Computers and Operations Research 89 (2018) 113–126

Contents lists available at ScienceDirect

Computers and Operations Research

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

Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay

Avinash Malik[a], Cameron Walker[b], Michael O'Sullivan[b], Oliver Sinnen[a,*]

[a] Department of Electrical and Computer Engineering, New Zealand
[b] Department of Engineering Science, University of Auckland, Auckland, New Zealand

ARTICLE INFO

Article history:
Received 14 November 2016
Revised 18 June 2017
Accepted 16 August 2017
Available online 24 August 2017

Keywords:
Parallel computing
Task scheduling with communication delays
SMT
MILP

ABSTRACT

In scheduling theory and practise for parallel computing, representing a program as a task graph with communication delays is a popular model, due to its general nature, its expressiveness and relative simplicity. Unfortunately, scheduling such a task graph on a set of processors in such a way that it achieves its shortest possible execution time ($P|pred, c_{ij}|C_{\max}$ in $\alpha|\beta|\gamma$ notation) is a strong NP-hard optimization problem without any known guaranteed approximation algorithm. Hence, many heuristics have been researched and are used in practise. However, in many situations it is necessary to obtain optimal schedules, for example, in the case of time-critical systems or for the evaluation of heuristics. Recent years have seen some advances in optimal algorithms for this scheduling problem, based on smart exhaustive state-space search or MILP (Mixed Integer Linear Programming) formulations. This paper proposes a novel approach based on SMT (Satisfiability Modulo Theory). We propose an elegant SMT formulation of the scheduling problem that only needs one decision variable and is very compact and comprehensible in comparison to the state-of-the-art MILP formulations. This novel optimal scheduling approach is extensively evaluated in experiments with more than a thousand task graphs. We perform experimental comparison with the best known MILP formulations, with attempts to further improve them, and deeply analyse the behaviour of the different approaches with respect to size, structure, number of processors, etc. Our proposed SMT-based approach in general outperforms the MILP-formulations and still possesses great potential for further optimization, from which MILP formulations have benefited in the past.

1. Introduction

Task scheduling has long been recognized as a crucial part of parallel computing. In any parallel computation, tasks need to be mapped to the available processors and ordered for execution. In the formalisation of this process, a directed acyclic graph (DAG) – a task graph – represents the program to be executed on a parallel system. The nodes or vertices of this graph represent the (sub-)tasks and the edges between them reflect the communication (data transfer) or dependencies between them. The weighted DAG representation, a very universal and flexible model, has been widely used in literature and practice to create schedules on a given set of processors (Drozdowski, 2009; Sinnen, 2007). It has also seen an increasing interest in High Performance Computing in dynamic runtime schedulers such as StarPU (Augonnet et al., 2011), KAAPI (Gautier et al., 2007), StarSS (Planas et al., 2009), and PaRSEC (Bosilca et al., 2013). Even the OpenMP standard now includes DAG scheduling constructs (Board, 2013).

Unfortunately, the optimization problem of minimising the execution time of a program, modelled as a weighted DAG, on a set of processors is a well known NP-hard problem (Sarkar, 1987), for all but some very simplified cases, e.g. a fork or a join graph on an unlimited number of processors (Chrétienne, 1989), or a tree with unit weights on two processors (El-Rewini and Ali, 1994). For that reason, many scheduling algorithms have been proposed in the past. The largest category of algorithms is based on list scheduling (Hu, 1961; Liu et al., 2005; Macey and Zomaya, 1998; Yang and Gerasoulis, 1993a). Other algorithms are based on clustering, where a second phase maps the clusters onto the limited number of processors (Cirou and Jeannot, 2001; Kadamuddi and Tsai, 2000; Kianzad and Bhattacharyya, 2006; Yang and Gerasoulis, 1993b). Also meta-heuristics, such a genetic algorithms have been applied to task scheduling (Daoud and Kharma, 2011; Omara and Arafa, 2010). In the general case there is no guaranteed constant approximation factor, only one based on the communication costs

* Corresponding author.
E-mail addresses: avinash.malik@auckland.ac.nz (A. Malik), cameron.walker@auckland.ac.nz (C. Walker), michael.osullivan@auckland.ac.nz (M. O'Sullivan), o.sinnnen@auckland.ac.nz (O. Sinnen).

http://dx.doi.org/10.1016/j.cor.2017.08.012
0305-0548/

of the graph (Hwang et al., 1989). Some scheduling algorithms might even produce schedules that are longer than the sequential schedule, where all tasks are executed on the same processors (Sinnen, 2007). This is especially true for large communication costs.

Given the importance of this task scheduling problem and the lack of guaranteed approximation algorithms, an attempt has been made to design algorithms and approaches to optimally solve this scheduling problem. An optimal solution can be crucial for critical systems, where the scheduled application is executed many times. A very important usage of optimal schedules is the evaluation of heuristic scheduling algorithms, given the lack of guaranteed optimality from heuristics. The quality of heuristic algorithms can be judged better when the optimal solutions are available for comparison. As a combinatorial optimization problem, the task scheduling problem can be tackled with various approaches. Searching through the exhaustive solution space is one method that has been proposed, using algorithms such as branch-and-bound or A* (Kafil and Ahmad, 1998; Kwok and Ahmad, 2005; Shahul and Sinnen, 2010; Sinnen, 2014a; Venugopalan and Sinnen, 2016). Another approach is using (Mixed) Integer Linear Programming (MILP) formulations with corresponding solvers. Despite the difficulty of solving scheduling problems with MILP formulations, there has been some significant progress recently (Davare et al., 2006a; Davidović et al., 2007; Venugopalan and Sinnen, 2015). As a natural generalization, constraint programming has been applied to solving the task scheduling problem (Kuchcinski, 2003). Another recent approach is to use a SAT (Boolean Satisfiability Problem) formulation to solve the scheduling problem (Liu et al., 2011).

