Computers & Operations Research 43 (2014) 201–214

Contents lists available at ScienceDirect

Computers & Operations Research

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

Reducing the solution space of optimal task scheduling

Oliver Sinnen

Department of Electrical and Computer Engineering, University of Auckland, New Zealand

ARTICLE INFO

Available online 18 October 2013

Keywords:
Parallel computing
Task scheduling
A*
State space pruning

ABSTRACT

Many scheduling problems are tackled by heuristics due to their NP-hard nature. Task scheduling with communication delays ($P|prec, c_{ij}|C_{max}$) is no exception. Nevertheless, it can be of significant advantage to have optimal schedules, e.g. for time critical systems or as a baseline to evaluate heuristics. A promising approach to optimal task scheduling with communication delays for small problems is the use of exhaustive search techniques such as A*. A* is a best first search algorithm guided by a cost function. While good cost functions reduce the search space, early results have shown that problem specific pruning techniques are paramount. This paper proposes two novel pruning techniques that significantly reduce the search space for $P|prec, c_{ij}|C_{max}$. The pruning techniques Fixed Task Order and Equivalent Schedules are carefully investigated based on observations made with simple graph structures such as fork, join and fork–join, yet they are generally applicable. An extensive experimental evaluation of computing more than two thousand schedules demonstrates the efficiency of the novel pruning techniques in significantly reducing the solution space.

1. Introduction

Scheduling is an essential and crucial part of parallel computing. In the scheduling problem addressed in this paper, a set of tasks with precedence constraints and communication delays, represented by a task graph, is to be scheduled on a set of identical processors in such a way that the schedule length, or makespan, is minimised. This problem is a classical scheduling problem and written as $P|prec, c_{ij}|C_{max}$ in the $\alpha|\beta|\gamma$ notation [7,28]. It is well known that this problem is NP-hard in the strong sense [19]. The only known guaranteed approximation algorithm [11] has an approximation factor depending on communication costs of the longest path in the schedule [6]. Many heuristics have been proposed for this classical problem, even quite recently [2,12–14,16,26].

Despite the NP-hardness of the problem, it can be of significant advantage to have an optimal schedule in certain situations. This can be the case for very time critical systems or for application areas where a schedule is reused many times, e.g. in an embedded system or where the schedule is enclosed by a loop. Moreover, having optimal solutions for scheduling instances allows to better judge the quality of heuristics and thereby to gain insights into their behaviour. The lack of good guaranteed approximation algorithms makes this very relevant.

The work presented in this paper addresses the optimal solution of the $P|prec, c_{ij}|C_{\max}$ scheduling problem. A task graph represents the program to be scheduled, where the nodes represent the tasks and the edges represent the communications among them. Weights represent computation and communication costs. The scheduling problem is the assignment of a processor and a start time to each task, in such a way that the schedule length is minimised. It is trivial to show that this is equivalent to finding a processor allocation and an ordering of the tasks [23]: the tasks just need to start as early as possible adhering to all constraints. In other words, our scheduling problem is a combinatorial optimisation problem. Such problems have been widely addressed with two different optimisation techniques: (i) Mixed Integer Linear Programming (MILP) and (ii) smartly enumerating all feasible solutions, e.g. using a branch-and-bound algorithm.

For our scheduling problem, there have been surprisingly few attempts at solving it optimally. Scheduling problems often have a weak LP relaxation (i.e. the problem without the integer constraint, which is solvable in polynomial time), which leads to long runtimes of MILP solvers. Further, with the communication costs in the scheduling model and the fact that local communication is cost free, it is non-trivial to formulate an efficient MILP for our problem. Examples of MILP formulations are [1] and more recently [3,4]. In [4], the communication cost leads to bilinear constraints which have to be linearised, significantly reducing the efficiency of such an approach. Sometimes the MILP approach is used to design an efficient heuristic for the scheduling problem [9].

In this paper we will look at the second technique of enumerating all feasible solutions. For a task graph with $|\mathbf{V}|$ tasks scheduled on $|\mathbf{P}|$ processors, the solution space is spawned by all possible orderings of the tasks ($|\mathbf{V}|!$) times all possible processor allocations $|\mathbf{P}|^{|\mathbf{V}|}$. It is clear that the exploration of the solution

E-mail address: o.sinnen@auckland.ac.nz

0305-0548/$ - see front matter © 2013 Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.cor.2013.09.004

space must be done very intelligently if we want to go beyond half a dozen of tasks and two processors. A* is a best-first search technique that can be applied to such solution space exploration [5]. In contrast to typical branch-and-bound algorithms [18], it does not follow a depth-first search, but always expands the state in the search space that looks most promising. In other words, A* uses a priority queue for states to expand, while branch-and-bound usually uses a LIFO (Last In First Out) queue. This comes at the cost of much higher memory consumption, but A* has the nice property that it will traverse the least number of states for a given cost estimation heuristic, compared to other exploration techniques [20]. In [22], we proposed an A* based scheduling algorithm, which was inspired by an earlier attempt [15,17].

From these earlier results, we gained two important insights. First, pruning techniques play a paramount role to reduce the search space, even with good A* cost estimate functions. Second, the performance of the A* based scheduling algorithm is very dependent on the structure and density of the task graph. Paradoxically, very simple graph structures like fork and join were more difficult to schedule, i.e. it took longer to find the optimal solution for the same number of tasks than more complex and dense structures like pipeline or stencil. The problem is that the simpler structures have more degrees of freedom and hence have a larger solution space.

