take the time $O(1)$. On $\lceil o^3/\log o\rceil$ processors the calculations have to be made $\lceil\log o\rceil$ 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 $\lceil\log o\rceil$ times, therefore, using $p=O(o^3/\log o)$ processors, the whole algorithm has a computational complexity:

$$T_{par}(p)=\lceil\log o\rceil O(\log o)=O(\log^2 o). \tag{12}$$

Finally, for the $J\|\sum f_i$ problem, all the $f_j(C_{l_j})$, where $C_{l_j}=a_{0,l_j}$, $j\in\mathcal{J}$, 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 $J\|f_{\max}$ problem, it is necessary to determine maximum of all values $f_j(C_{l_j})$, $j\in\mathcal{J}$. 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. □

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 $O(\log^2 o)$

Table 1
Times of $C_{\max}$ calculations (in milliseconds) due to the method from Theorem 1 on GPU.

$n \times m$	o	t_p^{av}	t_p^{min}	t_p^{max}	$o\log^2 o$
5×5	25	0.24	0.17	0.61	0.054
10×5	50	0.25	0.2	0.61	0.159
20×5	100	0.32	0.24	0.66	0.441
10×10	100	0.28	0.22	0.61	0.441
20×10	200	0.49	0.41	0.85	1.169
10×20	200	0.66	0.39	4.86	1.169
50×5	250	0.65	0.58	1.06	1.586
20×20	400	1.13	1.02	1.61	2.989
100×5	500	1.94	1.54	8.16	4.019
50×10	500	1.94	1.4	8.1	4.019
10×50	500	2.13	1.48	6.5	4.019
100×10	1000	4.97	3.58	8.86	9.932
50×20	1000	4.11	3.46	9.16	9.932
20×50	1000	4.34	3.41	8.34	9.932
100×20	2000	14.47	11.96	16.47	24.050
50×50	2500	21.49	17.6	23.22	31.853
100×50	5000	71.77	67.85	80.1	75.494

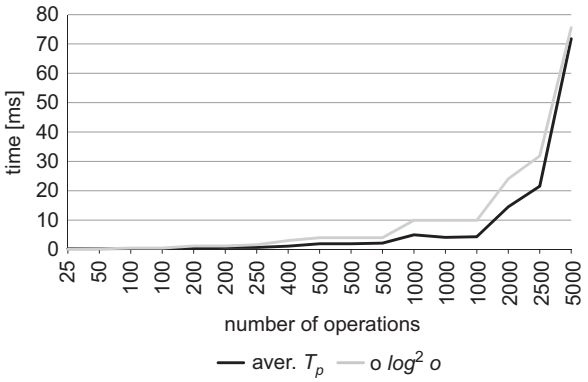

Fig. 5. Comparison of execution times of the matrix multiplication based procedure on a 32-processor GPU.

(because $O(o^2/\log o)$ processes was used to calculations instead of $O(o^3/\log o)$). The algorithm was using up to 32 physical processors, the number of processes was up to $\lceil o^2/\log o \rceil$. The computational experiments shown in Table 1 and in Fig. 5 fully confirm the theoretical results.

Theorem 2. *For a fixed feasible operations order π for the $J\|\sum f_i$ or $J\|f_{\max}$ problem, the value of the objective function can be determined in $O(\log o \log \Lambda)$ time on $O(o^3/\log o)$-processor CREW PRAMs, where Λ is an upper bound of the objective function value.*

Proof. The proof is similar to that of Theorem 1. It is enough to repeat the main loop $\lceil \log \Lambda \rceil$ times instead of $\lceil \log o \rceil$ times because the maximal number of vertices on the critical path is not greater than $\Lambda/p_{\min}$, where $p_{\min}$ is the shortest operation execution time among all operations from the set $\mathcal{O}$. Values of $p_{\min}$ are integers, minimal value equals 1, thus the maximal number of vertices on the critical path is Λ. To determine the length of the critical path it is enough to execute $\lceil \log \Lambda \rceil$ parallel steps (compare Theorem 1) which decreases computational complexity of the whole method to $O(\log o \log \Lambda)$ keeping the number of processors $O(o^3/\log o)$. □

According to studies on popular benchmark instances for the job shop problem [6,10,17,15] there is a conjecture in the paper of Steinhöfel et al. [14] that for the makespan of the optimal solutions λ_{opt} there exists a uniform upper bound $\lambda_{opt} \le p_{\max}(n+m)$, where $p_{\max}$ is the maximal processing time of all operations, n is the number of jobs and m is the number of machines. Let us assume that the number of operations $o=nm$, i.e. as in popular benchmarks [16]. If the conjecture of Steinhöfel et al. is true, decreasing the computational complexity from o to $\Lambda = \lambda_{opt}$ decreases the complexity from nm to $n+m$. 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 $O(o)$. 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 π for the $J\|C_{\max}$ problem, the number of layers from Definition1 of the graph $G(\pi)$ can be calculated in $O(\log^2 o)$ time on the CREW PRAMs with $O(o^3/\log o)$ processors.*

