Lessons Learned and Scalability Achieved when Porting Uintah to DOE Exascale Systems *

John K. Holmen[1][0000−0002−5934−2641], Marta García[2][0000−0002−9495−5443], Allen Sanderson[3], Abhishek Bagusetty[2][0000−0002−9642−921X], and Martin Berzins[3][0000−0002−5419−0634]

[1] Oak Ridge National Laboratory, Oak Ridge TN, USA holmenjk@ornl.gov
[2] Argonne National Laboratory, Lemont IL, USA {mgarcia,abagusetty}@anl.gov
[3] University of Utah, Salt Lake City UT, USA {allen,mb}@sci.utah.edu

Abstract. A key challenge faced when preparing codes for Department of Energy (DOE) exascale systems was designing scalable applications for systems featuring hardware and software not yet available at leadership-class scale. With such systems now available, it is important to evaluate scalability of the resulting software solutions on these target systems. One such code designed with the exascale DOE Aurora and DOE Frontier systems in mind is the Uintah Computational Framework, an open-source asynchronous many-task (AMT) runtime system. To prepare for exascale, Uintah adopted a portable MPI+X hybrid parallelism approach using the Kokkos performance portability library (i.e., MPI+Kokkos). This paper complements recent work with additional details and an evaluation of the resulting approach on Aurora and Frontier. Results are shown for a challenging benchmark demonstrating interoperability of 3 portable codes essential to Uintah-related combustion research. These results demonstrate single-source portability across Aurora and Frontier with scaling characteristics shown to 3,072 Aurora nodes and 9,216 Frontier nodes. In addition to showing results run to new scales on new systems, this paper also discusses lessons learned through efforts preparing Uintah for exascale systems.

Keywords: Asynchronous Many-Task Runtime System, Exascale, Performance Portability, Parallelism and Concurrency, Portability, Software Engineering

1 Introduction

A key challenge developers faced when preparing large-scale simulation codes for Department of Energy (DOE) exascale systems was designing scalable applications for systems featuring hardware and software not yet available at leadership-class scale. Specifically, the exascale DOE Aurora and DOE Frontier systems include Intel- and AMD-based graphics processing units (GPUs), respectively, while heterogeneous petascale high-performance computing (HPC) systems primarily featured NVIDIA-based GPUs. With Frontier in production and Aurora nearing production, it is important for developers to evaluate the scalability of resulting solutions on target systems. One such code designed with Aurora and Frontier in mind is the Uintah Computational Framework, an open-source asynchronous many-task (AMT) runtime system.

Uintah's preparation for exascale began in earnest in 2014 by the University of Utah's Carbon Capture Multidisciplinary Simulation Center and their participation in the National Nuclear Security Administration's Predictive Science Academic Alliance Program (PSAAP) II initiative. Since, development has continued through participation in the Argonne Leadership Computing Facility's Aurora Early Science Program (ESP) as a simulation project. Through these efforts, Uintah adopted a portable MPI+X hybrid parallelism approach using the Kokkos performance portability library (i.e., MPI+Kokkos). The resulting approach has allowed Uintah to run in a performance portable manner across several major HPC systems and DOE exascale testbeds.

The work presented here extends two recent evaluations of this approach on systems featuring AMD-, Intel-, and NVIDIA-based GPUs. Work in [15] evaluated large-scale use of the approach on the DOE Summit and National Science Foundation (NSF) Frontera systems using two exascale target benchmarks. Work in [10] evaluated single-node performance of the approach on DOE exascale testbeds using an intermediate benchmark. This paper builds on these prior works by evaluating large-scale use of the approach on Aurora and Frontier using the more complex of the two benchmarks used in [15]. In doing so, this paper documents Uintah's first efforts to scale across Intel- and AMD-based GPU nodes. In addition to showing results run to new scales on new systems, this paper also discusses lessons learned through efforts preparing Uintah for exascale systems.

