Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Task scheduling algorithm using minimized duplications in homogeneous systems

KwangSik Shin*, MyongJin Cha, MunSuck Jang, JinHa Jung, WanOh Yoon, SangBang Choi

HighTech-913, Inha University, 253 Yonghyun-dong, Nam-gu, Incheon 402-751, Republic of Korea

ARTICLE INFO

Article history:
Received 13 April 2007
Received in revised form
28 October 2007
Accepted 1 April 2008
Available online 8 April 2008

Keywords:
Load balancing and task assignment
Parallel processors
Scheduling and task partitioning
Distributed architectures

ABSTRACT

For fine grain task graphs, duplication-based scheduling algorithms are generally more efficient than list and cluster-based algorithms. However, most duplication-based heuristics try to duplicate all possible ancestor nodes of a given join node, in order to reduce the earliest start time (EST) of the join node, even though these ancestor nodes have already been allocated in previous steps. Thus, these duplication heuristics inevitably induce redundant duplications, which lead to the superfluous consumption of resources and generally deteriorate the scheduling result in the case of a bounded number of processors. When scheduling algorithms are used on an unbounded number of processors, the required number of processors grows excessively with the size of the task graph, thereby limiting the practicality of these algorithms for large task graphs. In this paper, we propose a novel algorithm designed to allocate join nodes without redundant duplications. In the proposed algorithm, if the ancestor nodes of a join node are duplicated when scheduling the join node, the original allocations of these ancestor nodes are removed using a very efficient method. The performance of the proposed algorithm, in terms of its normalized schedule length and efficiency, is compared with that of some of the recently proposed algorithms. The proposed algorithm generates better or comparable schedules with minimized duplication. Specifically, the simulation results show that it is most useful on a bounded number of processors.

1. Introduction

The scheduling of a parallel program on a set of homogeneous processors has been extensively studied. In the static scheduling model, the parallel program is represented by a weighted directed acyclic graph (DAG), in which nodes represent tasks and directed edges represent the partial order of the tasks. The objective function of the static scheduling algorithm is to minimize the completion time of a parallel program by mapping tasks to processors and ordering their executions so that task-precedence constraints are satisfied.

The scheduling of applications for parallel and distributed computing systems is one of the most challenging NP-complete problems in general cases, and polynomial time algorithms are known only for a few restricted cases. One of the main obstacles is the inevitable communication overhead that is incurred when tasks executed on different processors exchange data. Since efficient scheduling is imperative to obtain an improvement in performance from a parallel or distributed system, considerable research has been expended to develop heuristic-based algorithms by sacrificing the optimality of the scheduling.

Scheduling heuristics may be broadly classified as list-based, clustering-based, and duplication-based heuristics. In general, duplication-based heuristics are known to be more effective for fine grain task graphs and for networks with high communication latencies [11,2,4,19,5,3,15]. These duplication-based scheduling algorithms reduce the communication overhead by allocating some tasks to multiple processors. Given a task graph, duplication-based scheduling can mitigate the communication overhead by allocating some of the tasks to more than one processor.

Several different strategies can be used to duplicate ancestor nodes. As in the case of the Critical Path Fast Duplication (CPFD) [2] and Selective Duplication (SD) [4] algorithms, most of the duplication-based scheduling heuristics try to duplicate all possible ancestor nodes of a given join node, in order to reduce the earliest start time (EST) of the join node, even though these ancestor nodes have already been allocated in previous steps. Thus, these duplication heuristics inevitably induce redundant duplications. A duplication is referred to as *redundant* when it does not lead to an improvement in the performance or a reduction in the schedule length of an individual processor. Even if a duplicated task in a processor only reduces the schedule of the processor and not the schedule length of the whole system, it is not considered as redundant. Thus, redundant duplications consume unnecessary resources and generally deteriorate the scheduling result in the case of a bounded number of available processors. The required

* Corresponding author.
E-mail addresses: kwangsik@inha.ac.kr (K. Shin), sangbang@inha.ac.kr (S. Choi).

number of processors grows excessively with the size of the task graph, thereby limiting the practicality of scheduling tasks for large task graphs. Most duplication-based algorithms replicate the ancestor nodes of a given node in a bottom-up fashion in order to achieve the EST. Thus, the backward search which is performed for the purpose of duplicating the ancestor nodes of the node can cause a slight increase in the time complexity.

In this paper, we propose a novel scheduling algorithm designed to allocate join nodes without redundant duplications. In the case of an unbounded number of processors, the proposed algorithm can completely eliminate redundant duplications without increasing the schedule length. Reducing the number of processors to which tasks are assigned, it can avoid excessive resource usage, the drawback of existing duplication-based heuristics. Even though the proposed algorithm may not be able to eliminate redundant duplications perfectly in the environment of a bounded number of processors, it can still improve the schedule length of the whole system significantly. We assume that the processor network is fully connected and that each processor has dedicated communication hardware, so that communication and computation can take place simultaneously. In the proposed algorithm, if the ancestor nodes of a join node are duplicated during the scheduling of the join node, the original allocations of these ancestor nodes are removed using a very efficient method. Our objective is to prevent any redundant duplication from occurring when the join node is allocated to a processor, without causing any increase in the schedule length and time complexity. A fork node generally has to be duplicated in order to reduce the arrival time of the data that originates from it. Thus, we focus on eliminating redundant duplications at join nodes. The proposed algorithm can be used both for bounded and unbounded numbers of homogeneous processors. Its performance, in terms of the normalized schedule length and efficiency, is compared with that of some of the recently proposed algorithms. The proposed algorithm generates better or comparable schedules with minimized duplication. In particular, the simulation results show that it is most useful on a bounded number of processors.

