Flexible job shop problem – parallel tabu search algorithm for multi-gpu

Wojciech Bożejko[1]
Mariusz Uchroński[2] and Mieczysław Wodecki[3*]

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

[2]Wrocław Centre of Networking and Supercomputing
Wyb. Wyspańskiego 27, 50-370 Wrocław, Poland

[3]Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland

Abstract

In the paper we propose a new framework for the distributed tabu search algorithm designed to be executed with the use of a multi-GPU cluster, in which cluster of nodes are equipped with multicore GPU computing units. The proposed methodology is designed specifically to solve difficult discrete optimization problems, such as a flexible job shop scheduling problem, which we introduce as a case study used to analyze the efficiency of the designed synchronous algorithm.

1 Introduction

The paper considers solving of a flexible job shop problem, which can be summarized as follows. There is a given set of tasks and a set of machines. Each task consists of a number of operations that must be performed on a machine from a set of dedicated machines in the given order. Operations'

*Corresponding author: Mieczysław Wodecki, email: mwd@ii.uni.wroc.pl

execution cannot be interrupted and the machine can perform at most one task at a given point in time. The aim is to find such a scheduling (assigning operations to machines in time) that minimizes the maximum completion time criterion of tasks' execution ($C_{\max}$). The difficulty lies within generalization of the classical job shop problem (*job shop*), thus it belongs to a class of strongly NP-complete problems.

The literature proposes a variety of methods, ranging from simple and fast priority algorithms to complex algorithms based on the division and constraint method. In addition, exact algorithms basing on presentation of the solution in the form of disjunctive graphs were proposed by Pinedo ([9]). Nevertheless, the method is ineffective in reference to time for instances of more than 20 tasks and 10 machines. There is also a big number of approximate, mainly metaheuristics, algorithms proposed. We can refer here to the works of Dauz'e re-Pérè the Pauli [4] Mastrolilli and Gambardella [7] (tabu search algorithm) and Gao et al. [6] (hybrid genetic algorithm.) The overview of current trends in paralleling of flexible job shop problem can be found in the monograph of Bożejko [2].

In this paper we propose a new model of a distributed tabu search algorithm (*tabu search*), of metaheurystics dedicated to solving difficult problems of discrete optimization, such as the considered flexible job shop problem, using 'a cluster architecture' consisting of nodes equipped with the GPU units (*multi-GPU*) with distributed memory. We determine the theoretical number of processors for which speedup measure (*speedup*) takes the maximum value. Computational experiments were conducted on the multi-GPU NVIDIA Tesla S2050 installations with a 6-core CPU processor.

2 Problem formulation

The considered problem of parallel ordering of machines denoted in the literature by $FJ|m|C_{\max}$, can be formulated as follows: there is a given a set of tasks $\mathcal{J} = \{1, 2, \ldots, n\}$ to be performed on machines from the set $\mathcal{M} = \{1, 2, \ldots, m\}$. There is a division of a machine set into types, i.e. into such subsets of machines that have the same functional properties. The task is a sequence of certain operations. Each operation must be performed on the appropriate type of machine in the set time. The problem lies within the allocation of tasks to machines of the appropriate type and the designation of order of operations on machines to minimize the execution time of all tasks.

Let $\mathcal{O} = \{1, 2, \ldots, o\}$ be the set of all operations. The set can be broken down into sequences corresponding to the tasks, where the task $j \in \mathcal{J}$ is a sequence of o_j operations, which will in turn be performed on the respective

machines (in the technological line). The operations are indexed by numbers $(l_{j-1}+1,\ldots,l_{j-1}+o_j)$, in which $l_j=\sum_{i=1}^{j}o_i$ is the number of first operations $j=1,2,\ldots,n$, where $l_0=0$, and $o=\sum_{i=1}^{n}o_i$. The set of machines $\mathcal{M}=\{1,2,\ldots,m\}$ can be broken into q subset of machines of the same type (*slots*), wherein i-th ($i=1,2,\ldots,q$) type $\mathcal{M}^i$ includes m_i machines, which are indexed with numbers $(t_{i-1}+1,\ldots,t_{i-1}+m_i)$, including $t_i=\sum_{j=1}^{i}m_j$ as the number of the first i type, $i=1,2,\ldots,q$, where $t_0=0$, and $m=\sum_{j=1}^{m}m_j$.

