Early Experiences on the OLCF Frontier System with AthenaPK and Parthenon-Hydro

John K. Holmen
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory
Oak Ridge, USA
holmenjk@ornl.gov

Philipp Grete
Hamburg Observatory
University of Hamburg
Hamburg, Germany
pgrete@hs.uni-hamburg.de

Verónica G. Melesse Vergara
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory
Oak Ridge, USA
vergaravg@ornl.gov

***Abstract*—The Oak Ridge Leadership Computing Facility (OLCF) has been preparing the nation's first exascale system, Frontier, for production and end users. Frontier is based on HPE Cray's new EX architecture and Slingshot interconnect and features 74 cabinets of optimized 3rd Gen AMD EPYC CPUs for HPC and AI and AMD Instinct 250X accelerators. As a part of this preparation, "real-world" user codes have been selected to help assess the functionality, performance, and usability of the system. This paper describes early experiences using the system in collaboration with the Hamburg Observatory for two selected codes, which have since been adopted in the OLCF Test Harness. Experiences discussed include efforts to resolve performance variability and per-cycle slowdowns. Results are shown for a performance portable astrophysical magnetohydronamics code, AthenaPK, and a mini-application stressing the core functionality of a performance portable block-structured adaptive mesh refinement (AMR) framework, Parthenon-Hydro. These results show good scaling characteristics to the full system. At the largest scale, the Parthenon-Hydro miniapp reaches a total of 1.7×10^{13} zone-cycles/s on 9,216 nodes (73,728 logical GPUs) at $\approx 92\%$ weak scaling parallel efficiency (starting from a single node using a second-order, finite-volume method).**

***Index Terms*—Adaptive Mesh Refinement, Performance Portability, High-Performance Computing, Parallel Computing**

I. Introduction

Leadership-class high performance computing (HPC) systems are featuring increasingly more hardware, in particular, GPU accelerators. For example, the petascale U.S. Department of Energy (DOE) Titan supercomputer featured 18,688 AMD Opteron CPUs and 18,688 NVIDIA K20X GPUs across 18,688 nodes and the DOE Summit supercomputer, Titan's successor, features 9,360 IBM POWER9 processors and 28,080 NVIDIA V100 GPUs across 4,680 nodes. Among forthcoming exascale systems, the DOE Frontier system features 9,408 nodes with 9,408 AMD "Optimized 3rd Gen EPYC" CPU processors and 37,632 AMD MI250X GPUs and the DOE Aurora system will feature >20,000 Intel Xeon Max CPUs and >60,000 Intel PVC GPUs across >10,000 nodes.

The increasing size of these systems poses challenges for those preparing such systems for production and end users due to the increasing number of potential points of failure. Among challenges to consider are hardware resiliency and performance variability [1]. These challenges require greater attention to detail when working to ensure the functionality, performance, and usability of such leadership-class systems as a part of acceptance testing.

An important part of acceptance testing is code selection. For effective testing, it is desirable that selected codes are representative of those anticipated to be run on the system. An approach taken by the Oak Ridge Leadership Computing Facility (OLCF) as a part of Frontier acceptance testing to help ensure representativeness has been to collaborate with past users who plan to use the new system. Such collaborations are beneficial to both parties and helpful for selecting more "real-world" codes making diverse use of the system's software stack. For the OLCF, this helps identify issues sooner. For the user, this helps achieve scientific discoveries sooner.

This paper describes early experiences using the pre-production DOE Frontier supercomputer in collaboration with the Hamburg Observatory for two selected codes, which have since been adopted in the OLCF Test Harness [2]. Experiences discussed include efforts to resolve performance variability and per-cycle slowdowns. Results are shown for a performance portable astrophysical magnetohydronamics code, AthenaPK, and a mini-application stressing the core functionality of a performance portable block-structured adaptive mesh refinement (AMR) framework, Parthenon-Hydro. These results show good scaling characteristics to the full Frontier system. At the largest scale, the Parthenon-Hydro miniapp reaches a total of 1.7×10^{13} zone-cycles/s on 9,216 nodes (73,728 logical GPUs) at $\approx 92\%$ weak scaling parallel efficiency (starting from a single node using a second-order, finite-volume method). Note, zone-cycles/s is a measure of raw performance corresponding to the number of mesh cells updated per second.

The remainder of this paper is structured as follows. Section II provides background discussion. Section III describes the applications used. Section IV describes the scaling studies conducted. Section V describes performance variability issues and resolutions. Section VI describes system testing efforts. Section VII concludes this paper.

