Multi-thread parallel metaheuristics for the flow shop problem

Wojciech Bożejko, Jarosław Pempera, and Adam Smutnicki

Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland
{wojciech.bozejko, jaroslaw.pempera}@pwr.wroc.pl,
adam.smutnicki@student.pwr

Abstract. The matter of using scheduling algorithms in parallel computing environments is discussed. As the completion of our theoretical studies on parallel small-granularity single-thread approach [2], in this paper we propose and examine some parallel large-grain multi-thread approaches, namely: parallel variants of tabu search, simulated annealing, population-based search and scatter search. Computational experiments are done for the flow shop scheduling problem, the classical NP-hard case in combinatorial optimization, which plays here the role of exemplary test problem. Obtained results exhibit *superlinear speedup* in parallel environment and allow us to recommend parallel metaheuristics as the most suitable to solve scheduling instances derived from practice.

1 Introduction

Up to the end of eighties one had believed that exact algorithms, such as branch-and-bound, dynamic programming, linear integer programming, were the universal medicine on strong NP-hardness of combinatorial optimization problems. Although significant progress was done in the theory and special properties of these methods, they finally appear to be to poor to solve instances of practically applicable size, which especially refers to scheduling in manufacturing. This *barrier* follows from exponential computational complexity of problems and can be poorly eliminated by parallel processing (even the significant increase of computers power will result incomparably small increase of the size of instances we can solved). From that time, approximate approaches, chiefly metaheuristics, have been developed widely and extensively. Quality of the best solutions generated by search algorithms strongly depends on the number of analyzed solution, and thus on the running time. Time and quality have opposable tendency in that sense, that finding better solution requires significant growth of computation time. Through the parallel processing one can increase the number of checked solutions (per time unit) and thus can improve speed of calculations as well as quality of generated solutions. One can say that the hybrid combination of metaheuristics and parallel processing provides new direction is discrete optimization.

In the scope of multi-thread search, dedicated for uniform as well as non-uniform multiprocessor systems of large granularity (such as mainframe, clusters, distributed systems linked through the network) there have been designed

and examined experimentally parallel versions of the most promising combinatorial optimization methods (Tabu Search, Simulated Annealing, Genetic Search, Scatter Search) in application to selected scheduling problem. There have been discussed in detail various techniques of thread communication. The superlinear speedup effect has been observed. Single-thread search, for uniform multiprocessor system of small granularity, has been analyzed in our previous paper [2].

Throughout the paper we refer to several fundamental notions, see e.g. [8, 6, 4] for their definitions: theoretical parallel architectures, theoretical models of parallel computations, granularity, threads, cooperation, speed up, cost, computational complexity, real parallel architectures and parallel programming languages.

2 The problem

We consider as tester, accordingly to [2], strongly NP-hard well-known in the scheduling theory problem, called the permutation flow-shop problem with the makespan criterion (denoted by $F||C_{max}$ using Graham notation style). We also refer the reader to recent reviews and best up-to-now algorithms [12, 7, 13, 15].

The problem can be defined as follows. Jobs from the set $J = \{1, 2, \ldots, n\}$ have to be processed in a production system having m machines, indexed by $1, 2, \ldots, m$, organized in the line (sequential structure). Single job reflects one final product (or sub product) manufacturing. Every job is performed in m subsequent stages, in common way for all tasks. Stage i is performed by machine i, $i = 1, \ldots, m$. Every job $j \in J$ is split into sequence of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$ performed on machines in turn. Operation O_{ij} reflects processing of job j on machine i with processing time $p_{ij} > 0$. Once started job cannot be interrupted. Each machine can execute at most one job at a time, each job can be processed on at most one machine at a time.

The sequence of loading jobs into system is represented by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ on the set J. The optimization problem is to find the optimal sequence π^* so that

$$C_{max}(\pi^*) = \min_{\pi \in \Pi} C_{max}(\pi). \tag{1}$$

where $C_{max}(\pi)$ is the makespan for permutation π and Π is the set of all permutations. Denoting by C_{ij} the completion time of job j on machine i we have $C_{max}(\pi) = C_{m,\pi(n)}$. Values C_{ij} can be found by using formula

$$C_{i\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i\pi(j)}, \quad i = 1, 2, \ldots, m, \quad j = 1, \ldots, n, \tag{2}$$

with initial conditions $C_{i\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0\pi(j)} = 0$, $j = 1, 2, \ldots, n$.