Operation $v\in\mathcal{O}$ should be performed in a slot $\mu(v)$, i.e. on one of the machines from the set $\mathcal{M}^{\mu(v)}$ in time $p_{v,j}$, where $j\in\mathcal{M}^{\mu(v)}$. Let

$$\mathcal{O}^k=\{v\in\mathcal{O}:\ \mu(v)=k\}$$

be a set of operations performed in k-th ($k=1,2,\ldots,q$) slot. The sequence of operations set

$$\mathcal{Q}=[\mathcal{Q}^1,\mathcal{Q}^2,\ldots,\mathcal{Q}^m],$$

such as for every $k=1,2,\ldots,q$

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

we call *an allocation of operation set $\mathcal{O}$ to the machines of the $\mathcal{M}$ set*. The sequence $[\mathcal{Q}^{t_{k-1}+1},\mathcal{Q}^{t_{k-1}+2},\ldots,\mathcal{Q}^{t_{k-1}+m_k}]$ is ***allocation*** of operations to machines in i-th slot (in short *allocation* in i-th slot). In some particular cases a given machine might not perform any operation. Then, in the process of operation allocation there is an empty set of operations in a slot to be performed by this particular machine.

If operations were allocated to the machines, then designation of the optimum time for the operation (including the order of operations on the machines) is reduced to the solution of the classical scheduling problem, namely the job shop problem.

Let $K=[K_1,K_2,...,K_m]$, be the sequence of sets, where $K_i\in 2^{\mathcal{O}^i},\ i=1,2,...,m$. In particular the elements of this sequence might be empty sets. By $\mathcal{K}$ we denote a set of all such sequences. Power of the $\mathcal{K}$ set equals $2^{|\mathcal{O}^1|}\times 2^{|\mathcal{O}^2|}\times ...\times 2^{|\mathcal{O}^q|}$.

If $\mathcal{Q}$ is a free allocation of operation to a machine, then $\mathcal{Q}\in\mathcal{K}$ (undoubtedly, the set $\mathcal{K}$ contains also sequences which are not acceptable, i.e. they are not allocations of operations to machines). For any sequence of sets $K=[K_1,K_2,...,K_m]$ ($K\in\mathcal{K}$) by $\Pi_i(K)$ we denote a set of all permutations of elements from K_i. Next, let

$$\pi(K)=(\pi_1(K),\pi_2(K),...,\pi_m(K))$$

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

$$\pi(K) \in \Pi(K) = \Pi_1(K) \times \Pi_2(K) \times, ..., \Pi_m(K).$$

It is easy to notice that if $K = [K_1, K_2, ..., K_m]$ is a certain allocation of operations to machines, then a set $\pi_i(K)$ $(i = 1, 2, ..., m)$ contains all the permutations (possible sequences of execution) of operation from the set K_i on i machine. Next, let

$$\Phi = \{(K, \pi(K)) : \ K \in \mathcal{K} \ \wedge \ \pi(K) \in \Pi(K)\},$$

be the set of pairs whose first element is a sequence of sets, the second - concatenation of permutations of the elements of these sets. Any feasible acceptable solution of the problem is a pair $(\mathcal{Q}\pi(\mathcal{Q})) \in \Phi$, where $\mathcal{Q}$ is the assignment of operations to machines, and $\pi(\mathcal{Q})$ is concatenation of permutations designating the order of operations assigned to each machine. By $\Phi^\circ \subset \Phi$ we denote the set of feasible solutions.

