Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Waśniewski (Eds.)

LNCS 7204

Parallel Processing and Applied Mathematics

9th International Conference, PPAM 2011
Torun, Poland, September 2011
Revised Selected Papers, Part II

2 Part II

Springer

Volume Editors

Roman Wyrzykowski
Czestochowa University of Technology, Poland
E-mail: roman@icis.pcz.pl

Jack Dongarra
University of Tennessee, Knoxville, TN, USA
E-mail: dongarra@cs.utk.edu

Konrad Karczewski
Czestochowa University of Technology, Poland
E-mail: xeno@icis.pcz.pl

Jerzy Waśniewski
Technical University, Kongens Lyngby, Denmark
E-mail: jw@imm.dtu.dk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-31499-5 e-ISBN 978-3-642-31500-8
DOI 10.1007/978-3-642-31500-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012941360

CR Subject Classification (1998): D.2, H.4, D.4, C.2.4, D.1.3, H.3, F.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

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)

Parallel Cost Function Determination on GPU for the Job Shop Scheduling Problem

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. The goal of this paper is to propose a methodology of the effective cost function determination for the job shop scheduling problem in parallel computing environment. Parallel Random Access Machine (PRAM) model is applied for the theoretical analysis of algorithm efficiency. The methods need a fine-grained parallelization, therefore the approach proposed is especially devoted to parallel computing systems with fast shared memory. The methods proposed are tested with CUDA and OpenCL and ran on NVidia and ATI GPUs.

1 Introduction

In this work we are showing the method of parallelization of the job shop problem solving algorithm for GPGPU, consisting in parallelization of the cost function calculations. There are only few papers dealing with single-walk parallel algorithms for the job shop scheduling problem. Most of papers examine multiple-walk parallelization, e.g. parallel metaheuristics, without using of any theoretical properties of this scheduling problem. From the single-walk approaches, Bożejko et al. [2] proposed a simulated annealing metaheuristic for the job shop problem. Steinhöfel et al. [6] described the method of parallel cost function determination in $O(\log^2 o)$ time on $O(o^3)$ processors, where o is the number of all operations. Bożejko [3] considered a method of parallel cost function calculation for the flow shop problem which constitutes a special case of the job shop problem. Here we show a cost-optimal parallelization which takes a $O(d)$ time on $O(o/d)$ processors, where d is the number of layers in the topological sorted graph representing a solution. Finally, we conduct computational experiments on two types of GPU architectures (provided by nVidia and ATI) which fully confirm theoretical results.

2 The Job Shop Problem

Let us consider a set of jobs $\mathcal{J} = \{1,2,\ldots,n\}$, set of machines $M = \{1,2,\ldots,m\}$ and a set of operations $\mathcal{O} = \{1,2,\ldots,o\}$. A set $\mathcal{O}$ is decomposed into subsets connected with jobs. A job j consists of a sequence o_j operations indexed consecutively by $(l_{j-1}+1, l_{j-1}+2, \ldots, l_j)$ which have to be executed in the order, where $l_j = \sum_{i=1}^{j} o_i$ is a 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 the machine $v_i \in M$ without any idleness in the $p_i > 0$, $i \in \mathcal{O}$ time. Each machine can execute at most one operation in any moment of 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 constrains are fulfilled:

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

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

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

Certainly, $C_j = S_j + p_j$. An appropriate criterion function has to be added to the above constrains. The most frequently met are the following two criteria: minimization of the time of finishing of all the jobs and minimization of the sum of jobs' finishing times. From the formulation of the problem we have $\mathcal{C}_j \equiv C_{l_j}$, $j \in \mathcal{J}$.

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

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

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

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

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

Both described problems are strongly NP-hard and although they are similarly modeled, the second one is considered to be harder because of lack of some specific properties (so-called block properties, see [5]). They are used in optimization of execution time of solving algorithms.

2.1 Disjunctive Model

A disjunctive model is based on the notion of disjunctive graph $G^* = (\mathcal{O}^*, U^* \cup V)$. This graph has a set of vertices $\mathcal{O}^* = \mathcal{O} \cup \{0\}$ which represents operations (with an additional artificial beginning operation (0), for which $p_0 = 0$), a set of conjunctive arcs (directed) which show a technological order of operation's execution

$$U^* = U \cup U^0 = \bigcup_{j=1}^{n} \bigcup_{i=l_{j-1}+1}^{l_j-1} \{(i, i+1)\} \cup \bigcup_{j=1}^{n} \{(0, l_{j-1}+1)\} \qquad (6)$$

and the set of disjunctive arcs (non-directed) which shows a possible schedule of operations' realization on each machine

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

Disjunctive arcs $\{(i,j),(j,i)\}$ are in fact pairs of directed arcs with inverted directions connecting vertices i and j.

