Computers & Operations Research 40 (2013) 2322–2328

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization

Ahmad Al Badawi [a,*], Ali Shatnawi [b]

[a] Department of Computer Engineering, Taif University, Al-Taif 21974, P.O. Box 888, Saudi Arabia
[b] Department of Computer Engineering, Jordan University of Science and Technology, Irbid 22110, P.O. Box 3030, Jordan

ARTICLE INFO

Available online 17 April 2013

Keywords:
Combinatorial problems
Multiprocessor scheduling
Particle swarm optimization
NP-complete
Parallel processing
Graph theory

ABSTRACT

An efficient method based on particle swarm optimization (PSO) is developed to solve the Multiprocessor Task Scheduling Problem (MPTSP). To efficiently execute parallelized programs on a multiprocessor environment, a scheduling problem must be solved to determine the assignment of tasks to the processors, the execution order of the tasks, and the starting time of each task, such that some optimality criteria are met. The scheduling problem is known as an NP-complete problem even when the target processors are fully connected and no communication delay is considered among the tasks in the task graph. The complexity of the scheduling problem depends on the number of tasks (N), the number of processors (M), the task processing time and the precedence constraints. The Directed Acyclic Graph (DAG) was exploited to represent the tasks and their precedence constraints. The proposed algorithm was compared with the Genetic Algorithm (GA) and the Duplication Scheduling Heuristic (DSH). We also provide a systematic investigation on the effect of varying problem settings. The results show that the proposed algorithm could not outperform the DSH while it could outperform the GA in some cases.

1. Introduction

With many breakthroughs in the VLSI systems, device technology, computer architectures, hardware synthesis and software tools, the demand for multiprocessor systems has vastly increased in several applications [19]. In such multiprocessor systems, the scheduling of the tasks is a major step in the design process. On the other hand, the task scheduling is an important problem in other areas such as manufacturing, process control, economics, and operation research [19]. Basically, scheduling is simply to allocate a set of tasks (N) to resources (M) such that the optimum performance is obtained by, for example, minimizing the overall time required to execute all tasks. However, the scheduling problem is known to be NP-complete for the general case and even for many restricted cases [3,9]. The difficulty of the problem is due to the so many factors involved such as the nature of the task graph, the topology of the target multiprocessor system, and the uniformity of the task processing time. Because of its computational complexity, the scheduling problem is usually handled by heuristic methods which provide reasonable solutions [18].

In a broad sense, scheduling exists in two forms: static and dynamic. In static scheduling, which is usually done at compile time, the characteristics of a parallel program (such as task processing times, communication, data dependencies, and synchronization requirements) are known before the start of program execution [19]. A parallel program, therefore, can be represented by a Directed Acyclic Graph (DAG). In dynamic scheduling, only a few assumptions about the parallel program can be made before execution, and thus, the scheduling decisions have to be made dynamically while the program is running [19]. The optimization of a dynamic scheduling algorithm includes the minimization of the program completion time as well as the minimization of the scheduling overhead which constitutes a significant portion of the cost paid for running the scheduler. In this work, we only address the static scheduling problem. Hereafter, we refer to static scheduling as the scheduling.

Many heuristic methods have been proposed to solve the scheduling problem. These heuristics are highly diverse in terms of their assumptions about the structure of the parallel program and the targeted parallel architecture. Common simplification assumptions include uniform task processing times, zero inter-task communication times, contention-free communication, full connectivity of parallel processors, and availability of unlimited number of processors [19,9,14]. It is obvious that some of these assumptions may not hold in practical situations. For instance, it is not practical to assume that the task processing times of an application are uniform, since the

* Corresponding author. Tel.: +962 785466726.
E-mail addresses: a.badawi@tu.edu.sa, caesar.etos@gmail.com (A. Al Badawi), ali@just.edu.jo (A. Shatnawi).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.03.015

amount of computations encapsulated in tasks usually vary. It is also a violation of the physical nature when assuming a zero inter-task communication delay, and contention free communication.