To demonstrate task scheduling capabilities, MPI+Kokkos is used to make portable use of Intel- and AMD-based GPUs across the DOE Aurora and DOE Frontier systems, respectively. On Aurora, the approach is shown to scale to 36,864 Intel Data Center GPU Max Series Stacks using MPI+Kokkos with the Kokkos::SYCL backend. On Frontier, the approach is shown to scale to 73,728 AMD MI250X Graphics Compute Dies (GCD) using MPI+Kokkos with the Kokkos::HIP backend. Note, Stacks and GCDs refer to logical GPUs, which there are 2 of on each vendor's physical GPU. This portable scalability has been achieved with a challenging single-source benchmark demonstrating interoperability of 3 portable codes essential to Uintah-related combustion research.

The remainder of this paper is structured as follows. Section 2 provides an overview of the Uintah Computational Framework and its MPI+X task schedulers. Section 3 describes experiment setup and presents results gathered on the DOE Aurora and DOE Frontier systems. Section 4 describes lessons learned when preparing Uintah for exascale systems. Section 5 describes related work and Section 6 concludes this paper and discusses future work.

2 The Uintah Computational Framework

The Uintah Computational Framework is an open-source asynchronous many-task runtime system that has been widely ported. Uintah's application codes target large-scale simulation of fluid-structure interaction problems across diverse HPC systems. Examples of such systems include the DOE Titan, NSF Stampede, and DOE Mira systems [24]. A more detailed description of Uintah and its broad support for HPC systems can be found in a recent paper [15].

With emphasis on maintaining broad support, portability has been a primary focus for Uintah and its users. For this reason, Uintah was an early adopter of Kokkos [26]. Initial efforts improved portability of tasks themselves [11]. When adopted widely in application code, Kokkos was adopted through a Uintah-specific intermediate portability layer [12] to preserve legacy code and ease maintenance. More recent efforts improved portability of task scheduling infrastructure, specifically Uintah's heterogeneous MPI+Kokkos task scheduler [10].

Uintah's task schedulers have made scalability possible across some of the world's largest systems. These schedulers coordinate the selection and execution of tasks while also handling aspects such as communication and host-to-device data transfers. Uintah's heterogeneous MPI+Kokkos task scheduler is based on a production-grade MPI+PThreads+CUDA task scheduler and supports execution of portable tasks across diverse systems. A more detailed description and history of Uintah's MPI+X task schedulers can be found in a recent paper [14].

At the core of Uintah's MPI+X task scheduler is the task executor logic driving task selection and execution. This logic is continuously executed by one-to-many task executors in a given MPI process until all tasks have been processed. Uintah's initial MPI+X task scheduler used PThreads for task executors. However, this posed challenges during initial Kokkos adoption due to issues encountered when mixing PThreads and OpenMP threads. Uintah's initial MPI+Kokkos task scheduler [13] used Kokkos::OpenMP::partition_master for task executors. However, partition_master functionality was eventually deprecated in Kokkos. A visual representation and more complete description of task executor logic can be found in a recent paper [15].

As a part of this work, Uintah's MPI+Kokkos task scheduler has been simplified to make direct use of OpenMP for task executors. This scheduler was then merged into Uintah's `master` branch to replace Uintah's MPI+PThreads+CUDA task scheduler with a portable alternative. Figure 1 shows how OpenMP is used to parallelize task executor logic contained in *RunTasks* as well as general task scheduling capabilities. Tasks can be executed as legacy serial tasks or portable

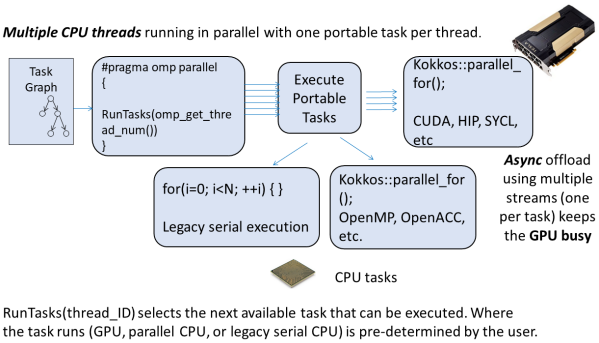

Fig. 1. Uintah's MPI+Kokkos task scheduling capabilities.

tasks using Kokkos backends. How and where a task is executed depends on compile-time and run-time parameters. In each task, developers provide macro-based tags to indicate which backend(s) a task supports. At compile-time, tags and the Kokkos build configuration are used to compile tasks for a given backend. At run-time, command-line options are used to indicate which Kokkos execution policy and related parameters are used. This approach has been demonstrated at large-scale on the NSF Frontera [15], DOE Lassen [14], and DOE Summit [15] systems as well as at the single-node scale on the DOE Crusher, DOE Polaris, and DOE Sunspot exascale testbeds [10]. A more detailed description of how Uintah manages multiple backends can be found in recent paper [12].

