Approaching the Final Frontier: Lessons Learned from the Deployment of HPE/Cray EX Spock and Crusher Supercomputers

Verónica G. Melesse Vergara, Reuben D. Budiardja, Matt Davis, Matt Ezell,
Jesse Hanley, Christopher Zimmer, Michael J. Brim, Wael Elwasif, Dan Dietz
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: vergaravg@ornl.gov

***Abstract*—In 2021, the Oak Ridge Leadership Computing Facility (OLCF) deployed Spock and Crusher, the first two user-facing HPE/Cray EX systems as precursors to Frontier. Both systems were transitioned to operations in 2021 with the goal of providing users with platforms to begin porting scientific applications in preparation for Frontier's arrival. While Spock's architecture is one-generation removed from Frontier's hardware, it exposed users earlier to the new HPE/Cray Programming Environment designed for AMD GPUs. Spock is a 36-node HPE/Cray EX supercomputer with one AMD EPYC 7662 processor and 4 AMD MI100 GPUs per node. Crusher, on the other hand, is a 192-node HPE/Cray EX supercomputer with one AMD EPYC 7A53 processor and 4 AMD MI250X GPUs per node with the same hardware as Frontier. In this paper, we present an overview of the challenges and lessons learned encountered during the deployment and the transition to operations of both systems. These include issues identified with the programming environment, layout and process binding via SLURM, and providing access to the center-wide file systems. We also discuss settings added locally to improve the user experience, current workarounds in-place, and the processes developed to capture the status of evolving issues in our user-facing documentation.**

***Index Terms*—high performance computing, system deployment, system testing, programming environment**

I. Introduction

In 2021, the Oak Ridge Leadership Computing Facility (OLCF) deployed the first two HPE/Cray EX user-facing systems precursors to Frontier: Spock [1] and Crusher [2]. Both systems were installed, deployed, and transitioned to operations in 2021 with the goal of providing Exascale Computing Project (ECP) and Center of Accelerated Application Readiness (CAAR) users with a platform to begin port scientific applications in preparation for Frontier's arrival.

While Spock's architecture is one-generation removed from Frontier's hardware, it has the advantage of exposing users earlier to the new HPE/Cray Programming Environment (CPE) specifically designed for AMD GPUs. Spock is a 36-node HPE/Cray EX supercomputer with one 64-core AMD EPYC 7662 processor, 256 GB of memory, and 4 AMD Instinct MI100 GPUs per node. Spock was the first user-facing AMD GPU-based HPE/Cray EX supercomputer deployed at the OLCF.

Prior to opening the Spock early access system to users, the OLCF worked closely with HPE and AMD to install and configure the hardware, system software, and user environment. On the system-software side, transitioning applications to use Slurm [3] for launching jobs revealed several bugs centered around placement and mapping of processes to cores and GPUs. In addition, due to the initial lack of support for IBM Spectrum Scale (GPFS) on Cray OS, we had to find alternatives that would allow us to provide the OLCF's center-wide GPFS file system on Spock.

On the user-environment side, initial challenges included compatibility issues identified between AMD's Radeon Open Compute Platform (ROCm) and several HPE/Cray products: Cray Compiling Environment (CCE), Cray Performance Analysis Tools (CrayPat), and Cray MPICH. We elaborate some of these challenges in this paper. In one case, for instance, we found that users with codes that rely on OpenMP offloading were unable to compile their application with CCE when moving from ROCm 4.2.0 to 4.3.0, which was found to be caused by a mismatch in the LLVM versions provided by each modulefile.

In addition, we identified several bugs in the HPE/Cray programming environment for AMD (PrgEnv-amd) that prevented users from successfully compiling applications. Being a fairly new addition, some of the Lua-based modulefiles required local patches in order to provide users with a fully functional user environment. For example, initially in CPE 21.07, loading the `PrgEnv-amd` modulefile disabled `cray-mpich` in the environment due to the fact that the corresponding modulefile was not found. This prevented users from compiling MPI

applications using `PrgEnv-amd` and a workaround was necessary until an official fix was available in CPE.

The Crusher test and development system (TDS) has the same processor and GPU as Frontier. It is the first system that will be available to users to begin porting activities for Frontier. Crusher is a 192-node HPE/Cray EX supercomputer with one 64-core AMD EPYC 7A53 "Optimized 3rd Gen EPYC" processor, 512 GB of DDR4 memory, and 4 AMD MI250X, each with 2 Graphics Compute Dies (GCDs). On Crusher, we encountered additional compatibility issues due to the rapidly evolving nature of ROCm and worked closely with HPE to address these issues. The more complex node architecture also uncovered edge cases that were not adequately supported in Slurm, as well as improvements needed in CPE to provide a more streamlined user experience.

In this paper, we present an overview of the challenges encountered, bugs and fixes identified, as well as lessons learned during the deployment and transition to operations of the Spock and Crusher systems. We also discuss settings added locally to improve the user experience, the current workarounds that remain in-place, and the processes developed to maintain the status of evolving issues automatically updated in our user-facing documentation. These lessons learned can be useful for the HPE/Cray user community currently deploying similar systems.

II. System Architectures

As part of the preparation for Frontier, the OLCF deployed the Spock early access system (EAS) and the Crusher test and development system (TDS). In this section, we detail information about the system architecture of both platforms as well as the center-wide Spectrum Scale parallel file system, Alpine.