2.1 Graph representation of the solution

Any feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi^\circ$ (where $\mathcal{Q}$ is an assignment of operations to machines, and $\pi(\mathcal{Q})$ is a sequence of operations' execution on each machine) of the considered problem can be presented in the form of a directed graph with burdened vertices (of network) $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$, where $\mathcal{V}$ is a set of vertices, and $\mathcal{R} \cup \mathcal{E}(\Theta)$ is a set of arcs, wherein:

1) $\mathcal{V} = \mathcal{O} \cup \{s, c\}$, where s i c are additional (fictitious) operations representing, respectively, the 'start'and 'end'. Vertex $v \in \mathcal{V} \setminus \{s, c\}$ can be characterized with two features:

   - $\lambda(v)$ – number of the machine on which we perform the operation$v \in \mathcal{O}$,
   - $p_{v,\lambda(v)}$ – weight of vertex equalling to time of operation $v \in \mathcal{O}$ on $\lambda(v)machine$.

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

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

   The set $\mathcal{R}$ has arcs connecting consecutive operations of the same task, and arcs from the vertex s to the first operation of each task and the arcs of the last operation of each task to the top of the c.

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

It is easy to notice that arcs from the set $\mathcal{E}(\Theta)$ combine operations performed on the same machine (π_k is a permutation of the operations performed on the machine M_k, i.e., the operation of a set $\mathcal{O}^k$).

Arcs from the set $\mathcal{R}$ determine the order of operations' execution in the tasks (technological order), and arcs from the set $\mathcal{E}(\Theta)$ the order of operations on each machine.

Remark 1 *A pair of* $\Theta = (\mathcal{Q}\pi(\mathcal{Q})) \in \Phi$ *is a feasible solution to the considered problem if and only when the graph* $G(\Theta)$ *does not contain cycles* .

The sequence of vertices $(v_1, v_2, \ldots, v_k)$ graph $G(\Theta)$ such that $(v_i, v_{i+1}) \in \mathcal{R} \cup \mathcal{E}(\Theta)$ for $i = 1, 2, \ldots, k-1$ is called *a path* (or *a thread*) from vertex v_1 to v_k.

By $C(v, u)$ we denote the longest path (called the *critical path*) in the graph graph $G(\Theta)$ of vertex v to u ($v, u \in \mathcal{V}$) and the $L(v, u)$ *length* (the sum of the weights of vertices) of the path.

It is easy to see that the time to perform all the operations $C_{\max}(\Theta)$ according to the allocation of operations $\mathcal{Q}$ and the order (scheduling) $\pi(\mathcal{Q})$ is equal to the length $L(s, c)$ of a critical path $C(s, c)$ in the graph $G(\Theta)$. Solving the job shop problem with parallel machine boils down to determining such feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$, for which the corresponding graph $G(\Theta)$ has the shortest critical path, i.e. minimizing the $L(s, c)$.

3 Parallel tabu search algorithm

In general, the method of tabu search (*tabu search*) is an iterative improvement of the current solution by local search. It begins with a certain initial (starting) solution. Then, we generate its surroundings (neighborhood) and determine the best solution in this neighborhood, which is taken as a starting solution for the next iteration. It is possible to increase the value of the goal function (in determining of a new starting solution), so as to increase the chance of achieving the global minimum. Such movements 'upwards' should be nevertheless controlled in some way, because otherwise after reaching a local minimum there would be rapid return to it. To prevent in new iterations the generation of solutions recently considered (the formation of cycles), we retain them in memory (their attributes) on the list of banned solutions, so-called tabu list (short term memory).

The first attempt to classify parallel tabu algorithms was made by Voß [10], referring to the classic division of parallel algorithms by Flynn [5] into SIMD, MIMD, MISD and SISD models. Voß's classification is placed 'close to' the general classification of parallel metaheuristic methods taking as a subject matter the distribution of number of paths, granulation and cooperation. Voß proposed the division of parallel tabu algorithms into four categories in reference to the fact whether parallel search threads compete for the same or different starting solutions and whether they use the same or different search strategy:

- SPSS (*Single (Initial) Point Single Strategy*) – one initial solution, one search strategy; model allowing for parallelism only at the lowest level, such as counting goal functions or parallel neighborhood search,
- SPDS (*Single (Initial) Point Different Strategies*) – all processors start with the same initial solution, but they use different search strategies (e.g., different lengths of tabu list, different items stored in the tabu list, etc.)
- MPSS (*Multiple (Initial) Point Single Strategy*) – processors begin operation from different initial solutions, using the same search strategy,
- MPDS (*Multiple (Initial) Point Different Strategies*) – processors begin operation from different initial solutions, using different search strategies, it is the widest class, embracing all previous categories as its special cases.

