A Task-Based Abstraction Layer for User Productivity and Performance Portability in Post-Moore’s Era Supercomputing

Steve Petruzza, Attila Gyulassy, Valerio Pascucci, and Peer-Timo Bremer

Abstract—The proliferation of heterogeneous computing architectures in current and future supercomputing systems dramatically increases the complexity of software development and exacerbates the divergence of software stacks. Currently, task-based runtimes attempt to alleviate these impediments, however their effective use requires expertise and deep integration that does not facilitate reuse and portability. We propose to introduce a task-based abstraction layer that separates the definition of the algorithm from the runtime-specific implementation, while maintaining performance portability.

1 INTRODUCTION

Recent and future supercomputing systems are shifting towards more heterogeneous computing architectures to address post-Moore’s era supercomputing (PMES) and to increase their power efficiency. At the same time, software infrastructures are becoming more heterogeneous, and a number of libraries and frameworks have been proposed to simplify accessing, aggregating and staging simulation data as well as facilitating the implementation of algorithms and integrate them into simulations [4], [6], [9], [13]. Despite these efforts, implementing scalable simulations and analysis algorithms remains challenging, requiring both domain-specific expertise, and in-depth knowledge of the chosen libraries or runtimes. While many runtimes promise genericity, instead, achieving performance comes from tightly coupling application to the underlying software stack, frequently even optimizing to specific machines. Certain simulations can currently afford this tight coupling as they are designed with performance foremost in mind with specific machines and software stack targets (e.g. NAMD [12], ChaNGa [3], Legion S3D [2]), however utility and analytics codes that are designed to be run across simulations cannot be re-implemented for every use case. As data movement becomes the driving bottleneck for exascale computation, these diverse algorithms must be deployed in-situ and their performance portability will determine the success or failure of scientific workflows.

To leverage the increasing complexity of modern machines, task-based runtime systems are being adopted to manage resources, scheduling and execution of tasks on heterogeneous architectures. While simulation codes choose to integrate with application software stack making the portability challenging, with the directed edges of the graph encoding data dependencies. A task-based model provides an inherent separation of concerns in which the algorithm developer is not exposed to any communication, synchronization or other runtime-related concepts, and is also insulated from the architectural properties of a target machine. Additionally, the design naturally allows over-decomposition, which is not only useful for runtimes that provide load balancing but also simplifies debugging at scale.

The task-based abstraction implemented as an Embedded Domain-Specific Language (EDSL) together with a thin software layer is sufficient to automatically adapt a task-based model to the specific runtime systems that are used in a scientific workflow. In contrast to existing approaches, where significant investment is required to deploy an algorithm on a specific runtime system, a task-based model and EDSL allows developers to maintain a single implementation of an algorithm that nevertheless provides a native interface and efficient implementation for a number of different software stacks.

The EDSL and software layer needs to have a frontend with a stable and easy to program interface to express various existing task based runtimes provide the ability to decompose every application in computational components of arbitrary granularity, enabling the definition and implementation of tasks for different hardware architectures. However, comparing the performance of the same algorithm on different runtimes becomes a very expensive task and introduce the big challenge of understanding how the same algorithm should be designed, implemented and executed by different runtimes taking advantage of their distinct data and execution models.

2 TASK ABSTRACTION LAYER

We propose a new programming paradigm for PMES systems using a task graph to encode the data dependencies between data processing stages in algorithms. In our model, a task simply represents the data transformation between inputs and outputs, and is the natural level at which programmers express computational dependencies. A task graph expresses not only dependencies between internal stages of an algorithm, but also dependencies introduced when data is decomposed for distributed computation. For instance, in figure 1, an in-situ distributed volume rendering algorithm is represented in terms of tasks for local rendering and compositing, with the directed edges of the graph encoding data dependencies. A task-based model provides an inherent separation of concerns in which the algorithm developer is not exposed to any communication, synchronization or other runtime-related concepts, and is also insulated from the architectural properties of a target machine. Additionally, the design naturally allows over-decomposition, which is not only useful for runtimes that provide load balancing but also simplifies debugging at scale.

The task-based abstraction implemented as an Embedded Domain-Specific Language (EDSL) together with a thin software layer is sufficient to automatically adapt a task-based model to the specific runtime systems that are used in a scientific workflow. In contrast to existing approaches, where significant investment is required to deploy an algorithm on a specific runtime system, a task-based model and EDSL allows developers to maintain a single implementation of an algorithm that nevertheless provides a native interface and efficient implementation for a number of different software stacks.

The EDSL and software layer needs to have a frontend with a stable and easy to program interface to express various
Much of the complexity of today's runtimes, i.e., MPI, Charm++, Legion, etc., is that they allow user to define an almost arbitrarily complex algorithm or dataflow at runtime. However, in many use cases, like visualization, analysis, or file I/O the necessary steps are known a priori and are static. Relying on this restriction can allow much simpler descriptions of such algorithms. Providing backends that translate a high level description to different task-based runtimes implementation and less on acquiring new skills to implement their application on yet another runtime. We demonstrated that software portability can be easily achieved transparently access simulation data and further uncouple the implementation of an algorithm from the specific application that uses it.

4 Conclusion

The unstoppable divergence of simulation and software stacks will continue in the Post-Moore's era of supercomputing. More than ever, we now need to focus on how to quickly and transparently port algorithms to different software stacks in order to easily integrate with different applications and maximize the utilization of the heterogeneous resources available on the machines. Separating the definition of the algorithm from the specific runtime implementation allows the user to focus more on the algorithm specification and less on acquiring new skills to implement their application on yet another runtime. We implemented and EDSL that translates a task based algorithm definition to different task-based runtimes implementation and demonstrated that software portability can be easily achieved as well as overall performance portability. We believe that this direction can increase user productivity and also allow small contributions, in terms of analysis and visualization algorithms, to find widespread adoption over multiple communities using different software stacks.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work is also supported in part by NSF: CGV: Award:1314896, NSF:IIP Award: 1602127 NSF:ACI:award 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375, and PIPER: ER26142
DE-SC0010498 and by the Department of Energy under the guidance of Dr. Lucy Nowell and Richard Carson. This research used the resources of the Supercomputing Laboratory at KAUST, Saudi Arabia.

REFERENCES