A. Alpine

The Alpine file system cluster is an existing platform within the OLCF. As a center-wide resource, the cluster provides a POSIX-based Spectrum Scale (formerly known as GPFS) namespace for computational and visualization systems [4]. Alpine has a maximum capacity of 250 PB and performance of up to 2.5 TB/s and 2.2 TB/s for sequential and random I/O, respectively. The cluster is attached to a Scalable I/O Network (SION3) based on EDR InfiniBand. A schematic describing the connectivity path between Alpine and Spock and Crusher is shown in Figure 1.

B. Spock

The Spock Early Access System (EAS) is one generation removed from the hardware that will be available on Frontier. Spock is comprised of 36 HPE Apollo 6500 nodes each with one 64-core AMD EPYC 7662 "Rome" CPU and four AMD MI100 GPUs. Each compute node has 256 GB of DDR4 memory and 128 GB of high-bandwidth memory (HBM2). The CPU is connected to the GPUs via PCIe Gen4 which delivers a bandwidth of 32+32 GB/s. All GPUs on a compute node are interconnected via AMD's Infinity Fabric (xGMI)

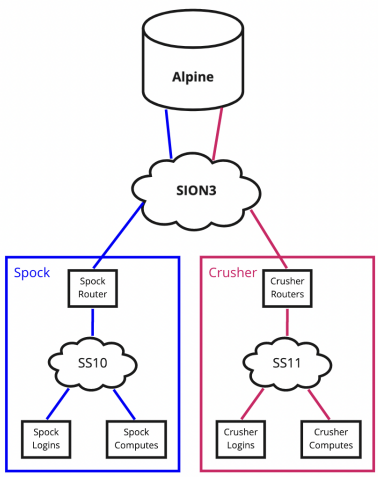

Fig. 1: Schematic describing Alpine connectivity to/from Spock and Crusher.

which delivers 46+46 GB/s between devices. Two login nodes provide front-end services for users to compile code and submit jobs. Compute nodes on Spock are interconnected via HPE's Slingshot 10 interconnect.

Spock was deployed and is managed with the HPE Performance Cluster Manager (HPCM) version 1.5. HPCM provides tools for node power and firmware management, image building, and monitoring. The compute nodes boot Cray OS (COS) version 2.1, which is based on SUSE Linux Enterprise Server (SLES) 15 SP2. The Alpine file system is natively mounted on each Spock compute node. Two gateway nodes outfitted with a Slingshot 10 interface and a Mellanox CX-6 InfiniBand interface provide Ethernet IP forwarding services to allow the compute nodes to communicate with the file system servers [5]. The routing table on each Spock compute node contains a static multipath route to reach Alpine's network range. Host level firewall rules on each Spock compute node permit Spectrum Scale traffic from the Alpine cluster. A corresponding multipath route entry and a set of host firewall rules permit each server in the Alpine cluster to reach the Slingshot 10 network on Spock. The compute, gateway, and utility nodes in Spock form a distinct Spectrum Scale cluster. This cluster is defined as a remote cluster to Alpine and is permitted to mount the Alpine namespace [6]. HPE Slingshot 10 (SS10) [7] is a 100Gbps network architecture composed of HPE Rosetta switches and Mellanox interfaces. The interfaces are Mellanox CX-5 100Gbps Ethernet adapters.

SS10 supports features seen in comparable Infiniband networks, such as adaptive routing and quality of service. Using Rosetta switches, SS10 also provides new network features, HPC optimized Ethernet, and state-of-the-art congestion control. Rosetta switches are 64 ports and support up to 200Gbps per interface. SS10 supports the dragonfly network topology to enable large and scalable networks. The Spock SS10 network is 3 groups with 12 computes per group providing 1.2Tbps of intragroup connectivity and 800Gbps of intergroup

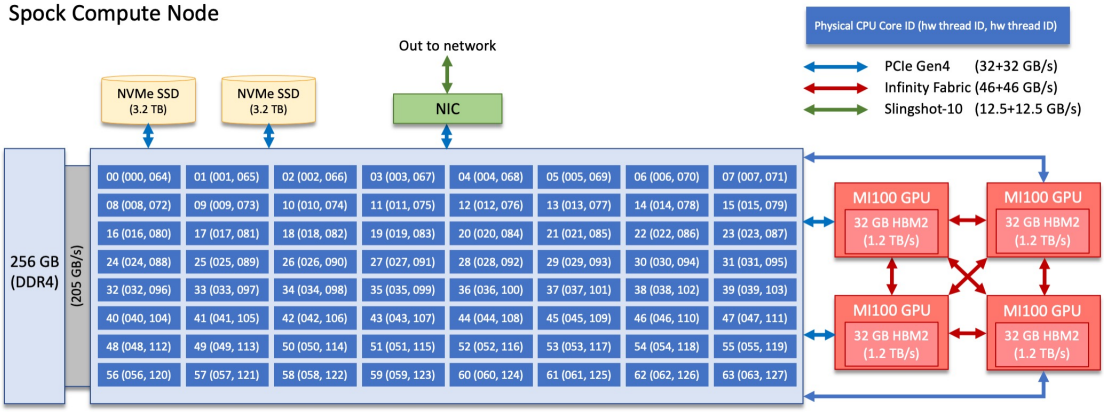

Fig. 2: Spock compute node diagram.

connectivity.