II. Background

A. Frontier

Frontier is an exascale supercomputer maintained at the OLCF. Frontier is currently Number 1 on November 2022's Top500 list with an High-Performance Linpack (HPL) score of 1.102 EFlop/s [3]. The system is comprised of 9,408 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). Figure 1 shows the node architecture. 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 Frontier 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 Frontier are interconnected via HPE's Slingshot 11 interconnect. Note, a GCD is referred to as a logical GPU for the remainder of this paper.

B. The OLCF Test Harness

The OLCF Test Harness [5] (OTH) is an open-source, python-based framework used to orchestrate acceptance testing for the OLCF's pre-production systems. The OTH was initially introduced in 2007 [6] and was used for acceptance testing of each Jaguar [7] upgrade, which brought the system from 25 teraflops in 2005 to more than 1 petaflop in 2008. This framework was designed to closely mimic the activities performed by real users of HPC systems. These activities include building a scientific application, generating a batch script, submitting the batch script to the job scheduler, and verifying application results after the job completes. The framework is also able to simulate the execution of "real-world" production workloads that fully occupy an HPC system for an extended period of time using a diverse set of applications. More details on the requirements, design, structure, and evolution of the framework can be found in a recent article [5].

The OLCF Test Harness is available on GitHub [2] with contributions welcome and encouraged.

C. Parthenon

Parthenon [8] is a performance portable block-structured adaptive mesh refinement framework whose roots go back to the host-only Athena++ [9] code. Performance portability in Parthenon is achieved through an intermediate abstraction layer that exposes a simplified interface to Kokkos [10] and is tailored to the ease of use by Parthenon (or downstream code) developers. Key features of Parthenon include a flexible, plug-in package system with dependencies (e.g., to logically separate solvers), abstract variables including "sparse" ones that may not need to be allocated on the entire mesh (e.g., to reduce memory consumption in multi-material simulations), particles, and multi-stage integrators with support for task-based parallelism.

From a performance point of view, the most relevant design decisions in Parthenon include a device-first approach (i.e., work data is only allocated in device memory and only transferred between host-device for IO), using asynchronous, one-sided MPI communication directly between device buffers, and transparent packing of data across blocks (to increase work size within kernels and reduce launch latency). The latter is controlled by the `pack_size` runtime parameter that determines how many blocks are logically packed in a larger data container. Based on past experience, the default value of `pack_size=-1`, indicating to create only a single pack containing all blocks belonging to a rank, resulted in the best performance for many application scenarios when running on GPUs.

The combination of those features allows for a flexible (though not necessarily non-trivial) configuration of runtime parameters that affect the overall performance of an identical simulation depending on the machine characteristics and scale. For example, an overdecomposition of the entire mesh in more than one block per rank allows for overlapping computation and computation. However, this changes various performance-critical parameters at the same time such as total number of messages exchanged, individual message sizes, or the "distance" of messages (i.e., rank-local, node-local, and inter-node). In addition, the logical block packing also affects performance as messages associated with a buffer-filling kernel can only be sent once the entire kernel is done. Therefore, there exists a trade-off between the number of buffers filled in a single kernel and the total work inside that kernel. These characteristics are explored in more detail in Sec. V.

Parthenon is a collaborative project developed on GitHub [11] with contributions welcome and encouraged.

In addition to AthenaPK and Parthenon-hydro described in the following two subsections, other downstream codes on top of Parthenon exist such as phoebus [12], RIOT, and KHARMA based on the iharm3d code [13].

D. AthenaPK

AthenaPK (Athena-Parthenon-Kokkos), the successor of K-Athena [14], is a general purpose (astro)physical magneto-hydronamics code which serves as a performance-portable (through Kokkos use), AMR-capable (through Parthenon use) conversion of Athena++. It implements the hydrodynamics solvers from Athena++ and supplemented them with a divergence cleaning magnetohydrodynamics solver. The code is used for simulations of magnetized galaxy clusters with feedback from active galactic nuclei, cloud crushing in galactic outflows, and magnetohydrodynamic turbulence.

AthenaPK is available on GitHub [15] with contributions welcome and encouraged.