3 Multi-thread search

There are two basic types of metaheuristics parallelization discussed in literature. The first one, called single-walk, is based on local search method's neighborhood decomposition onto concurrent working processors. Received solutions

are exactly the same as in sequential algorithm, but computing time is shorter. The algorithm subsequently selects and performs one single move. In multiple-walk parallelizations, a sequence of consecutive moves in the neighborhood (or genetic operators in a genetic algorithm) is made simultaneously. The second type of parallelization, called multiple-walk type (and considered here), is based on concurrent working metaheuristic threads, running on different processors. There are two sub-types of this parallelism: independent search, where there is no communication between threads, and cooperative search, with exchanging e.g. the best known solution found of the thread.

The philosophy of the approach described in the paper is to call several threads that perform simultaneously search through the solution space. Threads can communicate either each other or with the master thread (so called co-operative threads) or can run separately (independent threads). In the former case, the frequency of communication strongly influences on the efficiency of the whole method. In this section we present only four the most promising meta-heuristic methods (among variety of over 20 approaches, [15]), well-known for single-thread search, however here adopted to make parallel search throw the solution space.

3.1 Tabu search

There are two basic types of tabu search parallelization discussed in literature. The first one, called single-walk, is based on neighborhood decomposition onto concurrent working processors. Received solutions are exactly the same as in sequential algorithm, but computing time is shorter. The second type of parallelization, called multiple–walk type, is based on concurrent working tabu search threads, running on different processors. Classification of multiple-walk tabu search algorithm was created in [18]. Such an approach is proposed in [3].

Parallel version of the algorithm proposed here has been designed by us for the MIMD model, without shared memory. Processes are dispersed based on backtrack list, common for all processors, stored and updated on single dedicated master processor. This processor also stores the best up to now solution found. Algorithm has been designed to have relatively rare communication with master, particularly in the further phases of the algorithm. The algorithm has been implemented in Ada95 and run on Sun Enterprise 4x400 MHz computer under Solaris 7 operating system. Tuning parameters for TS are as follows: tabu list length (10), number of iterations without improving (10), number of processors (1,2,4), number of iterations (4000 for 1 processor, 2000 for 2 processors, 1000 for 4 processors). Common benchmarks instances are taken from [16], and includes particularly hard instances. Quality of the algorithm A has been measured by relative error $RE = 100 \cdot (C_{max}(\pi^A) - C^*)/C^*$, where C^* is the reference value (best known makespan) from [17]. Results are shown in Table 1, the best obtained values are in bold. The champion construction algorithm NEH follows from [11].

Superlinear speedup has been observed for both 2 and 4-processors implementations. Parallel algorithms obtain better solutions (in average) then 1-processor

$n \times m$	1 processor	2 processors	4 processors	NEH
20×5	1,96%	0,43%	**0,42%**	2,87%
20×10	2,84%	**1,09%**	1,30%	4,74%
20×20	1,82%	0,62%	**0,62%**	3,69%
50×5	0,33%	**0,08%**	0,14%	0,89%
50×10	2,81%	2,10%	**1,81%**	4,53%
50×20	3,49%	3,02%	**2,86%**	5,24%
100×5	0,25%	0,16%	**0,09%**	0,46%
average	1,79%	1,07%	**1,03%**	3,20%

Table 1. Relative error of parallel TS implementations under equal cost of calculations.

version with comparable costs of computations counted as a sum of iterations on all the processors.

3.2 Simulated annealing

Multiple-walk algorithm with rare communications (large grain model) is based on the *multiple Markov chains*. Synchronous implementation in [1] exchanges, in points of synchronization, information about best up to now solution; the length of the Markov chain is reduced globally. Alternative approach uses set of parallel cooperating threads, each of which has own annealing scheme, [10].

Parallel version of our algorithm is designed for MIMD machine without shared memory and with assumption that communication time between processors is significantly longer than communication inside single processor. There is used centralized model with separated master processor dedicated for storage results. We tested two strategies of communications: (A) slave processor obtains new best permutation π^* from master only if it want to distribute its own best permutation (rarely appears) - this model is denoted in Table 2 by BF, (B) slave processor communicates with master every K iterations, $K = 1, 10, 100, 1000$ or never INF(infinity).

