Leszek Rutkowski Marcin Korytkowski
Rafał Scherer Ryszard Tadeusiewicz
Lotfi A. Zadeh Jacek M. Zurada (Eds.)

LNAI 7268

Artificial Intelligence and Soft Computing

11th International Conference, ICAISC 2012
Zakopane, Poland, April/May 2012
Proceedings, Part II

2 Part II

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Leszek Rutkowski
Marcin Korytkowski
Rafał Scherer
Częstochowa University of Technology, Poland
E-mail: lrutko@kik.pcz.czest.pl,
{marcin.korytkowski, rafal.scherer}@kik.pcz.pl

Ryszard Tadeusiewicz
AGH University of Science and Technology, Kraków, Poland
E-mail: rtad@agh.edu.pl

Lotfi A. Zadeh
University of California, Berkeley, CA, USA
E-mail: zadeh@cs.berkeley.edu

Jacek M. Zurada
University of Louisville, KY, USA
E-mail: jacek.zurada@louisville.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-29349-8 e-ISBN 978-3-642-29350-4
DOI 10.1007/978-3-642-29350-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012934672

CR Subject Classification (1998): I.2, H.3, F.1, I.4, H.4, I.5

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Solving the Flexible Job Shop Problem on GPU

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

[1] Institute of Computer Engineering, Control and Robotics,
Wrocław University of Technology,
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

[2] Wrocław Centre of Networking and Supercomputing,
Wyb. Wyspańskiego 27, 50-370 Wrocław, Poland
mariusz.uchronski@pwr.wroc.pl

[3] Institute of Computer Science, University of Wrocław,
Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

Abstract. In this work we examine a model of flexible job shop problem in which for a given operation there is a possibility of a choice of the machine on which this operation will be carried out. This problem is a generalization of the classic job shop problem. We present a tabu search algorithm in which "a golf neighborhood" was applied. Because it is a huge neighborhood, the concurrent programming tools based on GPU platform were thus used to its searching. The computational results indicate that by acceleration of computations with the utilization of GPU one obtains very good values of a speedup. Computational experiments executed on benchmark instances show the efficiency of this approach.

1 Introduction

There is a set of tasks and machines of various types given. A flexible job shop problem (FJS problem in short) consists in assigning tasks to adequate machines and determining an optimal sequence of its execution. Machines of the same type, that is executing only one type of operation, create production nests. Tasks' operations must be executed on adequate machines in a fixed order called a production itinerary. For each operation there is created a nest in which this operation will be carried out. Thus, the times of execution of operations are given.

FJS problem consists in assigning operations to machines and defining the sequence of their execution in order to minimize the time of finishing tasks' completion ($C_{\max}$). The problem presented in this work also belongs to the strongly NP -hard class.

Although the exact algorithms based on a disjunctive graph representation of the solution have been developed (see Adrabiński and Wodecki [1], Pinedo [8]), they are not effective for instances with more than 20 jobs and 10 machines. From metaheuristic algorithms, Nowicki and Smutnicki [7] proposed a tabu search approach using block properties for the special case of the considered problem.

2 Flexible Job Shop Problem

A flexible job shop problem (FJSP), also called a general job shop problem with parallel machines, can be formulated as follows. Let $\mathcal{J} = \{1, 2, \ldots, n\}$ be a set of jobs which have to be executed on machines from the set $\mathcal{M} = \{1, 2, \ldots, m\}$. There exists a partition of the set of machines into types, i.e. subsets of machines with the same functional properties. A job constitutes a sequence of some operations. Each operation has to be executed on an adequate type of machine (nest) within a fixed time. The problem consists in allocating the jobs to machines of an adequate type and scheduling of jobs execution determination on each machine to minimize a total jobs' finishing time ($C_{\max}$).

Let $\mathcal{O} = \{1, 2, \ldots, o\}$ be the set of all operations. This set can be partitioned into sequences which correspond to jobs where the job $j \in \mathcal{J}$ is a sequence of o_j operations which have to be executed in an order on dedicated machines.