The contribution of this paper is the proposal of new pruning techniques to significantly reduce the solution space of the $P|prec, c_{ij}|C_{\max}$ scheduling problem. The two major techniques are Fixed Task Order pruning and Equivalent Schedule pruning. Both techniques are developed from observations made about the scheduling of simple task graph structures, namely independent tasks, fork graphs, join graphs and fork–join graphs. The techniques are investigated based on our A* scheduling algorithm, but the concepts and methods can be applicable to other exhaustive search techniques. They possibly extent to metaheuristics such as Genetic Algorithms [32,29], by reducing the search space on which the metaheuristic operators work. A further pruning technique proposed is the extended utilisation of schedules produced by heuristics, which can prune the solution space significantly under certain conditions.

This paper also contributes an extensive evaluation computing more than two thousand schedules. Scheduling task graphs optimally with the novel techniques demonstrates that we achieved our goal of significantly reducing the search space. The novel pruning techniques are especially successful for those graph structures that were most difficult to schedule before, which are fork, join, fork–join and trees.

Section 2 continues this paper with the background and definition of the scheduling problem, presenting the basic A* scheduling algorithm and discussing its cost estimate function f. Existing pruning techniques are reviewed in Section 3, leading to the proposal of the new pruning techniques in Sections 3.4, 4 and 5. They are evaluated in Section 6 and the paper concludes with Section 7.

2. Background

The problem to be addressed in this work is the scheduling of a Directed Acyclic Graph (DAG), called task graph, $G = (\mathbf{V}, \mathbf{E}, w, c)$ on a set of processors $\mathbf{P}$. Each node $n \in \mathbf{V}$ represents a *non-divisible* sequential task of the program modelled by the task graph. An edge $e_{ij} \in \mathbf{E}$ represents the communication from task n_i to task n_j. The positive weight $w(n)$ of task $n \in \mathbf{V}$ represents its computation cost and the non-negative weight $c(e_{ij})$ of edge $e_{ij} \in \mathbf{E}$ represents its communication cost. Fig. 1 depicts a sample task graph with 4 tasks; weights are displayed besides the tasks and edges. The set $\{n_x \in \mathbf{V} : e_{xi} \in \mathbf{E}\}$ of all direct predecessors (parents) of n_i is denoted by $\mathbf{parents}(n_i)$ and the set $\{n_x \in \mathbf{V} : e_{ix} \in \mathbf{E}\}$ of all direct successors (children) of n_i is denoted by $\mathbf{children}(n_i)$. A task $n \in \mathbf{V}$ without parents, $\mathbf{parents}(n) = \emptyset$, is named *source* task and if it is without children, $\mathbf{children}(n) = \emptyset$, it is named *sink* task.

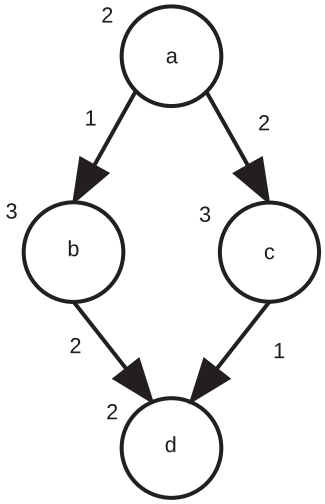

Fig. 1. Graph with four tasks.

The target parallel system consists of a finite set of *dedicated* processors $\mathbf{P}$ (there is *no preemption* of an executing task) connected by a communication network. It has the following properties: (i) local communication has zero cost; (ii) communication is performed by a communication subsystem; (iii) communication can be performed concurrently; and (iv) the communication network is fully connected. This system model and its idealising assumptions are sometimes referred to as the classic model [23], as opposed to more advanced models that consider communication contention [24,25].

Scheduling this task graph G on the processors $\mathbf{P}$ is the assignment of a *processor allocation* $proc(n)$ and a *start time* $t_s(n)$ to each task. The node's *finish time* is given by $t_f(n) = t_s(n) + w(n)$, i.e. the task's start time plus its computation costs, as homogeneous processors are assumed in this work. Let $t_f(P) = \max_{n \in \mathbf{V}: proc(n) = P}\{t_f(n)\}$ be the *processor finish time* of P and let $sl(\mathcal{S}) = \max_{n \in \mathbf{V}}\{t_f(n)\}$ be the *schedule length* (or makespan) of schedule $\mathcal{S}$, assuming $\min_{n \in \mathbf{V}}\{t_s(n)\} = 0$.

For such a schedule to be feasible, the following two conditions must be fulfilled for all tasks in G. The *Processor Constraint* enforces that only one task is executed by a processor at any point in time, which means for any two tasks $n_i, n_j \in \mathbf{V}$

$$proc(n_i) = proc(n_j) \Rightarrow \begin{cases} & t_f(n_i) \le t_s(n_j) \\ \text{or} & t_f(n_j) \le t_s(n_i) \end{cases}.$$

The *Precedence Constraint* enforces that for every edge $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$, the destination task n_j can only start after the communication associated with e_{ij} has arrived at n_j's processor P_{dst},

$$t_s(n_j) \ge t_f(n_i) + \begin{cases} 0 & \text{if } proc(n_i) = proc(n_j) \\ c(e_{ij}) & \text{otherwise} \end{cases}.$$

Considering all parents, n_j's start time on processor P is constrained by the so-called *Data Ready Time (DRT)* $t_{dr}(n_j, P)$. This is the time, when all communications from n_j's predecessors have arrived at P, given as

$$t_{dr}(n_j, P) = \max_{n_i \in \mathbf{parents}(n_j)} \left\{ t_f(n_i) + \begin{cases} 0 & \text{if } proc(n_i) = P \\ c(e_{ij}) & \text{otherwise} \end{cases} \right\} \quad (1)$$

If n_j is a source task then $t_{dr}(n_j, P) = 0$.