A vertex $i \in \mathcal{O}$ has a weight p_i which equals to the time of execution of the operation O_i. Arcs have the weight zero. A choice of exactly one arc from the set $\{(i,j),(j,i)\}$ corresponds to determination of a schedule of operations execution – "i before j" or "j before i". A subset $W \subset V$ consisting exclusively of 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 directions. 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 $G(W) = (\mathcal{O}, U \cup W)$ 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). Because 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 $O(o^2)$ time.

2.2 Combinatorial Model

In case of many applications a combinatorial representation of the solution is better than a disjunctive model for the job shop problem. It is voided of redundance, characteristic for the disjunctive graph, it denotes 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 the single, determined machine $k \in M$, $M_k = \{i \in \mathcal{O} : v_i = k\}$ and let $m_k = |M_k|$. A schedule of operations execution on a machine k is determined by a permutation $\pi_k = (\pi_k(1), \pi_k(2), \dots, \pi_k(m_k))$ of elements of the set M_k, $k \in M$, where $\pi_k(i)$ means such an element from M_k which is on the i position in π_k. Let $\Pi(M_k)$ be a set of all permutations of elements of M_k. A schedule of operations' execution on all machines is defined as $\pi = (\pi_1, \pi_2, \dots, \pi_m)$, where $\pi \in \Pi$, $\Pi = \Pi(M_1) \times \Pi(M_2) \times \dots \times \Pi(M_m)$. 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 technological order of operations execution inside a job and a set of arcs representing an order of operations' execution on machines is defined as

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

Each vertex $i \in \mathcal{O}$ has the weight p_i, each arc has the weight zero. A schedule π is feasible, if the graph $G(\pi)$ does not include a cycle. For a given feasible schedule π the process of determining of the cost function value requires the $O(o)$ time, thus shorter than for the disjunctive representation.

3 Sequential Determination of the Cost Function

Taking into consideration the constraints (1)–(3) presented in Section 2 it is possible to determine the time moments of operation completion C_j, $j \in \mathcal{O}$ and job beginning S_j, $j \in \mathcal{O}$ in time $O(o)$ on the sequential machine using the recurrent formula

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

where an operation i is a direct technological predecessor of the operation $j \in \mathcal{O}$ and an operation k is a directed machine predecessor of the operation $j \in \mathcal{O}$. The determination procedure of $S_j, j \in \mathcal{O}$ from the recurrent formula (9) 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 (9).

It is easy to observe that the order of determining S_j times corresponds to the index of the vertex of the $G(\pi)$ graph connected with a j operation 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 an element of indexing vertices to this algorithm, 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 the starting times for all machines and technological predecessors of the operations in the layer.

Definition 1. *The layer of the graph $G(\pi)$ is a maximal (due to the number of vertices) subsequence of the sequence of vertices ordered by the topological sort algorithm, such that there are no arcs between vertices of this subsequence.*

We will need this definition in the next paragraph.

4 Parallel Determination of the Cost Function

Theorem 1. *For a fixed feasible operations π order for the $J||C_{\max}$ problem, the number of layers from Definition 1 of the $G(\pi)$ graph can be calculated in $O(\log^2 o)$ time on the CREW PRAMs with $O\left(\frac{o^3}{\log o}\right)$ processors.*

Proof. Here we use the $G^*(\pi)$ graph with an additional vertex 0. Let $B = [b_{ij}]$ be an incidence matrix for the $G^*(\pi)$ graph, i.e., $b_{ij} = 1$, if there is an arc i, j in the $G^*(\pi)$ graph, otherwise $b_{ij} = 0$, $i, j = 0, 1, 2, \ldots, o$. 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 parallel Bellman-Ford algorithm (see [1]) – we need the time $O(\log^2 o)$ and CREW PRAMs with $O(o^3/\log o)$ processors.
2. We sort distances from the 0 vertex 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 the parallel mergesort algorithm. We obtain a sequence $Topo[i]$, $i = 0, 1, 2, \ldots, o$. The value of $Topo[o]$ equals d.

