Making Uintah Performance Portable for Department of Energy Exascale Testbeds *

John K. Holmen[1], Marta García[2], Abhishek Bagusetty[2], Allen Sanderson[3], and Martin Berzins[3]

[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. To help ease ports to forthcoming Department of Energy (DOE) exascale systems, testbeds have been made available to select users. These testbeds are helpful for preparing codes to run on the same hardware and similar software as in their respective exascale systems. This paper describes how the Uintah Computational Framework, an open-source asynchronous many-task (AMT) runtime system, has been modified to be performance portable across the DOE Crusher, DOE Polaris, and DOE Sunspot testbeds in preparation for portable simulations across the exascale DOE Frontier and DOE Aurora systems. The Crusher, Polaris, and Sunspot testbeds feature the AMD MI250X, NVIDIA A100, and Intel PVC GPUs, respectively. This performance portability has been made possible by extending Uintah's intermediate portability layer [18] to additionally support the Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends. This paper also describes notable updates to Uintah's support for Kokkos, which were required to make this extension possible. Results are shown for a challenging radiative heat transfer calculation, central to the University of Utah's predictive boiler simulations. These results demonstrate single-source portability across AMD-, NVIDIA-, and Intel-based GPUs using various Kokkos back-ends.

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

1 Introduction

Forthcoming Department of Energy (DOE) exascale systems pose new challenges for large-scale simulation codes. These challenges include understanding how to manage the increased concurrency, deep memory hierarchies, heterogeneity, and diversity of such systems. Most notable among challenges are the new hardware and software featured among systems such as the exascale DOE Frontier [12] and DOE Aurora [11], which include AMD- and Intel–based GPUs, respectively. This is a challenge as it is a significant departure from prior heterogeneous high performance computing (HPC) systems featuring NVIDIA-based GPUs.

Development for exascale systems is enabled and simplified by testbeds that have been made available to select users through early access programs such as the Aurora Early Science Program and the Frontier Center for Accelerated Application Readiness Program. These testbeds feature the same hardware and similar software as in their respective exascale system. This availability eases the preparation of user codes for forthcoming exascale systems by allowing developers to port their codes to the target architectures while waiting for exascale systems to enter production and open to users.

This paper describes how the Uintah Computational Framework, an open-source asynchronous many-task runtime system, has been extended to run in a performance portable manner across the DOE Crusher, DOE Polaris, and DOE Sunspot testbeds in preparation for portable simulations across the exascale DOE Frontier and DOE Aurora systems. The Crusher, Polaris, and Sunspot testbeds feature the AMD MI250X, NVIDIA A100, and Intel PVC GPUs, respectively. These runs have been made possible by extending Uintah's intermediate portability layer [18] to also support the Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends. This paper also describes notable updates to Uintah's support for Kokkos, which were required to make this extension possible. These updates include rewriting device-specific portions of Uintah's runtime to make use of portable Kokkos abstractions rather than raw CUDA.

To demonstrate Kokkos capabilities, a case study using Uintah's newly extended intermediate portability layer and runtime system are examined for a challenging radiative heat transfer calculation, central to the University of Utah's predictive boiler simulations. This case study shows single-source portability across AMD-, NVIDIA-, and Intel-based GPUs with various Kokkos back-ends. For AMD-based GPUs, single-source portability is shown using the Kokkos::HIP and Kokkos::OpenMPTarget back-ends. For NVIDIA-based GPUs, single-source portability is shown using the Kokkos::CUDA, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends. For Intel-based GPUs, single-source portability is shown using the Kokkos::OpenMPTarget and Kokkos::SYCL back-ends. Note, an experimental Kokkos::OpenACC [43] back-end providing cross-vendor support is also available. However, functionality of this back-end was not tested as a part of this work.

The remainder of this paper is structured as follows. Section 2 provides an overview of the Uintah Computational Framework. Section 3 describes the extension of Uintah's intermediate portability layer to support the Kokkos::HIP,

Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends and updates to Uintah's support for Kokkos. Section 4 describes the benchmark used for experiments. Section 5 describes the systems used for experiments and presents results gathered on the DOE Crusher, DOE Polaris, and DOE Sunspot testbeds. Section 6 describes related work and Section 7 concludes this paper.