C. Crusher

The Crusher Test and Development System (TDS) 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. Crusher uses HPCM version 1.6, which was the first version to officially support HPE Cray EX Mountain/Olympus hardware. ORNL augmented HPCM with its own software, Phoenix [8], to pre-discover the compute nodes and automate configuration file generation. Crusher uses 3 HPCM SU Leader nodes to distribute the load of node boot and management. The OLCF center-wide NFS home areas are served to the compute nodes via Cray's Data Virtualization Services (DVS). Although DVS is not formally supported with HPCM 1.6 and COS 2.2, the software is still included in the COS distribution. Local scripts were developed to generate a DVS node map and initialize the servers and clients. Similar to Spock, Crusher natively mounts the Alpine Spectrum Scale file system using three dedicated gateway nodes. Each node connects to both the internal high speed network and SION. The compute, login, and utility nodes of Crusher form a distinct Spectrum Scale cluster. Because of the center-wide nature of Alpine, remote clusters must have distinct non-overlapping subnet ranges. Careful planning ensures that the network address scheme implemented on Crusher and Spock do not interfere with each other, with existing clusters, or with future systems that mount Alpine. Slingshot 11 (SS11) upgrades SS10 by replacing the Mellanox CX-5 Ethernet adapters with HPE Cassini 200Gbps adapters. These adapters expand the features of the Slingshot network by incorporating hardware tag-matching, triggered messages, hardware-accelerated collectives, and atomic operations. With SS11, hardware-based congestion control can throttle at the NIC. The 192 nodes of Crusher represent two dragonfly groups where only one is fully populated. A Crusher node has 4 NICs per node totaling 768 200Gbps ports from the compute portion of the network.

III. Programming Environment

The Spock and Crusher systems leverage the HPE/Cray Programming Environment (CPE) which includes a software stack that utilizes the Cray Compiling Environment (CCE), AMD compiler (AMD), and GNU Compiler Collection (GCC) compilers. AMD's Radeon Open Compute (ROCm) platform is provided separately and includes both the ROCm device drivers and ROCm libraries and tools.

In this section, we describe the different components included as part of the user environment available on Spock and Crusher.

A. Compilers on Spock and Crusher

The Cray Compiling Environment (CCE) provides C, C++, and Fortran compilers. The CCE C and C++ compilers are based on the Clang compiler [9] and include support for the Unified Parallel C (UPC), Heterogeneous-Compute Interface for Portability (HIP) [10], and OpenMP parallel programming model [11]. The support for HIP is only available for AMD GPUs. The OpenMP support includes the full implementation of 4.5 specification and partial for 5.0 and 5.1 specifications. It also includes support for OpenMP target offload on Cray systems. The CCE Fortran compiler is a HPE/Cray proprietary Fortran compiler. It also fully implements OpenMP 4.5 specification with partial support for 5.0 and 5.1 specifications.

The AMD programming environment is derived from the Radeon Open Compute (ROCm) platform [12]. The ROCm compiler is built upon the LLVM open source compiler

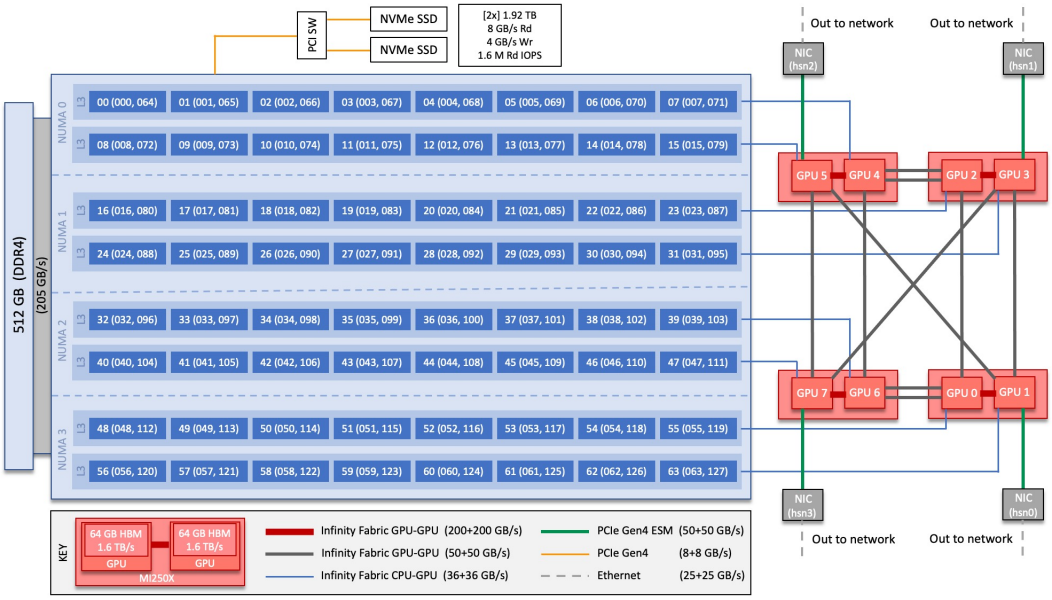

Fig. 3: Crusher compute node diagram.

framework with support for compiling GPU kernels written using the C++ HIP specification, as well as the OpenMP standard for parallel execution on the CPU and offloading to the MI100 and MI250X GPUs. The ROCm compiler supports C, C++, and Fortran using the the `amdclang`, `amdclang++` and `amdflang` drivers, respectively. It should be noted that Fortran support in ROCm is built upon the *classic Flang* code base [13], which is an open-sourced version of `pgfortran`, a commercial Fortran compiler from PGI/NVIDIA, and is not currently using the flang code base that is part of the LLVM monorepo. The latter code base is currently under active development.