We can also use a tree-based parallel maximum determination algorithm in Step 2, instead of mergesort. However, 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 2. *For a fixed feasible operations π order for the $J||C_{\max}$ problem, the value of cost function can be determined in $O(d)$ time on $O(o/d)$-processor CREW PRAMs, where d is the number of layers of the graph $G(\pi)$.*

Proof. Let Γ_k, $k = 1, 2, \ldots, d$, be the number of calculations of the operations finishing moments C_i, $i = 1, 2, \ldots, o$ in the k-th layer. Certainly $\sum_{k=1}^{d} \Gamma_k = o$. Let p be the number of processors used. The time of computations in a single layer k after having divided calculations into $\lceil \frac{\Gamma_k}{p} \rceil$ groups, each group containing (at most) p elements, is $\lceil \frac{\Gamma_k}{p} \rceil$ (the last group cannot be full). Therefore, the total computation time in all d layers equals $\sum_{k=1}^{d} \lceil \frac{\Gamma_k}{p} \rceil \leq \sum_{k=1}^{d} (\frac{\Gamma_k}{p} + 1) = \frac{o}{p} + d$. 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 job shop problem with the makespan criterion.

5 The GPU Algorithm

The main part of our parallel implementation of goal function calculation for the job shop problem constitutes calculating of the longest path between all vertices in graph. This part was parallelized with CUDA and OpenCL and ran on NVidia and ATI GPUs.

```
__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. 1. CUDA kernel code

Kernel code (Figure 1 – CUDA kernel code) is invoked o times where o is the number of nodes in the graph. At the k-th iteration, the kernel computes direct and the indirect distance between every pair of nodes in graph 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 in the graph. The inputs of the GPU kernel are the number of the graph nodes, the graph distance matrix and the iteration number. Figure 2 shows OpenCL implementation of computing the longest path between each pair of nodes in a graph.

6 Computational Experiments

The proposed parallel algorithm of goal function calculation for the job shop problem was coded in CUDA and OpenCL and tested on GPU servers in the Wroclaw Centre for Networking and Supercomputing. The algorithm was tested on the set of benchmark problem instances taken from Lawrence [4] and Taillard [7]. We run our algorithm on three different GPUs:

– NVidia GTX480 with 480 parallel processor cores and 1.4 GHz clock rate,
– ATI Radeon HD5870 with 20 compute units and 850 MHz clock rate,
– ATI Radeon HD5970 with 20 compute units and 725 MHz clock rate.