The set of machines $\mathcal{M} = \{1, 2, \ldots, m\}$ can be partitioned into q subsets of the same type (*nests*) where i-th type $\mathcal{M}^i$. An operation $v \in \mathcal{O}$ has to be executed on the machines type $\mu(v)$, i.e. on one of the machines from the set (nest) $\mathcal{M}^{\mu(v)}$ in the 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 executed in the k-th nest ($k = 1, 2, \ldots, q$). A sequence of operations sets being a partition of $\mathcal{O}$ (pair-disjoint)

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

we call an *assignment of operations from the set $\mathcal{O}$ to machines from the set $\mathcal{M}$.*

For an assignment of operation $\mathcal{Q}$, let $\pi(\mathcal{Q}) = (\pi_1(\mathcal{Q}), \pi_2(\mathcal{Q}), \ldots, \pi_m(\mathcal{Q}))$ where $\pi_i(\mathcal{Q})$ is a permutation of operations executed on a machine M_i. Any feasible solution of the FJSP is a pair $(\mathcal{Q}, \pi(\mathcal{Q}))$ where $\mathcal{Q}$ is an assignment of operations to machines and $\pi(Q)$ is a permutations concatenation determining the operations execution sequence which are assigned to each machine (see [3]).

3 Solution Method

There is an exponential number of possible jobs to machines assignments, due to the number of operations. Each feasible assignment generates a NP-hard problem (job shop) whose solution consists in determining an optimal jobs execution order on machines. One has to solve an exponential number of NP-hard problems.

Therefore, we will apply an approximate algorithm based on the tabu search method. The main element of this approach is a neighborhood – subsets of all feasible solutions set, generated from the current solution by transformations called *moves*. By searching a neighborhood we choose an element with the lowest cost function value, which we take as a new current solution in the next iteration of the algorithm. It is possible to generate another solution from the fixed solutiont by executing a move which consists in:

1. moving (transferring) an operation from one machine into another machine in the same nest (of the same type), or
2. changing operations execution order on machines.

Let $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ be a feasible solution, where $\mathcal{Q} = [\mathcal{Q}^1, \mathcal{Q}^2, \ldots, \mathcal{Q}^m]$ is an assignment of operations to machines, o_i is a number of operations executed on a machine M_i (i.e. $\pi(\mathcal{Q}) = (\pi_1(\mathcal{Q}), \pi_2(\mathcal{Q}), \ldots, \pi_m(\mathcal{Q}))$ is a concatenation of m permutations). A permutation $\pi_i(\mathcal{Q})$ determines an order of operations from the set $\mathcal{Q}^i$ which have to be executed on the machine M_i.

3.1 Transfer Type Moves

By $t^i_j(k, l)$ we define a *transfer* type move (*t-move*) which consist in moving of an operation from the position k in a permutation π_i into position l in a permutation $\pi_j(k)$. Execution of the move $t^i_j(k, l)$ generates from $\Theta = (\mathcal{Q}, \pi)$ a new solution $\Theta' = (\mathcal{Q}', \pi')$. Its computational complexity is $O(n)$.

If τ is a *t-move*, then we define a solution generated from Θ by execution of the τ move by $\tau(\Theta)$. It is possible that the solution $\tau(\Theta)$ is not feasible.

Let Θ be a feasible solution. The set

$$\mathcal{T}^i_j(\Theta) = \{t^i_j(k, l) : \ k \in Q^i \text{ and } l \in Q^j\}$$

includes all *t-moves* which transfers operations from a machine M_i into a machine M_j and

$$T(\Theta) = \sum^m_{i,j} \mathcal{T}^i_j(\Theta)$$

includes all *t-moves* for the solution Θ. The number of elements of this set has an upper bound $O(qm^2o^2)$.

3.2 Insert Type Moves

In order to simplify the problem, let us assume that a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(t))$ determines an operations' execution order on a machine.

