Chapter 3
Multi-GPU Tabu Search Metaheuristic for the Flexible Job Shop Scheduling Problem

Wojciech Bożejko, Mariusz Uchroński, and Mieczysław Wodecki

Abstract. We propose a new framework of the distributed tabu search metaheuristic designed to be executed using a multi-GPU cluster, i.e. cluster of nodes equipped with GPU computing units. The methodology is designed to solve difficult discrete optimization problems, such as a job shop scheduling problem, which we introduce to solve as a case study for the framework designed.

3.1 Introduction

The job shop problem can be briefly presented as follows (see [4]). There is a set of jobs and a set of machines. Each job consists of a number of operations which have to be processed in a given order, each one on a specified machine during a fixed time. The processing of an operation cannot be interrupted. Each machine can process at most one operation at a time. We want to find a schedule (the assignment of operations to time intervals on machines) that minimizes the *makespan*. The job shop scheduling problem, although relatively easily stated, is strongly NP-hard and it is considered as one of the hardest problems in the area of combinatorial optimization.

Because of NP-hardness of the problem heuristics and metaheuristics are recommended as 'the most reasonable' solution methods. The majority of these

Wojciech Bożejko
Institute of Computer Engineering, Control and Robotics,
Wrocław University of Technology Janiszewskiego 11-17, 50-372 Wrocław, Poland
e-mail: wojciech.bozejko@pwr.wroc.pl

Mariusz Uchroński
Wrocław Centre of Networking and Supercomputing, Wyb. Wyspańskiego 27,
50-370 Wrocław, Poland
e-mail: mariusz.uchronski@pwr.wroc.pl

Mieczysław Wodecki
Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland
e-mail: mwd@ii.uni.wroc.pl

R. Klempous et al. (eds.), *Advanced Methods and Applications in Computational Intelligence*,
Topics in Intelligent Engineering and Informatics 6,
DOI: 10.1007/978-3-319-01436-4_3,

methods refer to the makespan minimization. We mention here a few recent studies: Jain, Rangaswamy, and Meeran [20]; Pezzella and Merelli [28]; Grabowski and Wodecki [16]; Nowicki and Smutnicki [26]; Bożejko and Uchroński [6]. Heuristics algorithms based on dispatching rules are also proposed in papers of Holthaus and Rajendran [18], Bushee and Svestka [10] for the problem under consideration. For the other regular criteria such as the total tardiness there are proposed metaheuristics based on various local search techniques: simulated annealing [32], tabu search [2] and genetic search [25].

Here we propose the new framework of the distributed tabu search metaheuristic designed to solve difficult discrete optimization problems, such as the job shop problem, using a multi-GPU cluster with distributed memory. We also determine theoretical number of processors, for which the speedup measure has a maximum value. We experimentally determine what parallel execution time T_p can be obtained in real-world installations of multi-GPU clusters (i.e. nVidia Tesla S2050 with a 12-cores CPU server) for Taillard benchmarks [30] of the job shop scheduling problem, and compare them with theoretically determined values.

In the theoretical part of the chapter we consider the *Parallel Random Access Machine* (PRAM) model of the concurrent computing architecture, in which all processors have access to the common, shared memory with the constant access time $O(1)$. Additionally, we assume that we use the *Concurrent Read Exclusive Write* (CREW) model of the access to memory, in which processors can read concurrently from the same cell of memory, but it is forbidden to write concurrently to the same memory cell. Such a theoretical model is the most convenient for theoretical analysis, and it is very close to the practical hardware properties of the real GPU devices. The chapter constitutes a continuation of work presented in Bożejko et al. [5].

3.2 Job Shop Problem

Job shop scheduling problems result from many real-world cases, which means that they have good practical applications as well as industrial significance. Let us consider a set of jobs $\mathscr{J} = \{1,2,\ldots,n\}$, a set of machines $M = \{1,2,\ldots,m\}$ and a set of operations $\mathscr{O} = \{1,2,\ldots,o\}$. The set $\mathscr{O}$ 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 $p_i > 0$, $i \in \mathscr{O}$. Each machine can execute at most one operation at a time. A feasible solution constitutes a vector of times of the operation execution beginning $S = (S_1, S_2, \ldots, S_o)$ such that the following constraints are fulfilled

$$S_{l_{j-1}+1} \geq 0, \quad j = 1,2,\ldots,n, \tag{3.1}$$

$$S_i + p_i \leq S_{i+1}, \;\; i = l_{j-1}+1, \;\; l_{j-1}+2,\ldots,l_j-1, \;\; j = 1,2,\ldots,n, \tag{3.2}$$

$$S_i + p_i \leq S_j \quad \text{or} \quad S_j + p_j \leq S_i, \quad i, j \in O, \quad v_i = v_j, \quad i \neq j. \tag{3.3}$$

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 time of finishing all the jobs and minimization of the sum of job finishing times. From the formulation of the problem we obtain $\mathscr{C}_j \equiv C_{l_j}$, $j \in \mathscr{J}$.

The first criterion, the time of finishing all the jobs

$$C_{\max}(S) = \max_{1 \leq j \leq n} C_{l_j}, \tag{3.4}$$

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

$$C(S) = \sum_{j=1}^{n} C_{l_j}, \tag{3.5}$$

corresponds to the problem denoted as $J||\sum C_i$ in the literature.

