Journal of Computational Science 27 (2018) 303–319

Contents lists available at ScienceDirect

Journal of Computational Science

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

Demonstrating GPU code portability and scalability for radiative heat transfer computations

Brad Peterson[a,*], Alan Humphrey[a], John Holmen[a], Todd Harman[a], Martin Berzins[a], Dan Sunderland[b], H. Carter Edwards[c]

[a] Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112, USA
[b] Sandia National Laboratories, PO Box 5800/MS 1418, Albuquerque, NM 87175, USA
[c] NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, CA 95051, USA

ARTICLE INFO

Article history:
Received 29 December 2017
Received in revised form 21 April 2018
Accepted 10 June 2018
Available online 15 June 2018

Keywords:
Asynchronous many-task runtime
GPU
Scalability
Portability
Radiative heat transfer

ABSTRACT

High performance computing frameworks utilizing CPUs, Nvidia GPUs, and/or Intel Xeon Phis necessitate portable and scalable solutions for application developers. Nvidia GPUs in particular present numerous portability challenges with a different programming model, additional memory hierarchies, and partitioned execution units among streaming multiprocessors. This work presents modifications to the Uintah asynchronous many-task runtime and the Kokkos portability library to enable one single codebase for complex multiphysics applications to run across different architectures. Scalability and performance results are shown on multiple architectures for a globally coupled radiation heat transfer simulation, ranging from a single node to 16,384 Titan compute nodes.

1. Introduction

The need to solve larger and more complex simulation problems while at the same time not incurring additional power costs has led to an increasing focus on GPU and Intel Xeon Phi-based architectures. Many current and future high performance computing (HPC) systems rely on such architectures. In the case of the DOE Titan system, with a theoretical peak performance of 27 petaflops, over 90% of the computational power come from its 18,688 GPUs. These heterogeneous systems pose significant challenges in terms of programmability due to deep memory hierarchies, vendor-specific language extensions and memory constraints, e.g. less device-side memory compared to host memory per node. This paper focuses on scalability, portability, and programmability of multiphysics applications. This work covers (1) scalability improvements necessary to compute a radiation transport problem on 16,384 GPUs on Titan using the Uintah asynchronous many-task runtime, and (2) portability improvements necessary to utilize a single codebase capable of execution on nodes containing CPUs, GPUs, and/or Intel Xeon Phi processors. Nvidia GPUs receive particular emphasis as they introduce four challenges distinct from the CPU execution model: (1) task asynchrony, (2) multiple memory spaces, (3) an additional programming model (e.g. CUDA), and (4) another level of parallelism through partitioned execution units among streaming multiprocessors.

Uintah's emphasis on scalability across a diverse set of HPC architectures is currently driven by the target problem of the University of Utah Carbon Capture Multidisciplinary Simulation Center (CCMSC), funded by the NNSA Predictive Science Academic Alliance Program (PSAAP) II. The CCMSC aims to simulate, using petascale/exascale computing, a 1000 MWe oxy-fired clean coal boiler being developed by Alstom Power to deliver high efficiency electric power generation with carbon capture. A primary CCMSC focus is on using extreme-scale computing for reacting, large eddy simulation (LES)-based codes within the Uintah open source framework, using machines like Titan and the upcoming Summit system in a scalable manner. The physical size of the CCMSC target boiler simulations and the resolution required to resolve the dominant physical processes necessitates the use of systems like DOE Titan at near-capacity.

Radiation is the dominant mode of heat transfer in these boiler simulations. A principal challenge in modeling radiative heat transfer is the nonlocal nature of it. Thermal energy propagates across the entire computational domain from any point in space. Our radiation model, a reverse Monte Carlo ray tracing (RMCRT) technique [1], described further in Section 3 requires an all-to-all communi-

* Corresponding author.
E-mail address: bradp@cs.utah.edu (B. Peterson).

https://doi.org/10.1016/j.jocs.2018.06.005
1877-7503/

cation to replicate the radiative properties and boiler geometry on each node to facilitate local ray tracing. This challenge is addressed by leveraging Uintah's adaptive mesh refinement (AMR) capabilities, using Cartesian mesh patches to generate a fine mesh that is only used locally (close to each grid point) and a successively coarser mesh is used further away, via a level-upon-level approach. This approach is fundamental to the CCMSC target problem, where the entire computational domain needs to be resolved to adequately model the radiative heat flux. Using this approach enabled excellent strong scaling to over 256K CPU cores on the DOE Titan system for problem sizes that were previously intractable with a single fine mesh (single-level) RMCRT approach due to on-node memory constraints [1]. This scaling was consistent with the communication and computation model [1].

The challenges in moving from a CPU to a GPU-based multi-level RMCRT algorithm using this mesh refinement approach have extended well beyond what a typical GPU port of a CPU codebase might entail. A core Uintah design focuses on insulating the application developer from the underlying runtime, which requires more automation of runtime features. Uintah's runtime requires that all host-to-device and device-to-host data copies for computational task dependencies (inputs and outputs), as well as device context management must be handled automatically in the same way MPI messages are generated by the Uintah runtime system [2–4]. Meeting these challenges required numerous runtime changes to support the RMCRT problem on Titan's GPUs.