This paper is organized as follows. In Section 2, we describe the previously proposed static scheduling heuristics that are based on task duplications. In Section 3, we formulate the scheduling problem and discuss the inefficiencies of duplication-based algorithms, which motivate the need for more effective scheduling. In Section 4, we present the design principles of our approach, followed by some illustrative examples to demonstrate the operations of the proposed algorithm, as well as those of the well-known CPFD and SD algorithms. The performance results of the proposed algorithm are presented in the following section in comparison with those of the above two algorithms. Section 6 summarizes and concludes this paper.

2. Related works

The scheduling of task graphs on multiple processors is known to be an NP-complete problem in most cases, leading to solutions based on heuristics. The NP-completeness of the scheduling problem has inspired researchers to propose numerous heuristic algorithms. The complexity and quality of a heuristic largely depend on the task graph structure and the target machine model. Static task-scheduling heuristics can be classified into three main groups; list scheduling heuristics, clustering heuristics, and task duplication heuristics.

Most list-based scheduling algorithms maintain a list of all of the tasks of a given graph. The basic idea in generating a list is to assign priorities to the nodes of the DAG and sort the nodes in the list in descending order of priority. These priorities are based on the computation and communication costs in the task graph. The node with a higher priority is considered for scheduling before a node with a lower priority. If more than one node have the same priority, the tie is broken using a certain method. Algorithms in this group include the Modified Critical Path (MCP) [20], Dynamic Level Scheduling [18], Mapping Heuristic (MH) [8], and Dynamic Critical Path (DCP) [10] algorithms. List scheduling heuristics are generally more practical and provide better performance results with a lower scheduling time than the other types of algorithm. However, the performance tends to deteriorate drastically for fine grain task graphs having a high communication to computation cost ratio (CCR).

Clustering-based heuristics try to schedule heavily communicating tasks on the same processor, thereby trading off parallelism with interprocess communication. These cluster-based heuristics map the nodes of a cluster to the same processors. To accomplish this, at the beginning of the scheduling process, each node is considered as a cluster. In the subsequent steps, two clusters are merged if this results in a reduction in the completion time. This merging step continues until no more clusters can be merged. Two tasks that are assigned to the same cluster will be executed on the same processor. A clustering heuristic requires additional steps to generate a final schedule: a cluster merging step to ensure that the number of clusters is equal to the number of processors and a task ordering step to sort the mapped tasks for each processor [12]. Some examples of clustering-based heuristics are the Mobility Directed [20], Dominant Sequence Clustering (DSC) [21], Linear Clustering Method [9], and Clustering and Scheduling System (CASS) [13] algorithms.

The reasoning behind duplication-based heuristics is to achieve a reduction in the communication overhead by redundantly allocating some nodes to multiple processors [11,2,4,1,16,6]. Individual duplication-based algorithms differ in terms of the strategy used to decide which tasks to duplicate. The most common technique employed is to recursively duplicate ancestor nodes in a bottom-up fashion, as is done in the CPFD and SD algorithms. Allocating the nodes to more than one processor, in order to reduce the waiting time of the dependent tasks, is an interesting approach that can be combined with list or clustering based techniques. List or clustering based algorithms combined with duplication tend to perform better than non-duplication algorithms. However, the improvement in the performance of list- or cluster-based algorithms afforded by integrating the function of duplication causes an increase in the complexity, because these algorithms have to carry out backward searches for the purpose of duplicating the ancestor nodes. Algorithms in this group include the Bottom-Up Top-Down Duplication Heuristic (BTDH) [6], Papadimitriou and Yannakakis (PY) [14], Lower Bound (LWB) [7], Linear Clustering with Task Duplication (LCTD) [17], Critical Path Fast Duplication (CPFD) [2], and Selective Duplication (SD) [4] algorithms.

The CPFD algorithm [2] initially determines the critical path nodes (CPNs) and constructs the CPN-dominant list. Then, the algorithm selects a candidate node, which is the first unscheduled CPN in the CPN-Dominant list, and allocates the candidate node to the processor that gives the smallest value of the EST. The CPFD attempts to duplicate ancestor nodes into the available time slot of the processor until either the slot is used up or the start time of the node does not improve any further. If there is no improvement at all in the EST, then the algorithm undoes all of the duplications that it just performed.

The selective duplication (SD) [4] algorithm has been developed as a generic algorithm that improves the performance of any non-duplication-based list heuristic by selectively adding to it the function of duplication, to the extent that it helps in improving the performance. The algorithm is divided into three main steps, i.e. the task sequence generation phase, processor selection phase, and task assignment and duplication phase. The proposed