We propose the use of a distributed algorithm, tabu search multitrack version (model *multiple-walk*, Alba [1]) based on the MPDS model, *Multiple starting Point Different Strategies*. In addition, the MPI (*Message Passing Interface*) library was used for communication between distributed computing threads run on the GPU devices concurrently counting the value of the goal function.

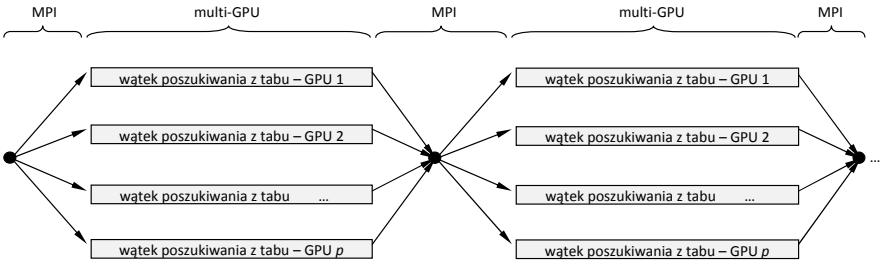

Figure 1: Skeleton of the Multi-Level Tabu Search metaheuristic.

Let us consider a single cycle of data broadcast with use of the MPI communication library, implementing tree-based broadcast scheme (*broadcasting*), and then we take into consideration the computation time on a multi-GPU cluster and collecting (including tree-diagram of logarithmic time complexity) of the obtained results. Let us suppose that a single communication procedure between two nodes of the cluster takes time T_{comm}, the sequence tabu search algorithm takes T_{seq} and parrallel $T_{parallelcalc} = \frac{T_{seq}}{p}$ (p is the number of cores in GPU devices.) Thus, the total computation time in a single communication cycle equals

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

By increasing the number of processors parallel computation time $\left(\frac{T_{seq}}{p}\right)$ is decreasing, while the communication time ($2T_{comm} \log p$) is increasing. We look for such number of processors p (let us call it p^*) in which T_p is minimal. By designating $\frac{\partial T_p}{\partial p} = 0$ we obtain

$$\frac{2T_{comm}}{p \ln 2} - \frac{T_{seq}}{p^2} = 0 \tag{1}$$

and then

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

which gives us the optimal number of processors p^* that minimizes the value of the parallel computing time T_p.

4 Solution method

We propose a solution to the considered flexible job shop problem divided in two stages, firstly, considering the problem of assigning operations to machines, secondly, solving the classic job shop problem (*job shop*) generated for this assignment. With the use of the tabu search algorithm in each step there is a neighborhood generated representing the assignment of operations to machines. Every element of the neighborhood represents a classic job shop problem (the second stage), and the best solution found for this problem which makes it possible to select the best elemnt of the neighborhood in the first stage. After scattering the calculations on a cluster using the MPI communication library, the value of the goal function is determined in one of the GPU. Proposed tabu search method uses MPDS strategy according to Vo ßclassification, in which optimization of classic job shop problem is made with the use of TSAB algorithm proposed by Nowicki and Smutnicki [8].

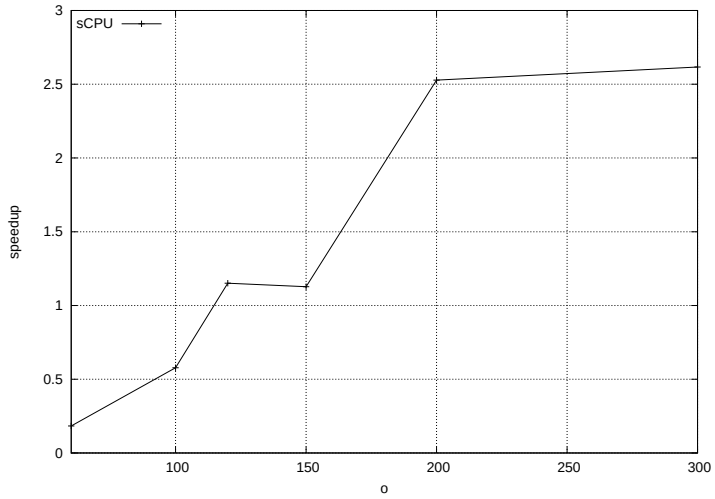

Figure 2: Speedup for test examples of Brandimart [3].

