A Parallelization and Performance Optimization Framework for Mesh-Based Simulations Using Target Execution Models

Zhi-guo ZHANG*, Qing-yin HE, Jin-yu LIU and Jing-yi SHAO
Sun Yat-Sen University, School of Data and Computer Science, Guangzhou 510006, China
*Corresponding author

Keywords: Parallelization, Performance Optimization, Mesh-based Applications, Target Execution Model.

Abstract. Mesh-based applications usually use very big mesh structures for simulation accuracy, and they are coded in different programming languages using different parallel runtime support systems. They run in various parallel computation systems. It is a big problem that in such a situation how can domain expert build a mesh-based application which well fit the different performance requirements for different target computation systems. Another problem is that it becomes harder to create well performed parallel programs for modern computing systems for domain experts even for computing specialists because modern computing systems become more complicated. In this paper an automatic parallelizing and performance optimizing framework is proposed and implemented for mesh-based application domains through a translation from domain description into internal programming model. Using internal programming models and their performance models with target system execution models, domain descriptions can be effectively parallelized and optimized into parallel code for target systems. Several CFD codes are tested in the framework as case-studies and experiments shown that more efficient codes, which are more scalable and of higher performance, can be generated for different target computation systems.

Introduction

Mesh-based simulations are widely used in many industrial fields such as aerospace, and etc. For simulation accuracy, the mesh structures in such applications are usually very big so that the simulation time in sequential computation is very long. Almost all industrial mesh-based applications are running in parallel in different clusters of computers or supercomputers as being programmed in different parallel programming paradigms and in different programming environments. A big problem in such a situation is how can domain expert build a mesh-based application which well fit the different performance requirements for different target computation systems. Mostly a mesh-based application is coded in a certain programming language with certain programming environment and targeted to certain runtime system and architecture. When they are expected to be optimized for and run to different runtime system and architecture which they are not targeted originally, they have to be reprogrammed for the new target runtime and architecture. Even for the originally targeted system with a new configuration, these programs may need being refactorized again. Another problem is that it becomes harder to create well performed parallel programs for modern computing systems for domain experts even for computing specialists because modern computing systems become more complicated.[1] You have to select different programming language and programming environment to code your programs. You have to consider some detailed configuration information of the target runtime system and target architecture for performance optimization. Many factors from runtime system and/or architecture can be take into account for performance optimization, and in many times these factors may conflict with each other when we want to make an improvement for performance of our programs.

Aiming to solve these two problems, we propose a framework for automatic parallelizing and optimizing mesh-based (simulation) applications, in which domain description is separated from
detailed parallelization and performance optimization for specified target programming environment, runtime system and architecture. Domain description is a purely domain algorithm description in a domain specific language in which mesh structure, kernel operations over mesh structure, and algorithms over mesh structure can be simply declared without consideration of parallelization and performance optimization. Parallelization and performance optimization can be realized in more steps by the framework. First, the domain description is translated into an internal programming model according to specified target programming environment. Then combined with a specified target execution model, detailed performance optimization can be further realized. Finally the optimized program can be tested and further optimized automatically, partially automatically or manually in the framework.

**Related Work**

To solve problems mentioned in last section, many research teams focus on decoupling the scientific specification from parallel implementation, by providing an abstraction layer between the specification and implementation. Such abstraction layer enables domain scientists to develop their programs easily with their specific domain knowledge, and enables computing scientists to implement the domain specification with near-optimal performance in the specific architectures. Different abstraction layers by different authors provide different angle to analyze and breakdown the domain problems. [4,8,9]

**Differential Equations**

This type of abstraction layer enables programmers to express the problem in the ways of differential equations. The FEniCS[2] project defines a unified form language UFL, and solve PDEs by finite element methods, the implementation for backend is left to its library. The abstraction of differential equations is ideal for programmer who is solving PDEs with a particular numeric methods. This kind of application is usually user-friendly, as user can focus only on their problem. However, this type of abstraction lack the flexibility of defining new data structure and new algorithm.

**Element-based computation**

The domain is discretized into the mesh, the elements on the mesh, and computation on these elements. OP2[5,7] and Liszt[3] provides the similar approach of providing such an abstraction layer. In the process of development, the programmer need to define the mesh and the algorithm on the elements, and then the analysis and back-end implementation is left to the applications. This abstraction is designed to be portable and flexible for not only solving new algorithms but also transforming the user code to a wide range of programming model and the execution on different architecture systems.

We select element-based description in our framework as a domain description language for mesh-based applications since we can sufficiently write a domain description in this form and can easily translate the description to our internal programming model and further to different architectures.