2 The Uintah Computational Framework

The Uintah Computational Framework is an open-source asynchronous many-task (AMT) runtime system and block-structured adaptive mesh refinement (SAMR) framework specializing in large-scale simulation of fluid-structure interaction problems. These problems are modeled by solving partial differential equations on structured adaptive mesh refinement grids. Uintah is based upon novel techniques for understanding a broad set of fluid-structure interaction problems [4].

Through its lifetime, Uintah has been ported to a diverse set of major HPC systems. Examples using Uintah's MPI+Kokkos capabilities include the NSF Frontera [20], DOE Lassen [19], DOE Summit [20], NSF Stampede 2 [17, 18], DOE Theta [35], and DOE Titan [35] systems. Other examples include the National Research Center of Parallel Computer Engineering and Technology (NRCPC) Sunway TaihuLight [44], DOE Titan [33, 23, 25], NSF Stampede [33, 17], DOE Mira [33, 4], and NSF Blue Waters [4] systems.

The work presented here extends past efforts by demonstrating Uintah's first successful ports to heterogeneous HPC systems featuring AMD- and Intel-based GPUs. This work makes use of Uintah's MPI+Kokkos capabilities through the runtime's heterogeneous MPI+Kokkos task scheduler [19]. This scheduler supports use of 1 GPU and a subset of CPU cores per MPI process on heterogeneous CPU+GPU nodes and executes tasks simultaneously across the host and device based upon user-specified tags indicating where a task can run. For this work, the majority of tasks were run on the GPU. Note, Uintah does not yet support GPU-aware MPI. More details on Uintah's heterogeneous MPI+Kokkos task scheduler can be found in a recent paper [19].

3 Extending Uintah's Intermediate Portability Layer

Introduced in 2019 [18], Uintah's intermediate portability layer consists of 3 components: (1) loop-level support providing application developers with framework-specific abstractions (e.g., generic parallel loop statements) that map to interface-specific abstractions (e.g., PPL-specific parallel loop statements), (2) application-level support that includes a tagging system to identify which interfaces are supported by a given loop, and (3) build-level support that includes selective compilation of loops to allow for incremental refactoring and simultaneous use of multiple underlying programming models for heterogeneous HPC systems. More details on Uintah's intermediate portability layer can be found in a recent paper [18].

The goal of this intermediate layer is for application developers to, hopefully, need only adopt the layer once to support current and future interfaces to underlying programming models. For application developers, this layer allows for easy adoption of underlying programming models without requiring knowledge of low-level implementation details. For infrastructure developers, this layer allows for easy addition, removal, and tuning of interfaces behind-the-scenes in a single location, reducing the need for far-reaching changes across application code.

As a part of this work, Uintah's intermediate portability layer has been extended to support the Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends. Prior to this extension, however, notable updates to Uintah's use of Kokkos were necessary to make use of the latest Kokkos releases. Section 3.1 describes these updates and the state of Uintah's support for Kokkos. Note, these updates and extensions were made without any required changes to user-facing abstractions.

3.1 State of Uintah's Support for Kokkos

Prior to this work, Uintah's support for Kokkos was limited to use of the Kokkos::OpenMP and Kokkos::CUDA back-ends. This support had two key limitations: (1) use of the Kokkos::CUDA back-end required use of a patched version of Kokkos release 2.7.00 from May 2018 with custom modifications from Uintah developers to add support for asynchronous execution [35], and (2) Uintah's heterogeneous MPI+Kokkos task scheduler [19] used raw CUDA behind-the-scenes for task scheduling, which is described further in [20]. Though they eased rapid research development, these limitations posed challenges when working to additionally support the Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends, which required use of Kokkos releases newer than 2.7.00.

Prior to adding support for new back-ends, limitation (1) was addressed first to deprecate Uintah's Kokkos 2.7.00 patch and modernize Uintah's use of Kokkos. This patch pre-dated Kokkos execution space instance functionality and implemented instance-like functionality to achieve asynchronous execution of parallel patterns when using the Kokkos::CUDA back-end. Deprecation was a critical first step as Kokkos development has been advancing rapidly with regular releases adding new functionality and back-end support. To deprecate this patch, the only changes required in Uintah were minor interface changes. Deprecation of this patch has been key for allowing Uintah's use of Kokkos to be updated more quickly as new Kokkos versions are released. Additionally, this allows us to stay up to date with third party libraries used by Uintah that also make use of the latest Kokkos releases (e.g., Hypre [13]).