The CCE, AMD, and GCC compilers are accessible via loading the environment module `PrgEnv-cray`, `PrgEnv-amd`, and `PrgEnv-gnu`, respectively. The compiler drivers `cc`, `CC`, and `ftn` for C, C++, and Fortran that invoke the underlying compilers are made available to provide a convenient mechanism to use common flags such as offloading to the GPU and adding the correct path to include files. During linking, these compiler drivers also automatically add the necessary flags to link to commonly used libraries such as the `cray-mpich` library for MPI and the Cray scientific library `libsci`.

In addition to the ROCm platform, AMD makes available the AOMP research compiler [14]. AOMP combines the latest updates from upstream LLVM development with AMD open development that have yet to be merged into upstream LLVM. Furthermore, the ROCm environment includes several libraries tuned for execution on AMD Instinct GPUs. Those libraries include `rocBLAS`, `rocfft`, `MIOpen` and other scientific and machine learning libraries. The libraries deployed as part of the ROCm environment are a subset of the open source libraries [15] that constitute the ROCm Software Platform. AMD also makes available the AFAR compiler, which uses the same LLVM code base as AOMP, but is packaged against a specific version of ROCm, along with the associated scientific libraries. While having access to research and *pre-release* versions of the compilers was generally beneficial, it was sometimes difficult to navigate feature availability and deployment plans, especially as new features are introduced and bugs fixed in officially released AMD tool chains.

Due to the inadequate support for Fortran and OpenMP offloading in the `amdflang` compiler, two more *hybrid* programming environments were added based on early application developer feedback. The `PrgEnv-gnu-amd` environment uses the AMD C and C+ compilers with the GNU `gfortran` compilers while the `PrgEnv-cray-amd` environment uses the AMD C and C++ compilers with the Cray CCE Fortran compiler. These mixed environments are supported using the standard CPE compiler drivers providing a simple interface to these hybrid environments.

B. Performance and Correctness Tools

The AMD Instinct accelerator series and its underlying CDNA architecture are fairly recent entries into the realm of general-purpose GPU-based computing. Many existing AMD tool products are focused on graphics-oriented workloads for the RDNA architecture GPUs, where the performance goals, methods, and associated GPU hardware components for optimizing performance are very different from general-purpose scientific computing. As a result, the necessary tooling support for performance measurement and analysis and functional correctness for CDNA GPUs remains in a state of active design and development.

For performance measurement and analysis, AMD provides the ROC-profiler and ROC-tracer API libraries that are loosely based on functionality provided by NVIDIA's CUDA Profiling

Tools Interface (CUPTI). These libraries provide the core functionality for profiling and tracing of GPU code, including collection of GPU hardware performance counters, and are crucial for supporting third-party performance tools for AMD GPUs. AMD also provides the `rocprof` command-line utility to orchestrate use of the profiling and tracing libraries and post-process performance results when running a local application. Profile data from `rocprof` is generated in comma-separated value (CSV) files, while trace data is produced in the JSON trace format used by the trace viewer embedded in Google's Chrome web browser. For functional correctness, AMD provides the Debugger API library for control and inspection of GPU code, and the `rocgdb` debugger that integrates the library's capabilities into the widely-used `gdb` debugger. Because AMD's tools are designed for single-process, single-host use, they are not currently suitable for performance analysis and debugging of long-running parallel applications spanning many hosts or using multiple processes per host, as is common for scientific HPC applications.

HPE's performance and correctness tools for AMD GPUs include the Cray Performance Measurement and Analysis Tools (CPMAT) and the Cray Debugger Support Tools (CDST). Both toolsets are dependent upon AMD's libraries and tools for much of their AMD GPU support. For example, the `perftools` and `perftools-lite` suite of performance analysis tools use the ROC-tracer API, and the `gdb4hpc` parallel debugger leverages `rocgdb`. As these toolsets are designed for HPC use, they automatically aggregate and summarize information collected from many parallel processes, and thus are the preferred method for analyzing scientific applications.

IV. Challenges of an Evolving Ecosystem

Given that the AMD GPU ecosystem is rapidly evolving, unsurprisingly, the user environment that is available on both Spock and Crusher has also had to quickly change. Initially, due to the specific packaging of ROCm, only a single version could be available on the system at a given time. This negatively impacted users as they were unable to compare changes between versions. Working with our vendor partners, more recent versions of ROCm began supporting parallel installations and removed hard-coded dependencies on specific paths (e.g., `/opt/rocm/`). As the ecosystem matured and became integrated with the HPE/Cray Programming Environment (CPE), we identified several potential gaps that needed to be addressed in order to provide users with a robust and flexible environment. In this section, we discuss specific issues that were identified, present workarounds currently in place, and describe suggestions on changes needed in future releases.