Sequential as well as parallel implementation uses so called *block properties* known for $F||C_{max}$, [7], and *reduced INS neighborhood*. Accepting function has been given as $\Psi_t(\pi) = \exp((C_{max}(\pi) - C_{max}(\pi^*))/t)$ with geometric annealing scheme $t_{i+1}{=}at_i$. If after T_iter iterations algorithm does not find better solution than the best known up to now π^*, we set $t_{i+1} = t_0$. Algorithm stops after Max_iter iterations. The algorithm has been implemented in Ada95 and run on Sun Enterprise 4x400 MHz computer under Solaris 7 operating system. Starting solution has been generated by NEH algorithm. The following tuning parameters have been used: initial temperature ($t_0 = 60$), cooling constant ($a = 0.98$), the number of iterations in fixed temperature ($L = n$), the number of iterations without improvement ($T\_iter = 10$), the total number of iterations ($Max\_iter = 200n$ for 1 processor and $50n$ for 4 processors). We use the same benchmarks as previously and identical methodology of analysis. Results are shown in Table 2.

$n \times m$	1 processor	4 processors					
		frequency of communications (K)					
		INF	1	10	100	1000	BF
20 × 5	1,23%	1,28%	1,11%	0,85%	1,24%	1,04%	0,72%
20 × 10	2,08%	1,65%	1,91%	1,77%	1,99%	1,87%	1,63%
20 × 20	1,97%	1,74%	1,84%	1,55%	1,68%	1,77%	1,77%
50 × 5	0,27%	0,12%	0,15%	0,06%	0,13%	0,14%	0,10%
50 × 10	2,04%	1,03%	0,68%	1,26%	0,88%	1,08%	1,00%
average	**1,52%**	**1,16%**	**1,14%**	**1,10%**	**1,18%**	**1,18%**	**1,04%**

Table 2. Mean RE for parallel SA algorithm

Similarly as for the TS approach, a superlinear speedup has been observed. The most efficient model was BF, which results was 32.5% better then the results of the sequential SA algorithm.

3.3 Genetic approach

There are three basic types of parallelization strategies which can be applied to the genetic algorithm: (a) global, single-walk, (b) diffusion model and (c) island model (migration model). Model (a) spreads on parallel processors calculations for single population, so offers *linear* speedup of the solution time. Parallel model (b) uses asynchronous processes run on identical processors. Each processor serves small subpopulation for which it calculates adaptation function and genetic operations. Selection and crossover is possible in the local neighborhood limited by topology of physical links between processors. Parallel model (c) assumes that each processor performs its own autonomous genetic algorithm with its own population. Communication between processors is used to migrate best or some individuals. It is dedicated for large grain applications. A survey of parallel genetic approach can be found in [5].

Below, a parallel genetic algorithm is proposed. It is based on the global model of parallelism. There is the MSXF (Multi – Step Crossover Fusion) operator used to extend the process of researching for better solutions of the problem. MSXF has been described in [14]. Its idea is based on local search, starting from one of the parent solutions, to find a new good solution where the other parent is used as a reference point. The population is located in the processor number 0. Because of the time-consuming of the MSXF operator, crossover is made in parallel on p processors. Obtained offspring is added to the population, which is sorted. Worse solutions are deleted to keep the constant number of the population. Neighborhood API was used to generate next permutation σ in MSXF. As the distance measure between permutations we take Kendall's τ.

Parallel genetic algorithm was tested on the Silicon Graphics SGI Altix 3700 Bx2 with 128 Intel Itanium2 1.5 GHz processors and cache-coherent Non-Uniform Memory Access (cc-NUMA), craylinks NUMAflex4 in fat tree topology

with the bandwidth 4.3 Gbps[1]. Up to 8 processors of the supercomputer were used. The algorithm was implemented in C++ language using MPI (mpich 1.2.7) library. Tests are carried out on the same benchmarks as previously, using the same evaluation methodology.

$n \times m$	Processors			
	1 ($iter$=800)	2 ($iter$ = 400)	4 ($iter$ = 200)	8 ($iter$ = 100)
20 × 5	0.12%	0.08%	0.18%	0.20%
20 × 10	0.33%	0.24%	0.48%	0.98%
20 × 20	0.21%	0.18%	0.25%	0.49%
50 × 5	0.03%	0.04%	0.10%	0.18%
50 × 10	0.77%	0.71%	0.96%	1.28%
average	**0.29%**	**0.25%**	**0.39%**	**0.63%**
t_{total} (h:min:sec)	2:03:14	1:02:57	0:32:39	0:17:15
t_{cpu} (h:min:sec)	2:03:11	2:05:42	2:10:17	2:17:28

Table 3. Average values of RE for parallel GA

Tables 3 presents results of computations of the parallel genetic algorithm for the number of iterations (as a sum of iterations on all the processors) equals to 800. The cost of computations, understood as a sum of time-consuming an all the processors (t_{cpu}), is about 2 hours for the all 50 benchmark instances of the flow shop problem. The best results (average percentage deviations to the best known solutions) has the 2-processors implementation, which is almost 2 times faster that 1-processor implementation (t_{total}).

