Computers & Operations Research 39 (2012) 2258–2264

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

On single-walk parallelization of the job shop problem solving algorithms

Wojciech Bożejko *

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

ARTICLE INFO

Available online 18 November 2011

Keywords:
Job shop problem
Parallel programming
PRAM

ABSTRACT

New parallel objective function determination methods for the job shop scheduling problem are proposed in this paper, considering makespan and the sum of jobs execution times criteria, however, the methods proposed can be applied also to another popular objective functions such as jobs tardiness or flow time. 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 (e.g. GPGPU, *General-Purpose computing on Graphics Processing Units*).

1. Introduction

In this paper we are proposing new parallel objective function determination methods for the job shop scheduling problems. We are considering an algorithm (e.g. metaheuristic: tabu search, scatter search, etc.) which employs a single process to guide the search. The thread performs in a cyclic way (iteratively) two leading tasks:

(A) objective function evaluation for a single solution or a set of solutions,
(B) management, e.g. solution filtering and selection, collection of history, updating.

Part (B) takes statistically 1–3% total iteration time, thus its acceleration is useless. Part (A) can be accelerated in a multi-thread environment in various manners—our aim is to find either *cost-optimal* method or non-optimal one in terms of cost while offering *the shortest running time*. It is noteworthy to observe that if Part (B) takes β percentage of the 1-processor algorithm and if it is not parallelizable, the speedup of the parallel algorithm for *any* number of processors p cannot be greater than $1/\beta$ (according to Amdahl's law). In practice, if Part (B) takes 2% of the total execution time, the speedup can achieve at most the value of 50.

There are only a few papers dealing with parallel algorithms for the job shop scheduling problem, which has a relatively simple formulation and which is very interesting from the theoretical and practical points of view—many real manufacturing systems can be modeled just as the job shop, i.e. in construction projects, chemistry, electronics, etc. It is also considered as an indicator of practical efficiency of the new scheduling algorithms (see [7]). Bożejko et al. [2] proposed a single-walk parallelization of the simulated annealing metaheuristic for the job shop problem. Steinhöfel et al. [14] described the method of parallel objective function determination in $O(\log^2 o)$ time on $O(o^3)$ processors, where o is the number of all operations. Bożejko [1] considered a method of parallel objective function calculation for the flow shop problem, which constitutes a special case of the job shop problem. Here we are proposing a more efficient version of the algorithm developed by Steinhöfel et al., which works in $O(\log^2 o)$ time on $O(o^3/\log o)$ processors. Besides, we show a cost-optimal parallelization which takes a time $O(d)$, where d is the number of layers in the topological sorted graph representing a solution. Finally, we prove that this method has a constant $O(1)$ time complexity if we know the value of the upper bound of the objective function value.

Algorithms proposed in this paper are placed in proofs of theorems – they are constructive ones. The main result – the algorithm of the objective function value determination – is placed in the proof of the Theorem 3, and its practical aspects are described in Section 6.

2. Parallel computations model

We make a complexity analysis of the objective function determination algorithms for their implementations on Parallel Random Access Machine (PRAM model). A PRAM consists of many cooperating processors, each being a random access machine (RAM), commonly used in theoretical computer science. Each processor can make local calculations, e.g. additions, subtractions, shifts, conditional and unconditional jumps and indirect addressing. All the processors in the PRAM model are synchronized and have access to a shared global memory in constant time $O(1)$.

* Tel.: +48 713202961.
E-mail address: wojciech.bozejko@pwr.wroc.pl

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.11.009

There is also no limit on the number of processors in the machine, and any memory cell is uniformly accessible from any processor. The amount of shared memory in the system is not limitable. We make use of two kinds of PRAMs: CREW (*Concurrent Read Exclusive Write*) where processors can read from the same memory cell concurrently, and EREW (*Exclusive Read Exclusive Write*) where the concurrency of reading is forbidden. Both models resemble the GPU programming model. We take advantage of the following well-known facts for the PRAM parallel computer model [5]:

Fact 1. *Sequence of prefix sums* $(y_1,y_2,\ldots,y_n)$ *of input sequence* $(x_1,x_2,\ldots,x_n)$ *such, that* $y_k=y_{k-1}+x_k=x_1+x_2+\cdots+x_k$ *for* $k=2,3,\ldots,n$ *where* $y_1=x_1$ *can be calculated in* $O(\log n)$ *time on the EREW PRAM machine with* $O(n/\log n)$ *processors.*

According to the above statement we can assume that the sum of n values can be calculated in $O(\log n)$ time on $O(n/\log n)$ – processors EREW PRAMs.