E. Parthenon-Hydro

Parthenon-Hydro is a minimal implementation of algorithms solving the Euler equations. It is a $\sim$1400 line C++ mini-application whose main purposes are to illustrate use of

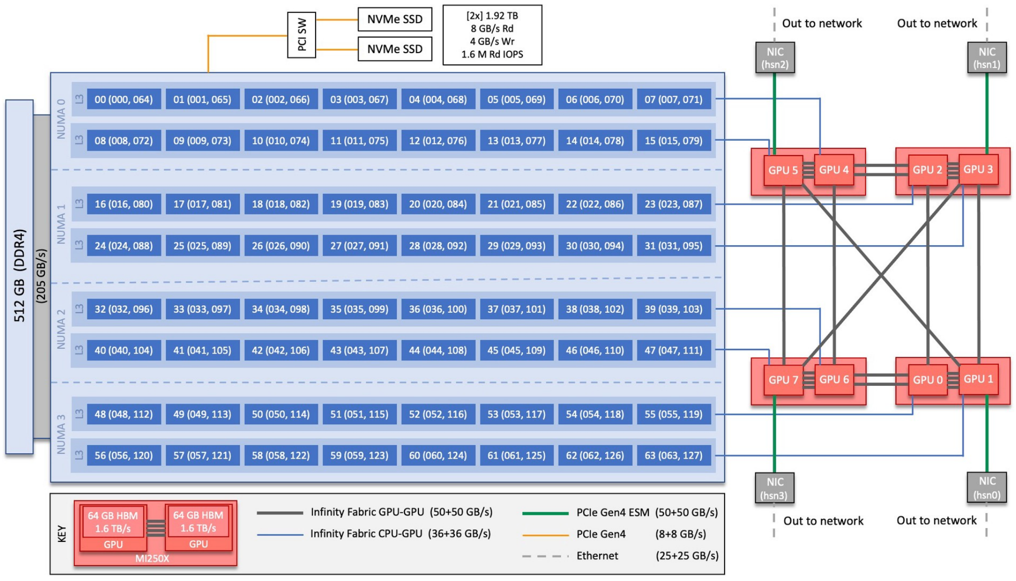
Fig. 1. Frontier node architecture [4]. Note, NICs are directly connected to the GPUs in contrast to earlier systems like Summit.

various Parthenon features combined in practice as well as to serve as an external integration and performance test. The mini-application supports 1D, 2D, and 3D compressible hydrodynamics on uniform and (static and adaptive) multi-level meshes.

Parthenon-Hydro is available on GitHub [16] with contributions welcome and encouraged.

III. Test Setup

All tests were conducted using the `linear_wave` problem generator that is implemented in both AthenaPK and Parthenon-Hydro. It allows a user to initialize various types of linear (magneto)hydrodynamic waves, e.g., an acoustic wave, that travels through the domain at an angle, i.e., not aligned with any mesh direction. This setup has multiple properties that are well-suited for testing. For example, the work per cell (i.e., also per block as a fixed block size is used) is constant across the entire mesh allowing for an easy distribution of work across devices and a predictable wallclock time per cycle. Similarly, the simulation timestep per cycle is effectively constant making deviations (e.g., from erroneous MPI messages) easy to spot at a high level. Likewise, given the known analytic solution to the evolution, error norms can be calculated to verify the correctness of the numerical solution (and its convergence). Finally, the simplicity of the setup allows for a straightforward modification of mesh and block sizes in combination with performance critical parameters like the number of blocks contained in a pack (i.e., handled jointly in a single kernel).

IV. Scaling Studies

Weak-scaling studies were performed on Frontier for both AthenaPK and Parthenon-Hydro. These studies used 8 MPI processes per node to provide 1 MPI process for each logical GPU on node. For both applications, problems were scaled such that each logical GPU used a 512^3 uniform, static mesh with block and mesh sizes adjusted to use a large portion of the available high-bandwidth memory (HBM2E). Here, a block corresponds to a unit of work submitted to the GPU for execution.

Figure 2 shows the weak-scaling results for AthenaPK for different numerical methods (requiring different amounts of work per block and different amounts of data to be communicated) and different block decompositions. In all cases, weak scaling efficiencies are $\gtrsim 90\%$ going from one to 9,216 Frontier nodes (73,728 logical GPUs). However, some subtle differences are already visible in the outliers (points) of the box plots. For both combinations of numerical methods (simple in the left two columns and more complex in the right two columns of Fig. 2) the test case using a single pack of four blocks (of size 128×512^2 cells each) showed significantly more outliers (points in the box plot) at larger (100+) node counts than the test case using two packs each containing one 256×512^2 block, cf., first and third panel versus second and fourth panel of Fig. 2. In the former case, a larger number of MPI messages are put on the network simultaneously as `MPI_Start` is called for all communication buffers of all blocks in a single pack effectively simultaneously. This is already a first indication of the interconnect related performance variability presented and discussed in the following Sec. V. In the best case, AthenaPK achieved a total of 5.4×10^{12} zone-cycles/s (3.4×10^{12}) for a second (third)-order, finite-volume magnetohydrodynamics method with 89% (91%) scaling efficiency across 8,192 Frontier nodes (65,536 logical GPUs) using a block size of $256 \times 512 \times 512$ and a pack size of 1.

