J. Parallel Distrib. Comput. 75 (2015) 53–66

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Memory-aware tree traversals with pre-assigned tasks

Julien Herrmann [a,*], Loris Marchal [a], Yves Robert [a,b]

[a] Ecole Normale Supérieure de Lyon, CNRS/INRIA, France
[b] University of Tennessee Knoxville, USA

HIGHLIGHTS

- Complexity of scheduling task graphs with two memories.
- Application to CPU/GPU hybrid programming.
- Heuristics to design efficient trade-offs between both peak memories.
- Analysis of post-order traversals and application to multifrontal methods.

ARTICLE INFO

Article history:
Received 7 June 2013
Received in revised form
22 July 2014
Accepted 2 October 2014
Available online 15 October 2014

Keywords:
Scheduling
Memory-aware
Sparse matrix factorization
Multifrontal method
Tree traversal
Bi-objective optimization

ABSTRACT

We study the complexity of traversing tree-shaped workflows whose tasks require large I/O files. We target a heterogeneous architecture with two resource types, each with a different memory, such as a multicore node equipped with a dedicated accelerator (FPGA or GPU). The tasks in the workflow are colored according to their type and can be processed if all their input and output files can be stored in the corresponding memory. The amount of used memory of each type at a given execution step strongly depends upon the ordering in which the tasks are executed, and upon when communications between both memories are scheduled. The objective is to determine an efficient traversal that minimizes the maximum amount of memory of each type needed to traverse the whole tree. In this paper, we establish the complexity of this two-memory scheduling problem, and provide inapproximability results. In addition, we design several heuristics, based on both post-order and general traversals, and we evaluate them on a comprehensive set of tree graphs, including random trees as well as assembly trees arising in the context of sparse matrix factorizations.

1. Introduction

Modern computing platforms are heterogeneous: a typical node is composed of a multi-core processor equipped with a dedicated accelerator, such as a FPGA or a GPU. Our goal is to study the execution of a computational workflow, described by an out-tree, onto such a heterogeneous platform, with the objective of minimizing the amount of memory of each resource needed for its processing. The nodes of the workflow tree correspond to tasks, and the edges correspond to the dependences among the tasks. The dependences are in the form of input and output files: each node accepts a (potentially large) file as input, and produces a set of files, each of them to be processed by a different child node. We consider in this paper that we have two different processing units at our disposal, such as a CPU and a GPU. For sake of generality, we designate them by a color (namely *blue* and *red*). Each task in the workflow is best suited to a given resource type (say a core or a GPU), and is *colored* accordingly. To execute a task of a given color, the input file and all the output files of the task must fit within the corresponding memory. As the workflow tree is traversed, tasks of different colors are processed, and capacity constraints on both memory types must be met. In addition, when a child of a task has a different color than its parent, say for example that a blue task has a red child, a communication from the blue memory to the red memory must be scheduled before the red child can be processed (and again, the input file and all output files of this red child must fit within the red memory). All these constraints require to carefully orchestrate the scheduling of the tasks, as well as the communications between memories, in order to minimize the maximum amount of each memory that is needed throughout the tree traversal.

Memory-aware scheduling is an important problem that has been the focus of many papers (see Section 2 for related work).

* Corresponding author.
E-mail addresses: julien.herrmann@ens-lyon.fr (J. Herrmann), loris.marchal@ens-lyon.fr (L. Marchal), yves.robert@ens-lyon.fr (Y. Robert).

http://dx.doi.org/10.1016/j.jpdc.2014.10.004
0743-7315/

This work mainly builds upon the pioneering work of Liu, who has studied tree traversals that minimize the peak amount of memory used on a homogeneous system, hence with a single memory type. Liu first restricted to depth-first traversals in [17], before dealing with an optimal algorithm for arbitrary traversals in [17]. In many situations, the optimal traversal is a depth-first traversal, but this is not always the case. An assessment of the relative performance of depth-first traversals versus optimal traversals is proposed by [14]. The main objective of this paper is to extend these results to colored trees with two memory types, and tasks belonging to a given type. Clearly, the traversal, i.e., the order chosen to execute the tasks, and to perform the communications, plays a key role in determining which amount of each memory is needed for a successful execution of the whole tree. The interplay between both memories dramatically complicates the scheduling: it is no surprise that the complexity of the problem, that was polynomial with a unique memory, now becomes NP-complete.