3 Experiments

Experiments discussed in this section used a modified Burns and Christon benchmark designed for recent experiments on the DOE Summit and NSF Frontera systems [15]. This benchmark is one of Uintah's exascale target benchmarks and stresses the interoperability of 3 portable codes central to Uintah-related combustion research: (1) Uintah's ARCHES turbulent combustion simulation component, (2) Uintah's standalone linear solver using Lawrence Livermore National Laboratory's (LLNL) *hypre*, and (3) Uintah's standalone reverse Monte-Carlo tracing (RMCRT) radiation model. A key feature making this an important problem for evaluating Uintah's infrastructure at scale are the complex hand-offs between codes due to shared data dependencies. More details on this problem can be found in a recent paper [15].

Simulations were performed on the exascale DOE Aurora and DOE Frontier systems. Aurora and Frontier are No. 2 and No. 1, respectively, on June 2024's TOP500 list[4] with HPL scores of 1.012 EFlop/s and 1.206 EFlop/s, respectively. Aurora[5] is maintained at the Argonne Leadership Computing Facility and is comprised of 10,624 HPE Cray EX nodes, each with two 52-core Intel Xeon CPU Max 9470 processors and six Intel Data Center GPU Max 1550 GPUs, each

[4] https://top500.org/lists/top500/2024/06/
[5] https://www.alcf.anl.gov/aurora

Table 1. Loop execution policy experiments.[a]

System	Nodes	Stacks	MDRange 2,2,2	MDRange 4,4,4	Range 16	Range 32	Team
Aurora[b]	12	144	241 (-)	189 (-)	187 (-)	201 (-)	204 (-)
	24	288	132 (91.6%)	88.3 (107%)	94.3 (98.9%)	101 (99.3%)	111 (91.8%)
	48	576	56.7 (107%)	40.9 (116%)	40.0 (116%)	43.9 (114%)	41.3 (123%)
	96	1152	27.2 (111%)	25.4 (93.0%)	21.7 (107%)	21.7 (115%)	22.2 (114%)

[a]Mean time per timestep reported in seconds (strong-scaling efficiency).
[b]This work was done on a pre-production supercomputer with early versions of the Aurora software development kit.

with 2 Stacks. Each Aurora compute node has 512 GB of DDR5 memory and 64 GB of high-bandwidth memory per CPU and 128 GB of high-bandwidth memory (HBM2E) per GPU. Frontier[6] is maintained at the Oak Ridge Leadership Computing Facility and is comprised of 9,408 HPE Cray EX235a nodes, each with one 64-core AMD EPYC 7A53 CPU and four AMD MI250X GPUs, each with 2 Graphics Compute Dies (GCDs). Each Frontier compute node has 512 GB of DDR4 memory on the CPU and 64 GB of high-bandwidth memory (HBM2E) per GCD. Compute nodes on both Aurora and Frontier are interconnected via HPE's Slingshot 11 interconnect.

Simulations were launched using 1 MPI process per logical GPU (i.e., Stack on Aurora, GCD on Frontier). On Aurora, simulations used MPI+Kokkos with the Kokkos::SYCL backend to target Intel-based GPUs. On Frontier, simulations used MPI+Kokkos with the Kokkos::HIP backend to target AMD-based GPUs. For Sections 3.1 and 3.3, Uintah `master` branch commit `3002edc`, Kokkos release tag `4.3.01`, and *hypre* release tag `v2.31.0` were used. For Section 3.2, Uintah `master` branch commit `28bb54a`, Kokkos release tag `4.2.01`, and *hypre* release tag `v2.31.0` were used. Section 3.1 used various Kokkos loop execution policies. Sections 3.2 and 3.3 use the MDRange policy with a tile size of [4,4,4].