Fact 2. *The minimal and the maximal value of input sequence* $(x_1,x_2,\ldots,x_n)$ *can be determined in* $O(\log n)$ *time on the EREW PRAM machine with* $O(n/\log n)$ *processors.*

If we do not have enough large number of processors, we can use the following fact to keep the same cost [5]:

Fact 3. *If the algorithm A works on p – processors PRAM in t time, then for every* $p'<p$ *there exists an algorithm* A' *for the same problem which works on* p' *– processors PRAM in* $O(pt/p')$ *time.*

The speedup and cost of a parallel algorithm as compared to a sequential algorithm are two commonly used criteria to evaluate parallel algorithms. Let us consider a problem Φ and a parallel algorithm A_{par}. Let us define $T_{A_{par}}(p)$ – time of calculations of the algorithm A_{par}, which is necessary to solve the problem Φ on the machine using p processors. Let $T_{A_{seq}}$ be a time of calculations of the best (the fastest) known sequential algorithm A_{seq} which solves the same problem Φ on the sequential machine with the processor identical to processors of the parallel machine. We define the *speedup* $S_{A_{par}}(p)=T_{A_{seq}}/T_{A_{par}}(p)$. The *cost* $C_{A_{par}}(p)$ of solving a problem by using an algorithm A_{par} in a p – processors parallel machine is defined as $C_{A_{par}}(p)=p\cdot T_{A_{par}}(p)$. This cost is the aggregated time that the processors require for solving the problem.

For sequential algorithms the problem solving time by the fastest known algorithm using one processor constitutes also its cost. We can state that a parallel algorithm is *cost-optimal* if its executing cost in a parallel system is linearly proportional to the execution time of the fastest known sequential algorithm on one processor.

3. The job shop problem

Let us consider a set of jobs $\mathcal{J}=\{1,2,\ldots,n\}$, a set of machines $M=\{1,2,\ldots,m\}$ and a set of operations $\mathcal{O}=\{1,2,\ldots,o\}$. The set $\mathcal{O}$ 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 $p_i>0$, $i\in\mathcal{O}$. Each machine can execute at most one operation at a time. A feasible solution constitutes a vector of times of the operation execution beginning $\mathcal{S}=(S_1,S_2,\ldots,S_o)$ such that the following constraints are fulfilled:

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

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

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

Certainly, $C_j=S_j+p_j$. An appropriate criterion function has to be added to the above constraints. The most frequent are the following two criteria: minimization of the makespan and minimization of the sum of job 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 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 job 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. In fact, we can distinguish a wider class of problems, with the objective function form of $f_{\max}$ and $\sum f_i$, where $f_{\max}=\max_{1\leq i\leq n}f_i(C_i)$ and $\sum f_i=\sum_{i=1}^{n}f_i(C_i)$, for any non-decreasing functions f_i (such as a flow time, sum of completion times, tardiness, makespan, etc.).

Both problems, with makespan and with the sum of job finishing times, are strongly NP-hard and although they are similarly modeled, the second one is found to be harder because of the lack of some specific properties (so-called block properties, see [11]) used in optimization of execution time of solution algorithms.

Because of NP-hardness of the problem heuristics and metaheuristics are recommended as 'the most reasonable' solution methods. The majority of these methods refer to the makespan minimization (e.g. [9,13,8,12,3]).

3.1. Models and properties

The most commonly used models of job shop scheduling problems are based on the disjunctive or the combinatorial approaches. Both these models are presented in this section.

3.1.1. Disjunctive model

The disjunctive model (see [11]) 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 so-called conjunctive arcs (i.e. directed) which show technological order of operation execution:

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

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

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

A sample disjunctive graph is presented on Fig. 1 (numbers near vertices are operation numbers, jobs are placed in rows and connected by solid arrows; disjunctive arcs are drawn as broken lines). Disjunctive arcs $\{(i,j),(j,i)\}$ are, in fact, pairs of directed arcs with inverted directions which connect vertices i and j. 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