Next, limitation (2) was addressed to remove the remaining raw CUDA used in Uintah's heterogeneous MPI+Kokkos task scheduler. This task scheduler used cudaMemcpyAsync for asynchronous host-to-device (H2D) and device-to-host (D2H) transfers and cudaStreamQuery to check the status of transfers. CUDA streams, CUDA memory allocations, and CUDA kernels were also used behind-the-scenes. Raw CUDA was replaced with portable alternatives by (1) replac-

ing use of CUDA streams with Kokkos execution space instances, (2) replacing use of CUDA memory allocations with Kokkos::kokkos_malloc, (3) replacing use of CUDA kernels with Kokkos::parallel_for, (4) replacing use of cudaMemcpyAsync with Kokkos::deep_copy, and (5) replacing use of cudaStreamQuery with Kokkos::fence.

With limitation (1) and (2) addressed as a part of this work, Uintah's use of Kokkos was then updated from the May 2018 release to 2023 releases. Specifically, March 2023 release 4.0.0 was used for this work. This update allowed Uintah to make use of newly supported Kokkos back-ends including Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL. To support these back-ends, Uintah's Kokkos::CUDA-specific device support was broadened to offer general device support through multiple back-ends. To ease parameter tuning, command-line options were also added to allow the user to specify the execution policy type (e.g., TeamPolicy, RangePolicy, and MDRangePolicy) and low-level parameters (e.g., chunk size and tile size) at run-time. Note, Uintah's use of Kokkos::OpenMP::partition_master was also deprecated as a part of this update due to the functionality being deprecated in Kokkos. A replacement was implemented using Kokkos::partition_space.

4 Radiation Modeling

Parallel reverse Monte-Carlo ray tracing (RMCRT) methods [23, 25] are one of several methods available within Uintah for solving the radiative transport equation. RMCRT models radiative heat transfer using random walks across rays cast throughout the computational domain. These rays are traced in reverse, towards their origin, to eliminate the need to trace rays that may never reach an origin. During ray traversal, the amount of incoming intensity absorbed by the origin is computed. This incoming intensity is then used to aid in solving the radiative transport equation. The work presented here uses Uintah's 2-Level RMCRT-based radiation model to solve the Burns and Christon benchmark problem described in [6]. More detailed information on RMCRT can be found in a recent dissertation [22].

5 Experiments

Experiments made use of the DOE Crusher, DOE Polaris, and DOE Sunspot testbeds at the Oak Ridge Leadership Computing Facility and Argonne Leadership Computing Facility, respectively. Crusher, Polaris, and Sunspot feature AMD-, NVIDIA-, and Intel-based GPUs, respectively. For this reason, performance portability is important.

5.1 Crusher

Crusher is a testbed featuring the same hardware and similar software as the exascale Frontier system. Crusher and Frontier are maintained at the Oak Ridge

Leadership Computing Facility. Frontier and Crusher (also known as Frontier TDS) are currently Number 1 and Number 32 on June 2023's Top500 list [38].

The Crusher testbed is comprised of 192 HPE Cray EX235a nodes, each with one 64-core AMD EPYC 7A53 "Optimized 3rd Gen EPYC" CPU and four AMD MI250X GPUs, each with 2 Graphics Compute Dies (GCDs). Each compute node has 512 GB of DDR4 memory and 512 GB of high-bandwidth memory (HBM2E), 64 GB per GCD. The CPU is connected to the GPUs via AMD's Infinity Fabric which delivers a bandwidth of 36+36 GB/s. All GCDs on a Crusher node are interconnected via Infinity Fabric delivering up to 50+50 GB/s for GCDs across GPUs, and up to 200+200 GB/s for GCDs on the same GPU. Compute nodes on Crusher are interconnected via HPE's Slingshot 11 interconnect.