Insert type move i^k_l (*i-move*) is moving an element $\pi(k)$ into the position l, generating a permutation $i^k_l(\pi) = \pi^k_l$. The number of all such moves (for a determined machines) is $t(t-1)$. For a fixed feasible solution Θ, let $\mathcal{I}_k(\Theta)$ be a set of all *i-moves* for the machine $M_k \in \mathcal{M}$ and let

$$I(\Theta) = \sum^m_{k=1} \mathcal{I}_k(\Theta)$$

be a set of all *i-moves* on all machines.

3.3 Graph Models

Any feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q}))$ (determining the operations' execution sequence on each machine) of the FJSP can be presented as a directed graph with weighted vertices $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$ where $\mathcal{V}$ is a set of vertices and a $\mathcal{R} \cup \mathcal{E}(\Theta)$ is a set of arcs, with:

1) $\mathcal{V} = \mathcal{O} \cup \{s, c\}$, where s and c are additional operations which represent 'start' and 'finish', respectively. A vertex $v \in \mathcal{V}$ possesses two attributes:
   - $\lambda(v)$ – a number of machines on which an operation v has to be executed,
   - $p_{v,\lambda(v)}$ – a weight of vertex which equals the time of operation $v \in \mathcal{O}$ execution on the assigned machine $\lambda(v)$ ($p_s = p_c = 0$).

2) A set $\mathcal{R}$ 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) Arcs from the set $\mathcal{E}(\Theta)$ connect operations executed on the same machine.

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

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

3.4 Golf Neighborhood

A transfer type move is like a long shot in golf: it moves an operation into other machines. In turn, an insert type move makes only a little modification of operation sequence on machines. An inspiration to these researches was a paper of Bożejko and Wodecki [2] which considered multimoves.

Let Θ be a feasible solution and let $\mathcal{T}(\Theta)$ be a set of all *t-moves*. We consider the move $t_j^i(k,l) \in \mathcal{T}(\Theta)$. It transfers an operation from the position k-th on the machine M_i into a position l-th on the machine M_j. This move generates the solution $\Theta' = t_j^i(k,l)(\Theta)$. The set $\mathcal{I}(\Theta')$ includes all *i-moves* connected with a solution Θ' and $\mathcal{I}_j(\Theta')$ – *i-moves* defined on operations executed on the machine M_j. Let $i_t^s \in \mathcal{I}_j(\Theta')$ be an *i-move*. Its execution generates a new solution $\Theta'' = i_t^s(\Theta')$. The transformation which generates a solution Θ'' from the Θ we call an *it-multimove*. It constitutes a product of the *t-move* $t_j^i(k,l)$ and *i-move* i_t^s. We will denote this move shortly as $i_t^s \circ t_j^i(k,l)$. Therefore $\Theta'' = i_t^s \circ t_j^i(k,l)(\Theta)$.

By $\mathcal{I} \circ \mathcal{T}(\Theta)$ we denote a set of all *it-multimoves* determined for a solution Θ. *The golf neighborhood* Θ is the set

$$\mathcal{N}(\Theta) = \{\lambda(\Theta) : \ \lambda \in \mathcal{I} \circ \mathcal{T}(\Theta)\}. \tag{1}$$

In the paper Bożejko et al. [3] a so-called elimination criterion was proved. It allows to eliminate all *i-moves* and *t-moves*, which generate unfeasible solutions. Moreover, multimoves are generating solutions, for which the cost function value is not less than $C_{\max}(\Theta)$ are also eliminated. Such a determined golf neighborhood will be applied to the tabu search algorithm.

3.5 Neighborhood Determination

Execution of a *t-move* can lead to a non-feasible solution, i.e., a graph connected with this solution can have a cycle. Therefore, checking feasibility equals checking if a graph has a cycle.