The simulation domain for this problem is a 2-level mesh decomposed into a collection of patches. Patches are distributed across MPI processes and correspond to the collection of cells executed by a loop. The base problem provides each logical GPU with thirty-two 64^3 fine mesh patches and four 32^3 coarse mesh patches using a mesh refinement ratio of 4. The problem was configured to cast 100 rays per cell. Results have been averaged over 7 consecutive timesteps.

3.1 Parameter Tuning Studies

Parameter tuning studies explored the impact of Kokkos execution policy type on time-to-solution and strong-scaling efficiency. These policies are used to manage how Kokkos parallel patterns are executed. Experiments used the MDRange policy with a tile size of [2,2,2] and [4,4,4], Range policy with a chunk size of 16 and 32, and Team policy. Table 1 shows strong-scaling results using 5 configurations across Aurora nodes for the modified Burns and Christon benchmark problem on a 2-level structured adaptive mesh refinement grid. For each pol-

[6] https://www.olcf.ornl.gov/frontier

icy, the base problem was strong-scaled from 12 to 96 Aurora compute nodes. Results show that time-to-solution improvements are achievable. However, the impact on strong-scaling efficiency is unclear due to atypical scaling of this problem on Aurora nodes (e.g., superlinear scaling). This unexpected behaviour is attributed to per-timestep variability and under investigation. Note, Aurora was a pre-production system with early versions of the Aurora software development kit at the time of submission. For remaining experiments discussed here, the MDRange policy with a tile size of [4,4,4] was used.

3.2 Feasibility Studies

Feasibility studies explored the impact of a fixed mesh refinement ratio on weak-scaling efficiency to gauge suitability of the problem for full-system runs. Experiments used a fixed refinement ratio (RR) of 4 when weak-scaling the base problem to larger node counts. Table 2 shows strong-scaling results using 3 problem sizes weak-scaled across Frontier nodes for the modified Burns and Christon benchmark problem on a 2-level structured adaptive mesh refinement grid. L0

Table 2. Fixed mesh refinement ratio experiments.[a]

System	Nodes	GCDs	L0: 96 - 32^3 L1: 768 - 64^3 RR:4	Nodes	GCDs	L0: 576 - 32^3 L1: 4608 - 64^3 RR:4	Nodes	GCDs	L0: 6144 - 32^3 L1: 49.1K - 64^3 RR:4
Frontier	3	24	94.0 (-)	18	144	165 (-)	192	1536	- (-)
	6	48	47.5 (98.9%)	36	288	82.6 (100%)	384	3072	- (-)
	12	96	24.4 (96.3%)	72	576	41.6 (99.2%)	768	6144	- (-)
	24	192	12.5 (94.0%)	144	1152	21.3 (96.7%)	1536	12288	385 (-)

[a]Mean time per timestep reported in seconds (strong-scaling efficiency).

and L1 correspond to coarse and fine mesh patch counts and sizes, respectively, and RR corresponds to refinement ratio. When weak-scaled, the 18- and 192-node problems provide per-device work equivalent to the base problem due to the fixed refinement ratio. Results show that good strong-scaling efficiency is achievable. However when comparing time-to-solution across a given row, the problem is shown to weak-scale poorly (e.g., 1.76x and 18.1x increases when weak-scaling from 3 to 18 and 144 to 1,536 nodes, respectively). Such increases are problematic given limited system access. For this reason, runs with the largest problem were not completed and the problem was not weak-scaled further to reach the full-system. Note, this result is not unexpected due to communication costs related to the global all-to-all nature of radiation. One solution for full-system runs is to use aggressive mesh refinement, which has been shown to improve this problem's ability to weak-scale [17].

3.3 Large-Scale Studies

Large-scale studies explored the impact of an increasing mesh refinement ratio on weak-scaling efficiency to determine if aggressive mesh refinement will

make full-system runs possible. Experiments used refinement ratios of 4, 8, and 16 when weak-scaling the base problem to larger node counts. Figure 2 shows strong-scaling results using 3 problem sizes weak-scaled across Aurora and Frontier nodes for the modified Burns and Christon benchmark problem on a 2-level structured adaptive mesh refinement grid. L0 and L1 correspond to coarse and