5.2 Polaris

Polaris is a testbed featuring a similar heterogeneous CPU+GPU node configuration and similar software as the forthcoming exascale Aurora system. Polaris and Aurora are maintained at the Argonne Leadership Computing Facility. Polaris is currently Number 19 on June 2023's Top500 list [38].

The Polaris testbed is comprised of 560 HPE Apollo Gen10+ nodes, each with one 32-core AMD EPYC Milan 7543P CPU and four NVIDIA A100 GPUs connected via NVLink. Each compute node has 512 GB of DDR4 memory and 160 GB of high-bandwidth memory (HBM2), 40 GB per GPU. Compute nodes on Polaris are currently interconnected via HPE's Slingshot 10 interconnect and are scheduled to be upgraded to Slingshot 11 in 2023.

5.3 Sunspot

Sunspot is a testbed featuring the same hardware and similar software as the forthcoming Aurora system. Sunspot and Aurora are maintained at the Argonne Leadership Computing Facility.

The Sunspot testbed is comprised of 128 HPE Cray EX nodes, each with two Intel Xeon CPU Max Series(Sapphire Rapids) processors and six Intel Data Center GPU Max Series (Ponte Vecchio/PVC), each with 2 Stacks. Each compute node has 512 GB of DDR5 memory, 512 GB of high-bandwidth memory (HBM) on each CPU, and 128 GB of high-bandwidth memory (HBM) on each GPU. Compute nodes on Sunspot are interconnected via HPE's Slingshot 11 interconnect.

5.4 Testbed Comparisons

For testbed comparisons, experiments explored performance of various GPU architectures (i.e., AMD MI250X, NVIDIA A100, and Intel PVC) using problems sized to provide each GCD/GPU/Stack with eight 128^3 fine mesh patches. Note, additional demonstration of Uintah's portable capabilities can be found among

Table 1. Single-node timings across AMD MI250X, NVIDIA A100, and Intel PVC GPUs. For Crusher, 1 MPI process is used per AMD MI250X GCD. For Polaris, 1 MPI process is used per NVIDIA A100 GPU. For Sunspot, 1 MPI process is used per Intel PVC Stack.

Testbed Used	Devices Used	Back-End Used	Execution Policy	Mean Time per Timestep (s)
Crusher	1x MI250X (2 GCDs)	Kokkos::HIP	MDRange	51.7
	1x MI250X (2 GCDs)	Kokkos::OpenMPTarget	MDRange	53.3
Polaris	1x A100	Kokkos::CUDA	Range	28.9
	1x A100	Kokkos::OpenMPTarget	Team	48.7
	1x A100	Kokkos::SYCL	Range	37.1
Sunspot	1x PVC (2 Stacks)	Kokkos::OpenMPTarget	Team	55.6
	1x PVC (2 Stacks)	Kokkos::SYCL	MDRange	48.6
Crusher	4x MI250X (8 GCDs)*	Kokkos::HIP	MDRange	64.1
	4x MI250X (8 GCDs)*	Kokkos::OpenMPTarget	MDRange	75.2
Polaris	4x A100*	Kokkos::CUDA	Range	39.2
	4x A100*	Kokkos::OpenMPTarget	Team	60.3
	4x A100*	Kokkos::SYCL	Range	47.4
Sunspot	6x PVC (12 Stacks)*	Kokkos::OpenMPTarget	Team	82.0
	6x PVC (12 Stacks)*	Kokkos::SYCL	MDRange	65.9

*This configuration corresponds to use of a full node.

MPI+Kokkos results gathered on the NSF Frontera [20], DOE Lassen [19], DOE Summit [20], NSF Stampede 2 [17, 18], DOE Theta [35], and DOE Titan [35] systems.

Simulations were launched using 1 MPI process per GCD/GPU/Stack. Experiments were performed to identify optimal run configurations using each Kokkos back-end that worked on the target architecture with various Kokkos execution policy types and parameters. Optimal run configurations were used for the results to follow.

The simulation domain is decomposed into a collection of patches, which are distributed across MPI processes. Here, a patch refers to the collection of cells executed by a loop. The radiation modeling problem also uses adaptive mesh refinement to coarsen/refine patches [24]. Domain decomposition and, thus, patch size is user-specified at run-time and remains fixed throughout the simulation.