**The Framework Architecture**

Our framework has two main functions: parallelization and performance optimization. Domain description written in domain description language will be translated into target parallel code. During the translation, first parallelization changes the domain description into a parallel code in internal programming model. This translation step will be guided by a performance model in more abstraction level. The parallel code is further refined with performance optimization according to target execution model and efficiency model. This translation step may be done several times automatically, or manually. All these translations are source translations. The framework takes an active library
approach[6], and the generated code supports different programming environments and different hardware architectures, and can be directly compiled by the certain specified compilers.

![Figure 1. Framework Architecture.](image)

**Domain Description**

Domain description is written in a domain specific programming language. A domain description consists of 3 declaration segments: a declaration of a mesh structure, a declaration of kernel operations on nodes of mesh, and a declaration of algorithms over mesh structure using kernel operations. A domain description can be understood as a program in data parallel programming model and taking the form of sequential program. The computation of domain algorithms usually are some loops over the nodes of mesh structure. We use C syntax as our domain description language so that a domain description can almost directly be compiled by a C compiler and run as a sequential program.

The mesh structure can be formed by some relations on some sets. The kernel operations on the mesh structure are defined as a series of kernel functions. The entire domain computation process can be defined through some iterative loops of the kernel functions on the mesh structure.

**Internal Programming Models and Performance Models**

The internal programming model is a hierarchy of groups. There are 3 levels of groups. The top level group is called communicator which consists of many processes, each of which is running in its own addressing space. The middle level group is called a team which consists of many tasks. Each running task, understood as a thread, in a team shares the team addressing space. A low level group is called a bunch. The elements of bunch are called instances which run synchronously under a control of single instruction stream. But each instance in a bunch runs with its own data. A communicator can be viewed as a parallel program of many processes. In each process, there is a team of tasks which can be viewed as multithreading programs sharing common addressing space. In each task, there may be a bunch of instances. The bunch of instances can be viewed as a vectorized program for a SIMD unit in a certain computation architecture.

For performance consideration, we will build for each program a performance model which consists of many program parameters such as arithmetic intensity, maximal arithmetic intensity, memory access stride, message size, etc. Building performance model can be done automatically or manually by analyzing the transformed internal model from domain description. Performance models actually consists of many levels of a hierarchy. An abstraction level is used for the top level of internal programming model and more concrete levels (the efficiency models) are used for further architecture-aware optimizations.
Target Execution Models and Efficiency Models

An execution model consists of a cluster of nodes, each of which consists of many execution contexts, each of which consists of a certain group of computation units. These constitute a hierarchy of 3 levels. There are 3 kinds of execution entities: processes, threads, and silks. A cluster node can run some processes; an execution context can run one thread; a computation unit can run one silk. For each execution model, there is a set of parameters representing its computing and communication capabilities. This set is called the configuration (or profile) of the execution model. These parameters include processor cycle time, the width of computation units, latency and bandwidth for accessing memory, cache size, the size of cache line, latency and bandwidth of interaction between nodes, number of cores, number of execution context for each core, etc. According to the configuration of the execution model a more concrete performance model, called efficiency model, will be built for internal parallel code.

Case Studies and experiment results

We have tested the translated codes in different cases and different modern multi-core and many-core architectures using two benchmark applications aero[10] and airfoil[11]. Aero has a mesh structure with 180700 nodes, 12000 edges, and 180000 elements. We have selected 9 kernel operations on its mesh structure. Airfoil has a mesh structure with 2883601 nodes, 5757200 edges, and 280000 elements. We have selected 5 kernel operations on its mesh structure. In almost all cases our generated codes perform better than the original codes with the speed up from 1.03 to 3.8. The more detailed results are presented in another submitted paper titled with Performance Optimization for Mesh-Based Applications Targeting Many-Core and SIMD Architectures.

Conclusion

In this work, we have proposed and implemented a parallelizing and optimizing application framework for mesh-based domains. The domain problems can be easily described in a domain specific language with sequential programming semantics (or data parallel semantics). The domain description is translated into parallel codes (of an internal programming model) in a different range of programming environments and for a different range of target architectures. Performance optimizations and refinements are done through the translation according to the hierarchy of performance models taking into account the execution models of the target architectures.

This framework separates the domain descriptions from its implementations which are realized for a range of target architectures with more detailed performance optimization and refinements consideration with target execution models. The test experiments show that the framework generates better quality codes of better performance in various cases for many different many-core architectures. We make sure that the framework will also fit the extreme scale systems and very huge mesh applications.

Many properties of the framework need to be tested further. Some optimization techniques need to be considered and implemented within the framework.

Acknowledgement

This research was financially supported by the National Key R&D Program of China (2017YFB0202400, 2017YFB0202403).

References