Similar weak-scaling results are shown for Parthenon-Hydro in Fig. 3 for varying block sizes. 1.7×10^{13} zone-cycles/s (for

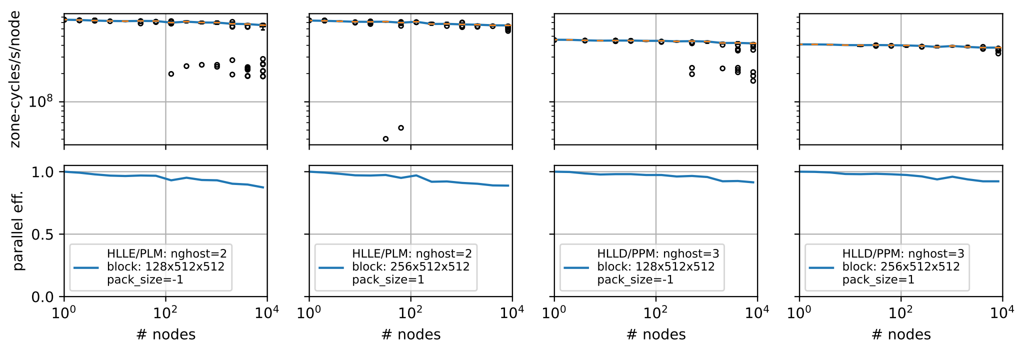

Fig. 2. AthenaPK weak-scaling performance from 1 to 9,216 Frontier nodes. Top panels show a box plot over 50 cycles with the boxes being so compact that they are not visible. Individual points are boxplot outliers. Bottom panels show the corresponding parallel efficiency. The two left columns use a simpler numerical method (HLLE Riemann solver with piecewise linear, PLM, reconstruction) and the right two columns a more complex method (HLLD Riemann solver with piecewise parabloic, PPM, reconstruction require one more ghost cell layer to be communicated). Each logical GPU handled 512^3 cells split into either one pack (`pack_size=-1`) of four 128×512^2 blocks or two packs containing one 256×512^2 block each.

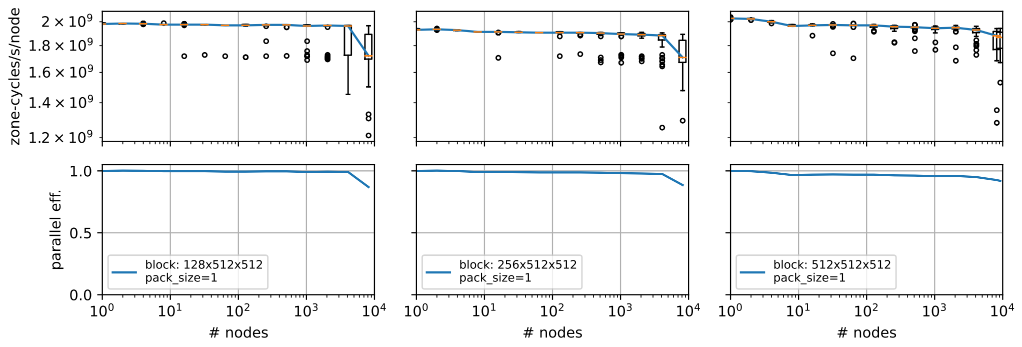

Fig. 3. Parthenon-Hydro weak-scaling performance from 1 to 9,216 Frontier nodes. Same plot type as Fig. 2. Here, the `pack_size` is fixed to 1, i.e., there are as many independent packs as there are blocks on each logical GPU: four blocks of 128×512^2 cells, two blocks of 256×512^2 cells, and one 512^3 block, respectively, from left to right.

a second-order, finite volume hydrodynamics method) were achieved with 92% scaling efficiency across 9,216 Frontier nodes (73,728 logical GPUs) using a block size of 512^3 and a pack size of 1, but in all cases the parallel efficiency remained $\gtrsim 90\%$ up to 9,216 Frontier nodes.