Let $\Theta = (\mathcal{Q}, \pi)$ be a feasible solution. We consider two machines M_i and M_j from the same nest. A permutation π_i determines a processing order of operations from the set $\mathcal{Q}^i$ on the machine M_i and π_j – a processing order of operations from the set $\mathcal{Q}^j$. For an operation $\pi_i(k) \in \mathcal{Q}^i$ we define two parameters connected with paths in the graph $G(\Theta)$. The first parameter is

$$\eta_j(k) = \begin{cases} 1, & \text{if there is no path } C(\pi_j(v), \pi_i(k)) \ \forall\, v = 1, 2, \ldots, \varrho_j, \\ 1 + \max\limits_{1 \le v \le \varrho_j} \{\text{there is a path } C(\pi_j(v), \pi_i(k))\}, & \text{otherwise.} \end{cases} \tag{2}$$

Thus, there is no path to the operation (vertex) $\pi_i(k)$ from any of the operations placed in the permutation π_j in positions $\eta_j(k), \eta_j(k) + 1, \ldots, \varrho_j$.

The second parameter is

$$\rho_j(k) = \begin{cases} 1 + \varrho_j, & \text{if there is no path } C(\pi_j(v), \pi_i(k)) \ \forall\, v = 1, 2, \ldots, \varrho_j, \\ 1 + \min\limits_{\eta_j(k) \le v \le \varrho_j} \{\text{there is a path } C(\pi_i(k), \pi_j(v))\}, & \text{otherwise.} \end{cases} \tag{3}$$

From the definition formulated above it follows that in the graph there is no path from a vertex $\pi_i(k)$ to any operation placed in positions $\eta_j(k), \eta_j(k)+1, \ldots, \rho_j(k)$ in the permutation π_j.

Theorems for characterizing a *t-moves* whose execution generates an unfeasible solutions can be found in our previous work [3]. The structure of assumptions allows an easy implementation in the parallel computing environment, such as GPUs. Basically, in order to check solution feasibility the information about the longest paths between all vertexes in graph connected with flexible job shop problem solution are needed. Calculating the longest paths between all vertexes in graph (Floyd–Warshall algorithm) with sequential algorithm takes the time $O(o^3)$. Parallel algorithm using o^2 processors calculate the longest paths between all vertexes in graph with the time $O(o)$. This algorithm can be easily implemented in the parallel computation environment such as GPU.

4 Computational Experiments

Proposed algorithm for the flexible job shop problem was coded in C++ and C(CUDA) and run on HP workstation with CPU and NVIDIA GTX480 GPU. Algorithms were tested on the set of benchmark problem instances taken from Brandimarte [5]. Table 1 shows computational times for parallelized part of golf neighborhood determination procedure. Particular columns in Table 1 denote:

- p - number of GPU threads,
- t_s - computational time for sequential algorithm on CPU,
- t_p - computational time for parallel algorithm on GPU,
- s - speedup. A measured speedup increases with a number of processors (in all cases it is greater than one) but still quite small in comparison with a number of used GPU processors. There are two reasons for a relative small speedup in considered parallel GPU algorithm. Firstly, it is connected with the time needed for transferring data between CPU and GPU. This is a well known bottleneck of GPU computations. The second reason is an unoptimized access to the GPU memory during computation for this particular implementation.

Table 1. Speedup for parallelized part of algorithm

problem	p	$t_s[ms]$	$t_p[ms]$	s
Mk01	55	5.98876	4.32949	1.38325
Mk02	58	7.09983	4.46129	1.59143
Mk03	150	154.99	15.8564	9.7746
Mk04	90	29.8697	7.68228	3.88813
Mk05	106	51.3419	9.52336	5.39115
Mk06	150	150.047	16.5776	9.05119
Mk07	100	43.043	8.79722	4.8928
Mk08	225	517.379	25.3213	20.4326
Mk09	240	628.237	27.3808	22.9444
Mk10	240	637.619	27.6397	23.069

A parallel speedup of the algorithm is a measure of the success of the parallelization process. All algorithms contain some parts that can be parallelized and some parts which cannot undergo this process. The time spent in the parallelized parts of the algorithm is reduced by using increasing number of processors, but the sequential parts remain the same. Finally, the execution time of the algorithm is dominated by the time taken to compute the sequential part, which is an upper limit of the expected speedup. This effect is known as Amdahl's law and can be formulated as [6] $s = \frac{1}{(f_{par}/p+(1-f_{par}))}$ where f_{par} is parallel fraction of the code and p is the number of processors.