Both problems described are strongly NP-hard and although they are similarly modelled, the second one is found to be harder because of the lack of some specific properties (so-called block properties, see [26]) used in optimization of execution time of solution algorithms.

3.2.1 Disjunctive Model

The disjunctive model 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 conjunctive arcs (directed) which show technological order of operation execution

$$U = \bigcup_{j=1}^{n} \bigcup_{i=l_{j-1}+1}^{l_j - 1} \{(i, i+1)\} \tag{3.6}$$

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

$$V = \bigcup_{i,j \in O, i \neq j, v_i = v_j} \{(i, j), (j, i)\}. \tag{3.7}$$

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 $\{(i, j), (j, i)\}$, 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 always feasible, if it includes cycles. A feasible solution is generated by a complete representation W such that the graph $G(W) = (O, U \cup W)$ is acyclic. For a feasible schedule values S_i of the vector of operations execution starting times S= (S_1, S_2, ..., S_o) can be determined as a length of the longest path incoming to the vertex i (without p_i). As the graph $G(W)$ includes o vertices and $O(o^2)$ arcs, therefore determining the value of the cost function for a given representation W takes the time $O(o^2)$ by using Bellman algorithm of paths in graphs determination.

3.2.2 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. It is void of redundance, characteristic of the disjunctive graph, that is, the situation where many disjunctive graphs 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 execution on all machines is defined as $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, where $\pi \in \Phi_n$, $\Phi_n = \Phi_n(M_1) \times \Phi_n(M_2) \times \ldots \times \Phi_n(M_m)$. 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

$$E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k - 1} \{(\pi_k(i), \pi_k(i+1))\} \tag{3.8}$$

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, the terms of operations' beginning can be determined in time $O(o)$ from the recurrent formula

$$S_j = \max\{S_i + p_i, S_k + p_k\}, j \in \mathcal{O}. \tag{3.9}$$

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. For a given feasible schedule π the process of determining the cost function value requires the time $O(o)$, which is thus shorter than for the disjunctive representation.

3.3 Flexible Job Shop Problem

Flexible job shop problem constitute a generalization (hybridization) of the classic job shop problem. In this section, we discuss a flexible job shop problem in which operations have to be executed on one machine from a set of dedicated machines. Next, as a job shop problem it also belongs to the strongly NP-hard class. Although exact algorithms based on a disjunctive graph representation of the solution have been developed (see Pinedo [29]), they are not effective for instances with more than 20 jobs and 10 machines.

Many approximate algorithms, chiefly metaheuristic, have been proposed (i.e. tabu search of Dauzère-Pérès and Pauli [12] and Mastrolilli and Gambardella [24]). Many authors have proposed a method of assigning operations to machines and then determining sequence of operations on each machine. This approach was followed by Brandimarte [8] and Pauli [27]. These authors solved the assignment problem (i.e., using dispatching rules) and next applied metaheuristics to solve the job shop problem. Genetic approaches have been adopted to solve the flexible job shop problem, too. Recent works are those of Jia et al. [21], Kacem et al. [22], Pezzella et al. [23] and Bożejko et al. [7]. Gao et al. [14] proposed the hybrid genetic and variable neighborhood descent algorithm for this problem.

3.3.1 Problem Formulation

The flexible job shop problem (FJSP), also called the general job shop problem with parallel machines, can be formulated as follows. Let $\mathscr{J} = \{1,2,\ldots,n\}$ be a set of jobs which have to be executed on machines from the set $\mathscr{M} = \{1,2,\ldots,m\}$. There exists a partition of the set of machines into types, so-called *nests* – subsets of machines with the same functional properties. A job constitutes a sequence of some operations. Each operation has to be executed on a dedicated type of machine (from the nest) within a fixed time. The problem consists in the allocation of jobs to machines of dedicated type and in determining the schedule of jobs execution on each machine to minimize the total jobs finishing time. The following constraints have to be fulfilled:

(*i*) each job has to be executed on only one machine of a determined type at a time,
(*ii*) machines cannot execute more than one job at a time,
(*iii*) there are no idle times (i.e., the job execution must not be broken),
(*iv*) the technological order has to be obeyed.

Let $\mathscr{O} = \{1,2,\ldots,o\}$ be the set of all operations. This set can be partitioned into sequences corresponding to jobs, where the job $j \in \mathscr{J}$ is a sequence of o_j operations, which have to be executed in an order on dedicated machines (i.e., in the so-called technological order). Operations are indexed by numbers

$(l_{j-1}+1,\ldots,l_{j-1}+o_j)$ where $l_j=\sum_{i=1}^{j} o_i$ is the number of operations of the first j jobs, $j=1,2,\ldots,n$, where $l_0=0$ and $o=\sum_{i=1}^{n} o_i$.

The set of machines $\mathscr{M}=\{1,2,\ldots,m\}$ can be partitioned into q subsets of the same type (*nests*) where the i-th $(i=1,2,\ldots,q)$ type $\mathscr{M}^i$ includes m_i machines which are indexed by numbers $(t_{i-1}+1,\ldots,t_{i-1}+m_i)$, where $t_i=\sum_{j=1}^{i} m_j$ is the number of machines in the first i types, $i=1,2,\ldots,q$, where $t_0=0$ and $m=\sum_{j=1}^{q} m_j$.