A. Programming Environment

The HPE/Cray Programming Environment (CPE) has multiple packages dependent on ROCm libraries. These packages include: *LibSci_acc*, which provides accelerated versions of scientific libraries for Cray systems with AMD MI100 or MI250X targets; CCE compilers for the OpenMP target offloading support; and Cray-MPICH with its GPU Transport Layer (GTL) to enable the use of GPU-aware MPI.

Due to the rapid development of ROCm, frequent linking failures due to breaking changes to the API have been encountered when the libraries from the above packages are needed. With ROCm-4.2, AMD began using versioned symbols to indicate when an interface had either changed or been created. The older versions of these symbols are not exported in subsequent releases, which leads to errors when the symbol versions cannot be resolved by the dynamic linker. In this case the versioned symbols are doing the work that a change in the library `.so` name would normally do, but the resulting linking errors are encountered at various times in the build and run process rather than immediately.

Among the CPE packages, GTL's library for AMD GPUs has been the most frequently involved in link failures due to its reliance on the less stable ROCm interfaces and the ubiquity of its usage among the MPI applications running on Spock and Crusher. To contrast, the *LibSci_ACC* package utilizes symbols that have been stable since ROCm-4.2 and CCE uses symbols which have been stable in all ROCm-4.x releases.

With the release of ROCm-5.0, the library `libamdhip64.so` changed its `DT_SONAME` to indicate the change in major version. This has prohibited the usage of ROCm-5.x with ROCm-dependent CPE packages prior to version CPE-22.04, as they mark the previous releases' `DT_SONAME` as needed. In April 2022, HPE's CPE release included partial support for ROCm 5.0. Most notably this included a build for GTL that linked to the newer ROCm-5.x libraries and a `PrgEnv-amd` environment with full support for ROCm-5.0.2 and ROCm-5.1.0. However, CCE support for ROCm-5.x is lagging.

GTL is packaged with Cray MPICH and is released as a single build rather than individual builds for each compiler. This has important consequences due to the way that CPE packages are set as default and how their libraries are found at runtime. When setting a default release of a CPE package, the package's modulefile is set as the default version and its libraries are added to the `ld.so.cache` file to be found at runtime without modification to the user's environment. With the current mechanisms for setting defaults, this leaves only one version of the GTL library for AMD GPUs to be found without environment modifications. When compatibility is broken between GTL and ROCm releases, this necessitates the setting of defaults modules for `PrgEnv-amd` to versions which match the ROCm compatibility of `PrgEnv-cray`.

Table I shows versions compatibility between Cray-MPICH and ROCm. They are determined by matching the `DT_SONAME` from ROCm to the `DT_NEEDED` from Cray-MPICH library. If all needed symbols are successfully resolved, the versions are deemed compatible.

Due to the fact that ROCm is not provided within PE, we deployed site local `rocm` modulefiles. These can be loaded independent from the specific `PrgEnv-*` loaded. The advantage of this approach is that we can quickly

cray-mpich	ROCm
8.1.10	4.3, 4.5
8.1.11	4.3
8.1.12	4.5
8.1.13	4.5
8.1.14	4.5
8.1.15	5.0, 5.1

TABLE I: Cray-MPICH and ROCm version compatibility. Compatibility was determined by matching the HIP library's DT_SONAME to the MPICH library's DT_NEEDED, and the successful resolution of all needed symbols.

add new ROCm versions as soon as released. However, because `PrgEnv-amd` only supports up to specific versions of ROCm, we found that conflicting modules could be simultaneously loaded. For example, a user could load `rocm/5.0.2` and `amd/4.5.0` which provide a different version of `amdclang`, `amdclang++`, `hipcc` and the corresponding libraries. As the `PrgEnv-amd` matured, we also no longer had the need for the `rocm-compiler` so opted to hide that in our environment.

B. Process placement and mapping

Task placement and distribution through Slurm proved to be challenging on both Spock and Crusher. Spock sets `CR_Pack_Nodes` in its Slurm configuration to pack tasks as tightly as possible onto the first node of the allocation before placing tasks on the next node and repeating the process. This packing behavior initially ignored task-to-GPU locality despite the job step being given flags to control such placement and binding. This issue has since been resolved by SchedMD.

Crusher's unique node architecture has also presented unique allocation challenges. With `l3cache_as_socket` configured, Slurm allocates CPU cores cyclically across L3 domains; each one with its own xGMI link directly attached to a single MI250X GCD. The order in which each GCD is allocated to the tasks does not follow the same cyclic order on a Crusher compute node. This leads to cases where the job step's allocated CPU cores and GCDs are frequently not connected by a single xGMI link, but instead must route through multiple links. A fix to enforce the allocation and binding to the nearest GCD is expected to be released in Slurm 22.05.