In this paper, we concentrate on memory usage, but we are fully aware that performance aspects are important too, and that even more difficult trade-offs are to be found between parallel performance and memory consumption. One could envision a fully general framework, where tasks have different execution-times for each resource type (instead of being tied to a given resource as in this paper), and where concurrent execution of several tasks on each resource type is possible (instead of the fully sequential processing of the task graph that is assumed in this paper). Altogether, this study is only a first step towards the design of memory-aware schedules on modern heterogeneous platforms with two memory types. However, despite the apparent simplicity of the model, our results show that we already face a difficult bi-criteria optimization problem when dealing with two different memory types. We firmly believe that the results presented in this paper will help to lay the foundations for memory-aware scheduling algorithms on modern heterogeneous platforms such as those equipped with multicores and GPUs. Indeed, one key contribution of the paper is the derivation of several complexity results: NP-completeness of the problem, and inapproximability within a constant (α, β) factor pair of both absolute minimum memory amounts. Here the absolute minimum memory of a given type is computed when assuming an infinite amount of memory of the other type.

Another major contribution is the study of depth-first traversals and related variants. We show how to extend Liu's algorithm to compute the best depth-first traversal, which simultaneously minimizes both memory usages. However, while depth-first traversals were natural algorithms with a single memory, they severely constrain the activation of communication nodes with two memories. We show that the optimization problem is still NP-complete when relaxing the firing of communication nodes in depth-first traversal, which leads us to go beyond depth-first traversals and to introduce general heuristics. These heuristics extends Liu's optimal algorithm along various (greedy) decision criteria to trade-off the usage of both memory types.

Finally, the third major contribution is a comprehensive assessment of all these heuristics using both randomly generated trees, and actual elimination trees that arise from the multifrontal factorization of sparse linear systems.

The rest of the paper is organized as follows: We start with an overview of related work in Section 2. Then we detail the framework in Section 3. The next four sections constitute the heart of the paper. We deal with complexity results in Section 4. Section 5 is devoted to the study of depth-first traversals, a first class of (widely-used) heuristics. Then we introduce additional heuristics in Section 6. The experimental evaluation of all the heuristics is conducted in Section 7. Finally we provide some concluding remarks and hints for future work in Section 8.

2. Related work

The work presented in this paper builds upon previous results related to memory-aware scheduling, but its applications are relevant to the field of sparse matrix factorization and of hybrid computing. In this section, we present related work for each domain.

2.1. Sparse matrix factorization

Determining a memory-efficient tree traversal is very important in sparse numerical linear algebra. The elimination tree is a graph theoretical model that represents the storage requirements, and computational dependences and requirements, in the Cholesky and LU factorization of sparse matrices. In a previous study, we have described how such trees are built, and how the multifrontal method organizes the computations along the tree [14]. This is the context of the founding studies of Liu [17,18] on memory minimization for postorder or general tree traversals mentioned in Section 1. Memory minimization is still a concern in modern multifrontal solvers when dealing with large matrices. In particular, efforts have been made to design dynamic schedulers that takes into account dynamic pivoting (which impacts the weights of edges and nodes) when scheduling elimination trees with strong memory constraints [11], or to consider both task and tree parallelism with memory constraints [1]. Recently, still in the context of a single memory type, an extension of these results to parallel machines has been proposed in [20]. While these studies try to optimize memory management in existing parallel solvers, we aim at designing a simple model to study the fundamental underlying scheduling problem.

2.2. Scientific workflows

The problem of scheduling a task graph under memory constraints also appears in the processing of scientific workflows whose tasks require large I/O files. Such workflows arise in many scientific fields, such as image processing, genomics or geophysical simulations. The problem of task graphs handling large data has been identified in [22] which proposes some simple heuristic solutions. Surprisingly, in the context of quantum chemistry computations, Lam et al. [16] have recently rediscovered the algorithm published in 1987 in [18].

2.3. Pebble game and its variants

On the more theoretical side, this work builds upon the many papers that have addressed the pebble game and its variants. Scheduling a graph on one processor with the minimal amount of memory amounts to revisiting the I/O pebble game with pebbles of arbitrary sizes that must be loaded into main memory before *firing* (executing) the task. The pioneering work of Sethi and Ullman [24] deals with a variant of the pebble game that translates into the simplest instance of the problem with a unique memory and where all files have weight 1. The concern in [24] was to minimize the number of registers that are needed to compute an arithmetic expression. The problem of determining whether a general DAG can be traversed with a given number of pebbles has been shown NP-hard by Sethi [23] if no vertex is pebbled more than once (the general problem allowing recomputation, that is, re-pebbling a vertex which have been pebbled before, has been proven PSPACE complete [9]). However, this problem has a polynomial complexity for tree-shaped graphs [24]. Recently, still in the contact of a single memory type, an extension of these results to parallel machines base been proposed in [19].