The Multiprocessor Task Scheduling Problem (MPTSP) is not new, and the importance of this topic led to a great variety of intensive studies. Early scheduling algorithms were typically designed with some simplification assumptions about the DAG and processor network model [2,6]. For instance, the nodes in the DAG were assumed to be of unit processing time and zero communication time. Coffman and Graham [4] developed an algorithm for generating optimal schedule for arbitrary structured task graphs with unit-weighted tasks and zero inter-task times for a system of two homogeneous processors. The complexity of their algorithm is $O(V^2)$ and the scheduling process takes $O(V^2)$ time. Ibarra and Kim [8] proposed an algorithm which can be used to schedule a set of tasks without dependencies onto a heterogeneous multiprocessor system. Their algorithm, which called min–min, is a simple heuristic in which the tasks are scheduled based on their completion time. The one which has the minimum completion time is assigned to the processor on which it completes faster. The min–min heuristic is very simple, easy to implement and it was one of the fastest algorithms. Braun et al. [1] present a comparison of eleven heuristics for mapping meta-tasks onto a heterogeneous cluster of processors without taking into consideration the communication delays. Kruatrachue and Lewis [12] proposed Duplication Scheduling Heuristic (DSH) algorithm to solve the MPTSP. The main idea behind DSH is to eliminate communication delays by duplicating some predecessors in different processors so that their children can start as earlier as possible. The duplication is triggered once an idle time slot is detected in a processor. For example, if node n_i creates an idle time slot in one processor, the parent node n_p of this node is considered to be duplicated, given that it is not scheduled on the same processor. Several studies have been proposed to solve the MPTSP using Genetic Algorithms (GA). GAs try to mimic the natural evolution process and generally starts with an initial population of chromosomes. Each chromosome represents a potential solution of the problem. In each generation, the population goes through the processes of crossover, mutation, fitness evaluation and selection. During crossover, parts of two chromosomes of the population are exchanged in order to create two entirely new chromosomes which replace the chromosomes from which they evolved. Each chromosome is selected for crossover with a probability of crossover rate. Mutation alters one or more genes in a chromosome with a probability of mutation rate. A fitness function evaluates each chromosome, i.e., it decides how good a particular solution is. In the selection process, the chromosomes with highest fitness value are chosen to be carried onto the next generation, while the rest are dropped out. Jin et al. [9] present a performance study of nine multiprocessor task scheduling heuristics. They modified the algorithms taking into consideration the precedence constraint and inter-task communication times. They evaluated their work using two well-known problems of linear algebra: LU decomposition and Gauss–Jordan elimination. Amongst the nine heuristics were the DSH and a general implementation of GA described above. They found that the DSH had provided short scheduling time with the shortest makespan, but at the expense of duplicating the tasks on multiple processors. We care most about this study as we compare our algorithm with their DSH and GA algorithms.

In this paper, we address the MPTSP with the following properties:

- Precedence constrained task graphs.
- DAGs are used to model the problem.
- Non-zero inter-task communication time.
- The target system consists of a fixed number of homogeneous processors.
- The processors are fully connected with contention free network.

This paper is organized as follows. First, we present the problem, the model used, and some related definitions. A detailed demonstration of the algorithm is presented in Section 3. In Section 4, we present the simulation environment, the test beds, and the simulation results. Finally, Section 5 concludes this work and provides the future work.

2. Problem formulation

As mentioned above, the objective of task scheduling is to minimize the overall program finish-time by proper allocation of the tasks to the target processors. Scheduling is done in such a manner that the precedence constraints among the program tasks are preserved. The overall finish-time of a parallel program is commonly called the schedule length or the makespan [19]. Here, we review some basic definitions.

2.1. Multiprocessor scheduling

The scheduling problem can be divided into two sub-problems: processor allocation and finding the start time of each task. Processor allocation is the process of finding nodes order and which processor each node is assigned to. Given the optimal processor allocation, the start time of each task can be computed using the End technique, which will be covered in Section 3, in polynomial time. This implies that the scheduling problem can be simplified to finding the nodes order and the processor allocation which still known as NP-hard problem [19].

2.2. Directed acyclic graph model

A parallel program can be represented by a DAG $G(V, E)$, where V is a set of nodes and E is a set of directed edges. A node in the DAG represents a task composed of a set of instructions that must execute sequentially on the same processor without pre-emption. Hereafter, we will use the terms node and task interchangeably. The weight of a node n_i is called the processing time and is denoted by $w(n_i)$. An edge in the DAG, identified by its end nodes (n_i, n_j), represents the communication link from node n_i to node n_j, and specifies the precedence constraints among these two nodes. The weight of an edge is called the communication time of the edge and is denoted by $c(n_i, n_j)$. The source node of an edge is called the parent node while the sink node is called the child node. A node with no parent is called an entry node and a node with no child is called an exit node [19,9,18,10].

The precedence constraints of a DAG dictate that a node cannot start execution before it gathers all of the messages from its parent nodes. The communication time between two tasks assigned to the same processor is assumed to be zero. If node n_i is scheduled to run on some processor, then $ST(n_i)$ and $FT(n_i)$ have to be computed, where $ST(n_i)$ and $FT(n_i)$ denote the start-time and finish-time of node n_i, respectively. After all the nodes have been scheduled, the schedule length, makespan, is determined by $\max(FT(n_i))$ for all nodes on all processors. The goal of an optimal scheduling algorithm is to minimize this parameter. A simple DAG with 10 tasks is illustrated in Fig. 1. As we can see in the figure, each node has two attributes, the processing time: which is the time the task will take once executed, and the node height: which is the number of edges from the node to the entry node.