C. Application workloads

1) GPU-Aware MPI Collectives: Early on during Crusher's deployment, we noticed that some applications that relied on GPU-Aware MPI will result in nodes rebooting unexpectedly. One application suite that was able to reliably reproduce the problem was the ROCm-enabled version of the OSU Microbenchmarks [16]. In particular, the `osu_ialltoallv` test when using 8 ranks per node on two or more nodes will result in at least one node crashing. The issue was impacting different applications including GenASiS, Cholla, and multi-node ML workloads using PyTorch and the RCCL backend. To work around this problem, users had to set `MPICH_SMP_SINGLE_COPY_MODE=NONE`. The issue is still being investigated but it appears to be related to an interaction between XPMEM and specific versions of the AMD GPU driver. A new Integrated Firmware Image (IFWI 042) version was deployed on Crusher in early April to partially address this problem [17].

D. File system availability and I/O

The hardware capabilities and software stack of Crusher created a number of challenges for directly accessing Alpine with native Spectrum Scale clients. The compilation flags and specific version of the kernel first deployed on Crusher fell outside of normal interoperability testing performed by IBM. This led to the stock Spectrum Scale client software failing to function without manual intervention and changes to the GPFS portability layer (GPL). Various Spectrum Scale client versions were tested on Crusher as an initial troubleshooting step before settling on version `5.1.3`.

One kernel compilation flag difference involved `CONFIG_RANDOMIZE_MEMORY`. To create a functioning Spectrum Scale client, the GPL was updated to check for the presence of another compilation flag instead. Without this change, the client software would fail with a variety of error signatures including a refusal to start the Spectrum Scale daemon or a failure to mount Alpine followed by a stream of errors in the Spectrum Scale log. Though this change allowed the Spectrum Scale client to successfully start on the Crusher login nodes, compute nodes continued to fail during daemon startup. Extensive troubleshooting discovered that the high amount of GPU memory in the Crusher compute nodes led to a conflict between the AMD GPU driver and Spectrum Scale software. Vendor support resolved this incompatibility with a new release of Spectrum Scale.

Issues with the underlying network can produce failure modes in the Spectrum Scale data path. Each additional entry in a multipath route introduces another potential point of failure because packets could be directed to a failed nexthop. In an attempt to avoid such a routing pattern, the `net.ipv4.fib_multipath_use_neigh sysctl` setting was enabled on Spock, Crusher, and Alpine [18]. This configuration improved the resiliency of the Spectrum Scale data path but has not fully mitigated against errors introduced by a failed gateway node.

A change to atomic opens in the upstream kernel meant that workloads encountered a `Permission denied` message when creating a copy of a file that lacked the `write` permission bit. This issue surfaced both when some workloads attempted to restore files from archives and when cloning particular `git` repositories. This behavior was corrected in Spectrum Scale version `5.1.3.0`.

E. Cassini Hardware Resources

The SS11 NIC is an advanced RDMA adapter that provides many facilities for offloading tasks that are traditionally performed using CPU cycles. Examples of this are tag matching, rendezvous offloading, and message triggering. To accomplish this, the NIC has a set of hardware resources that are allocated to each communicating process on the node. There are a limited number of these hardware resources and when they are all utilized, the communication subsystem, specifically `libfabric`, must fall back to traditional software mechanisms for performing this work. This led to some initial issues on Crusher, when the automatic fallback was not yet released. Using GPCNet as part of the initial network testing created a workload on the system that utilized 20 ranks per node with phases generating a significant amount of unexpected messages on the network. To accommodate unexpected messages in hardware, the NIC uses a resource called portal table list entries. The workload of GPCNet caused these resources to be totally consumed and without software fallback the tests failed. Further evaluation found that this issue could be exacerbated with as little as 10 ranks per node. HPE was able to provide a workaround by setting `FI_CXI_MSG_OFFLOAD=0` effectively disabling this offload and posting the messages to a request buffer. This workaround was no longer necessary with the release of HPE Slingshot version 1.7.0 (Field Notice #6735b).

F. Compilers and Programming Models

For Frontier, HIP and OpenMP are the primary programming models to target GPU. While HIP is based on C++ dialect, OpenMP is a directive-based language available for the most commonly used programming languages in HPC: C, C++, and Fortran. As such, OpenMP with target offloading must be supported by the vendor compilers on Spock, Crusher, and eventually, Frontier.

Support for GPU offloading was first introduced in OpenMP 4.0 specification. It was significantly refined in OpenMP 4.5 specification. OpenMP 5.0 and 5.1 versions of the specification extended it further with new concepts such as `declare variant`, `metadirective`, and `task detach`.

For testing of functionality related to target offload in OpenMP, we rely on the verification and validation test suite SOLLVE_VV [19] that is developed as part of the Exascale Computing Project (ECP) "Scaling OpenMP With LLVm for Exascale Performance and Portability (SOLLVE)" [20]. This test suite aims primarily at validating different compilers implementations of OpenMP offloading support starting with version 4.5 of the OpenMP specification as well as new features that are added to the specification that may not involve GPU offloading (e.g., the `loop` construct that was added in OpenMP 5.0).

We use the SOLLVE_VV test suite to track and evaluate functional support for OpenMP in the two main compiler suites on Spock and Crusher. During the evaluation period, the test suite experienced rapid development as more tests were added to cover more OpenMP features, while concurrently the compilers gained better support for more features.