```
// create the device memory for graph
cl_mem clGraph = clCreateBuffer(context, CL_MEM_READ_WRITE,
                                dataSize, NULL, NULL);
// transfer the input data into device memory
clEnqueueWriteBuffer(commands, clGraph,
                     CL_TRUE, 0, dataSize,
                     graph, 0, NULL, NULL);
size_t local  = o+1;
size_t global = (o-1)*(o+1);
// set the arguments to the compute kernel
clSetKernelArg(kernel, 0, sizeof(int), &o);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &clGraph);
for (int iter=1; iter <= size+1; iter++)
{
  for(int i=0; i<=o; ++i)
  {
    clSetKernelArg(kernel, 2, sizeof(int), &i);
    // execute the kernel
    clEnqueueNDRangeKernel(commands, kernel, 1, NULL,
                           &global, &local, 0, NULL, NULL);
    // wait for all commands to complete
    clFinish(commands);
  }
}
// read back the results from the device
clEnqueueReadBuffer(commands, clGraph, CL_TRUE, 0, dataSize,
                    graph, 0, NULL, NULL );
```

Fig. 2. OpenCL code

This GPUs are installed in servers with Intel Core i7 CPU with 3.20 GHz clock rate working under 64-bit GNU/Linux Ubuntu 10.10 operating system. The proposed algorithm uses o^2 processors for layers determination (basing on Theorem 1. scaled to the time $O(o \log o)$). The sequential algorithm (based on the method presented in the Section 3) using one CPU processor has been coded with the aim of determining the *absolute* speedup value which can be obtained with a parallel algorithm.

Our CUDA (OpenCL) implementation of the parallel goal function calculation for the job shop problem uses o blocks (work groups) with o threads (work items). The maximum work item size per dimension for HD5870 and HD5970 is equal to 256. Therefore we could ran our parallel algorithm on ATI GPUs only for Lawrence test instances (up to 255 operations). The maximum number of threads per block for NVidia GTX480 GPU is equal to 1024. On this GPU we can calculate a goal function for the job shop problem with up to 1024 operations.

Figures 3 and 4 show the comparison of computation time for sequential (run on CPU) and parallel algorithm coded in CUDA/OpenCL and run on NVidia/ATI GPUs. The measured time contains the time needed for data transfer between CPU and GPU. As shown in Figure 3 the considered algorithm coded in CUDA is faster than the algorithm coded in OpenCL for all Lawrence test instances. The algorithm coded in OpenCL is faster than the algorithm for CPU

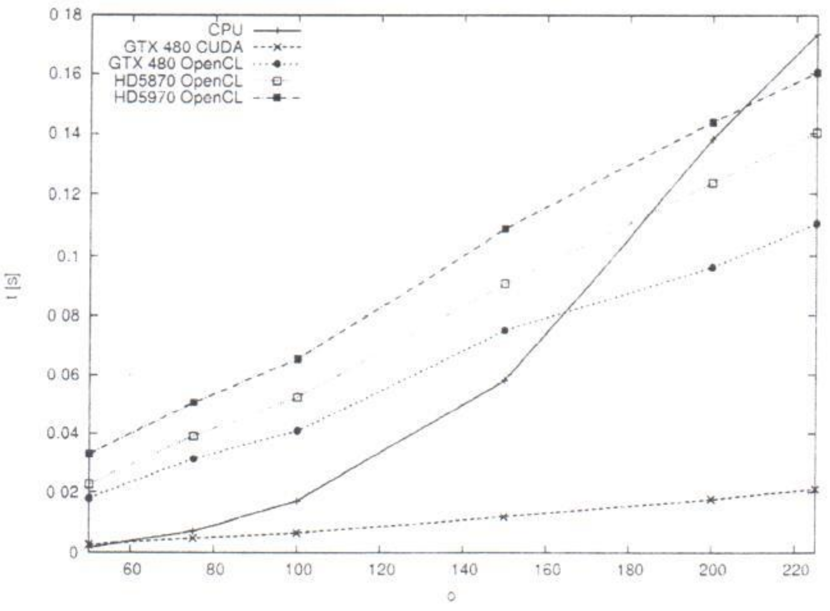

Fig. 3. Computation time for Lawrence test instances

for the number of operations grater than 225. The comparison of computation time for OpenCL code on different hardware shows that running the same code on Nvidia GTX 480 take less time than on used ATI GPUs. Also, the computation time for ATI Radeon HD5870 is less than ATI Radeon HD5970. This situation is caused by clock rate for GPU hardware used for tests evaluation.