Overall, these results for AthenaPK and Parthenon-Hydro are encouraging as this is among the largest number of logical GPUs that both applications have been run across. Additional weak- and strong-scaling results for these applications and cross-system comparisons can be found in a recent article [8].

V. Performance Variability

During strong-scaling studies with AthenaPK, random per-cycle performance slowdowns were observed. Figure 5 shows the magnitude of slowdowns for a problem using a 1024^3 uniform, static mesh strong-scaled from 8 to 512 Frontier nodes. The likelihood of per-cycle slowdowns increases as the node count increases with individual cycles being $\gtrsim 100\times$ slower than expected at the largest scale. Moreover, different performance characteristics seem to emerge. For example, while for 8, 64, and 128 nodes slower cycles are not clustered, multiple clusters exist in the 256 nodes case spreading over several tens of cycles between 90-100 % of peak performance. At 512 nodes, bands of cycles exist at the 35 % level indicating a significant loss in performance (in addition to the extreme cases of $\lesssim 1$ %).

To better understand these differences, a series of experiments were performed across 1024 Frontier nodes using a $16384 \times 8192 \times 8192$ uniform, static mesh. These experiments explored different block sizes, numbers of blocks per logical GPU, numbers of blocks per pack, and numbers of packs per logical GPU. The block size impacts the total number of MPI messages being sent (and their size). For smaller block sizes more messages of smaller sizes (on average) need to be exchanged whereas for larger blocks fewer, larger messages are being sent. In the extreme case of using a single block per device only, one message is being sent/received to/from each of the 26 neighboring blocks. The number of blocks per pack implicitly impacts the number of messages being put on the network simultaneously. More specifically, a single pack translates to filling all communication buffers with all blocks in a single kernel and then calling *MPI_Start* back-to-back for all communication buffers. Having more than one pack allows for data of the first pack to be already sent while the other buffers are being filled.

Figure 5 shows the per-cycle performance for these experi-

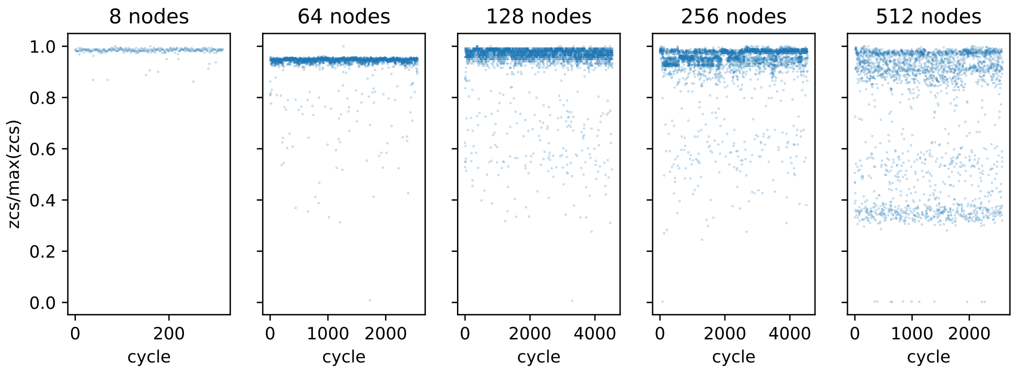

Fig. 4. AthenaPK per-cycle performance (in normalized zcs, zone-cycles per seconds) in a strong-scaling setup (1024^3 mesh) from 8 to 512 Frontier nodes.

ments. The first three top panels compare the per-cycle performance for a block size of 128^3 (i.e., 64 blocks per GCD) split into 1, 64, and 2 packs, respectively. When using few packs (1 or 2) the performance of individual cycles peaks at 1.9×10^{12} zone-cycles/s but their median value is only 0.5×10^{12} zcs. In contrast, using 64 packs, i.e., one pack per block, the performance is consistent at 1.4×10^{12} zcs. Given the identical number of messages (and their sizes) in all three cases, these per-cycle performance variations are a strong indicator that the number of messages being put on the network simultaneously is a significant performance concern at these scales on Frontier. This also aligns with the results for block sizes of 128×512^2 split into a single pack (top right panel of Fig. 5) or four packs (bottom left panel). Again in the former case where more messages are being sent/received simultaneously, per-cycle variations between 0.5–2.1×10^{12} zcs are observed whereas in the latter case the performance is effectively constant. This constancy in performance also applies to the test cases with blocks of 256×512^2 and 512^3 cells, for which fewer (larger on average) messages are being exchanged by construction.