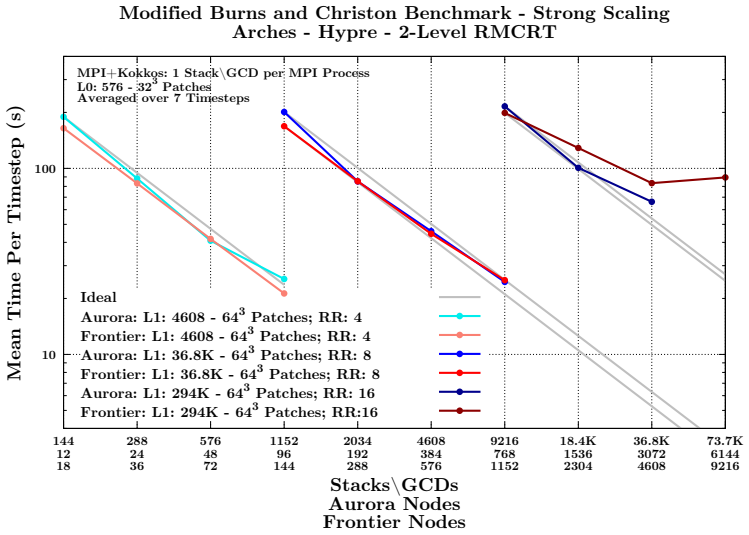

Fig. 2. Increasing mesh refinement ratio experiments. This work was done on a pre-production supercomputer with early versions of the Aurora software development kit.

fine mesh patch counts and sizes, respectively, and RR corresponds to refinement ratio. When increasing refinement ratio, the 18- and 192-node problems provide fewer per-device coarse mesh patch counts than the base problem. When comparing the topmost data point in each strong-scaling trend, results show that good weak-scaling efficiency is achievable with aggressive mesh refinement. However, good strong-scaling can be difficult to achieve with the largest problem used. This difficulty is attributed to poor scaling of reductions across ARCHES simulation variables. These results are encouraging as they demonstrate Uintah's ability to portably run across 29% of Aurora nodes and 98% of Frontier nodes. Further, this has been achieved using largely naive ports of ARCHES loops to device-side Kokkos backends with room for improvement. Note, larger node counts were not explored on Aurora due to limited system availability at the time of submission. Note also, Section 3.1's atypical scaling was also encountered at large-scale here.

4 Lessons Learned

4.1 Continued Portability of AMT Runtime Systems

Over the last decade, Uintah's asynchronous many-task runtime system has had great success in making challenging applications scale. This is a result of the self-

adaptive approach used by the underlying runtime in Uintah and other AMT runtime systems. Such runtime systems ease porting and scaling challenges for application developers by offloading low-level details for making efficient use of the underlying hardware to the runtime itself. However, this is achieved by shifting these challenges to the runtime system developers. Continuous exploration and integration of portable support for emerging architectures helps ensure long-term portability of runtime systems.

4.2 Pre-Production Work Environments

Resilience is an essential skill for working on pre-production systems. It has been particularly important for exascale systems as the path to Aurora (and to a lesser extent, Frontier) has been long and challenging. For example, developers preparing for Aurora faced a product cancellation and ported codes to not only novel hardware but also novel software stacks. Additional complexity was added by the several testbeds used through the 2017 to 2024 Aurora ESP and a six-week software development kit (SDK) upgrade cadence that required continuous testing and bug reporting across all areas of the software stack (e.g., MPI to Kokkos). For example, testbeds spanned multiple generations of GPU and include: Iris (Intel Iris Pro Graphics P580), Arcticus (ATS Intel GPU B Silicon 32GB 960EU SIMD256b), Florentia (PVC Intel GPU A Silicon 128GB 1024EU SIMD512b), and Sunspot (Intel Data Center GPU Max Series). As a result, care was needed to help maintain team cohesion, focus, and motivation as the success of the process is essential for providing confidence for future systems. An approach used to help ensure success was the use of dedicated contacts for application teams and constant involvement of ESP personnel through the process. Difficulties aside, the authors are forever grateful for the opportunity to be a part of the DOE's path to exascale.

