Improving Uintah's Scalability Through the Use of Portable Kokkos-Based Data Parallel Tasks

John K. Holmen
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, Utah 84112
jholmen@sci.utah.edu

Alan Humphrey
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, Utah 84112
ahumphrey@sci.utah.edu

Daniel Sunderland
Sandia National Laboratories
PO Box 5800 / MS 1418
Albuquerque, New Mexico 87175
dsunder@sandia.gov

Martin Berzins
Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, Utah 84112
mb@sci.utah.edu

ABSTRACT

The University of Utah's Carbon Capture Multidisciplinary Simulation Center (CCMSC) is using the Uintah Computational Framework to predict performance of a 1000 MWe ultra-supercritical clean coal boiler. The center aims to utilize the Intel Xeon Phi-based DOE systems, Theta and Aurora, through the Aurora Early Science Program by using the Kokkos C++ library to enable node-level performance portability. This paper describes infrastructure advancements and portability improvements made possible by the integration of Kokkos within Uintah. This integration marks a step towards consolidating Uintah's MPI+PThreads and MPI+CUDA hybrid parallelism approaches into a single MPI+Kokkos approach. Scalability results are presented that compare serial and data parallel task execution models for a challenging radiative heat transfer calculation, central to the center's predictive boiler simulations. These results demonstrate both good strong-scaling characteristics to 256 Knights Landing (KNL) processors on the NSF Stampede system, and show the KNL-based calculation to compete with prior GPU-based results for the same calculation.

KEYWORDS

Hybrid Parallelism, Kokkos, Knights Landing, Many-Core, MIC, Parallel, Portability, Radiation Modeling, Reverse Monte-Carlo Ray Tracing, Scalability, Stampede, Uintah, Xeon Phi

1 INTRODUCTION

This study is motivated by the University of Utah's participation in the DOE/NNSA's Predictive Science Academic Alliance Program (PSAAP) II initiative. For this project, the University of Utah's Carbon Capture Multidisciplinary Simulation Center (CCMSC) has been using large-scale simulation to predict performance of a next-generation, 1000 MWe ultra-supercritical clean coal boiler. These predictions support the design and evaluation of an existing boiler, which has been developed by Alstom (GE) Power.

CCMSC predictive boiler simulations have been made possible through the use of the Uintah Computational Framework [4] and large high-performance computing (HPC) systems such as the NSF Stampede system and the DOE Mira and Titan systems. Looking ahead, the next phase of simulations will utilize the DOE Theta [10] and Aurora [9] systems through participation in the Aurora Early Science Program. Theta is a 9.65 petaflop many-core system featuring 3,624 nodes based on Intel's second-generation Xeon Phi processor, Knights Landing. Aurora is a next-generation many-core system set to feature upwards of 50,000 nodes based on Intel's forthcoming third-generation Xeon Phi processor, Knights Hill.

The Intel Xeon Phi is a many-core device based on Intel's Many Integrated Core (MIC) Architecture [21, 32]. This architecture delivers high degrees of parallelism by presently offering up to 72 out-of-order cores featuring 4-way hyper-threading and 512-bit SIMD instructions. The latest generation Xeon Phi processor, Knights Landing, is binary compatible with past generations of Intel processors. Though easy to start using, the Xeon Phi poses new challenges for Uintah by requiring greater attention to data movement, thread-scalability, and vectorization to achieve performance. Early studies exploring Uintah's performance on first-generation Xeon Phi co-processors, Knights Corner, have helped demonstrate some of these challenges [14, 27, 28].

For GPU-based systems, NVIDIA CUDA-based data parallel tasks have been introduced within Uintah [17, 18]. However, Uintah's existing serial tasks are still being used for MIC-based HPC systems. This poses challenges due to the large core/thread counts offered by the Xeon Phi. Though suitable for CPU-based HPC systems, Uintah's parallel execution of serial tasks within an MPI process is not viable for large-scale simulations on MIC-based HPC systems.

This challenge is addressed by using the Kokkos C++ library [8] within Uintah's infrastructure. Within an MPI process, the underlying Kokkos back-ends allow data parallel tasks to run on CPU-, GPU-, and MIC-based architectures. For MIC-based systems, this has helped overcome a scalability barrier pertaining to strict domain decomposition requirements. This has allowed us to achieve good strong-scaling characteristics to 256 Knights Landing processors with Uintah's reverse Monte-Carlo ray tracing-based radiation model, more on which will be discussed in Section 3.

The remainder of this paper is structured as follows. Section 2 provides an overview of the Uintah Computational Framework. Section 3 introduces Uintah's reverse Monte-Carlo ray tracing-based radiation model, RMCRT. Section 4 describes the integration of the Kokkos C++ library within Uintah. Section 5 details the evolution of Uintah's MPI+X hybrid parallelism approach. Section 6 presents

strong-scaling results gathered on Knights Landing processors in the NFS Stampede system. Section 7 outlines related work and section 8 concludes this paper.