3.4 Scatter search

The main idea of the scatter search method is presented in [9]. The algorithm is based on the idea of evaluation of the so-called starting solutions set. In the classic version a linear combination of the starting solution is used to construct a new solution. In case of a permutational representation of the solution using linear combination of permutations gives us an object which is not a permutation. Therefore, in this paper a path relinking procedure is used to construct a path from one solution of the starting set to another solution from this set. The best element of such a path is chosen as a candidate to add to the starting solution set.

The base of the path relinking procedure used here, which connects two solutions π_1, $\pi_2 \in \Pi$, is a multi-step crossover fusion (MSXF) described by Reeves and Yamada [14]. Its idea is based on a stochastic local search, starting

[1] Calculations were done in the Wrocław Center of Networking and Supercomputing

from π_1 solution, to find a new good solution where the other solution π_1 is used as a reference point.

The neighborhood $N(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be achieved from π by exactly one adjacent pairwise exchange operator which exchanges the positions of two adjacent jobs of a problem's solution connected with permutation π. The distance measure $d(\pi,\sigma)$ is defined as a number of adjacent pairwise exchanges needed to transform permutation π into permutation σ. Such a measure is known as Kendall's τ measure.

Algorithm 1. Path-relinking procedure

Let π_1, π_2 be reference solutions. Set $x = q = \pi_1$;
repeat
  For each member $y_i \in N(\pi)$, calculate $d(y_i, \pi_2)$;
  Sort $y_i \in N(\pi)$ in ascending order of $d(y_i, \pi_2)$;
  repeat
    Select y_i from $N(\pi)$ with a probability inversely
      proportional to the index i; Calculate $C_{sum}(y_i)$;
    Accept $y_i$with probability 1 if $C_{sum}(y_i) \leq C_{sum}(x)$, and with
      probability $P_T(y_i) = exp((C_{sum}(x) - C_{sum}(y_i)) \ / \ T)$ otherwise
      (T is temperature);
    Change the index of y_i from i to n and the indices of
      y_k, $k = i{+}1,...,n$ from k to $k{-}1$;
  until y_i is accepted;
  $x \leftarrow y_i$;
  if $C_{sum}(x) < C_{sum}(q)$ **then** $q \leftarrow x$;
until *some termination condition is satisfied*;
return q { q is the best solutions lying on the path from π_1 to π_2 }

The condition of termination consisted in exceeding 100 iterations by the path relinking procedure. The parallel algorithm was projected to execute on the cluster of 152 dual-core Intel Xeon 2.4 GHz processors connected by Gigabit Ethernet with 3Com SuperStack 3870 swiches installed in the Wrocław Center of Networking and Supercomputing. Algorithms were implemented in C++ language using MPI (mpich 1.2.7) library and executed under the OpenPBS batching system which measures times of processor's usage.

Algorithm 2. Parallel scatter search algorithm for the SIMD model without shared memory

parfor $p := 1$ **to** number_of_processors **do**
  for i := 1 **to** *iter* **do**
    Step 1. if $(p = 0)$ **then** {only procesor number 0}
      Generate a set of unrepeated starting
      solutions S, $|S| = n$.
      Broadcast a set S among all the processors.

else {other processors}
Receive from the procesor 0 a set of starting solutions S.
end if;
Step 2. For randomly chosen $n/2$ pair from the S
apply path relinking procedure to generate a
set S' - of $n/2$ solutions which lies on paths.
Step 3. Apply local search procedure to improve
value of the cost function of solutions from the set S'.
Step 4. **if** $(p \neq 0)$ **then**
Send solutions from the set S' to procesor 0
else {only processor number 0}
Receive sets S' from other processors
and add its elements to the set S
Step 5. Leave in the set S at most n
solutions by deleting the worst and
repeated solutions.
if $|S| < n$ **then**
Add a new random solutions to the
set S such, that elements in the set
S does not duplicate and $|S| = n$.
end if;
end if;
end for;
end parfor.

$n \times m$	Processors				
	1	2	4	8	16
	iter=9600	2 *iter* = 4800	*iter* = 2400	8 *iter* = 1200	*iter* = 600
20 × 5	0.000%	0.000%	0.000%	0.000%	0.096%
20 × 10	0.097%	0.080%	0.066%	0.039%	0.109%
20 × 20	0.039%	0.062%	0.048%	0.031%	0.031%
50 × 5	0.007%	0.000%	0.007%	0.007%	0.000%
50 × 10	0.345%	0.278%	0.148%	0.238%	0.344%
average	**0.098%**	**0.084%**	**0.054%**	**0.063%**	**0.097%**
t_{total} (h:min:sec)	30:04:40	14:38:29	6:58:59	3:15:34	1:32:46
t_{cpu} (h:min:sec)	30:05:02	29:16:14	27:54:19	26:03:33	24:41:24

Table 4. Mean values of RE for parallel scatter search algorithm. The sum of iterations's number for all processors is 9600.