4.3 Rapidly Evolving Software

Uintah's portability relies on the Kokkos performance portability layer and third party libraries using Kokkos (e.g., LLNL's *hypre* library of linear solvers). Both the Kokkos and *hypre* teams have been outstanding in their support of Uintah while at the same time making significant advancements to support Intel- and AMD-based GPUs. Combined with Aurora's novel software stack, this has made for a rapidly evolving software ecosystem that has increased the complexity of Uintah's development as the ecosystem stabilizes. For example, it is important to test new third party library releases alongside research software engineering to help identify issues early. Maintaining a patient understanding of the challenges associated with such a fast-moving ecosystem helps maintain healthy collaborations and reasonable expectations. Such collaborations helped ease the modernization of Uintah's use of Kokkos when updating to the latest releases and deprecating a patched version of a 2018 Kokkos release used to accelerate Uintah development [10]. To put the pace into perspective, this patched version was 25 releases behind the Kokkos 4.2.01 release used for this work. Similarly,

the fixed release of `hypre` previously used was 19 releases behind the `hypre` 2.31.0 release used for this work.

4.4 Node Complexity

Increasing node-level parallelism and deep memory hierarchies pose challenges for making efficient use of a node. This is a result of nodes having various non-uniform memory access (NUMA) domains and network interface controller (NIC) card configurations. For example, Frontier has 4 NUMA domains per node with 2 L3 cache regions per NUMA domain for a total of 8 L3 cache regions and 1 NIC connected directly to each MI250X GPU for a total of 4 NICs per node. In addition to high-bandwidth memory on each GPU, Aurora nodes also feature high-bandwidth memory on each CPU [22] with 4 NICs connected directly to the PCIe switch of each socket for a total of 8 NICs per node. This is complicated by non-obvious mapping of resources across NUMA domains. For example, GPU 4 is associated with the L3 cache region of cores 0 through 7 on Frontier. Consulting user guides, reviewing a system's node topology, and verifying that you are running sensibly with regard to NUMA domains and NIC configurations helps improve node use.

4.5 Parameter Tuning

The great flexibility of Kokkos execution policies poses potential parallel pattern design challenges. This is a result of the many ways in which a parallel pattern can be written for a target architecture. Offering mechanisms for users to easily change between execution policy types (e.g., MDRange, Range, Team) and low-level parameters (e.g., chunk size, tile size) helps ease parameter tuning when working to identify optimal run configurations. An example of a mechanism used by Uintah is that command-line options (e.g., `-kokkos_policy range -kokkos_chunk_size 64`) can be used to modify the default parallel pattern design at run-time (i.e., `-kokkos_policy mdrange -kokkos_tile_size 4 4 4`).

5 Related Work

As described in [10], Uintah is one of many asynchronous many-task runtime systems and block-structured adaptive mesh refinement (SAMR) frameworks. Examples of similar AMT runtime systems include Charm++ [20], HPX [19], IRIS [21, 25], Legion [3], PaRSEC [4], and StarPU [2]. Examples of similar SAMR frameworks include BoxLib [29] (superseded by AMReX [28]), Cactus [8], and Parthenon [9]. A review of representative SAMR frameworks, including Uintah, can be found in a recent survey [5].

General lessons learned when moving to Exascale and making use of Kokkos are readily available (e.g., [6, 1, 7]). Kokkos is one of many performance portability layers. Examples of similar performance portability layers include OCCA [23], RAJA [16], and SYCL/DPC++ [27]. A review of exascale challenges and performance portable programming models can be found in a recent survey [18].

6 Conclusions and Future Work

The work presented here captures the culmination of Uintah's decade long preparation for DOE exascale systems through highly collaborative multidisciplinary efforts pursued as a part of PSAAP II and the Aurora Early Science Program. Specifically, this work documents Uintah's first large-scale portable use of the exascale DOE Aurora and DOE Frontier systems. This use has been made possible through Uintah's adoption of a portable MPI+X hybrid parallelism approach using the Kokkos performance portability library (i.e., MPI+Kokkos).

MPI+Kokkos capabilities have been shown across the DOE Aurora and DOE Frontier systems when using this approach to make portable use of Intel- and AMD-based GPUs, respectively. On Aurora, strong-scaling characteristics were shown to 3,072 nodes using MPI+Kokkos at scale with the Kokkos::SYCL backend. On Frontier, strong-scaling characteristics were shown to 9,216 nodes using MPI+Kokkos at scale with the Kokkos::HIP backend. Preliminary experiments used to identify approaches needed to reach this scale and general lessons learned were also discussed.