Problems were sized to provide each MPI process with eight 128^3 fine mesh patches. The problem is weak-scaled when moving from one device to a full node. The problem was configured to cast 50 rays per cell and used a mesh refinement ratio of 4. Results have been averaged over 7 consecutive timesteps. Note, this problem does not weak scale as configured due to communication requirements. However, excellent weak-scaling is possible through the use of aggressive mesh refinement to reduce communication requirements [22].

Table 1 shows single-node timings across MI250X, A100, and PVC for the Burns and Christon benchmark problem on a 2-level structured adaptive mesh refinement grid. For each testbed, results were gathered using problems sized to provide each MPI process with $256 \times 256 \times 256$ cells on the fine mesh and $64 \times 64 \times 64$ cells on the coarse mesh, respectively, for one fine mesh patch

size (128^3 cells per fine mesh patch). These results are encouraging as they demonstrate the portability of Kokkos across DOE exascale testbeds. In addition to being able to run across multiple architectures with Kokkos, we were also able to run across multiple Kokkos back-ends on a given architecture. Note, Kokkos back-ends are supported to varying degrees on individual architectures. For example, use of Kokkos::SYCL is not tested or officially supported on AMD-based GPUs as of this writing.

6 Related Work

Uintah is one of many asynchronous many-task runtime systems and block-structured adaptive mesh refinement frameworks. Examples of similar AMT runtime systems include Charm++ [28], HPX [27], IRIS [29, 34], Legion [2], PaRSEC [5], and StarPU [1]. Other examples of AMT-based approaches include the combination of OpenMP tasking and target offloading [42]. Examples of similar SAMR frameworks include BoxLib [46] (superseded by AMReX [45]), Cactus [14], and Parthenon [15]. An analysis of performance portability for representative AMT runtime systems, including Uintah, can be found in a recent technical report [3]. A review of representative SAMR frameworks, including Uintah, can be found in a recent survey [9].

Kokkos is one of many performance portability layers offering a single interface to multiple underlying programming models (e.g., CUDA, HIP, OpenACC, OpenCL, OpenMP, etc). Examples of similar performance portability layers are OCCA [32], RAJA [21], and SYCL [37] / DPC++ [36]. Examples of Exascale Computing Project codes using Kokkos can be found in a recent survey [10]. Examples of "real-world" applications using Kokkos include ArborX [30], BabelStream [8], K-Athena [16] (superseded by AthenaPK [15]), kEDM [39], LAMMPS [40], and Octo-Tiger [7, 31] with a more extensive collection of applications using Kokkos documented on their GitHub [41]. A review of exascale challenges and performance portable programming models can be found in a recent survey [26].

7 Conclusions and Future Work

This study has helped demonstrate Uintah's preparedness for forthcoming DOE exascale systems. Specifically, this work documents Uintah's first portable use of AMD- and Intel-based GPUs, whereas prior work was limited to NVIDIA-based GPUs. This preparedness has been made possible by the extension of Uintah's intermediate portability layer [18] to additionally support the Kokkos::HIP, Kokkos::OpenMPTarget, and Kokkos::SYCL back-ends. This extension has been made possible by notable updates to Uintah's support for Kokkos.

Kokkos capabilities have been shown across the DOE Crusher, DOE Polaris, and DOE Sunspot testbeds when using Uintah's intermediate portability layer to make portable use of AMD-, NVIDIA-, and Intel-based GPUs, respectively, for the benchmark examined. At the device- and node-level, single-source portability

is shown across AMD-, NVIDIA-, and Intel-based GPUs using various Kokkos back-ends. This portability offers encouragement as we prepare to make portable use of the DOE Aurora and DOE Frontier systems. Next steps include portable simulations across the DOE Aurora and DOE Frontier systems to help better understand how Uintah scales across exascale systems.

8 Acknowledgments

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 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. This work was supported by the Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. 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. The authors would like to thank the ALCF and OLCF for early access to exascale testbeds. The authors would also like to thank the Aurora Early Science Program and Kokkos developer communities for their continued support with special thanks to Daniel Arndt, Rahulkumar Gayatri, Varsha Madananth, and Patrick Steinbrecher.