This paper is an extended form of the workshop paper [5], which addressed scalability and runtime improvements necessary to run this difficult globally-coupled, all-to-all problem to 16,384 GPUs on the DOE Titan system. This extended paper addresses the portability challenges of implementing RMCRT into one single portable codebase using the Kokkos portability library and executing this code on CPUs, GPUs, and Intel Xeon Phi Knights Landing (KNL) architectures.

Prior to this work, Uintah's use of Kokkos has been limited to support for CPU and Xeon Phi processors [6]. This work extends portability support to the GPU. Special focus is given to GPU portability enabling Kokkos to now efficiently execute Uintah's fine-grained tasks on GPUs. In particular, modifications are made to Kokkos itself to enable GPU asynchronous and performant execution of parallel work loops with fewer iterations (i.e. an iteration range the low hundreds). This work also describes modifications for GPU portability that affected Xeon Phi performance and describes how it was addressed to enable one portable codebase across three architectures. Intel Xeon Phi portable performance has been addressed in prior work [6], and does not need to be extended here.

The contributions from the original paper [5] are:

(i) Leveraging Uintah's AMR infrastructure in a novel way to reduce the volume of communication sufficiently so as to allow scalability. Uintah's AMR capabilities are introduced in Section 2, along with an overview of Uintah.
(ii) Changing the way that AMR meshes are stored on the GPU to overcome the limited available GPU global memory. This has entailed a significant extension of the Uintah `GPU DataWarehouse` system [7] to support a mesh-level database, a repository for shared, per-mesh-level variables such as global radiative properties. This has allowed multiple mesh patches, each with associated GPU tasks, to run concurrently on the GPU while sharing coarse, radiation mesh data. This extension of the `GPU DataWarehouse` is discussed in Section 3, which also gives an overview on radiation transport and describes a GPU-based multi-level RMCRT model.
(iii) The introduction of novel non-blocking, thread-scalable data structures for managing asynchronous MPI communication requests, replacing previously problematic mutex-protected vectors of MPI communication records. To be non-blocking a wait, failure, or resource allocation by one thread cannot block progress on any other thread. Non-blocking data-structures are lock-free if at all steps at least one thread is guaranteed to make progress, and are wait-free if at any step all threads are guaranteed to make progress [8]. Section 4 describes these changes and their motivation, and also shows speedups in local MPI communication times made possible through these infrastructure improvements.
(iv) A vastly improved memory allocation strategy to reduce heap fragmentation is covered in Section 4. This strategy allows running simulations at the edge of the nodal memory footprint on machines like Titan.
(v) Determining optimal fine mesh patch sizes to yield GPU performance while maintaining over-decomposition of the computational domain to hide latency. This is covered in Section 5 with strong scaling results over a wide range of GPU counts up to 16,384 GPUs, and also show the results of differing over-decomposition configurations across this range of GPUs.

The four major extensions to this paper from the prior paper [5] are:

- Using and modifying Kokkos to improve performance portability on GPUs. Kokkos's current GPU execution model is bulk synchronous, where a parallel loop is partitioned into many CUDA blocks and the GPU distributes those blocks throughout the GPU device. However, the Uintah asynchronous many-task runtime is designed to asynchronously execute many overlapping finer-grained tasks, many of which require only one CUDA block each. Section 6 describes modifications to Kokkos's GPU execution model so that it is no longer bulk synchronous and can instead overlap many smaller asynchronous execution units.
- Section 7 reviews prior Intel Xeon Phi performance portability work [6] for Uintah and describes portability challenges relating to architecture specific iteration patterns.
- The integration of GPU portability into Kokkos and Uintah is given in Section 8. The challenges here include using Uintah's own memory management system within Kokkos, using Kokkos's portable random number library, and supplying task execution parameters for many architectures.
- Results of running portable code on multiple architectures is given in Section 9. In particular, the results compare three codebases: (1) prior CPU code, (2) prior GPU code, and (3) Kokkos-enabled code. Portability is demonstrated on CPUs, GPUs, and Intel Xeon Phi KNLs. GPU portability is shown before and after Kokkos modifications from Section 6.

An overview of related work is given in Section 10, and the paper concludes in Section 11 with future work in this area.

2. The Uintah code

The Uintah asynchronous many-task (AMT) runtime [2,9] is open-source (MIT License) software that has been widely ported and used for many different types of problems involving fluids, solids, and fluid-structure interaction problems [9], with the latest release in September 2017 [10]. Uintah consists of a set of parallel software components and libraries that facilitate the solution of partial differential equations on structured AMR grids. Uintah presently contains four main simulation components: (1) the multi-material ICE [11] code for compressible flows; (2) the particle-based code MPM [12] for structural mechanics; (3) the combined fluid-structure interaction (FSI) algorithm MPM-ICE [13], and (4) the ARCHES turbulent reacting CFD component [14] that