Table 4 presents results of computations of the parallel scatter search method for the number of iterations (as a sum of iterations on all the processors) equals to 9600. The cost of computations, understanding as a sum of time-consuming

an all the processors, is about 7 hours for the all 50 benchmark instances of the flow shop problem. The best results (average percentage deviations to the best known solutions) has the 4-processors version of the global model of the scatter search algorithm, which are 44.8% better comparing to average 1-processor implementation(0.054% vs 0.098%). For the 2, 4 and 8-processors implementation of the parallel scatter search algorithm the average results of RE are better than RE of the 1-processors versions, but the times-consuming on all the processors (t_{cpu}) are *shorter*. So these algorithm obtain better results with a smaller cost of computations - the speedup is superlinear. This anomaly can be understood as the situation where the sequential algorithm executes its search threads such that there is a possibility to chose a better path of the solutions space trespass, which the parallel algorithm do.

4 Conclusions

We have proposed a new parallel approaches based on metaheuristics: tabu search, simulated annealing, genetic algorithm and scatter search, designed for solving permutational scheduling problems. Multiple-thread search parallelization increased the quality of obtained solutions keeping comparable costs of computations. Superlinear speedup is observed in cooperative model of parallelism. Computer experiments show, that the parallel algorithm is considerably more efficient and stable in comparison to the sequential algorithms. Especially, parallel metaheuristics are the modern and efficient tool to solve strongly NP-hard scheduling problems, thus can be recommended for the use in the scheduling practice and theory.

References

1. Aarts E.H.L., de Bont F.M.J., Habers J.H.A., van Laarhoven P.J.M., Parallel implementations of the statistical cooling algorithm, Integration **4** (1986), 209–238.
2. Bożejko W., Pempera J., Smutnicki A., Parallel single-thread strategies in scheduling. Proceedings of ICAISC 2008, Lecture Notes in Artificial Intelligence, Springer 2008 (accepted).
3. Bożejko W., Wodecki M., Parallel tabu search method approach for a very difficult permutation scheduling problems, Proceedings of PARELEC 2004, IEEE Computer Society P2080 (2004), 156-161.
4. Bożejko W., Parallel job scheduling algorithms (in Polish). Report PRE 29/2003, Ph.D. dissertation, Wroclaw University of Technology (2003).
5. Cantu-Paz E., Efficient and Accurate Parallel Genetic Algorithms, Springer (2000).
6. Crainic T.G., Toulouse M., Parallel metaheuristics, in Fleet management and logistics (T.G. Crainic and G. Laporte, eds.), 205–251, Kluwer (1998).
7. Grabowski J., Pempera J., New block properties for the permutation flow shop problem with application in tabu search. Journal of Operational Research Society **52** (2000), 210–220.
8. Grama A, Kumar V., State of the Art in Parallel Search Techniques for Discrete Optimization Problems, IEEE Transactions on Knowledge and Data Engineering **11**, (1999), 28-35

9. James T., Rego C., Glover F., Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem, IEEE Intelligent Systems Vol. **20** Issue **4** (2005), 58-65.
10. Miki M., Hiroyasu T., Kasai M., Application of the temperature parallel simulated annealing to continous optimization problems, IPSL Transaction **41** (2000), 1607–1616.
11. Nawaz M., Enscore Jr. E.E. and Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA International Journal of Management Science **11** (1983), 91–95.
12. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research **91** (1996), 160–175.
13. Nowicki E., Smutnicki C., Some aspects of scatter search in the flow-shop problem, European Journal of Operational Research **169** (2006), 654–666.
14. Reeves C. R. and Yamada T. , Genetic algorithms, path relinking and the flowshop sequencing problem. Evolutionary Computation **6** (1998), 45–60.
15. Smutnicki C., Scheduling algorithms (in Polish), EXIT, Warsaw (2002).
16. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64** (1993) 278–285
17. Taillard E., Home page, http://www.eivd.ch/ina/collaborateurs/etd/default.htm
18. Voss S., Tabu search: Applications and prospects, in Network Optimization Problems (Du D.Z., Pardalos P.M., eds.), World Scientific (1993).