References

1. 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)
2. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: Expressing locality and independence with logical regions. In: Proceedings of the international conference on high performance computing, networking, storage and analysis. p. 66. IEEE Computer Society Press (2012)
3. Bennett, J., Clay, R., Baker, G., Gamell, M., Hollman, D., Knight, S., Kolla, H., Sjaardema, G., Slattengren, N., Teranishi, K., Wilke, J., Bettencourt, M., Bova, S., Franko, K., Lin, P., Grant, R., Hammond, S., Olivier, S., Kale, L., Jain, N., Mikida, E., Aiken, A., Bauer, M., Lee, W., Slaughter, E., Treichler, S., Berzins, M., Harman, T., Humphrey, A., Schmidt, J., Sunderland, D., McCormick, P., Gutierrez, S., Schulz, M., Bhatele, A., Boehme, D., Bremer, P., Gamblin, T.: ASC ATDM level 2 milestone #5325: Asynchronous many-task runtime system analysis and assessment for next generation platforms. Tech. rep., Sandia National Laboratories (2015)
4. Berzins, M., Beckvermit, J., Harman, T., Bezdjian, A., Humphrey, A., Meng, Q., Schmidt, J., Wight, C.: Extending the uintah framework through the petascale modeling of detonation in arrays of high explosive devices. SIAM Journal on Scientific Computing **38**(5), 101–122 (2016)

5. 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)
6. Burns, S., Christon, M.: Spatial domain-based parallelism in large-scale, participating-media, radiative transport applications. Numerical Heat Transfer **31**(4), 401–421 (1997)
7. Daiß, G., Simberg, M., Reverdell, A., Biddiscombe, J., Pollinger, T., Kaiser, H., Pflüger, D.: Beyond fork-join: Integration of performance portable kokkos kernels with hpx. In: 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 377–386. IEEE (2021)
8. Deakin, T., Price, J., Martineau, M., McIntosh-Smith, S.: Evaluating attainable memory bandwidth of parallel programming models via babelstream. International Journal of Computational Science and Engineering **17**(3), 247–262 (2018)
9. 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)
10. 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)
11. Argonne Leadership Computing Facility: Aurora (2023), https://www.alcf.anl.gov/aurora
12. Oak Ridge Leadership Computing Facility: Frontier (2023), https://www.olcf.ornl.gov/frontier/
13. Falgout, R.D., Li, R., Sjögreen, B., Wang, L., Yang, U.M.: Porting hypre to heterogeneous computer architectures: Strategies and experiences. Parallel Computing **108**, 102840 (2021). https://doi.org/https://doi.org/10.1016/j.parco.2021.102840
14. 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)
15. Grete, P., Dolence, J.C., Miller, J.M., Brown, J., Ryan, B., Gaspar, A., Glines, F., Swaminarayan, S., Lippuner, J., Solomon, C.J., et al.: Parthenon—a performance portable block-structured adaptive mesh refinement framework. The International Journal of High Performance Computing Applications p. 10943420221143775 (2022)
16. Grete, P., Glines, F.W., O'Shea, B.W.: K-athena: a performance portable structured grid finite volume magnetohydrodynamics code. IEEE Transactions on Parallel and Distributed Systems **32**(1), 85–97 (2020)
17. 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)
18. 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). https://doi.org/10.1109/P3HPC49587.2019.00009