2 THE UINTAH COMPUTATIONAL FRAMEWORK

The open-source (MIT License) Uintah Computational Framework consists of a set of simulation components and libraries enabling the simulation and analysis of complex chemical and physical reactions. These reactions are modeled by solving partial differential equations on structured adaptive mesh refinement grids. This framework was originally developed as part of the University of Utah's Center for the Simulation of Accidental Fires and Explosions (C-SAFE) initiative in 1997. Since then, Uintah has been widely ported and used to develop novel techniques for understanding a broad set of fluid-structure interaction problems. [3]

Though small-scale simulations are supported, Uintah emphasizes scalability across some of the largest HPC systems available today. Recent studies have demonstrated good scaling characteristics to 96K, 262K, and 512K CPU cores on the NSF Stampede, DOE Titan, and DOE Mira systems, respectively [3, 16, 27]. For Intel Xeon Phi-based systems, recent studies have demonstrated good strong-scaling characteristics to 16 Knights Corner coprocessors on the NSF Stampede system [28]. For the target application, recent studies have demonstrated good strong-scaling characteristics to 262K CPU cores [16] and 16K GPUs [18] on the DOE Titan system. The work presented here extends this by demonstrating good strong-scaling characteristics to 256 Knights Landing processors on the NSF Stampede system.

Released in May of 2017, Uintah release 2.0.0 features four primary simulation components:

- *ARCHES*: This component targets the simulation of turbulent reacting flows with participating media radiation, more on which will be discussed in Section 3.
- *ICE*: This component targets the simulation of both low-speed and high-speed compressible flows.
- *MPM*: This component targets the simulation of multi-material, particle-based structural mechanics.
- *MPM-ICE*: This component corresponds to the combination of the ICE and MPM components for the simulation of fluid-structure interactions.

3 RADIATION MODELING WITHIN UINTAH

The CCMSC uses the ARCHES turbulent combustion simulation component for its predictive boiler simulations. In these simulations, radiation is the dominant mode of heat transfer and consumes a majority of compute time per timestep. At large-scale, additional simulation challenges are faced due to the global, all-to-all nature of radiation [16].

ARCHES is a three-dimensional Large Eddy Simulation (LES) code described further in [34]. This code leverages a low-Mach number (M <0.3), variable density formulation to model heat, mass, and momentum transport in reacting flows. ARCHES was initially developed using the parallel discrete ordinates method [23] and P1 approximation [24] to solve the radiative transport equation. Though scalable, this approach resulted in solution of the associated sparse linear systems being the main computational cost for reacting flow simulations.

To reduce this cost, attention has been given to potentially more efficient reverse Monte-Carlo ray tracing (RMCRT) methods [19, 35]. This has led to the development of a stand-alone RMCRT-based radiation model suitable for use within Uintah's simulation components [17]. With Monte-Carlo ray tracing methods (forward or backward), two approaches are considered to parallelize the computation for structured grids: (1) parallelize by patch-based domain decomposition with local information only and pass ray information at patch boundaries via MPI, and (2) parallelize by patch-based domain decomposition with global information and reconstruct the domain for the quantities of interest on each node by passing domain information via MPI [16]. For the CCMSC's predictive boiler simulations, the first approach becomes intractable due to the ray counts used. These ray counts are orders of magnitude larger than the ray counts used to produce results presented within this paper, which were between 209.5 million and 13.5 billion rays.

In the second approach, the primary difficulty is efficiently constructing the global information for millions of cells in a spatially decomposed (patch-based) domain. With this approach, an all-to-all communication phase is incurred for the radiative properties across the computational domain. Uintah has adopted the second RMCRT parallelization approach, providing global reconstruction of the radiative properties on each node to enable local ray tracing. This model has since been (1) extended to support adaptive mesh refinement (AMR) to achieve scalability by reducing communication volume and computational complexity [16], (2) further adapted to run on GPUs at large-scale using this novel AMR approach [18], and (3) used to explore performance on the Knights Corner coprocessor [14]. Uintah offers multiple RMCRT-based radiation modeling approaches, ranging from a single-level approach to the AMR approach used in [16] and [18].

RMCRT uses random walks to model radiative heat transfer by tracing rays in reverse, *towards* their origin. During traversal, the amount of incoming radiative intensity absorbed by the emitter is computed to aid in solving the radiative transport equation. Figure 1 depicts how a ray is traced backwards from **S** to the emitter, **E**, for single-level RMCRT in a structured grid. Figure 2 depicts how ray traversal might be accomplished when using a 3-level mesh coarsening scheme.

RMCRT lends itself to scalable parallelism by allowing multiple rays to be traced simultaneously at any given cell and/or timestep. Additionally, RMCRT eliminates the need to trace rays that may never reach an origin. However, RMCRT does not eliminate the global, all-to-all nature of radiation. Within Uintah, RMCRT has been parallelized by spatially decomposing the computational domain into patches and tracing rays within a given patch to termination.

As new architectures are introduced within HPC systems, it is important for these RMCRT-based radiation modeling approaches to be portable. Uintah presently maintains separate CPU- and GPU-based implementations. However, this is cumbersome as it requires radiation model changes to be updated across two implementations. The introduction of a Kokkos-based implementation marks

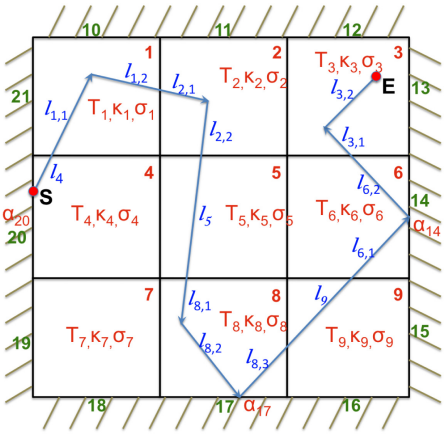

Figure 1: Two-dimensional outline of reverse Monte-Carlo ray tracing for the single-level approach. [17].