Table II shows the evolution of the SOLLVE_VV test suite and the number of passing tests for OpenMP 4.5 and OpenMP 5.0 features. The compilers used are the latest releases available on the testing platform at time of testing. As the table shows, there was a significant progress in support for OpenMP in both compilers during the evaluation period, especially for 5.0 Features. Both compiler suites more than doubled the number of passing tests for OpenMP 5.0 features (while the number of tests in the test suite itself also increased). Work is still ongoing to fill any functional gaps in support for OpenMP 5.0 and subsequent versions of the OpenMP specification.

Beyond the issues uncovered using SOLVVE_VV tests, more were discovered when we used the compilers to build applications on Spock and Crushers. These issues can be broadly categorized as (1) issues in the base language implementation (i.e., C, C++, or Fortran) itself, (2) issues in the OpenMP implementation independent of the target CPU or GPU (offload) backends, and (3) issues that are directly related to compilers support for the the AMD GPU. Most of the applications we use to test the compilers have been developed, tested, and verified on the OLCF current system Summit. Therefore issues that are only on Spock or Crusher are marked as high priority since they represent functional regressions from Summit. We have reported these issues to the vendors whom have since made a good progress in resolving them.

G. Performance and Correctness Tools

As mentioned in Section III, HPE's performance and correctness tools for AMD GPUs are dependent upon AMD's libraries and tools for much of their AMD GPU support. One of the unfortunate implications of these dependencies is that due to rapidly evolving AMD development, the HPE tools are often incompatible with the most recently released ROCm software release. The integration required to adapt to breaking API changes and subsequent release testing of HPE tools often leads to compatibility gaps lasting months. Further compounding the compatibility issue is the desire to quickly adopt new ROCm releases to address bugs or issues that have been preventing application development progress.

Both AMD's and HPE's sets of tools suffer from a lack of adequate documentation and usage examples. For example, although GPU hardware performance counters can be collected, AMD documentation provides only the counter names and a brief description for each counter. There is no guidance related to sets of counters that may be used to investigate various performance issues, such as whether a given GPU kernel is compute-bound or memory-bound. This is primarily due to developers focusing on delivering needed functionality at the expense of generating useful documentation. Furthermore, both originally only supported GPU applications using the HIP programming model, due to ongoing delays in the development of OpenMP offload support for AMD GPUs in the vendor compilers.

OpenMP Version	SOLVE_VV Tests 4.5	5.0	CCE 4.5	5.0	AMD 4.5	5.0
April 2021 (Spock)	206	166	172	61	195	77
April 2022 (Crusher)	229	226	203	130	205	157

TABLE II: Results of compilers execution of SOLVE_VV test suite. The first column indicates the date on which the test suite was pulled from its repository. The second column shows the number of tests for OpenMP 4.5 and 5.0. The third and fourth columns show the number of passing tests for the CCE and AMD compilers, respectively.

H. Known Issue Tracking

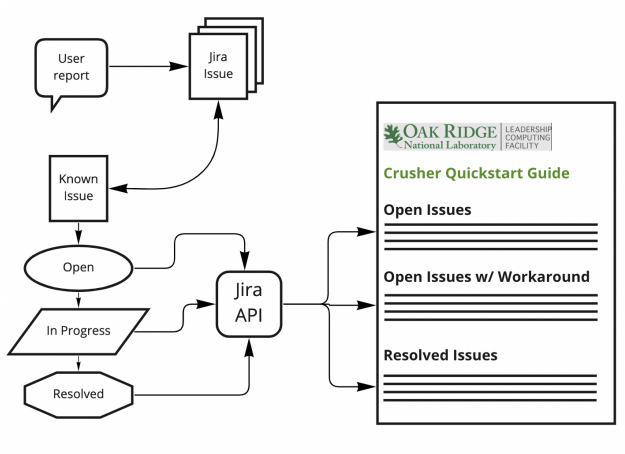

Fig. 4: Automated Jira to Sphinx-based documentation in RST.

As known issues (KIs) were discovered during the deployment and testing of the Crusher Test and Development System, we tracked these issues through Jira. To get these KIs published so that users could see them and their status, we developed a simple Python script that connected the API output of Jira to the `reStructuredText` (RST) input of our documentation system as illustrated in Figure 4. By moving the Jira issues between different statuses (i.e., Open, In Progress, or Resolved), the KIs would change status automatically on the published documentation. The KIs were also classified on the documentation system through a set of categories (e.g., running, compiling) that we were already using to categorize help tickets (we used the same Jira project for help tickets), to further aid in the readability of this information.

This setup reduced the overhead and increased the accuracy of KI tracking through several means. It offered a single, centralized and synchronized place to track incoming help requests, internal tracking of the KIs, and the published KI information to the end users. When we had an update to a KI, a single edit to a Jira issue would update internal tracking and the published KI information in one step. In previous deployments, we sometimes found the internal tracking of KIs would diverge from the externally published lists as updates got left out of one or the other over time. Additionally, the KIs and help tickets could be linked inside Jira, so that we could quickly see how many help tickets a particular KI had generated, and when a KI was resolved or a workaround was found, we could quickly see which tickets needed following up with the user.

V. Lessons Learned

Throughout this work, we have described several areas in the deployment of Spock and Crusher that required workarounds or site-specific updates. As a result, we have identified the recommendations and lessons learned that are listed in this section.