The portable scalability shown here offers encouragement as we prepare to scale further across Aurora. Next steps include further investigation of performance variability encountered on Aurora nodes and ARCHES-related scalability barriers. Once understood, we aim to stress Uintah's infrastructure across the full Aurora system to help identify issues and scalability barriers encountered at larger scales. Such efforts will help further improve Uintah's readiness for large-scale portable science runs across DOE exascale systems.

Acknowledgement. This material is based upon work originally supported by the Department of Energy, National Nuclear Security Administration, under Award Number(s) DE-NA0002375. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work is associated with an ALCF Aurora Early Science Program project. This work was supported by the Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Support for Allen Sanderson comes from the University of Texas at Austin under Award Number(s) UTA19-001215 and a gift from the Intel One API Centers Program. We would like to thank the ALCF and OLCF for early system access with special thanks to Varsha Madananth for her continued support through the Aurora Early Science Program.

References

1. Abdelfattah, A., Barra, V., Beams, N., Bleile, R., Brown, J., Camier, J.S., Carson, R., Chalmers, N., Dobrev, V., Dudouit, Y., Fischer, P., Karakus, A., Kerkemeier, S., Kolev, T., Lan, Y.H., Merzari, E., Min, M., Phillips, M., Rathnayake, T.,

Rieben, R., Stitt, T., Tomboulides, A., Tomov, S., Tomov, V., Vargas, A., Warburton, T., Weiss, K.: GPU algorithms for Efficient Exascale Discretizations. Parallel Computing **108**, 102841 (2021)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience **23**(2), 187–198 (2011)
3. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: Expressing locality and independence with logical regions. In: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. p. 66. IEEE Computer Society Press (2012)
4. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J.J.: PaRSEC: Exploiting Heterogeneity to Enhance Scalability. Computing in Science Engineering **15**(6), 36–45 (Nov 2013)
5. Dubey, A., Almgren, A., Bell, J., Berzins, M., Brandt, S., Bryan, G., Colella, P., Graves, D., Lijewski, M., Löffler, F., O'Shea, B., Schnetter, E., Straalen, B.V., Weide, K.: A survey of high level frameworks in block-structured adaptive mesh refinement packages. Journal of Parallel and Distributed Computing (2014)
6. Dubey, A., McInnes, L.C., Thakur, R., Draeger, E.W., Evans, T., Germann, T.C., Hart, W.E.: Performance portability in the exascale computing project: Exploration through a panel series. Computing in Science & Engineering **23**(5), 46–54 (2021)
7. Evans, T.M., Siegel, A., Draeger, E.W., Deslippe, J., Francois, M.M., Germann, T.C., Hart, W.E., Martin, D.F.: A survey of software implementations used by application codes in the exascale computing project. The International Journal of High Performance Computing Applications **36**(1), 5–12 (2022)
8. Goodale, T., Allen, G., Lanfermann, G., Massó, J., Radke, T., Seidel, E., Shalf, J.: The cactus framework and toolkit: Design and applications. In: Palma, J.M.L.M., Sousa, A.A., Dongarra, J., Hernández, V. (eds.) High Performance Computing for Computational Science — VECPAR 2002. pp. 197–227. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
9. Grete, P., Dolence, J.C., Miller, J.M., Brown, J., Ryan, B., Gaspar, A., Glines, F., Swaminarayan, S., Lippuner, J., Solomon, C.J., Shipman, G., Junghans, C., Holladay, D., Stone, J.M., Roberts, L.F.: Parthenon—a performance portable block-structured adaptive mesh refinement framework. The International Journal of High Performance Computing Applications **37**(5), 465–486 (2023)
10. Holmen, J.K., García, M., Bagusetty, A., Sanderson, A., Berzins, M.: Making Uintah Performance Portable for Department of Energy Exascale Testbeds. In: Euro-Par 2023: Parallel Processing. pp. 1–12 (2024)
11. Holmen, J.K., Humphrey, A., Sunderland, D., Berzins, M.: Improving Uintah's Scalability Through the Use of Portable Kokkos-Based Data Parallel Tasks. In: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact. pp. 27:1–27:8. PEARC17, ACM, New York, NY, USA (2017)
12. Holmen, J.K., Peterson, B., Berzins, M.: An approach for indirectly adopting a performance portability layer in large legacy codes. In: 2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC). pp. 36–49 (2019)
13. Holmen, J.K., Peterson, B., Humphrey, A., Sunderland, D., Diaz-Ibarra, O.H., Thornock, J.N., Berzins, M.: Portably Improving Uintah's Readiness for Exascale Systems Through the Use of Kokkos. Tech. Rep. UUSCI-2019-001, SCI Institute (2019)