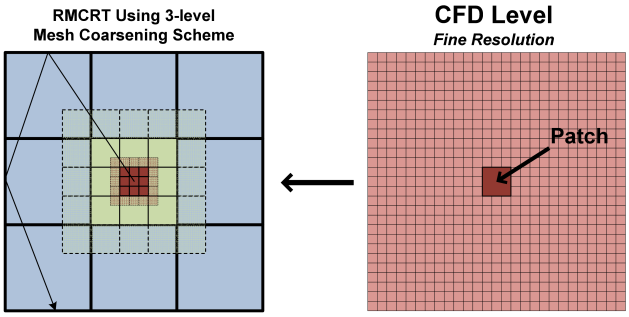

Figure 2: Two-dimensional outline of reverse Monte-Carlo ray tracing for a 3-level mesh refinement approach, illustrating how rays from a fine-level patch (right) may be traced across a coarsened domain (left) [16].

a step towards consolidating RMCRT-based radiation modeling approaches to a single implementation.

The results presented within this paper utilized the following implementations of RMCRT:

- Single-Level RMCRT:CPU
- Single-Level RMCRT:Kokkos
- 2-Level RMCRT:CPU
- 2-Level RMCRT:GPU
- 2-Level RMCRT:Kokkos

4 INCREMENTAL INTEGRATION OF KOKKOS

The Kokkos C++ library [8] is being used to avoid possible code bifurcation when extending Uintah to accelerators and many-core devices. Kokkos reduces the gap between development time and our ability to run on newly introduced systems by enabling performance portability. Here, performance portability refers to the maximization of portability across diverse architectures while striving to achieve performance comparable to hand-tuned code. For these advantages, Kokkos is believed to play a critical role in preparing Uintah for future HPC systems.

Developed by Sandia National Laboratories, Kokkos allows developers to write portable, thread-scalable code optimized for CPU-, GPU-, or MIC-based architectures. This library provides developers with data structures and architecture-aware parallel algorithms (e.g. *parallel_for*, *parallel_reduce*, and *parallel_scan*). When using these tools, Kokkos selects the memory layout and iteration scheme to utilize for a target architecture at compile time. This is enabled through various back-ends, which currently support NVIDIA CUDA, OpenMP, and PThreads programming models. Detailed usage information and demonstrations of performance portability can be found within [7].

As a large existing code base, Uintah must be extended incrementally. An overview of how this may be accomplished has been provided in [36]. To begin our integration, the first step was to incorporate support for Kokkos parallel loops within Uintah's infrastructure. This starting point was chosen to allow application developers to refactor code while simultaneously incorporating Kokkos further within Uintah's infrastructure. Results from early refactoring efforts [30, 36] have been encouraging, motivating our wider-spread integration of Kokkos. Next steps presently underway include the integration of Kokkos parallel loops within ARCHES, addition of support for Kokkos data structures, and further refinement of Uintah's hybrid parallelism approach.

5 EVOLUTION OF UINTAH'S HYBRID PARALLELISM APPROACH

Introduced within [26], the multi-threaded MPI scheduler marks Uintah's adoption of hybrid parallelism. Here, hybrid parallelism refers to the MPI+X programming model, where MPI is used for distributed memory parallelism and X (e.g. OpenMP, PThreads, etc) is used for shared memory parallelism. The work in [26] used MPI+PThreads to overcome scalability barriers imposed by memory footprint limitations on the NSF Kraken system and the DOE Jaguar system. In a similar sense, MIC-specific scalability barriers have necessitated the improvement of Uintah's hybrid parallelism approach.

Figure 3 presents an overview of Uintah's multi-threaded MPI scheduler. Within an MPI process, this scheduler uses several PThreads-based worker threads to execute tasks in parallel. For GPU-based architectures, worker threads execute NVIDIA CUDA-based data parallel tasks. For CPU-based architectures, worker threads execute serial tasks. For MIC-based architectures, worker threads can be used to execute Uintah's existing serial tasks. However, this is not conducive to scalability on many-core systems as will be demonstrated in Section 6.

With past CPU-based simulations, the largest core/thread counts that we have encountered within a node were 16 cores featuring 4 hardware threads on the DOE Mira system. The NSF Stampede system exceeds this by offering 68 cores featuring 4 hardware threads on a Knights Landing node. Though serial tasks have suited CPU-based architectures well, they are undesirable for many-core systems given this increase in core/thread counts. This holds due to the associated domain decomposition requirements. To utilize more worker threads within an MPI process, the computational domain must be subdivided to create an additional patches for each new worker thread.