In this paper we propose an approach based on a Satisfiability Modulo Theory (SMT) formulation. The formulation is simple and elegant as it is mostly based on the constraints that naturally emerge from the scheduling model. We only employ one binary decision variable to determine whether a task is scheduled on a given processor or not. In terms of theory solver for the inequalities, we employ Quantifier Free Linear Arithmetic Logic (QF_LRA).

To evaluate the performance and usefulness of the proposed SMT formulation, we compare it with state-of-the-art MILP formulations of the same problem (El Cadi et al., 2014; Mallach, 2016; Venugopalan and Sinnen, 2015). In an attempt to further improve these formulations, we propose new variants that address the typical symmetry issue of MILP formulations for scheduling problems.

The experimental evaluation is based on a large set of 120 task graphs, which are scheduled with all the SMT and MILP formulations onto different numbers of processors. These graphs cover different structures, sizes and weightings. As solvers for the SMT and MILP formulations we use Z3 (De Moura and Bjørner, 2008) and Gurobi Optimization (2015), respectively. We perform a thorough analysis of the results, uncovering performance behaviour of the formulations with respect to structure and other characteristics of the task graphs and the processor number. Two types of statistical classification are carried out, namely logistic regression and decision tree classification. While some of the obtained results are intuitive and expected (e.g. larger graphs are more difficult to schedule optimally), other results are unexpected and can lead to further improvement in the future. The evaluation shows that the SMT formulation outperforms the MILP based approaches, especially for a lower number of processors, solving more problems optimally within the given time limit. This holds a lot of promise for the novel SMT approach, given the potential of optimization, which MILP formulations have already enjoyed.

The remainder of this paper is organized as follows. Section 2 defines the scheduling model and problem. We propose our SMT formulation of the problem in Section 3 and revisit MILP formulations in Section 4. The experimental evaluation of all approaches is conducted and discussed in Section 5. We study related work in Section 6 and conclude the paper in Section 7.

2. Scheduling model

A task graph $G(V, E)$ is a directed acyclic graph (DAG), where $V = \{v_1, \ldots, v_n\}$ is the set of computation tasks, and $E \subseteq V \times V$ is the set of data dependency constraints (or communications) between tasks. A homogeneous multiprocessor platform $P = \{p_1, \ldots, p_m\}$ consists of m identical processors connected by a communication network. Each processor is connected to every other processor. Furthermore we also assume a dedicated communication sub-system so that computation and communication can be executed in parallel. Tasks are executed sequentially without preemption on a processor. Every task is able to run on any processor. Execution Time (ET) of task $v \in V$ is given by $t(v)$, and Communication Latency (CL) for edge $(v_i, v_j) \in E$ is given by $c(v_i, v_j)$. If two tasks, with data dependence, are mapped to the same processor, inter-task communication is implemented by data sharing in local memory, and no communication latency is incurred. If two tasks, with data dependence, are mapped to different processors, communication between them is implemented by data transfer through communication links, and communication latency corresponds to the CL (e.g., $c(v_i, v_j)$, $(i, j) \in E$). We define $deg^-(v)$ ($deg^+(v)$) as being the in-degree (out-degree) of the vertex v, i.e. the number of entering (leaving) edges. In this work, we assume the ET values of tasks and CL values of intertask communications are known, and focus on optimizing the mapping and scheduling of a task graph under these assumptions.

Given a task graph $G(V, E)$ and a homogeneous multi-processor platform P, the optimization problem is to find a mapping $M: V \rightarrow P$ for each task in V to a processor in P and a schedule $S : V \rightarrow \mathbb{N}$ for the tasks assigned to processors, where each task v in V mapped to a processor in P is assigned a natural number indicating its starting time s for execution on P. This schedule (from now on the term schedule implies both the mapping and the start time assignment to tasks) must adhere to two constraints resulting from the above definitions. The *processor constraint*, meaning that for any two tasks on the same processor, one task must finish before the other one starts. And the *precedence constraint*, which enforces that for any edge $(v_i, v_j) \in E$, task v_j can only start after task v_i is completed (i.e. $s(v_j) \geq s(v_i) + t(v_i)$) plus the communication time, if the tasks are mapped to different processors (i.e. $s(v_j) \geq s(v_i) + t(v_i) + c(v_i, v_j)$). The goal of the optimization is to minimize the *makespan* $m(G)$. Makespan, also called schedule length, is defined as the time difference between the start time of the earliest task and the finish time of the latest one. Assuming that the earliest task's start time is 0, the makespan is given by $m(G) = max_{v \in V}[s(v) + t(v)]$. The problem of finding the schedule that minimizes the makespan is known to be NP-hard (Sarkar, 1987).

3. Satisfiability modulo theory (SMT) formulation

Satisfiability Modulo Theory (SMT) (Barrett et al., 2009) is a technique for solving boolean constraint problems modulo theories. SMT has been used successfully in verification of hardware and software programs. Fig. 1 shows the difference between a SAT solver and a SMT solver. A SAT solver is used to solve propositional formulas. These formulas consist of atomic propositions (or their logical negation), in the Boolean domain, combined using logical connectives such as $\vee$ (disjunction), and $\wedge$ (conjunction). A SAT solver uses the well known DPLL (Davis et al., 1962) solving technique for finding a *model*, i.e., an assignment for atomic propositions that satisfies the propositional formula, if one exists. A SMT solver extends a SAT solver with theory solvers in order to solve