An operation $v\in\mathscr{O}$ has to be executed on machines of the type $\mu(v)$, i.e., on one of the machines from the set (nest) $\mathscr{M}^{\mu(v)}$ in time p_{vj}, where $j\in\mathscr{M}^{\mu(v)}$.

Let

$$\mathscr{O}^k=\{v\in\mathscr{O}:\ \mu(v)=k\} \tag{3.10}$$

be a set of operations executed in the k-th nest $(k=1,2,\ldots,q)$. A sequence of operations sets

$$\mathscr{Q}=(\mathscr{Q}^1,\mathscr{Q}^2,\ldots,\mathscr{Q}^m), \tag{3.11}$$

such that for each $k=1,2,\ldots,q$

$$\mathscr{O}^k=\bigcup_{i=t_{k-1}+1}^{t_{k-1}+m_k}\mathscr{Q}^i \text{ and } \mathscr{Q}^i\cap\mathscr{Q}^j=\emptyset,\ i\neq j,\ i,j=1,2,\ldots,m, \tag{3.12}$$

we call an *assignment of operations from the set $\mathscr{O}$ to machines from the set $\mathscr{M}$* (or shortly, machine workload).

A sequence $(\mathscr{Q}^{t_{k-1}+1},\mathscr{Q}^{t_{k-1}+2},\ldots,\mathscr{Q}^{t_{k-1}+m_k})$ is an *assignment* of operations to machines in the i-th nest (shortly, an assignment in the i-th nest). In a special case a machine can execute no operations; then a set of operations assigned to be executed by this machine is an empty set.

If the assignment of operations to machines has been completed, then the optimal schedule of operations execution determination (including a sequence of operations execution on machines) leads to the classic scheduling problem solving, that is, the job shop problem (see Section 3.2 and Grabowski and Wodecki [16]).

Let $K=(K_1,K_2,\ldots,K_m)$ be a sequence of sets where $K_i\in 2^{\mathscr{O}^i}$, $i=1,2,\ldots,m$ (in a particular case elements of this sequence can be empty sets). By $\mathscr{K}$ we denote the set of all such sequences. The number of elements of the set $\mathscr{K}$ is $2^{|\mathscr{O}^1|}\cdot 2^{|\mathscr{O}^2|}\cdot\ldots\cdot 2^{|\mathscr{O}^m|}$.

If $\mathscr{Q}$ is an assignment of operations to machines, then $\mathscr{Q}\in\mathscr{K}$ (of course, the set $\mathscr{K}$ includes also sequences which are not feasible; that is, such sequences do not constitute assignments of operations to machines).

For any sequence of sets $K=(K_1,K_2,\ldots,K_m)$ $(K\in\mathscr{K})$ by $\Pi_i(K)$ we denote the set of all permutations of elements from K_i. Thereafter, let

$$\pi(K)=(\pi_1(K),\pi_2(K),\ldots,\pi_m(K)) \tag{3.13}$$

be a concatenation of m sequences (permutations), where $\pi_i(K)\in\Pi_i(K)$. Therefore

$$\pi(K)\in\Pi(K)=\Pi_1(K)\times\Pi_2(K)\times,\ldots,\Pi_m(K). \tag{3.14}$$

It is easy to observe that, if $K = (K_1, K_2, \ldots, K_m)$ is an assignment of operations to machines, then the set $\pi_i(K)$ $(i = 1, 2, \ldots, m)$ includes all permutations (possible sequences of execution) of operations from the set K_i on the machine i. Further, let

$$\Phi = \{(K, \pi(K)) : K \in \mathscr{K} \wedge \pi(K) \in \Pi(K)\} \tag{3.15}$$

be a set of pairs where the first element is a sequence set and the second – a concatenation of permutations of elements of these sets. Any feasible solution of the FJSP is a pair $(\mathscr{Q}, \pi(\mathscr{Q})) \in \Phi$ where $\mathscr{Q}$ is an assignment of operations to machines and $\pi(Q)$ is a concatenation of permutations determining the operations execution sequence which are assigned to each machine fulfilling constraints (i)–(iv). By Φ° we denote a set of feasible solutions for the FJSP. Of course, there is $\Phi^\circ \subset \Phi$.

3.3.2 Graph Models

Any feasible solution $\Theta = (\mathscr{Q}, \pi(\mathscr{Q})) \in \Phi^\circ$ (where $\mathscr{Q}$ is an assignment of operations to machines and $\pi(\mathscr{Q})$ determines the operations execution sequence on each machine) of the FJSP can be presented as a directed graph with weighted vertices $G(\Theta) = (\mathscr{V}, \mathscr{R} \cup \mathscr{E}(\Theta))$ where $\mathscr{V}$ is a set of vertices and $\mathscr{R} \cup \mathscr{E}(\Theta)$ is a set of arcs with:

1) $\mathscr{V} = \mathscr{O} \cup \{s, c\}$, where s and c are additional (fictitious) operations which represent 'start' and 'finish', respectively. A vertex $v \in \mathscr{V} \setminus \{s, c\}$ possesses two attributes:
   - $\lambda(v)$ – a number of machines on which an operation $v \in \mathscr{O}$ has to be executed,
   - $p_{v,\lambda(v)}$ – a weight of vertex which equals the time of operation $v \in \mathscr{O}$ execution on the assigned machine $\lambda(v)$.