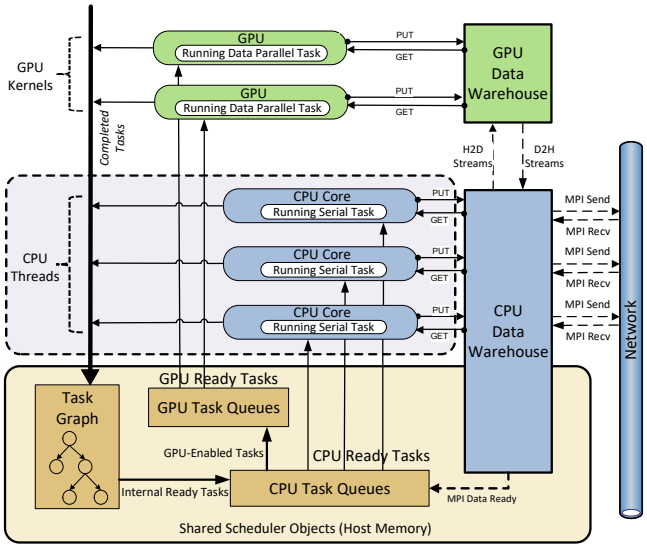

Figure 3: Uintah's multi-threaded MPI scheduler.

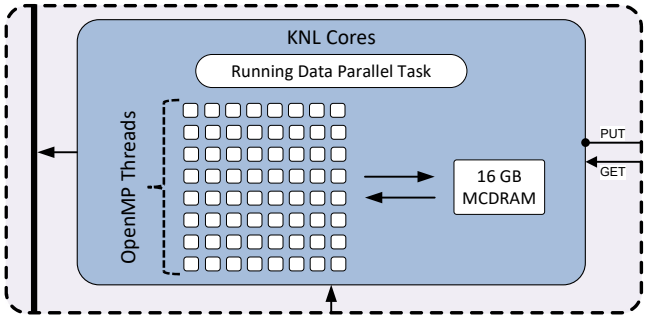

Figure 4: A Kokkos-based data parallel task, illustrating how a Knights Landing processor may be used at the node-level.

To address this scalability barrier in a portable manner, Uintah's dynamic MPI scheduler [29] has been extended to support MPI+Kokkos. The work in [29] introduced an MPI-only scheduler supporting asynchronous, out-of-order task execution to better overlap communication and computation. This extension has enabled the serial execution of Kokkos-based data parallel tasks within an MPI process. Figure 4 demonstrates how a Kokkos-based data parallel task may be used when running on a Knights Landing node.

The serial execution of Kokkos-based data parallel tasks eliminates the need to create an additional patch for each thread used within an MPI process. This offers greater control over patch sizes when balancing local and global communication at scale. Additionally, these tasks offer a potential to improve node-level performance through better utilization of a microarchitecture. For the Knights Landing processor, data parallel tasks allow per-patch work to be computed cooperatively using, for example, all hardware threads within a pair of cores, which share an L2 cache.

The preliminary RMCRT:Kokkos implementations utilized within this paper encapsulated the existing RMCRT:CPU tasks within a C++ functor to enable the use of Kokkos parallel algorithms. Once encapsulated, Uintah utilizes the OpenMP Kokkos back-end to enable data parallel tasks. As opposed to Uintah's worker thread scheduler, per-thread work is now scheduled dynamically by OpenMP.

6 STRONG-SCALING STUDIES

The strong-scaling studies presented within this section solve the Burns and Christon benchmark problem described in [5] and used for recent CPU- and GPU-based studies in [16] and [18], respectively. This problem exercises the radiation physics needed for predictive boiler simulations and the main features of Uintah's AMR support. Specifically, this problem calculates the radiative-flux divergence for each cell within the computational domain. An accuracy analysis verifying Uintah's RMCRT-based radiation model against the Burns and Christon benchmark problem can be found in [20]. More details on Uintah's RMCRT-based radiation model can be found in [16].

The results presented within this section utilized the following implementations of RMCRT:

- *Single-Level RMCRT:CPU*: This is an existing implementation of single-level RMCRT written to use serial tasks.
- *Single-Level RMCRT:Kokkos*: This is a preliminary implementation of single-level RMCRT written to use Kokkos-based data parallel tasks.
- *2-Level RMCRT:CPU*: This is an existing implementation of 2-level RMCRT written to use serial tasks.
- *2-Level RMCRT:GPU*: This is an existing implementation of 2-level RMCRT written to use NVIDIA CUDA-based data parallel tasks.
- *2-Level RMCRT:Kokkos*: This is a preliminary implementation of 2-level RMCRT written to use Kokkos-based data parallel tasks.

Aside from domain decomposition subtleties discussed later in this section, experiments have been run as in [16] and [18] with results averaged over 7 consecutive timesteps. The absorption coefficient was initialized per [5] with a uniform temperature field. For single-level RMCRT simulations, 100 rays were used to compute the radiative-flux divergence for each cell. For 2-level RMCRT simulations, 100 rays were used to compute the radiative-flux divergence for each cell on the fine-level.

With the exception of GPU-based results in Figure 7, these results have been gathered on the KNL Upgrade of the NSF Stampede system [37]. This portion of Stampede features the Intel Xeon Phi 7250 Knights Landing processor and offers a variety of memory and cluster mode configurations. These studies were conducted on Knights Landing processors configured for *Cache-Quadrant* mode. With this in mind, each problem size explored fits within the 16 GB memory footprint of MCDRAM.

