Journal of Computational Science 18 (2017) 106–116

Contents lists available at ScienceDirect

Journal of Computational Science

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

Porting HPC applications to the cloud: A multi-frontal solver case study

Bartosz Balis*, Kamil Figiela, Konrad Jopek, Maciej Malawski, Maciej Pawlik

AGH University of Science and Technology, Department of Computer Science, Al. Mickiewicza 30, 30-059 Krakow, Poland

ARTICLE INFO

Article history:
Received 16 March 2016
Received in revised form
19 September 2016
Accepted 21 September 2016
Available online 26 September 2016

Keywords:
HPC in the cloud
Multi-frontal direct solver
Scientific workflows
Mesh-based solver

ABSTRACT

In this paper we argue that scientific applications traditionally considered as representing typical HPC workloads can be successfully and efficiently ported to a cloud infrastructure. We propose a porting methodology that enables parallelization of communication – and memory-intensive applications while achieving a good communication to computation ratio and a satisfactory performance in a cloud infrastructure. This methodology comprises several aspects: (1) task agglomeration heuristic enabling increasing granularity of tasks while ensuring they will fit in memory; (2) task scheduling heuristic increasing data locality; and (3) two-level storage architecture enabling in-memory storage of intermediate data. We implement this methodology in a scientific workflow system and use it to parallelize a multi-frontal solver for finite-element meshes, deploy it in a cloud, and execute it as a workflow. The results obtained from the experiments confirm that the proposed porting methodology leads to a significant reduction of communication costs and achievement of a satisfactory performance. We believe that these results constitute a valuable step toward a wider adoption of cloud infrastructures for computational science applications.

1. Introduction

Communication overhead in the infrastructure-as-a-service (IaaS) clouds is arguably the primary performance bottleneck for High Performance Computing (HPC) applications [1] which are typically communication-sensitive. For this reason, clouds have been mostly used for more loosely-coupled high-throughput computations [2], including parameter studies and scientific workflows [3]. However, even though HPC clusters outperform clouds in terms of raw performance, clouds can still produce better turnaround times, even when considering typical HPC applications, because of their elastic on-demand resource provisioning model, eliminating long wait times in the job queues, common in HPC systems [4]. The advantage of clouds for HPC also comes from a good price/performance ratio which motivates large industry enterprises to move their HPC workloads to the cloud in order to reduce costs related to maintaining an on-premises HPC infrastructure [4].

In this paper, we study an important class of problems, traditionally considered as HPC applications, that can be solved using the finite element method and multi-frontal solvers. We focus on a large class of non-symmetric elliptic or Maxwell problems that can be solved on two dimensional adaptive meshes with uniform polynomial order of approximation. We follow the definition of hierarchical basis functions from the book of Demkowicz [5]. However, we restrict our class of meshes to uniform second order polynomials, which is the most common use choice for FEM computations. We utilize H-adaptive finite element method with constrained approximation and hanging nodes technique, as described in [5]. The particular implementation of the multi-frontal solver presented in our paper is described in Ref. [6], chapter 8.

The multi-frontal solver algorithm introduced by Duff and Reid [7,8] is the state of the art solver for mesh-based computations. The solver algorithm can be decomposed into a graph of tasks. This has been already done for one or two-dimensional finite difference method [9], as well as for two-dimensional [10,11] and three-dimensional [12] adaptive finite element method. Existing multi-frontal solvers have been designed for and executed exclusively on HPC clusters, shared-memory machines, or GPUs, utilizing MPI for interprocess communication [13–15], multi-threading [16,17], CUDA [18], or a hybrid of those [19].

We propose a methodology allowing efficient porting of such applications to cloud infrastructures. This methodology consists in representing the computational problem as a scientific workflow and applying several design-time and execution-time optimizations, such as efficient task agglomeration and mapping, that lead to significant reduction of the communication to computation ratio

* Corresponding author.
E-mail address: balis@agh.edu.pl (B. Balis).

http://dx.doi.org/10.1016/j.jocs.2016.09.006
1877-7503/

and achievement of a good performance and speedup in the cloud infrastructure. To evaluate the proposed methodology, we perform experiments using a system for execution of multi-frontal solver workflows in the cloud, implemented on the basis of the solver described in [20] and our workflow runtime environment HyperFlow [21]. Our target environment is the infrastructure-as-a-service (IaaS) cloud, which allows for dynamic on-demand creation of computing nodes in the form of virtual machines (called instances), but our approach can also be used on other infrastructures, such as traditional commodity clusters.

Scientific workflows are successfully used for automation of computational problems in a variety of scientific disciplines [3]. Using workflows lets the user focus on describing a computational problem as an abstract graph of tasks, while the *workflow management system* transparently takes care of its execution that includes, amongst others, provisioning of the required computing resources, task scheduling, deployment of application components, and data staging. A workflow defined once enables transparent utilization of diverse Distributed Computing Infrastructures (DCIs), such as clusters, grids or clouds. Workflows have been typically used for loosely-coupled computations, where each task is executed as an independent program (executable) with input and output data transferred as files. In this work, our intent is to investigate the usage of the workflow model and a workflow system to more tightly-coupled application.