   Weights of additional vertices $p_s = p_c = 0$.

2) $$\mathscr{R} = \bigcup_{j=1}^{n} \left[\bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \cup \{(l_{j-1} + o_j, c)\} \right]. \tag{3.16}$$

   A set $\mathscr{R}$ includes arcs which connect successive operations of the job, arcs from vertex s to the first operation of each job and arcs from the last operation of each job to vertex c.

3) $$\mathscr{E}(\Theta) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|\mathscr{O}^k| - 1} \{(\pi_k(i), \pi_k(i+1))\}. \tag{3.17}$$

It is easy to notice that arcs from the set $\mathscr{E}(\Theta)$ connect operations executed on the same machine (π_k is a permutation of operations executed on the machine M_k, that is, operations from the set $\mathscr{O}^k$).

Arcs from the set $\mathscr{R}$ determine the operations execution sequence inside jobs (a technological order) and arcs from the set $\mathscr{E}(\pi)$ the operations execution sequence on each machine.

Remark 1. *A pair $\Theta = (\mathscr{Q}, \pi(\mathscr{Q})) \in \Phi$ is a feasible solution for the FJSP if and only if the graph $G(\Theta)$ does not include cycles.*

A sequence of vertices $(v_1, v_2, \ldots, v_k)$ in $G(\Theta)$ such that an arc $(v_i, v_{i+1}) \in \mathscr{R} \cup \mathscr{E}(\Theta)$ for $i = 1, 2, \ldots, k-1$, we call a *path* from vertex v_1 to v_k. By $C(v,u)$ we denote the longest path (called a *critical path* in a Operational Research issues) in the graph $G(\Theta)$ from the vertex v to u ($v, u \in \mathscr{V}$) and by $L(v,u)$ we denote a *length* (sum of vertex weights) of this path.

It is easy to notice that the time of all operations execution $C_{\max}(\Theta)$ related with the assignment of operations $\mathscr{Q}$ and schedule $\pi(\mathscr{Q})$ equals the length $L(s,c)$ of the critical path $C(s,c)$ in the graph $G(\Theta)$. A solution of the FJSP amounts to determining a feasible solution $\Theta = (\mathscr{Q}, \pi(\mathscr{Q})) \in \Phi^\circ$ for which the graph connected with this solution $G(\Theta)$ has the shortest critical path, that is, it minimizes $L(s,c)$.

Let $C(s,c) = (s, v_1, v_2, \ldots, v_w, c)$, $v_i \in \mathscr{O}$ $(1 \le i \le w)$ be a critical path in the graph $G(\Theta)$ from the starting vertex s to the final vertex c. This path can be divided into subsequences of vertices

$$\mathscr{B} = (B^1, B^2, \ldots, B^r), \tag{3.18}$$

called *blocks* in the permutations on the critical path $C(s,c)$ (Grabowski [15], Grabowski and Wodecki [16]) where:

(a) a block is a subsequence of verticesfrom the critical path including successive operations executed directly one after another,
(b) a block includes operations executed on the same machine,
(c) a product of any two blocks is an empty set,
(d) a block is a maximum (according to the inclusion) subset of operations from the critical path fulfilling constraints (a)–(c).

Next, only these blocks are considered for which $|B^k| > 1$, i.e., non-empty blocks. If B^k $(k = 1, 2, \ldots, r)$ is a block on the machine M_i $(i = 1, 2, \ldots, m)$ from the nest t $(t = 1, 2, \ldots, q)$, then we shall denote it as follows

$$B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k)), \tag{3.19}$$

where $1 \le a^k < b^k \le |Q^i|$. Operations $\pi(a^k)$ and $\pi(b^k)$ in the block B^k are called *the first* and *the last*, respectively. In turn, a block without the first and the last operation we call an *internal block*. The definitions given are presented in Figure 3.1.

In the work of Grabowski [15] there are theorems called *elimination criteria* of blocks in the job shop problem.

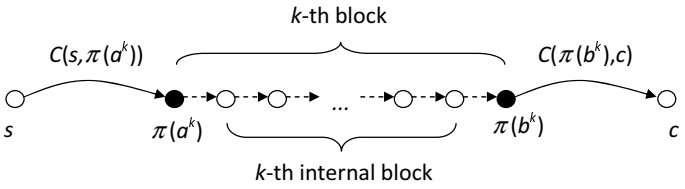


Fig. 3.1. Blocks on the critical path

Theorem 1 ([15]). *Let $\mathscr{B} = (B^1, B^2, \ldots, B^r)$ be a sequence of blocks of the critical path in the acyclic graph $G(\Theta)$, $\Theta \in \Phi^\circ$. If the graph $G(\Omega)$ is feasible (i.e., it represents a feasible solution) and if it is generated from $G(\Theta)$ by changing the order of operations execution on some machine and $C_{\max}(\Omega) < C_{\max}(\Theta)$ then in the $G(\Omega)$:*

(i) at least one operation from a block B^k, $k \in \{1,2,\ldots,r\}$ precedes the first element $\pi(a^k)$ of this block, or
(ii) at least one operation from a block B^k, $k \in \{1,2,\ldots,r\}$ occurs after the last element $\pi(b^k)$ of this block.