These studies emphasize strong-scaling due to the fixed target problem that the CCMSC aims to simulate at large-scale. As such, weak-scaling is not addressed due to the nature of communication growth for this problem, which has been characterized in [16]. Specifically, communication grows quadratically as $O(n^2)$ with respect to the problem size, where n corresponds to the number of communicating MPI processes.

Here, strong-scaling refers to the subdivision of a fixed size problem to support increasing node counts. A fixed number of

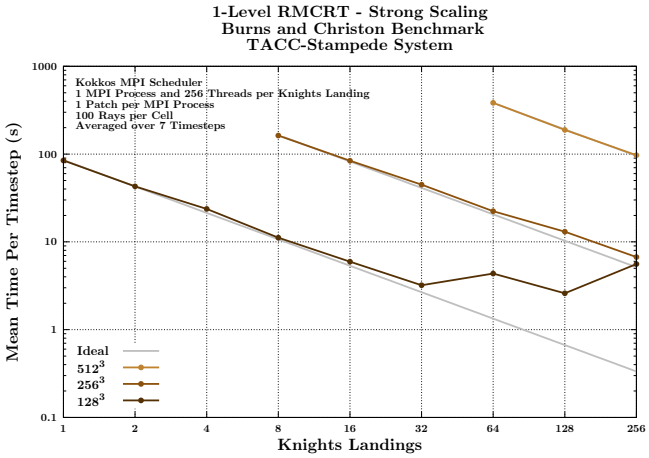

Figure 5: Strong-scaling results to 256 nodes for single-level RMCRT:Kokkos with data parallel tasks on Stampede's Knights Landing processors.

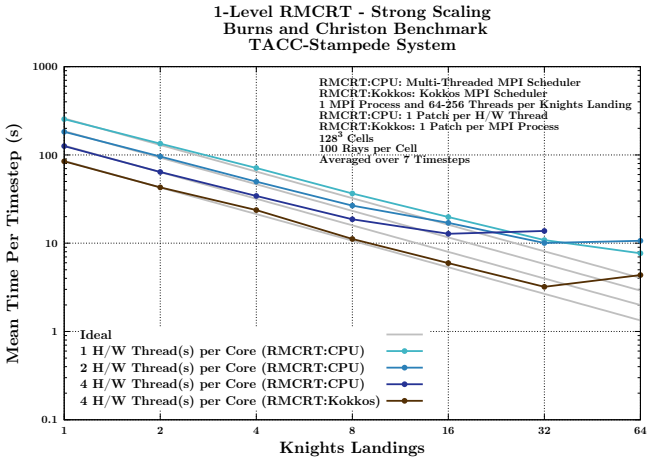

Figure 6: Strong-scaling results to 64 nodes for single-level RMCRT:CPU with serial tasks and single-level RMCRT:Kokkos with data parallel tasks on Stampede's Knights Landing processors.

patches was maintained per node and their size reduced to create additional patches for additional nodes. For simulations using serial tasks, patches were sized to enforce 1 patch per hardware thread. For simulations using data parallel tasks, patches were sized to enforce 1 patch per MPI process unless noted otherwise. For 2-level RMCRT simulations, coarse-level patch sizes were fixed and fine-level patch sizes were reduced as described above.

As a whole, simulations were launched using 1 MPI process per Knights Landing node. Within an MPI process, threads were launched in multiples of 64 to ease domain decomposition. For simulations using serial tasks, PThreads were launched without thread affinity. For simulations using data parallel tasks, OpenMP threads were launched via Kokkos with default affinity settings.

Figure 5 depicts strong-scaling of single-level RMCRT:Kokkos. This implementation features serial execution of data parallel tasks within an MPI process. This figure presents results for three problem sizes (128^3, 256^3, and 512^3 cells). For each problem size, MPI processes were launched with 256 threads to utilize 4 hardware threads per core.

Figure 6 depicts strong-scaling of single-level RMCRT:CPU. This implementation features parallel execution of serial tasks within an MPI process. This figure presents results for three thread counts (64, 128, and 256 threads per MPI process to utilize 1, 2, and 4 hardware threads per core, respectively). For each thread count, a problem size of 128^3 cells was utilized. To enable comparisons, this plot also features single-level RMCRT:Kokkos results from Figure 5 for the corresponding problem size and node counts.

Figure 7 depicts strong-scaling for 2-level RMCRT:CPU. This implementation features parallel execution of serial tasks within an MPI process. This figure presents results for three problem sizes (128^3, 256^3, and 512^3 cells on the fine mesh with 32^3, 64^3, and 128^3 cells on the coarse mesh, respectively). For each problem size, MPI processes were launched with 256 threads to utilize 4 hardware threads per core. To enable comparisons, this plot also features prior 2-level RMCRT:GPU results gathered on the DOE NVIDIA Tesla K20X-based Titan system for the corresponding problem sizes