Proof. Here we use the graph $G^*(\pi)$ with an additional 0 vertex. Let $B=[b_{ij}]$ 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 $b_{ij}=0$, $i,j=1,2,\ldots,o$. 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 $O(\log^2 o)$ and CREW PRAMs with $O(o^3/\log o)$ 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 $O(\log o)$ and CREW PRAMs with $o+1=O(o)$ processors, using Cole's parallel merge sort algorithm [4]. We obtain a sequence $Topo[i]$, $i=0,1,2,\ldots,o$.
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) $Topo[i+1]$, $i=0,1,\ldots,o-1$ is the same as $Topo[i]$ considered by the processor i, we assign $c[i]\leftarrow 1$, and $c[i]\leftarrow 0$ if $Topo[i+1] \ne Topo[i]$. This step requires the time $O(1)$ and o processors. Next, we add all values $c[i]$, $i=0,1,\ldots,o-1$. To make this step we need the time $O(\log o)$ and CREW PRAMs with $O(o)$ processors. We get $d=1+\sum_{i=0}^{o-1} c[i]$ 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 $O(\log^2 o)$ and the number of processors $O(o^3/\log o)$ of the CREW PRAMs. □

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

Proof. Let Γ_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 $\sum_{i=1}^{d} \Gamma_i = o$. Let p be the number of processors used. The time of computations in a single layer k after having divided calculations into $\lceil \Gamma_i/p \rceil$ groups, each group containing (at most) p elements, is $\lceil \Gamma_i/p \rceil$ (the last group cannot be full). Therefore, the total computation time in all d layers equals $\sum_{i=1}^{d} \lceil \Gamma_i/p \rceil \le$

Table 2
Times of $C_{\max}$ calculations (in milliseconds) due to the layers based method from Theorems 3 and 4 on 480-processor GPU.

$n \times m$	o	T_{seq}	τ_{par}^{GPU}	S_1	T_{par}^{GPU}	S_2
10 ×5	50	0.11	0.01	18.68	0.01	9.37
15 ×5	75	0.429	0.016	27.594	0.037	11.599
20 ×5	100	1.224	0.028	44.496	0.076	16.057
15 ×10	150	6.323	0.069	91.399	0.202	31.295
20 ×10	200	19.013	–	–	0.459	41.444
15 ×15	225	27.690	–	–	0.633	43.721
30 ×10	300	79.570	–	–	1.736	45.836
Average		**16.948**	**0.029**	**45.334**	**0.404**	**26.923**

$\sum_{i=1}^{d}(\Gamma_i/p+1) = o/p+d$. To obtain the time of computations $O(d)$we should use $p = O(\frac{o}{d})$ 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 $C_{\max}$ calculations due to the layer-based method from Theorems 3 and 4 tested on the set of Lawrence [10] benchmark instances and ran on the 480-processor nVidia GTX480 GPU with CUDA support. We use the following notions: T_{seq} – sequential time obtained by using 1 processor of the GPU, τ_{par}^{GPU} – 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 48KB), T_{par}^{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 $o^3/\log o$, as in Theorem 3), so the computational complexity was scaled from $O(\log^2 o)$ to $O(o^2 \log o)$. The computational experiments shown in Fig. 5 confirm the theoretical results.

6. Practical aspects of the objective function value determination

Step 1 consists in determining the longest paths (in the sense of vertex number) in the graph. We are interested in the lengths of paths from the vertex 0 to each other vertex. In Step 2 we should sort the obtained lengths. We make use of a two-row table and we sort it with reference to the second row together with the first row:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & \dots & o \\ C_1 & C_2 & C_3 & C_4 & \dots & C_o \end{bmatrix},$$

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

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 3 & 6 & 5 & 1 & 5 & 2 & 3 & 7 & 6 \end{bmatrix}.$$

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

$$\begin{bmatrix} 1 & 5 & 7 & 2 & 8 & 3 & 4 & 6 & 10 & 9 \\ 1 & 1 & 2 & 3 & 3 & 4 & 5 & 5 & 6 & 7 \end{bmatrix}.$$

This two-row table will be called $Topo[1..o][1..2]$. To avoid concurrent readings, we can multiply this table to the second, identical, table $Topo2[1..o][1..2]$ using $O(o)$ processors in constant time $O(1)$. Each processor $i = 1, 2, \dots, o-1$ compares a value $Topo[i][1]$ with $Topo2[i+1][1]$. If $Topo[i][1] < Topo2[i+1][1]$, 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 *Topo* value: $d = Topo[o][1]$ because it corresponds to the longest path (understood as the number of vertices) from the vertex 0 (table *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 $first\_in[1..o]$ 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 $first\_in[Topo[i][1]] := i$. In this way we obtain the d-elementary table *first_in* from which, together with the table *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.