Changing the order of operations in any block does not generate a solution with lower value of the cost function. At least one operation from any block should be moved before the first or after the last operation of this block to generate a solution (graph) with smaller weight of the critical path. We use this property to reduce the neighborhood size, i.e., do not generate solutions with greater values (compared to the current solution) of the cost function.

3.4 Determination of the Cost Function

Both in classic job shop problem presented in the Section 3.2 and in flexible job shop problem from the Section 3.3 the method of the cost function calculation is similar because operations of machines assignment (in the flexible job shop problem) leads us to consider a number of job shop problems. Therefore, taking into consideration the constraints (3.1)–(3.3) presented in Section 3.2, it is possible to determine the time moments of job completion C_j, $j \in \mathscr{O}$ and job beginning S_j, $j \in \mathscr{O}$ in time $O(o)$ on the sequential machine using the recurrent formula

$$S_j = \max\{S_i + p_i, S_k + p_k\}, j \in \mathscr{O}. \tag{3.20}$$

where an operation i is a direct technological predecessor of the operation $j \in \mathscr{O}$ and an operation k is a directed machine predecessor of the operation $j \in \mathscr{O}$. The determination procedure of $S_j, j \in \mathscr{O}$ from the recurrent formula (3.20) 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 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 (3.20).

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 simplistic 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.

Definition 3.1. The layer of the graph $G(\pi)$ is a subsequence of the sequence of vertices ordered by the topological sort algorithm, such that there are no arcs between vertices of this subsequence.

Now we show another approach to determine cost function value, which is more time-consuming, but cost-optimal. First, we need to determine the number of layers d of the graph $G(\pi)$.

Theorem 2. *For a fixed feasible operations order π for the $J||C_{\max}$ problem, the number of layers from Definition 3.1 of the graph $G(\pi)$ can be calculated in time $O(\log^2 o)$ on the CREW PRAMs with $O\left(\frac{o^3}{\log o}\right)$ processors.*

Proof. Here we use the graph $G^*(\pi)$ with additional vertex 0. Let $B = [b_i j]$ be an incidence matrix for the graph $G^*(\pi)$, i.e., $b_i j = 1$, if there is an arc i, j in the graph $G^*(\pi)$, otherwise $b_i j = 0$, $i, j = 1, 2, \ldots, o$. 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 classic parallel Bellman-Ford algorithm in this step in 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 each 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 parallel merge sort algorithm. 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 $Topo[i+1] \neq Topo[i]$. 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 $O\left(\frac{o^3}{\log o}\right)$ of the CREW PRAMs. ■

Theorem 3. *For a fixed feasible operations order π for the $J||C_{\max}$ problem, the value of cost function can be determined in time $O(d)$ 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 $\lceil \frac{\Gamma_i}{p} \rceil$ groups, each group containing (at most) p elements, is $\lceil \frac{\Gamma_i}{p} \rceil$ (the last group cannot be full). Therefore, the total computation time in all d layers equals $\sum_{i=1}^{d} \lceil \frac{\Gamma_i}{p} \rceil \leq \sum_{i=1}^{d} (\frac{\Gamma_i}{p} + 1) = \frac{o}{p} + d$. 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 cost function value for the classic job shop problem with the makespan criterion. We will use it to determine the cost function value for the flexible job shop problem, after fixing of the operations-to-machines assignment.

3.5 Data Broadcasting

Here we propose a solution method to the flexible job shop problem in the distributed computing environments, such as multi-GPU clusters. Tabu search algorithm is executed in concurrent working threads, as in *multiple-walk* model of parallelization [1] (MPDS,*Multiple starting Point Different Strategies* in the Voß [31] classification of parallel tabu search metaheuristic). Additionally, MPI library is used to distribute calculations among GPUs (see Fig. 3.2). GPU devices are used for concurrent cost function calculations as it was mentioned in the Section 3.4.

Now let us consider a single cycle of the MPI data broadcasting, multi-GPU computations and batching up of the results obtained. Let us assume that the single communication procedure between two nodes of a cluster takes the time T_{comm}, the time of sequential tabu search computations is T_{seq} and the computations time of parallel tabu search is $T_{calc} = \frac{T_{seq}}{p}$ (p is the number of GPUs). Therefore, the total parallel computations time of the single cycle is

$$T_p = 2T_{comm} \log_2 p + T_{calc} = 2T_{comm} \log_2 p + \frac{T_{seq}}{p}.$$

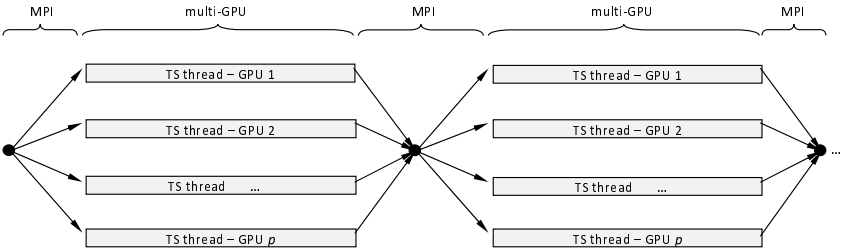
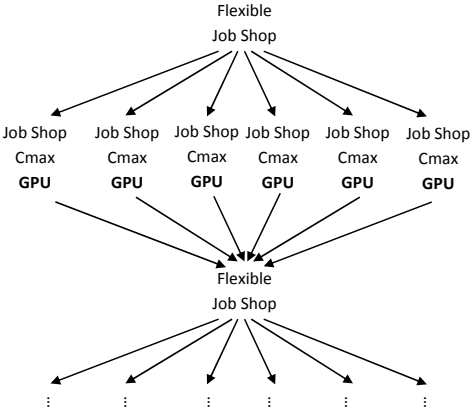