5 Computational experiments

A parallel algorithm for multi-GPU solving of a flexible job shop problem was implemented in C language with CUDA and MPI parallel computing libraries run on multi-GPU NVIDIA Tesla S2050 installation with a computing server equipped with 6-core Intel Core i7 X980 CPU, working under control of a 64 -bit Linux operating system Ubuntu 10.04. Test instances were taken from the work of Brandimarte [3]. Figure 2 and Table 1 present the speedup values (*speedup*) obtained for the implementation of MPI + CUDA Tesla S2050 GPU. Corresponding columns in Table 1 mean:

- o – number of operations in the considered instance of flexible job shop problem,
- s_{CPU} – value of absolute speedup (i.e., compared to the time of sequential algorithm run on the CPU),
- s_{GPU} – value of relative, orthodox speedup (compared to the time of sequential algorithm run on a single core GPU).

The speedup value was determined on the basis of formula: $s = \frac{T_{seq}}{T_{par}}$, where T_{seq} is the time of calculation of sequential algorithm and T_{par} is the time of the parallel algorithm running.

The obtained results indicate that the use of parallel methods for determining the value of the goal function in metaheuristics solving flexible job

Table 1: Speedup for MPI+CUDA implementation on GPU Tesla S2050.

instance	o	s_{CPU}	s_{GPU}	instance	o	s_{CPU}	s_{GPU}
Mk01	60	0.18	11.32	Mk06	150	1.12	121.33
Mk02	60	0.19	12.53	Mk07	100	0.57	49.92
Mk03	120	1.15	122.38	Mk08	200	2.52	335.24
Mk04	120	0.46	34.20	Mk09	200	2.65	-
Mk05	60	0.63	47.70	Mk10	300	2.61	-

shop problem resulted in shortening of computational time for number of more than 120 operations. The obtained mean absolute speedup of 2.5-fold (in reference to the CPU), and more than 120-fold relative orthodox speedup (in reference to the GPU) for instances of 200 and 300 operations.

6 Summary

In this paper we propose a parallel algorithm for solving difficult problems of discrete optimization, such as tasks scheduling flexible job shop problem in parallel and distributed architectures without shared memory. Example of this type of architecture are the cluster nodes equipped with GPU devices (so-called multi-GPU clusters), recently growing in popularity. The presented multi-step methodology for the construction of parallel algorithms can be particularly effective for large-scale instances of difficult discrete optimization problems, such as flexible scheduling problems with parallel machines and discrete transport systems.

References

[1] Alba E.: Parallel Metaheuristics. A New Class of Algorithms. Wiley & Sons Inc. (2005).

[2] Bożejko W.: A new class of parallel scheduling algorithms. Wrocław University of Technology Publishing House, 1–280 (2010).

[3] Brandimarte P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41, 157–183 (1993).

[4] 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).

[5] Flynn M.J.: Very highspeed computing systems. Proceedings of the IEEE 54, 1901–1909 (1966).

[6] 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).

[7] Mastrolilli M., Gambardella L.M.: Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling 3(1), 3–20 (2000).

[8] Nowicki E., Smutnicki C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8(2), 145–159 (2005).

[9] Pinedo M.: Scheduling: theory, algorithms and systems, Englewood Cliffs, NJ, Prentice-Hall (2002).

[10] Voß S.: Tabu search: Applications and prospects. In: D.Z. Du and P.M. Pardalos (eds.), Network Optimization Problems, pp. 333–353. World Scientific Publishing Co., Singapore (1993).