Theorem 5. *For a fixed feasible operations order π for the $J\|C_{\max}$ problem, the value of objective function can be determined in $O(\Lambda)$ time on $O(o)$-processor CREW PRAMs, where Λ 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 $C_{\max}/p_{\min}$ ($p_{\min}$ is a minimal processing time of all operations), where $C_{\max}$ 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 $C_{\max}/p_{\min}$ vertices. Therefore, its length (as a sum of weights) would be greater than $C_{\max}$, which is impossible because $C_{\max}$ is the length of the longest path in $G^*(\pi)$. We can use the upper bound of the objective function Λ instead of $C_{\max}$, as well as a minimal $p_{\min}$ value which is 1, because $C_{\max}/p_{\min} < \Lambda/1$ 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 $d \le \Lambda$ estimation and the number of processors $o/d \le o$. □

From the above theorem a surprising conclusion can be drawn, namely: if we can determine the upper bound of the objective function value Λ, the calculations take constant time $O(\Lambda) = O(1)$. The trivial upper bound Λ of the makespan is the sum of the processing times of all operations. Although the algorithm for determining the d value has computational complexity $O(\log^2 o)$ (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

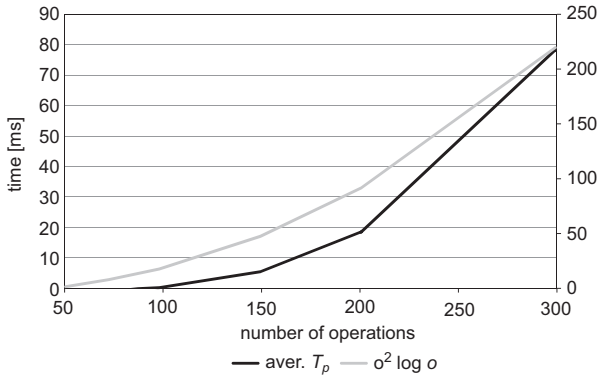

Fig. 6. Execution times comparison of the layers based procedure of $C_{\max}$ determination for the job shop problem on 480-processor GPU with scaled theoretical complexity.

value is changing after having executed an *insert* or *swap* move, and this can be done in constant time $O(1)$.

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 $O(\log^2 o)$, $O(\log o \log \Lambda)$, $O(d)$ and $O(\Lambda)$, where o is the number of operations, Λ 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 $f_{\max}$ 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

[1] Bożejko W. Solving the flow shop problem by parallel programming. Journal of Parallel and Distributed Computing 2009;69:470–81.

[2] Bożejko W, Pempera J, Smutnicki C. Parallel simulated annealing for the job shop scheduling problem. In: Allen G, editor. ICCS 2009, part I, Lecture notes in computer science, vol. 5544. Springer; 2009. p. 631–40.

[3] Bożejko W, Uchroński M. A neuro-tabu search algorithm for the job shop problem. In: Rutkowski L, editor. Proceedings of the ICAISC 2010, Lecture notes in artificial intelligence, vol. 6114. Springer; 2010. p. 387–94.

[4] Cole R. Parallel merge sort. SIAM Journal on Computing 1988;17(4):770–85.

[5] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms.MIT Press and McGraw-Hill; 1990.

[6] Fisher H, Thompson GL. Industrial scheduling. Englewood Cliffs, NJ: Prentice-Hall; 1963.

[7] Glover F, Kochenberger GA. Handbook of metaheuristics.Kluwer Academic Publishers; 2003.

[8] Grabowski J, Wodecki M. A very fast tabu search algorithm for job shop problem. In: Rego C, Alidaee B, editors. Metaheuristic optimization via memory and evolution. Tabu search and scatter search. Boston: Kluwer Academic Publishers; 2005. p. 117–44.

[9] Jain AS, Rangaswamy B, Meeran S. New and stronger job-shop neighborhoods: a focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics 2000;6(4):457–80.

[10] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania; 1984.

[11] Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 1996;91: 160–75.

[12] Nowicki E, Smutnicki C. An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 2005;8:145–59.

[13] Pezzella F, Merelli E. A tabu search method guided by shifting bottleneck for the job-shop scheduling problem. European Journal of Operational Research 2000;120:297–310.

[14] Steinhöfel K, Albrecht A, Wong CK. Fast parallel heuristics for the job shop scheduling problem. Computers and Operations Research 2002;29:151–69.

[15] Storer RH, Wu SD, Vaccari R. New search spaces for sequencing problems with application to job shop scheduling. Management Science 1992;38: 1495–509.

[16] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64:278–85.

[17] Yamada T, Nakano R. A genetic algorithm applicable to large-scale job shop problems. In: Manner R, Manderick B, editors. Parallel problem solving from nature II. Amsterdam: North-Holland; 1992. p. 281–90.