In general, using 4 or more blocks per pack resulted in greater variation and using 1 block per pack resulted in the most consistent performance. The run configuration that best balanced peak performance and performance variability was a block size of 512^3, a block count of 1 per logical GPU, a block count of 1 per pack, and a pack count of 1 per logical GPU. In other words, an overdecomposition of the mesh in more blocks than devices is currently not recommended on Frontier. Similar past tests on different machines with similar architecture, e.g., the JUWELS Booster with four Nvidia A100 GPUs with direct connection to four NICs, did not show any significant variations.

In addition to AthenaPK parameter tuning, experiments were also performed with Parthenon-Hydro to explore the impact of the MPICH_RANK_REORDER_METHOD environment variable on performance. This environment variable is used to specify various types of MPI rank placement and has 4 options: (1) round robin rank ordering, which is specified by setting the environment variable to 0, (2) SMP style rank ordering, which is specified by setting the environment variable to 1, (3) folded rank ordering, which is specified by setting the environment variable to 2, and (4) a custom user-defined rank ordering, which is specified by setting the environment variable to 3.

For (1), the first rank is on the first node, the second rank is on the next node, and so on, until all nodes have one rank, then the next rank after that is on the next available core on the first node. For (2), ranks are placed consecutively until the node is filled up, then on to the next node. For (3), ranks are populated down the node list like (1) but then the next ranks are populated back up the node list in reverse order.

Using Parthenon-Hydro, experiments were performed across 128 Frontier nodes to examine the impact of each rank order method. The fold method was found to have the most positive impact on runs, dropping an MPI_Allreduce bottleneck from 18.4% to 4.1% and dropping execution time from 12.1 seconds to 9.5 seconds while also smoothing per-cycle consistency. More details on MPICH_RANK_REORDER_METHOD can be found on the Blue Waters User Portal [17].

VI. System Testing

A. Code Selection

Code selection is an important part of system testing. The OLCF has found selecting user codes helpful for their ability to stress interesting third-party library combinations while making “real-world” use of a system. AthenaPK and Parthenon-Hydro were strategically chosen to be added to the OTH for a variety of reasons. A primary reason has been that these codes have successfully demonstrated large-scale performance portability across various leadership-class HPC systems [8]. The codes achieve this by making effective use of the Kokkos performance portability layer at scale. This is of particular interest to the OLCF due to the widespread use of Kokkos among Exascale Computing Project codes [18]. Other reasons include that the code is well documented, easy to build, run, and scale, and the developer community is friendly and supportive.

B. Test Design

Frontier system tests fall in two basic categories: (1) large-scale tests and (2) “every node” tests. Large-scale tests tend to

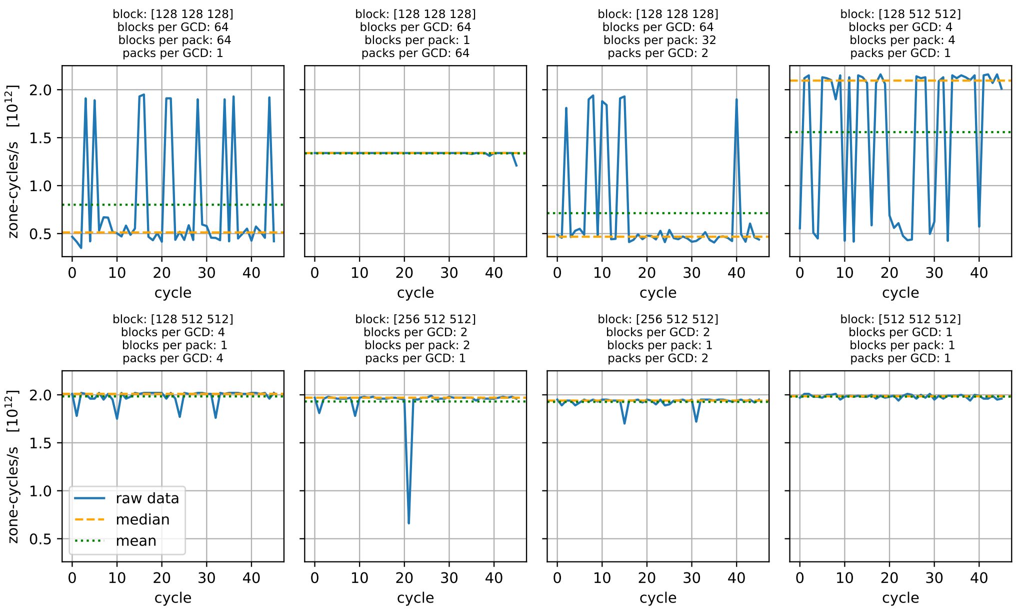

Fig. 5. AthenaPK per-cycle performance across 1024 Frontier nodes. The mesh size was fixed to 16384 × 8192 × 8192 and block size, block count, and pack count were varied.

target 1/8, 1/4, 1/2, and full system use with scales in between stressed as needed. “Every node” tests are based on single-node tests and used to launch a collection of individual single-node jobs across the system.