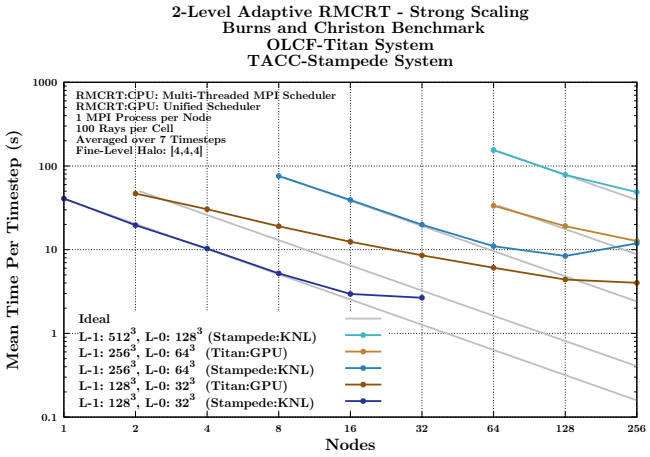

Figure 7: Strong-scaling results to 256 nodes for 2-level RMCRT:CPU with serial tasks on Stampede's Knights Landing processors and 2-level RMCRT:GPU with data parallel tasks on Titan's K20X GPUs.

and node counts. More details on 2-level RMCRT:GPU can be found in [18].

Figure 8 depicts strong-scaling for 2-level RMCRT:Kokkos. This implementation features serial execution of data parallel tasks within an MPI process. This figure presents results for two problem sizes (128^3 and 256^3 cells on the fine mesh with 32^3 and 128^3 cells on the coarse mesh, respectively). For each problem size, MPI processes were launched with 256 threads to utilize 4 hardware threads per core. To enable comparisons, this plot also features a portion of 2-level RMCRT:CPU results from Figure 7 for the corresponding problem sizes and node counts. For 2-level RMCRT:Kokkos, fine-level patches were sized to enforce 8 fine-level patches per MPI process.

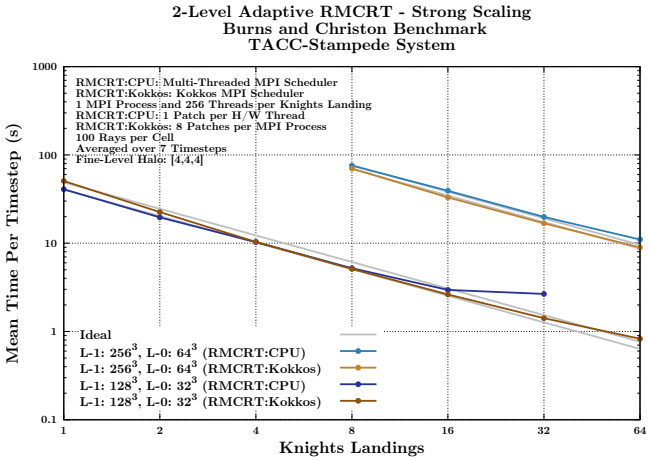

Figure 8: Strong-scaling results to 64 nodes for 2-level RMCRT:Kokkos with data parallel tasks and 2-level RMCRT:CPU with serial tasks on Stampede's Knights Landing processors.

Results presented within Figure 6 demonstrate that as more hardware threads were utilized per core, node-level performance increased at the expense of reductions in strong-scaling efficiency. This is attributed to the strict domain decomposition requirements imposed by the serial task execution model. Though it improved node-level performance, the utilization of more threads per core required immediate reductions in patch size to accommodate additional threads within an MPI process. This expedited the breakdown of scalability, which is attributed to per-patch computation no longer sufficing to hide communication.

While this approach has suited CPU-based architectures well [16, 26], these observations suggest that serial tasks are undesirable for large-scale simulations on many-core systems. Supporting this conclusion, single-level RMCRT:Kokkos results included within Figure 6 suggest that we have overcome the scalability barrier posed by strict domain decomposition requirements through the use of data parallel tasks. This has been achieved with an accompanying improvement in node-level performance, which is believed to be attributed to improvements in microarchitecture utilization enabled via data parallel tasks.

Comparing results presented within Figure 5 to those within Figure 7, single-level RMCRT:Kokkos exhibited strong-scaling characteristics comparable to those of 2-level RMCRT:CPU. This is encouraging as the AMR approach utilized within 2-level RMCRT:CPU enabled strong-scaling characteristics to 262K CPU cores on the DOE Titan system that were previously unattainable via single-level RMCRT:CPU [16]. Further, node-level performance on Stampede's Knights Landing processors outperformed that of Titan's K20X GPUs. Though the K20X is dated, this is encouraging as Titan is one of the largest systems that we currently utilize. For upcoming Theta and Aurora simulations, this suggests a potential for improving boiler performance predictions through the use of finer mesh resolutions and/or more simulated time.

7 RELATED WORK

Uintah is one of many computational frameworks supporting structured adaptive mesh refinement-based applications. MPI+X is commonly used by these frameworks and other codes emphasizing large-scale simulations. MPI+OpenMP is often used for CPU- and MIC-based architectures. MPI+CUDA is often used for GPU-based architectures. An evaluation of these and other MPI+X variants can be found in [25]. Examples of similar frameworks with applications utilizing MPI+OpenMP include BoxLib [38] and Cactus [11]. A survey of representative frameworks, including Uintah, can be found in [6].

Kokkos is one of several libraries aiming to address performance portability with attention to memory access patterns. Examples of similar libraries include HEMI [12] and RAJA [15], developed by NVIDIA and Lawrence Livermore National Lab, respectively. At Sandia National Labs, Kokkos has been integrated within Trilinos [13] and used in codes such as Albany [33] and LAMMPS [31]. Uintah is an early adopter of the MPI+Kokkos hybrid parallelism approach. An analysis of performance portability for Uintah and similar asynchronous many-task runtime systems (Charm++ [22] and Legion [1]) can be found in [2].