14. Holmen, J.K., Sahasrabudhe, D., Berzins, M.: A Heterogeneous MPI+PPL Task Scheduling Approach for Asynchronous Many-Task Runtime Systems. In: Proceedings of the Practice and Experience in Advanced Research Computing 2021 on Sustainability, Success and Impact (PEARC21). ACM (2021)
15. Holmen, J.K., Sahasrabudhe, D., Berzins, M.: Porting Uintah to Heterogeneous Systems. In: Proceedings of the Platform for Advanced Scientific Computing Conference. Association for Computing Machinery, New York, NY, USA (2022)
16. Hornung, R.D., Keasler, J.A.: The RAJA portability layer: overview and status. Tech. rep., Lawrence Livermore National Laboratory, Livermore, CA (2014)
17. Humphrey, A., Berzins, M.: An Evaluation of An Asynchronous Task Based Dataflow Approach For Uintah. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). vol. 2, pp. 652–657 (July 2019)
18. Johnson, A.: Area Exam: General-Purpose Performance Portable Programming Models for Productive Exascale Computing (2020)
19. Kaiser, H., Diehl, P., Lemoine, A.S., Lelbach, B.A., Amini, P., Berge, A., Biddiscombe, J., Brandt, S.R., Gupta, N., Heller, T., Huck, K., Khatami, Z., Kheirkhahan, A., Reverdell, A., Shirzad, S., Simberg, M., Wagle, B., Wei, W., Zhang, T.: HPX - The C++ Standard Library for Parallelism and Concurrency. Journal of Open Source Software **5**(53), 2352 (2020)
20. Kale, L.V., Krishnan, S.: 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. pp. 91–108. OOPSLA '93, ACM, New York, NY, USA (1993)
21. Kim, J., Lee, S., Johnston, B., Vetter, J.S.: IRIS: A Portable Runtime System Exploiting Multiple Heterogeneous Programming Systems. In: 2021 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–8 (2021)
22. McCalpin, J.D.: Bandwidth Limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors. In: International Conference on High Performance Computing. pp. 403–413. Springer (2023)
23. Medina, D.S., St-Cyr, A., Warburton, T.: OCCA: A unified approach to multithreading languages. arXiv preprint arXiv:1403.0968 (2014)
24. Meng, Q., Humphrey, A., Schmidt, J., Berzins, M.: Investigating applications portability with the uintah DAG-based runtime system on PetaScale supercomputers. In: SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 96:1–96:12 (2013)
25. Monil, M.A.H., Miniskar, N.R., Liu, F.Y., Vetter, J.S., Valero-Lara, P.: LaRIS: Targeting Portability and Productivity for LAPACK Codes on Extreme Heterogeneous Systems by Using IRIS. In: 2022 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA). pp. 12–21 (2022)
26. Peterson, B., Xiao, N., Holmen, J.K., Chaganti, S., Pakki, A., Schmidt, J., Sunderland, D., Humphrey, A., Berzins, M.: Developing uintah's runtime system for forthcoming architectures. Tech. rep., SCI Institute (2015)
27. Reinders, J., Ashbaugh, B., Brodman, J., Kinsner, M., Pennycook, J., Tian, X.: Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL. Springer Nature (2021)
28. Zhang, W., Almgren, A., Beckner, V., Bell, J., Blaschke, J., Chan, C., Day, M., Friesen, B., Gott, K., Graves, D., Katz, M., Myers, A., Nguyen, T., Nonaka, A., Rosso, M., Williams, S., Zingale, M.: AMReX: a framework for block-structured adaptive mesh refinement. Journal of Open Source Software **4**(37), 1370 (2019)
29. Zhang, W., Almgren, A.S., Day, M., Nguyen, T., Shalf, J., Unat, D.: Boxlib with tiling: An AMR software framework. CoRR **abs/1604.03570** (2016)