Fig. 3.2. Skeleton of the Multi-Level Tabu Search metaheuristic

Fig. 3.3 Scheme of the two levels parallelization model. Flexible job shop problem is converted into a number of classic job shop problems and distributed among GPU devices. Results are collected and converted into flexible job shop problem, etc.



In case of using more processors, the parallel computing time $\left(\frac{T_{seq}}{p}\right)$ decreases, whereas the time of communication $(2T_{comm}\log p)$ increases. We are looking for such a number of processors p (let us call it p^*) for which T_p is minimal. By calculating $\frac{\partial T_p}{\partial p} = 0$ we obtain

$$\frac{2T_{comm}}{p\ln 2} - \frac{T_{seq}}{p^2} = 0 \tag{3.21}$$

and then

$$p = p^* = \frac{T_{seq}\ln 2}{2T_{comm}}, \tag{3.22}$$

which provides us with an optimal number of processors p^* which minimizes the value of the parallel running time T_p.

The fraction of communication time is $O(\log_2 p)$ in this tree-based data broadcasting method, therefore this is another situation than for linear-time broadcasting (discussed in [3]), for which the overall communication and calculation efficiency is much lower. On the other hand the linear-time broadcasting is similar to described by Brooks' Law [9] for project management, i.e. the expected advantage from

splitting development work among n programmers is $O(n)$ but the communications cost associated with coordinating and then merging their work is $O(n^2)$.

3.6 Solution Method

Considered flexible job shop problem has been solved with a tabu search method. In each step of the tabu search method the neighbourhood of operations to machines assignment is generated (see Fig. 3.3). Each element of the neighbourhood generated in this way is a feasible solution of a classic job shop problem and its goal function is calculated in order to find the best solution from the neighbourhood. The value of cost function is determined on GPU. MPI library has been used for the communication implementation. The proposed tabu search method uses multiple point single strategy which means that MPI processes begins from different starting solutions using the same search strategy. For each MPI process one GPU is assigned for accelerating the goal function determination. For the best solution from neighbourhood TSAB algorithm is executed and solution obtained with this algorithm used as base solution in the next iteration. Also, after a number of iterations, the best solution with corresponding tabu list is broadcasted among MPI processes and each MPI process continues calculation from this solution. The best solution interchange between processors guarantees different search paths for each processor.

3.6.1 GPU Implementation Details

In our Multi-GPU tabu search metaheuristic for the flexible job shop problem we adopt the Floyd-Warshall algorithm for the longest path computation between each pair of nodes in a graph (see the proof of the theorem 2). The main idea of the algorithm can be formulated as follows. Find the longest path between nodes υ_i and υ_j containing the node υ_k. It consist of a sub-path from υ_i to υ_k and a sub-path υ_k to υ_j. This idea can be formulated with the following formula

$$d_{ij}^{(k)} = \max\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \tag{3.23}$$

where $d_{ij}^{(k)}$ is the longest path from υ_i to υ_j such that all intermediate nodes on the path are in set $\upsilon_1, \ldots, \upsilon_k$. Figure 3.4 shows CUDA implementation of the longest path computation between each pair of nodes. The CUDA kernel (Figure 3.5) is invoked o times, where o is the number of nodes in the graph. At the k-th iteration, the kernel computes two values for each pair of nodes in the graph: the direct distance between them and the indirect distance through the node υ_k. The larger of the two distances is written back to the distance matrix. The final distance matrix reflects the lengths of the longest paths between each pair of nodes. The inputs of the CUDA kernel constitutes the number of the graph nodes, path distance matrix and the iteration (step) number.

```
extern "C" void FindPathsOnGPU(const int &o, int *graph)
{
  int *graphDev;
  const int dataSize = (o+1)*(o+1)*sizeof(int);
  const int size=(int)(log((double)o)/log(2.0));
  dim3 threads(o+1);
  dim3 blocks(o+1);
  cudaMalloc( (void**) &graphDev, dataSize);
  cudaMemcpy( graphDev, graph, dataSize, cudaMemcpyHostToDevice);
  for (int iter=1; iter <= size+1; ++iter)
  {
    for(int k=0; k<=o; ++k)
    {
      PathsKernel<<<blocks, threads>>>(o, graphDev, k);
      cudaThreadSynchronize();
    }
  }
  cudaMemcpy( graph, graphDev, dataSize, cudaMemcpyDeviceToHost);
  cudaFree(graphDev);
}
```