This paper is a substantial extension of our earlier conference publication [22] where preliminary ideas for the approach were presented and evaluated on a small scale. The main contributions of this paper are:

- We discuss the parallelization of the multifrontal solver as a workflow using the partitioning, communication, agglomeration and mapping (PCAM) methodology [23].
- We propose new optimizations of task agglomeration based on estimated memory consumption, and a mapping that improves data locality in order to minimize the communication between tasks considerably.
- We develop new execution services and data exchange mechanisms for the HyperFlow workflow system to take advantage of the optimized parallel execution.
- We evaluate our solution on large-scale workflows with hundreds of thousands of tasks that can be agglomerated to less than a thousand tasks using our method. The tests on up to 64 VMs on Amazon EC2 show the speedup of over 39 times.

The paper is organized as follows. Section 2 describes related work. Section 3 presents the porting methodology. In Section 4, the implementation of the environment for development and execution of solver workflows is described. Section 5 contains experimental evaluation of the proposed methodology. Section 6 discusses the evaluation results. Section 7 concludes the paper and outlines directions for future research.

2. Related work

2.1. Multi-frontal solvers

The classical multi-frontal solvers work based on the ordering obtained from some heuristic algorithm, like e.g. nested-dissections [24] working on the sparsity graph of the global matrix. The ordering is transformed internally into the element partition tree inside the solver. It has been shown recently [20] that it is possible to construct multi-frontal direct solver based on the element partition tree, that defines recursive partitions of the computational mesh [25–27].

For some class of grids the orderings following from the element partition trees outperforms the ordering obtained based on the analysis of the sparsity of the global matrix. An alternative approach is to consider the adjacency graph for nodes of finite element mesh [16,17,28,29]. Based on the mesh, the frontal (elimination) tree is generated which prescribes the order of elimination of mesh nodes. This approach was used in [28]. In our approach, we generate our ordering based on the element partition tree, prescribing the recursive partitions of the mesh. The element partition tree is sometimes confused with the elimination tree, however the first one is obtained from the mesh, and the second one implies from the ordering and the sparse matrix. For adaptive mesh based computations, namely for grids with point and edge singularities, it is possible to construct element partition trees with lightweight computational tasks close to the root of the element partition tree [25,27]. This concerns computational meshes in both two and three dimensions.

A similar representation of the multifrontal solver computation as a tree has been used for efficient parallelization [30]. The key to achieving efficiency was the scheduling of tasks based on memory and work estimation to achieve a good load-balancing. In our work we also use the memory estimation for the optimal load-balancing of workflow tasks.

2.2. HPC in the cloud

Cloud infrastructures have not been widely adopted for HPC workloads yet, but there are initiatives to bridge this gap. Here we report on some most notable examples.

Early studies regarding HPC in the cloud were conducted in the scope of Magellan project [31]. The final report concluded that although loosely-coupled applications tend to perform well on clouds, the dedicated HPC systems are needed for typical communication-intensive and tightly coupled workloads.

Scientific workflows have been also evaluated in the cloud, in terms of cost and performance, e.g. in [32,33]. These classes of applications executed by Pegasus workflow management system are large-scale, but they typically consist of loosely-coupled tasks that communicate via file transfers, which is not applicable to the problem we are addressing in this paper. Similar reasoning also applies to large-scale many-task computing applications, which have been studied in clouds [34].

A comparative study of Amazon EC2 cluster to typical HPC clusters presented in [4] focused on turnaround time and total cost of execution. The turnaround time includes the time of a job waiting in a queue of a HPC system or VM acquisition time. As such, the turnaround time is a clear advantage of cloud services over typical HPC systems. In terms of cost, the results depend on the application and workload type, and the cost model of HPC system in question.

The initiative of UberCloud promotes the usage of cloud systems for technical and scientific computing. The experimental results [35,36] have shown that the performance of HPC in the cloud is sufficient for the application of a large finite element analysis, and that run time can be optimized by properly selecting a configuration of CPU, memory, and interconnect.

Gupta et al. [1] studied the performance behavior of several HPC applications in the cloud. However, they do not consider improvements to the applications' parallelization scheme toward reducing the performance bottlenecks of the cloud. They address the problem of heterogeneity of the cloud resources by dynamic load-balancing of application tasks among virtual machines [37] or by scheduling VMs on physical resources to optimize performance of HPC applications [38]. This is different from our approach, where we propose a parallelization method to significantly reduce the communication costs. We also do not assume the control over the physical infrastructure, but rather we focus on public clouds such as Amazon EC2.