19. 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)
20. Holmen, J.K., Sahasrabudhe, D., Berzins, M.: Porting uintah to heterogeneous systems. In: Proceedings of the Platform for Advanced Scientific Computing Conference. PASC '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3539781.3539794
21. Hornung, R.D., Keasler, J.A.: The raja portability layer: overview and status. Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2014)
22. Humphrey, A.: Scalable Asynchronous Many-Task Runtime Solutions to Globally Coupled Problems. Ph.D. thesis, School of Computing, University of Utah (2019)
23. Humphrey, A., Harman, T., Berzins, M., Smith, P.: A scalable algorithm for radiative heat transfer using reverse monte carlo ray tracing. In: Kunkel, J.M., Ludwig, T. (eds.) High Performance Computing, Lecture Notes in Computer Science, vol. 9137, pp. 212–230. Springer International Publishing (2015)
24. Humphrey, A., Meng, Q., Berzins, M., Harman, T.: 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 (2012)
25. Humphrey, A., Sunderland, D., Harman, T., Berzins, M.: 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). pp. 1222–1231 (May 2016)
26. Johnson, A.: Area exam: General-purpose performance portable programming models for productive exascale computing (2020)
27. 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). https://doi.org/10.21105/joss.02352
28. 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)
29. 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). https://doi.org/10.1109/HPEC49654.2021.9622873
30. Lebrun-Grandié, D., Prokopenko, A., Turcksin, B., Slattery, S.R.: Arborx: A performance portable geometric search library. ACM Trans. Math. Softw. **47**(1) (dec 2020). https://doi.org/10.1145/3412558
31. Marcello, D.C., Shiber, S., De Marco, O., Frank, J., Clayton, G.C., Motl, P.M., Diehl, P., Kaiser, H.: Octo-Tiger: A New, 3D Hydrodynamic Code for Stellar Mergers That Uses Hpx Parallelization. Monthly Notices of the Royal Astronomical Society **504**(4), 5345–5382 (04 2021). https://doi.org/10.1093/mnras/stab937
32. Medina, D.S., St-Cyr, A., Warburton, T.: Occa: A unified approach to multithreading languages. arXiv preprint arXiv:1403.0968 (2014)
33. Meng, Q., Humphrey, A., Schmidt, J., Berzins, M.: Investigating applications portability with the uintah DAG-based runtime system on PetaScale supercom-

puters. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 96:1–96:12 (2013)
34. 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). https://doi.org/10.1109/RSDHA56811.2022.00007
35. Peterson, B., Humphrey, A., Holmen, J.K., Harman, T., Berzins, M., Sunderland, D., Edwards, H.C.: Demonstrating GPU code portability and scalability for radiative heat transfer computations. Journal of Computational Science **27**, 303–319 (2018)
36. 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)
37. Rovatsou, M., Howes, L., Keryell, R.: Khronos Group SYCL 2020 Specification (2023), https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf
38. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: June 2023 — TOP 500 (2023), https://top500.org/lists/top500/2023/06/
39. Takahashi, K., Watanakeesuntorn, W., Ichikawa, K., Park, J., Takano, R., Haga, J., Sugihara, G., Pao, G.M.: kedm: A performance-portable implementation of empirical dynamic modeling using kokkos. In: Practice and Experience in Advanced Research Computing. PEARC '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3437359.3465571
40. Thompson, A.P., Aktulga, H.M., Berger, R., Bolintineanu, D.S., Brown, W.M., Crozier, P.S., in 't Veld, P.J., Kohlmeyer, A., Moore, S.G., Nguyen, T.D., Shan, R., Stevens, M.J., Tranchida, J., Trott, C., Plimpton, S.J.: Lammps - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Computer Physics Communications **271**, 108171 (2022). https://doi.org/https://doi.org/10.1016/j.cpc.2021.108171
41. Trott, C.: Apps Using Kokkos (2018), https://github.com/kokkos/kokkos/issues/1950
42. Valero-Lara, P., Kim, J., Hernandez, O., Vetter, J.: Openmp target task: Tasking and target offloading on heterogeneous systems. In: Chaves, R., B. Heras, D., Ilic, A., Unat, D., Badia, R.M., Bracciali, A., Diehl, P., Dubey, A., Sangyoon, O., L. Scott, S., Ricci, L. (eds.) Euro-Par 2021: Parallel Processing Workshops. pp. 445–455. Springer International Publishing, Cham (2022)
43. Valero-Lara, P., Lee, S., Gonzalez-Tallada, M., Denny, J., Vetter, J.S.: Kokkacc: Enhancing kokkos with openacc. In: 2022 Workshop on Accelerator Programming Using Directives (WACCPD). pp. 32–42 (2022). https://doi.org/10.1109/WACCPD56842.2022.00009
44. Yang, Z., Sahasrabudhe, D., Humphrey, A., Berzins, M.: A preliminary port and evaluation of the uintah amt runtime on sunway taihulight. In: 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2018). IEEE (May 2018)
45. 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 (May 2019). https://doi.org/10.21105/joss.01370
46. 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)