8 CONCLUSIONS AND FUTURE WORK

This work has helped advance Uintah's preparedness for large-scale simulations on many-core systems. Perhaps more important, it has improved our readiness for forthcoming simulations on the DOE Theta and Aurora systems. Such readiness promotes more productive use of our Aurora Early Science Program allocation when predicting boiler performance for the PSAAP II project.

These advancements have been made possible by our integration of Kokkos within Uintah. Though already supported for GPU-based architectures, Kokkos back-ends enable data parallel tasks for CPU- and MIC-based architectures. These data parallel tasks have helped overcome a MIC-specific scalability barrier pertaining to strict domain decomposition requirements. The resulting flexibility in domain decomposition allows Uintah to accommodate larger thread counts within an MPI process and offers greater control over the balance between local and global communication. This has been accomplished by allowing MPI processes to utilize fewer, yet larger, patches, improving our ability to hide communication.

These results offer encouragement as we prepare to incorporate support for parallel execution of Kokkos-based data parallel tasks within an MPI process. By adopting the MPI+Kokkos hybrid parallelism approach, we aim to increase Uintah's preparedness for a wider-range of HPC systems (e.g. the upcoming DOE NVIDIA Volta-based Summit system). This continued integration of Kokkos fosters better utilization of a node at the microarchitecture level while avoiding possible code bifurcation.

ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. An award of computing time was provided by the NSF Extreme Science and Engineering Discovery Environment (XSEDE) program. This research used resources of the Texas Advanced Computing Center, under Award Number(s)

MCA08X004 - "Resilience and Scalability of the Uintah Software". We would like to thank TACC for their support through the Stampede Knights Landing Early Science Program, with special thanks to the administrators for making this research possible. Additionally, we would like to thank all of those involved with the CCMSC and Uintah past and present.

REFERENCES

[1] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. 2012. Legion: Expressing locality and independence with logical regions. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*. IEEE Computer Society Press, 66.

[2] J. Bennett, R. Clay, G. Baker, M. Gamell, D. Hollman, S. Knight, H. Kolla, G. Sjaardema, N. Slattengren, K. Teranishi, J. Wilke, M. Bettencourt, S. Bova, K. Franko, P. Lin, R. Grant, S. Hammond, S. Olivier, L. Kale, N. Jain, E. Mikida, A. Aiken, M. Bauer, W. Lee, E. Slaughter, S. Treichler, M. Berzins, T. Harman, A. Humphrey, J. Schmidt, D. Sunderland, P. McCormick, S. Gutierrez, M. Schulz, A. Bhatele, D. Boehme, P. Bremer, and T. Gamblin. 2015. *ASC ATDM level 2 milestone #5325: Asynchronous many-task runtime system analysis and assessment for next generation platforms.* Technical Report. Sandia National Laboratories.

[3] M. Berzins, J. Beckvermit, T. Harman, A. Bezdjian, A. Humphrey, Q. Meng, J. Schmidt, , and C. Wight. 2016. Extending the Uintah Framework through the Petascale Modeling of Detonation in Arrays of High Explosive Devices. *SIAM Journal on Scientific Computing* 38, 5 (2016), 101–122.

[4] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C.A. Wight, and J.R. Peterson. 2010. Uintah: A Scalable Framework for Hazard Analysis. In *Proceedings of the 2010 TeraGrid Conference*. ACM, 3.

[5] S.P. Burns and M.A. Christon. 1997. Spatial domain-based parallelism in large-scale, participating-media, radiative transport applications. *Numerical Heat Transfer* 31, 4 (1997), 401–421.

[6] A. Dubey, A. Almgren, John Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler, B. OfiShea, E. Schnetter, B. Van Straalen, and K. Weide. 2014. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *J. Parallel and Distrib. Comput.* (2014).

[7] H. C. Edwards, C. R. Trott, and J. Amelang. 2015. Kokkos Tutorials. (2015). https://github.com/kokkos/kokkos-tutorials.

[8] H. C. Edwards, C. R. Trott, and D. Sunderland. 2014. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel and Distrib. Comput.* 74, 12 (2014), 3202 – 3216.

[9] Argonne Leadership Computing Facility. 2017. Aurora Web Page. (2017). http://aurora.alcf.anl.gov/.

[10] Argonne Leadership Computing Facility. 2017. Theta Web Page. (2017). https://www.alcf.anl.gov/theta.

[11] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. 2003. *The Cactus Framework and Toolkit: Design and Applications.* Springer Berlin Heidelberg, Berlin, Heidelberg, 197–227.

[12] M. Harris. 2017. Hemi: Simpler, More Portable CUDA C++. (2017). http://harrism.github.io/hemi/.

[13] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. 2005. An Overview of the Trilinos Project. *ACM Trans. Math. Softw.* 31, 3 (Sept. 2005), 397–423.

[14] J. K. Holmen, A. Humphrey, and M. Berzins. 2015. Chapter 13 - Exploring Use of the Reserved Core. In *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*, J. Reinders and J. Jeffers (Eds.). Vol. 2. Morgan Kaufmann, Boston, MA, USA, 229 – 242.