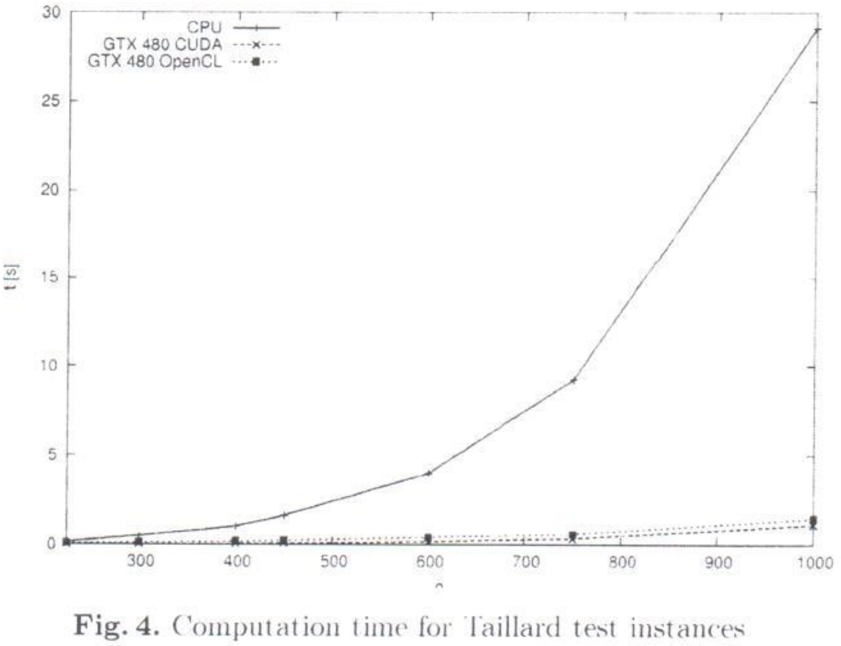

Fig. 4. Computation time for Taillard test instances

Figure 5 and Table 1 report the comparison of speedup obtained for CUDA and OpenCL implementation on NVidia GTX480 GPU. The particular columns in Table 1 denote:

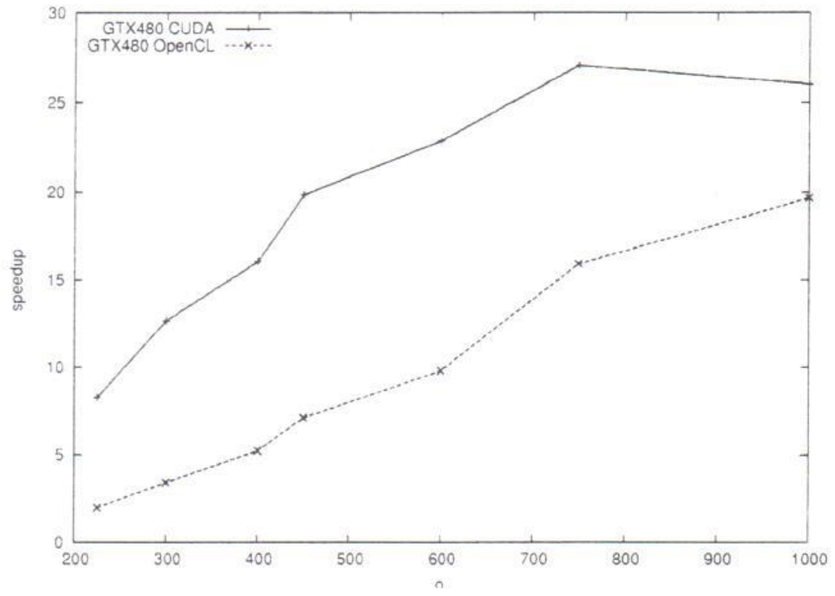

Fig. 5. Speedup for Taillard test instances

Table 1. Speedup for CUDA and OpenCL impementation obtained on NVidia GTX480 GPU

problem	o	s_{OpenCL}	s_{CUDA}
tail01-10	225	1.97731	8.25768
tail11-20	300	3.40649	12.6401
tail21-30	400	5.22559	16.0617
tail31-40	450	7.10766	19.855
tail41-50	600	9.7754	22.8953
tail51-60	750	15.9346	27.0535
tail61-70	1000	19.6828	25.9972

- o – number of operations in considered job shop problem instance.
- s_{OpenCL} – speedup for OpenCL implementation (average for each 10 instances).
- s_{CUDA} – speedup for CUDA implementation (average for each 10 instances).

The obtained results show that parallel goal function computation for the job shop problem on GPU results in shorter calculation time for the number of operations greater than 225. The considered algorithm coded in CUDA allow to obtain greater speedup than for the algorithm coded in OpenCL for all tested benchmarks. Our parallel algorithm reached 25x speedup for CUDA implementation and 19x speedup for OpenCL implementation on NVidia GTX480 GPU.

7 Conclusions

The algorithm proposed in this paper can be used for computation acceleration in metaheurisics solving the job shop problem. The calculation time of goal function in algorithms which solve the job shop problem take even 90% of the whole algorithm computation time. The use of parallel algorithm for goal function calculation might result in significant decreasing of algorithm execution time for solving the job shop problem.

References

1. Bożejko, W.: A new class of parallel scheduling algorithms. Monographs series. Wroclaw University of Technology Publishing House (2010)
2. Bożejko, W., Pempera, J., Smutnicki, C.: Parallel Simulated Annealing for the Job Shop Scheduling Problem. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part I. LNCS, vol. 5544, pp. 631–640. Springer, Heidelberg (2009)
3. Bożejko, W.: Solving the flow shop problem by parallel programming. Journal of Parallel and Distributed Computing 69, 470–481 (2009)
4. Lawrence, S.: Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate school of industrial administration, Carnegie Mellon University, Pittsburgh (1984)
5. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the job shop problem. Management Science 42, 797–813 (1996)
6. Steinhöfel, K., Albrecht, A., Wong, C.K.: Fast parallel heuristics for the job shop scheduling problem. Computers & Operations Research 29, 151–169 (2002)
7. Taillard, E.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)