Fig. 3.4. CUDA implementation of computing the longest path in a graph

```
__global__ void PathsKernel(const int o, int *graph, const int i)
{
  int x = threadIdx.x;
  int y = blockIdx.x;
  int k = i;
  int yXwidth = y * (o+1);

  int dYtoX = graph[yXwidth + x];
  int dYtoK = graph[yXwidth + k];
  int dKtoX = graph[k*(o+1) + x];

  int indirectDistance = dYtoK + dKtoX;
  int max = 0;
  int tmp = 0;

  if(dYtoK !=0 and dKtoX !=0)
  {
    tmp = indirectDistance;
    if(max < tmp)
      max = tmp;
  }
  if(dYtoX < max)
  {
    graph[yXwidth + x] = max;
  }
}
```

Fig. 3.5. CUDA kernel

3.6.2 Computational Experiments

Parallel multi–GPU algorithm for solving flexible job shop problem was coded in C (CUDA) and MPI library. The proposed algorithm was run on the Tesla S2050 GPU

and tested on the benchmark problems taken from Brandimarte [8] and Hurink [19]. The GPU was installed on the server based on the Intel Core i7 CPU X980 processor working under 64-bit Linux Ubuntu 10.04 operating system. Figure 3.6 and Table 3.1 report the comparison of speedup obtained for MPI+CUDA implementation on the Tesla S2050 GPU. For Tables 3.1 and 3.2 a particular column means:

- o – number of operations in considered flexible job shop problem instance,
- s_{CPU} – speedup value calculated for sequential CPU time,
- s_{GPU} – speedup value calculated for sequential GPU time (for one GPU thread).

The speedup value is given by the following formula:

$$s = \frac{T_{seq}}{T_{par}}, \tag{3.24}$$

where T_{seq} is the calculations time of the sequential algorithm and T_{par} is the calculations time of the parallel algorithm.

The obtained results show that the usage of parallel goal function computation method for computations acceleration in metaheuristics which solves flexible job shop problem results in shorter calculation time for the number of operations greater than 120 (bdata) and 200 (edata). The use of the parallel algorithm for goal function calculation in tabu search method results in about 2.5x (bdata) and 2.9x (edata) absolute speedup (in comparison with the sequential algorithm run on CPU) and

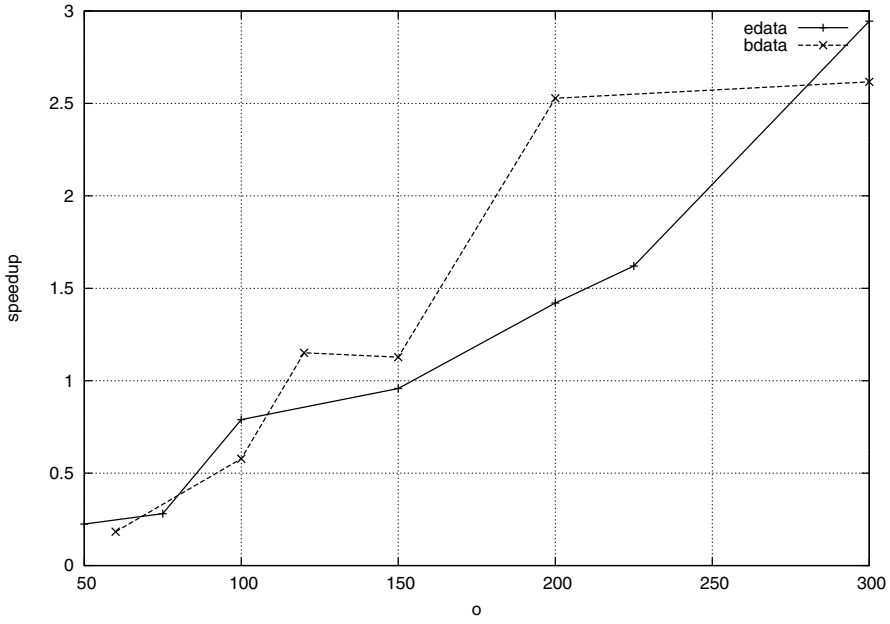


Fig. 3.6. Speedup for Brandimarte (*bdata*) and Hurink (*edata*)test instances

over 120x orthodox speedup (i.e. comparing with the sequential algorithm run on single GPU core, calculated for Mk01-Mk06 instances) for the flexible job shop problem instances with 200 and 300 operations.

Table 3.1. Speedup for MPI+CUDA implementation obtained on Tesla S2050 GPU

instance	o	s_{CPU}	s_{GPU}
Mk01	60	0.18	11.32
Mk02	60	0.19	12.53
Mk03	120	1.15	122.38
Mk04	120	0.46	34.20
Mk05	60	0.63	47.70
Mk06	150	1.12	121.33
Mk07	100	0.57	-
Mk08	200	2.52	-
Mk09	200	2.65	-
Mk10	300	2.61	-

Table 3.2. Speedup for MPI+CUDA implementation obtained on Tesla S2050 GPU

instance	$n \times m$	o	s_{CPU}
la01-05	10×5	50	0.2247
la06-10	15×5	75	0.2806
la11-15	20×5	100	0.4377
la16-20	10×10	100	0.7891
la21-25	15×10	150	0.9579
la26-30	20×10	200	1.4203
la31-35	30×10	300	2.9453
la36-40	15×15	225	1.6201

3.7 Conclusion

In this chapter we propose a framework designed to solve difficult problems of combinatorial optimization in distributed parallel architectures without shared memory, such as clusters of nodes equipped with GPU units (i.e. multiu-GPU clusters). The methodology can be especially effective for large instances of hard to solve optimization problems, such as flexible scheduling problems as well as discrete routing and assignment problems.