We compare a theoretical speedup (calculated with Amdahl's law) of proposed tabu search algorithm with an experimentally measured speedup. Table 2 shows our results. Particular columns in Table 2 denote:

- f_{par} - parallel fraction of the metaheuristic,
- s_t - theoretical speedup calculated with Amdahl's law,
- s_e - speedup measured experimentally.

For small values of the parallel fraction of the code (Mk02, Mk04 and Mk06) the difference between theoretical and experimentally measured speedup is small.

Table 2. Tabu search algorithm speedup

problem	p	f_{par}	$t_s[s]$	$t_p[s]$	s_t	s_e
Mk01	55	0.70098	0.85433	0.71478	3.20757	1.19523
Mk02	58	0.20656	3.43712	3.09198	1.25471	1.11162
Mk03	150	0.90118	17.1986	3.23384	9.53945	5.31832
Mk04	90	0.32281	9.25294	7.65297	1.46892	1.20907
Mk05	106	0.88954	5.77173	1.59326	8.4139	3.62259
Mk06	150	0.02064	247.16	254.661	1.02093	0.97054
Mk07	100	0.80061	5.37627	1.97047	4.82169	2.72842
Mk08	225	0.97586	53.0176	3.78734	35.1208	13.9986
Mk09	240	0.84383	74.4503	14.4714	6.26241	5.14465
Mk10	240	0.78271	81.4627	20.2549	4.53415	4.02188

5 GPU Implementation Details

In our tabu search algorithm for flexible job shop problem we adopt Floyd-Warshall algorithm for computing the longest path between each pair of nodes in a graph. The main idea of the algorithm is as follows. Find the longest path

```
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 = 0; 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. 1. 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. 2. CUDA kernel

between node v_i and v_j containing the node v_k. It consist of a sub-path from v_i to v_k and a sub-path v_k to v_j. This idea can be formulated as follows: $d_{ij}^{(k)} = \max\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$ where $d_{ij}^{(k)}$ is the longest path from v_i to v_j such that all intermediate nodes on the path are in set $v_1, \ldots, v_k$. Figure 1 shows CUDA implementation of of computing the longest path between each pair of nodes. The CUDA kernel (Figure 2) 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 every pair of nodes in the graph. The direct distance between them and the indirect distance through node v_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 are the number of the graph nodes, path distance matrix and the iteration (step) number.

6 Conclusions

We present the tabu search based algorithm with the golf neighborhood for the flexible job shop problem. In the golf neighborhood generation procedure parallel GPU acceleration has been applied, which allows to obtain an absolute speedup (in comparison to CPU) for each tested algorithm bigger than 1, and in extreme cases even the 14th times bigger.

Acknowledgement. The work was partially supported by the Polish Ministry of Science and Higher Education, grant N N514 232237.

References

1. Adrabiński, A., Wodecki, M.: An algorithm for solving the machine sequencing problem with parallel machines. Applicationes Mathematicae XVI 3, 513–541 (1979)
2. Bożejko, W., Wodecki, M.: On the theoretical properties of swap multimoves. Operations Research Letters 35(2), 227–231 (2007)
3. Bożejko, W., Uchroński, M., Wodecki, M.: Parallel hybrid metaheuristics for the flexible job shop problem. Computers and Industrial Engineering 59, 323–333 (2010)
4. Bożejko, W., Uchroński, M., Wodecki, M.: The new golf neighborhood for the flexible job shop problem. Procedia Computer Science 1, 289–296 (2010)
5. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41, 157–183 (1993)
6. Chapman, B., Jost, G., van der Pas, R.: Using OpenMP Portable Shared Memory Parallel Programming. The MIT Press (2007)
7. Nowicki, E., Smutnicki, C.: The flow shop with parallel machines: A tabu search approach. European Journal of Operational Research 106, 226–253 (1998)
8. Pinedo, M.: Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood Cliffs (2002)