[15] R. D. Hornung and J. A. Keasler. 2014. *The RAJA portability layer: overview and status.* Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

[16] A. Humphrey, T. Harman, M. Berzins, and P. Smith. 2015. A Scalable Algorithm for Radiative Heat Transfer Using Reverse Monte Carlo Ray Tracing. In *High Performance Computing*, Julian M. Kunkel and Thomas Ludwig (Eds.). Lecture Notes in Computer Science, Vol. 9137. Springer International Publishing, 212–230.

[17] A. Humphrey, Q. Meng, M. Berzins, and T. Harman. 2012. Radiation Modeling Using the Uintah Heterogeneous CPU/GPU Runtime System. In *Proceedings of the first conference of the Extreme Science and Engineering Discovery Environment (XSEDE'12)*. Association for Computing Machinery.

[18] A. Humphrey, D. Sunderland, T. Harman, and M. Berzins. 2016. Radiative Heat Transfer Calculation on 16384 GPUs Using a Reverse Monte Carlo Ray Tracing Approach with Adaptive Mesh Refinement. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1222–1231.

[19] I. Hunsaker. 2013. *Parallel distributed, reciprocal Monte Carlo radiation in coupled, large eddy combustion simulations.* Ph.D. Dissertation. Dept. of Chemical Engineering, University of Utah.

[20] I. Hunsaker, T. Harman, J. Thornock, and P.J. Smith. 2011. Efficient parallelization of RMCRT for large scale LES combustion simulations. In *20th AIAA Computational Fluid Dynamics Conference*. 3770.

[21] J. Jeffers and J. Reinders. 2013. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers Inc., Boston, MA, USA.

[22] L. V. Kale and S. Krishnan. 1993. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '93)*. ACM, New York, NY, USA, 91–108.

[23] G. Krishnamoorthy, R. Rawat, and P.J. Smith. 2004. Parallel computations of radiative heat transfer using the discrete ordinates method. *Numerical Heat Transfer* 47, 1 (2004), 19–38.

[24] G. Krishnamoorthy, R. Rawat, and P. J. Smith. 2006. Parallelization of the p-1 radiation model. *Numerical Heat Transfer, Part B: Fundamentals* 49, 1 (2006), 1–17.

[25] M. Martineau, S. McIntosh-Smith, M. Boulton, and W. Gaudin. 2016. An Evaluation of Emerging Many-Core Parallel Programming Models. In *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM'16)*. ACM, New York, NY, USA, 1–10.

[26] Q. Meng, M. Berzins, and J. Schmidt. 2011. Using Hybrid Parallelism to improve memory use in Uintah. In *Proceedings of the TeraGrid 2011 Conference*. ACM.

[27] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins. 2013. Investigating Applications Portability with the Uintah DAG-based Runtime System on PetaScale Supercomputers. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. 96:1–96:12.

[28] Q. Meng, A. Humphrey, J. Schmidt, and M. Berzins. 2013. Preliminary Experiences with the Uintah Framework on Intel Xeon Phi and Stampede. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE 2013)*. 48:1–48:8.

[29] Q. Meng, J. Luitjens, and M. Berzins. 2010. Dynamic Task Scheduling for the Uintah Framework. In *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*. 1–10.

[30] B. Peterson, N. Xiao, J. K. Holmen, S. Chaganti, A. Pakki, J. Schmidt, D. Sunderland, A. Humphrey, and M. Berzins. 2015. *Developing Uintah's Runtime System For Forthcoming Architectures.* Technical Report. SCI Institute.

[31] S. Plimpton. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.* 117, 1 (1995), 1 – 19.

[32] J. Reinders, J. Jeffers, and A. Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming Knights Landing Edition.* Morgan Kaufmann Publishers Inc., Boston, MA, USA.

[33] A. G. Salinger, R. A. Bartett, Q. Chen, X. Gao, G. Hansen, I. Kalashnikova, A. Mota, R. P. Muller, E. Nielsen, J. Ostien, and et al. 2013. Albany: A Component-Based Partial Differential Equation Code Built on Trilinos. *ACM Transaction on Mathematical Software* (Oct 2013).

[34] P. J. Smith, R.Rawat, J. Spinti, S. Kumar, S. Borodai, and A. Violi. 2003. Large eddy simulations of accidental fires using massively parallel computers. In *16th AIAA Computational Fluid Dynamics Conference*. 3697.

[35] X. Sun. 2009. *Reverse Monte Carlo ray-tracing for radiative heat transfer in combustion systems.* Ph.D. Dissertation. Dept. of Chemical Engineering, University of Utah.

[36] D. Sunderland, B. Peterson, J. Schmidt, A. Humphrey, J. Thornock, , and M. Berzins. 2016. An Overview of Performance Portability in the Uintah Runtime System Through the Use of Kokkos. In *Proceedings of the Second Internationsl Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*. IEEE Press, Piscataway, NJ, USA, 44–47.

[37] Texas Advanced Computing Center. 2017. TACC Stampede User Guide. (2017). https://portal.tacc.utexas.edu/user-guides/stampede.

[38] W. Zhang, A. S. Almgren, M. Day, T. Nguyen, J. Shalf, and D. Unat. 2016. BoxLib with Tiling: An AMR Software Framework. *CoRR* abs/1604.03570 (2016).