“Every node” tests are a new type of test added to the OTH for Frontier acceptance testing. An example of an “every node” test is launching 9,408 single-node AthenaPK tests across Frontier in a single job. Running in this manner has been helpful for isolating “bad” nodes and node failures. These tests have been valuable for their ability to quickly pinpoint problematic nodes in the system.

C. Early Challenges

When initially designing “every node” tests, issues were encountered with individual AthenaPK cycle-steps performing orders of magnitudes slower than others. Figure 6 shows an example of these performance differences. Upon further investigation, it was discovered that the manner in which the code checks for a manual output trigger was the source of the slowdowns. Specifically, during runtime rank 0 checks once per cycle for the existence of a file called `output_now` in the run directory. While this is no issue for large scale simulation as only a single rank performs the check, it caused problems for the “every node” tests that were originally all launched within the same run directory located on a shared file system. In the most extreme case, the `stat()` system call would be called several 10,000× per second on the same file (more than one call per second from each of the 9,408 ranks) resulting in significant delays. Shifting to use of individual run directories for per-node tests resolved the slowdowns. Figure 7 shows an example of the resulting performance.

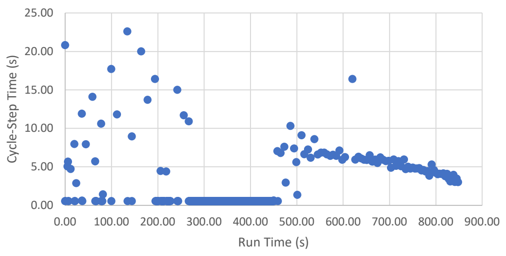

Fig. 6. Sample AthenaPK ”every node” test performance (in wallclock seconds per cycle) of a random node when using a single directory on the shared file system as the run directory for all nodes.

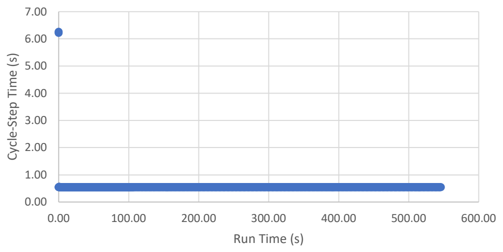

Fig. 7. Sample AthenaPK ”every node” test performance (in wallclock seconds per cycle) of a random node when using individual run directories on the shared file system for all nodes.

VII. Conclusions

The increasing size of leadership-class HPC systems poses challenges for those preparing such systems for production and end users. When working to ensure the functionality, performance, and usability of such leadership-class systems, code selection is an important part of acceptance testing. An approach taken by the OLCF as a part of Frontier acceptance testing to help ensure the representativeness of selected codes has been to collaborate with past users who plan to use the new system.

This paper described early experiences using the pre-production ORNL's exascale Frontier system in collaboration with the Hamburg Observatory for two selected codes, which have since been adopted in the OLCF Test Harness. Experiences discussed included efforts to resolve performance variability and per-cycle slowdowns. Results were shown for a performance portable astrophysical magnetohydronamics code, AthenaPK, and a mini-application stressing the core functionality of a performance portable block-structured adaptive mesh refinement (AMR) framework, Parthenon-Hydro. These results showed good scaling characteristics to the full Frontier system. At the largest scale, the Parthenon-Hydro mini-application reached a total of 1.7×10^{13} zone-cycles/s on 9,216 nodes (73,728 logical GPUs) at $\approx 92\%$ weak scaling parallel efficiency (starting from a single node using a second-order, finite-volume method).

For the OLCF, next steps include continuing to extend Frontier's test coverage by identifying similar opportunities to collaborate with users. A specific goal of this effort is to identify codes using various performance portability libraries to extend coverage of performance portability layers such as Kokkos [10], OCCA [19], RAJA [20], and SYCL [21] / DPC++ [22] on Frontier. For the Hamburg Observatory, next steps include science runs on Frontier targeting magnetized plasma jets from active galactic nuclei. Moreover, a more detailed evaluation of the ordering of filling buffers and sending messages (e.g., by effective distance, i.e., rank-local, node-local, or inter-node) and decoupling the global block packing from a communication related packs (to optimize raw compute and communication separately) is planned.