- While the timely provision of the newest ROCm release is important to assist in development efforts for users of the systems, it also creates friction when that release has multiple incompatibilities with the installed CPEs. We developed a set of scripts to quickly test for the dynamic linking compatibility between new ROCm libraries and Cray PE. The discovered incompatibilities are now documented in the Crusher Quickstart guide [2] to help minimize the time users spend troubleshooting these issues.
- With the relatively disproportionate amount of computing power delivered through GPUs for Spock, Crusher, and eventually Frontier, it is of utmost importance that we treat GPUs as first class resources. With each Slurm release, testing has been primarily focused on the placement, distribution, and binding of tasks relative to GPUs to ensure that progress is made towards the optimal utilization of hardware resources. Continued work with vendor partners will ensure that GPU capabilities are accessible in a user-friendly fashion.
- The rapidly evolving ecosystem for AMD GPUs necessitated efficient mechanisms to communicate known issues, workarounds, and fixes with the early users on both Spock and Crusher. As a result we developed a system that integrated the OLCF's internal bug tracking system (JIRA) with the individual system pages. This automatic mechanism to expose known issues and announces fixes has allowed us to keep users apprised of any changes in the status of open bugs.
- Careful coordination with vendor partners is critical in order to successfully deploy any new system and integrate it with center-wide resources. As described in Section IV-D, unexpected issues impacted our ability to make Alpine available on Spock and Crusher. Working closely with both HPE and IBM, we were able to receive the key bug fixes to complete successful deployment of both systems.

VI. Conclusion

In preparation for Frontier, the OLCF has deployed two separate systems that are available to the OLCF user community: Spock and Crusher. Spock is a MI100-based system which is one generation removed from the hardware that will be available on Frontier. Crusher is a test and development MI250X-based system with the same hardware as will be available on Frontier. Crusher has provided a platform to test the rapidly evolving AMD GPU ecosystem and has allowed users to get early experiences using the AMD MI250X GPU. As we have discussed here, the deployment of both test systems has helped OLCF get familiar with the programming environment, identify issues, put workarounds in place, and develop best practices that would be helpful for future Frontier users.

VII. Acknowledgement

The authors would like to acknowledge Cathy Willis for her guidance on modifications that we could apply locally to the HPE/Cray PE to better suit the needs of OLCF users.

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.

References

[1] "Spock: a pre-exascale system at ORNL," 2021. [Online]. Available: https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html

[2] "Crusher: an early-access testbed for Frontier at ORNL," 2021. [Online]. Available: https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html

[3] "Slurm Workload Manager," 2022. [Online]. Available: https://slurm.schedmd.com/

[4] Oak Ridge Leadership Computing Facility. Alpine IBM Spectrum Scale Filesystem. [Online]. Available: https://docs.olcf.ornl.gov/data/index.html#alpine-ibm-spectrum-scale-filesystem

[5] J. Hanley, C. Muzyn, and M. Ezell, "Improved I/O using native Spectrum Scale (GPFS) clients on a Cray XC system," in *Proceedings of the Cray User Group 2018 conference*, 2018.

[6] IBM Corporation. Mounting a remote GPFS file system. [Online]. Available: https://www.ibm.com/docs/en/spectrum-scale/5.1.3?topic=system-mounting-remote-gpfs-file

[7] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[8] "Phoenix Cluster Provisioning and Management Tool," https://github.com/olcf/phoenix.

[9] "Clang: a C language family frontend for LLVM," 2022. [Online]. Available: https://clang.llvm.org

[10] "HIP: C++ Heterogeneous-Compute Interface for Portability," 2022. [Online]. Available: https://github.com/ROCm-Developer-Tools/HIP

[11] "The OpenMP API specification for parallel programming," 2022. [Online]. Available: https://openmp.org

[12] "ROCm: Platform for GPU Enabled HPC and UltraScale Computing ," 2022. [Online]. Available: https://github.com/RadeonOpenCompute

[13] "Flang," 2022. [Online]. Available: https://github.com/flang-compiler/flang

[14] "AOMP - V 15.0-1," 2022. [Online]. Available: https://github.com/ROCm-Developer-Tools/aomp

[15] "ROCm Software Platform Repository," 2022. [Online]. Available: https://github.com/ROCmSoftwarePlatform

[16] "OSU Microbenchmarks," http://mvapich.cse.ohio-state.edu/benchmarks.

[17] "OLCFDEV-495: Some GPU Aware MPI collectives can cause node failures," 2022. [Online]. Available: https://docs.olcf.ornl.gov/systems/crusher_quick_start_guide.html#olcfdev-495-some-gpu-aware-mpi-collectives-can-cause-node-failures

[18] The Kernel Development Community. IP Sysctl. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/ip-sysctl.html

[19] J. M. Diaz, S. Pophale, O. Hernandez, D. E. Bernholdt, and S. Chandrasekaran, "Openmp 4.5 validation and verification suite for device offload," in *Evolving OpenMP for Evolving Architectures*, B. R. de Supinski, P. Valero-Lara, X. Martorell, S. Mateo Bellido, and J. Labarta, Eds. Cham: Springer International Publishing, 2018, pp. 82–95.

[20] "SOLLVE: Scaling OpenMP With LLVm for Exascale Performance and Portability," 2022. [Online]. Available: https://sollve.github.io/