Exascale-Ready Programming Models for Climate 1 December 2015

Paul Ullrich
University of California, Davis
Gabriele Jost (Intel Federal RND), Bryce Adelstein Lelbach, Hans Johansen (LBNL)

Abstract

It is widely acknowledged that most modern applications codes are ill-prepared for achieving performance and scalability on pre-exascale computing systems. Modern climate modeling systems are no exception, and in many cases scientific compromises have been made to achieve maximum performance. Unfortunately, part of this is due to the dominance of synchronous parallel programming paradigms, such as Communicating Sequential Processes (CSP) and Fork-Join Threading, typically implemented using the Message Passing Interface (MPI) and OpenMP, respectively. These programming models demonstrate satisfactory scalability for climate applications on current parallel systems, mostly due to static, well-balanced workloads. However, the shift towards multi-core architectures has necessitated the use of both models simultaneously – the "MPI+OpenMP" approach. While this has enabled performance improvements on current systems, the approach suffers from the programming challenges inherent in the compromises of using different programming models together. Additionally, future applications are expected to be more dynamic with a high demand for runtime-orchestrated load balancing, to deal with science-driven variations in work load, as well as system reliability and performance variability, which are expected to worsen in future exascale supercomputers.

To address these issues, we propose investigating parallel runtime systems implementing new, task-based programming models. Out of the available runtimes, we will focus on High Performance ParalleX (HPX) [1], Legion [2,3], Open Community Runtime (OCR) [4] and Kokkos [5]. These models typically are asynchronous, and message- or event-driven. Task parallelism, in combination with oversubscription, facilitates dynamic load balancing, latency hiding, and better fault tolerance. These systems abstract the underlying hardware and provide implicit mechanisms for tolerating performance variability and hardware faults. This paper proposes to explore, assess and compare alternative programming models for climate simulations, by developing a software framework that is adaptable to an asynchronous task-based infrastructure.

Background/Research to Date

Several approaches have been proposed for coping with the challenges of scaling applications to exascale. Expected complexities of large-scale systems include a vastly increased number of cores and heterogeneous memories which are possibly non-coherent. A successful transition to exascale performance must address the following critical issues: runtime-orchestrated asynchronous execution, dynamic load balancing, energy efficiency, and fault tolerance.

To address these issues, task-based programming is undergoing rapid and active research as a potential replacement for the MPI+OpenMP model. The task-based model breaks down the active workload into chunks of work, referred to as tasks. Parallelism is orchestrated by a scheduler which oversees the distribution of tasks to CPU cores, so as to achieve an optimum of load balance and data locality. By abstracting the notion of parallelism from the application developer, the task-based model allows for seamless integration with heterogeneous architectures and implicitly supports fault tolerance and load balancing.

Although the ecosystem of programming models is diverse, there are some key differences between many of the frontrunners that have the potential to dramatically impact performance and ease-of-implementation. In OCR the main programming abstractions are event driven tasks, data

blocks, and events. Events are used for synchronizing task execution. Producer tasks deliver a data block to an event. The data block is consumed by tasks dependent on this event. Models such as HPX support task suspension and communication during task execution. HPX is based on the concurrency model from the C++11 standard, and fully supports C++ data structures. A thorough investigation of the impacts of these design decisions on existing climate model codes is necessary to minimize future investment costs. Some preliminary efforts addressing this need are now underway with a block-structured finite element MPI-enabled C++ framework, Tempest [6], as part of the FastForward 2 initiative (https://asc.llnl.gov/fastforward/).

Proposed Direction of Work

The goal of this work would be to assess and compare several modern programming models in the context of global climate modeling. Since all of the above models naturally operate with C/C++, frameworks such as Tempest would be well-suited for supporting the comparison. Once an optimal candidate was selected from the available suite of programming models, this technology could then be adapted for use in other global climate modeling systems. Concerted efforts need to be made to adapt sections of legacy code (such as cloud and radiation parameterizations) to a task-driven model, with less reliance on MPI-segregated shared data.

Connections to Math, Comp Sci & and Climate Science

This project effectively bridges climate science and computer science by investigating how cutting-edge parallel computing technology can improve the performance of climate modeling systems on large-scale parallel computing systems. It further provides valuable feedback to the programming model developers on the capability of application developers to integrate their model into large application codes. It should also be able to demonstrate techniques for testing and validating codes as they evolve from MPI-only to task-driven, shared memory run-times.

Potential Impact on the Field

Supercomputing systems that operate at the exascale are expected to emerge in the next several years, and are an inevitability by the middle of the next decade. The design of climate models that can operate effectively on these architectures is necessary to reach the high model resolutions that are desired for the accurate resolution of meteorological features. In particular, parallel performance, heterogeneous architectures, and fault tolerance are three key issues that will require the adoption of new programming models going forward.

References

[1] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, Dietmar Fey (2014) "HPX – A Task Based Programming Model in a Global Address Space." PGAS 2014: The 8th International Conference on Partitioned Global Address Space Programming Models.

[2] Michael Bauer, Sean Treichler, Elliott Slaughter and Alex Aiken (2012). "Legion: Expressing Locality and Independence with Logical Regions." *Proceedings of the international conference on high performance computing, networking, storage and analysis*. IEEE Computer Society Press, 2012.

[3] Michael Bauer (2014) *Legion: Programming Distributed Heterogeneous Architectures with Logical Regions*. Diss. Stanford University.

[4] Tim Mattson, Rob van der Wijngaart, Zoran Budimlic, Vincent Cave, Sanjay Chatterjee, Romain Cledat, Bala Sshasayee and Vivek Sarkar, *OCR the open community runtime interface, version 0.95*, Technical Report, 2015.

[5] H. Carter Edwards and Daniel Sunderland (2012) "Kokkos Array performance-portable manycore programming model." *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores*. ACM.

[6] Paul A. Ullrich (2014) "A global finite-element shallow-water model supporting continuous and discontinuous elements." *Geosci. Model Dev.*, Volume 7, pp. 5141-5182, doi: 10.5194/gmd-7-3017-2014.