Acknowledgment

The authors would like to thank the Parthenon and Kokkos communities for being open, approachable and supportive. The authors would also like the thank the OLCF for early access to Frontier. 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 project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101030214.

References

[1] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman, "Not all gpus are created equal: characterizing variability in large-scale, accelerator-rich systems," *arXiv preprint arXiv:2208.11035*, 2022.

[2] OLCF. (2023) Olcf test harness. [Online]. Available: https://github.com/olcf/olcf-test-harness

[3] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. (2022) November 2022 — TOP 500. Https://top500.org/lists/top500/2022/11/.

[4] OLCF. (2023) Frontier user guide - olcf user documentation. [Online]. Available: https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

[5] V. G. V. Larrea, M. J. Brim, A. Tharrington, R. Budiardja, and W. Joubert, "Towards acceptance testing at the exascale frontier," in *Proceedings of the Cray User Group 2020 conference*, 2020.

[6] A. Tharrington, "An overview of nccs xt3/4 acceptance testing," in *Proceedings of the Cray User Group 2007 conference*, 2007.

[7] A. S. Bland, W. Joubert, R. A. Kendall, D. B. Kothe, J. H. Rogers, and G. M. Shipman, "Jaguar: The world's most powerful computer system– an update," *Cray Users Group*, 2010.

[8] P. Grete, J. C. Dolence, J. M. Miller, J. Brown, B. Ryan, A. Gaspar, F. Glines, S. Swaminarayan, J. Lippuner, C. J. Solomon *et al.*, "Parthenon—a performance portable block-structured adaptive mesh refinement framework," *The International Journal of High Performance Computing Applications*, p. 10943420221143775, 2022.

[9] J. M. Stone, K. Tomida, C. J. White, and K. G. Felker, "The athena++ adaptive mesh refinement framework: Design and magnetohydrodynamic solvers," *The Astrophysical Journal Supplement Series*, vol. 249, no. 1, p. 4, 2020.

[10] C. Trott, D. Lebrun-Grandie, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. A. Ibanez *et al.*, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[11] Parthenon Collaboration. (2023) Parthenon. [Online]. Available: https://github.com/parthenon-hpc-lab/parthenon

[12] Miller, Jonah and Ryan, Ben and Dolence, Josh and Roberts, Luke. (2023) phoebus. [Online]. Available: https://github.com/lanl/phoebus

[13] B. S. Prather, G. N. Wong, V. Dhruv, B. R. Ryan, J. C. Dolence, S. M. Ressler, and C. F. Gammie, "iharm3d: Vectorized general relativistic magnetohydrodynamics," *Journal of Open Source Software*, vol. 6, no. 66, p. 3336, 2021. [Online]. Available: https://doi.org/10.21105/joss.03336

[14] P. Grete, F. W. Glines, and B. W. O'Shea, "K-athena: a performance portable structured grid finite volume magnetohydrodynamics code," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 85–97, 2020.

[15] Parthenon Collaboration. (2023) Athenapk. [Online]. Available: https://github.com/parthenon-hpc-lab/athenapk

[16] ——. (2023) Parthenon-hydro. [Online]. Available: https://github.com/parthenon-hpc-lab/parthenon-hydro

[17] NCSA. (2022) Blue waters user portal - topology considerations. [Online]. Available: https://bluewaters.ncsa.illinois.edu/topology-considerations

[18] T. M. Evans, A. Siegel, E. W. Draeger, J. Deslippe, M. M. Francois, T. C. Germann, W. E. Hart, and D. F. Martin, "A survey of software implementations used by application codes in the exascale computing project," *The International Journal of High Performance Computing Applications*, vol. 36, no. 1, pp. 5–12, 2022.

[19] D. S. Medina, A. St-Cyr, and T. Warburton, "Occa: A unified approach to multi-threading languages," *arXiv preprint arXiv:1403.0968*, 2014.

[20] R. D. Hornung and J. A. Keasler, "The raja portability layer: overview and status," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2014.

[21] M. Rovatsou, L. Howes, and R. Keryell, "Khronos Group SYCL 2020 Specification," 2019, https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf.

[22] J. Reinders, B. Ashbaugh, J. Brodman, M. Kinsner, J. Pennycook, and X. Tian, *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL.* Springer Nature, 2021.