References

1. Alba, E.: Parallel Metaheuristics. A New Class of Algorithms. Wiley & Sons Inc. (2005)
2. Armentano, V.A., Scrich, C.R.: Tabu search for minimizing total tardiness in a job shop. International Journal of Production Economics 63(2), 131–140 (2000)
3. Bożejko, W.: A new class of parallel scheduling algorithms, pp. 1–280. Wroclaw University of Technology Publishing House (2010)
4. Bożejko, W.: On single-walk parallelization of the job shop problem solving algorithms. Computers & Operations Research 39, 2258–2264 (2012)
5. Bożejko, W., Uchroński: Distributed Tabu Search Algorithm for the Job Shop Problem. In: Proceedings of the 14th International Asia Pacific Conference on Computer Aided System Theory, Sydney, Australia, February 6-8 (2012)
6. Bożejko, W., Uchroński, M.: A Neuro-tabu Search Algorithm for the Job Shop Problem. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS, vol. 6114, pp. 387–394. Springer, Heidelberg (2010)
7. Bożejko, W., Uchroński, M., Wodecki, M.: Parallel Meta^{2}heuristics for the Flexible Job Shop Problem. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS, vol. 6114, pp. 395–402. Springer, Heidelberg (2010)
8. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41, 157–183 (1993)
9. Brooks Jr., F.P.: The Mythical Man-Month, anniversary edn. Addison-Wesley, Reading (1995)
10. Bushee, D.C., Svestka, J.A.: A bi-directional scheduling approach for job shops. International Journal of Production Research 37(16), 3823–3837 (1999)
11. Crainic, T.G., Toulouse, M., Gendreau, M.: Parallel asynchronous tabu search in multicommodity locationallocation with balancing requirements. Annals of Operations Research 63, 277–299 (1995)
12. Dauzère-Pérès, S., Pauli, J.: An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem using tabu search. Annals of Operations Research 70(3), 281–306 (1997)
13. Flynn, M.J.: Very highspeed computing systems. Proceedings of the IEEE 54, 1901–1909 (1966)
14. Gao, J., Sun, L., Gen, M.: A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research 35, 2892–2907 (2008)
15. Grabowski, J.: Generalized problems of operations sequencing in the discrete production systems. Monographs, vol. 9. Scientific Papers of the Institute of Technical Cybernetics of Wrocław Technical University (1979)
16. Grabowski, J., Wodecki, M.: A very fast tabu search algorithm for the job shop problem. In: Rego, C., Alidaee, B. (eds.) Adaptive Memory and Evolution, Tabu Search and Scatter Search. Kluwer Academic Publishers, Dordrecht (2005)
17. Hanafi, S.: On the Convergence of Tabu Search. Journal of Heuristics 7, 47–58 (2000)
18. Holthaus, O., Rajendran, C.: Efficient jobshop dispatching rules: further developments. Production Planning and Control 11, 171–178 (2000)
19. Hurink, E., Jurisch, B., Thole, M.: Tabu search for the job shop scheduling problem with Multi-purpose machine, Oper. Res. Spektrum 15, 205–215 (1994)
20. Jain, A.S., Rangaswamy, B., Meeran, S.: New and stronger job-shop neighborhoods: A focus on the method of Nowicki and Smutnicki (1996). Journal of Heuristics 6(4), 457–480 (2000)

21. Jia, H.Z., Nee, A.Y.C., Fuh, J.Y.H., Zhang, Y.F.: A modified genetic algorithm for distributed scheduling problems. International Journal of Intelligent Manufacturing 14, 351–362 (2003)
22. Kacem, I., Hammadi, S., Borne, P.: Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Transactions on Systems, Man, and Cybernetics, Part C 32(1), 1–13 (2002)
23. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the Flexible Job-schop Scheduling Problem. Computers & Operations Research 35, 3202–3212 (2008)
24. Mastrolilli, M., Gambardella, L.M.: Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling 3(1), 3–20 (2000)
25. Mattfeld, D.C., Bierwirth, C.: An efficient genetic algorithm for job shop scheduling with tardiness objectives. European Journal of Operational Research 155(3), 616–630 (2004)
26. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8(2), 145–159 (2005)
27. Pauli, J.: A hierarchical approach for the FMS schduling problem. European Journal of Operational Research 86(1), 32–42 (1995)
28. Pezzella, F., Merelli, E.: A tabu search method guided by shifting bottleneck for the job-shop scheduling problem. European Journal of Operational Research 120, 297–310 (2000)
29. Pinedo, M.: Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood Cliffs (2002)
30. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
31. Voß, S.: Tabu search: Applications and prospects. In: Du, D.Z., Pardalos, P.M. (eds.) Network Optimization Problems, pp. 333–353. World Scientific Publishing Co., Singapore (1993)
32. Wang, T.Y., Wu, K.B.: An eficient configuration generation mechanism to solve job shop scheduling problems by the simulated annealing. International Journal of Systems Science 30(5), 527–532 (1999)