Polaris and Acceptance Testing

Brian Homerding*, Ben Lenard*, Cyrus Blackworth, Carissa Holohan, Alex Kulyavtsev, Gordon McPheeters, Eric Pershy, Paul Rich, Doug Waldron, Michael Zhang, Kevin Harms, Ti Leggett, William Allcock
Leadership Computing Facility
Argonne National Laboratory
Lemont, IL
{bhomerding,blenard,cblackworth,carissa,alexku,gmcpheeters,pershy,
richp,dwaldron,mzhang,harms,tleggett,allcock}@anl.gov

***Abstract*—The Argonne Leadership Computing Facility (ALCF) is home to Polaris, a 44 peak PetaFLOP (PF) system developed in collaboration with Hewlett Packard Enterprise (HPE) and NVIDIA. Polaris is a heterogeneous system with 560 nodes utilizing NVIDIA GPUs along with a HPE Slingshot Interconnect and a HDR200 Infiniband network to storage. Due to hardware availability the delivery was performed in multiple stages. We introduce both hardware and software components of Polaris and discuss the performance of our thorough benchmarking analysis. ALCF policy is to perform a rigorous multi-week acceptance testing (AT) evaluation for every major system to ensure the capabilities of that system can support ALCF users' science application needs and meet ACLF system operational metrics. The various system components are thoroughly tested to ensure the system will be stable for production operation, function correctly, and fulfill performance expectations for scientific workloads. We will discuss how ALCF used Jenkins and ReFrame to perform the AT of the base Polaris system as well as a second AT to evaluate the Polaris CPU upgrade. We will present our approach for deploying Jenkins to streamline the AT evaluation with benchmarking improvements and lessons learned from the successful acceptance of the heterogeneous system, Polaris.**

***Index Terms*—deployment, integration, acceptance**

I. Introduction

In 2022, the Argonne Leadership Computing Facility (ALCF) acquired the Polaris system developed in collaboration with HPE and NVIDIA. At deployment, Polaris is a 44 peak PetaFLOP (PF) heterogeneous system. Polaris is comprised of 560 nodes utilizing NVIDIA GPUs for acceleration. Each compute node consists of an AMD EPYC "Milan" 7543P CPU with four NVIDIA HGX A100s. Polaris utilizes the HPE Slingshot Interconnect network system for high-speed, low latency communications. To access the external ACLF production Lustre filesystems, Eagle and Grand, Polaris utilizes HDR200 InfiniBand via gateway nodes. ALCF performs a rigorous multi-week acceptance testing (AT) process for its leadership class systems to ensure the capabilities of the system for supporting ALCF users' science applications and the Facility's ability to meet system operational metrics.

The Polaris compute nodes have an aggergate 160 GB HBM2 memory on the GPUs with 1.6 TB/s memory bandwidth per NVIDIA HGX A100 GPU. Additionally, each node has 512 GB total DDR4 memory with 204.8 GB/s memory bandwidth. There are two NVMe SSDs per node with a total capacity of 3.2 TB and two HPE Slingshot Cassini network adapters per node providing a total injection bandwidth of 50 GB/s. The PCIe Gen4 bandwidth on the node is 64 GB/s and the GPU NVLink bandwidth is 600 GB/s. The HPE Slingshot Interconnect provides multiple quality of service levels along with aggressive adaptive routing and advanced congestion control. This provides very low average and tail latency and high-performance multicast and reduction operations.

The AMD EPYC "Milan" 7543P CPU on each compute node provides 32 Zen3 cores with 64 total threads. The total shared L3 cache is 256 MB with a private L2 cache of 512 KB per core and a private L1 cache per core of 32 KB. Each of the NVIDIA HGX A100s on each compute node is capable of 9.7 TF at FP64 precision, this grows to 19.5 TF when utilizing the FP64 tensor core support. Each of the A100 GPUs has 40 GB of HBM2 memory.

To ensure that the Polaris system met the desired functionality, performance, and stability requirements many tests were implemented. These tests ranged from system operational tests, benchmarks designed to stress system components, to full scientific applications. ALCF leveraged the Jenkins automation framework and ReFrame regression testing framework, both open-source technologies, to perform the AT process for Polaris as well as acceptance of the Polaris upgrade. A rigorous multi-week system test of scientific applications was performed to ensure the stability of the system for steady-state production operation. The scientific applications covered a range of scientific domains and characteristics - these included HACC, QMCPack, NekRS, LAMMPS, and CosmicTagger. Each of these scientific applications were utilized at various scales - from single node up to full machine jobs - with the expected output and performance being validated. In all, 70 different scientific application configurations were utilized for the Polaris AT process.

ReFrame is a python framework for the development of HPC system tests and includes built-in support for HPC job schedulers, including Polaris's job scheduler, PBSPro, allowing for the easy submission of jobs and tracking of job progress. Each ReFrame test was staged in a unique directory with all execution dependencies, allowing for multiple concurrent independent ReFrame test launches. ReFrame pro-

*Both authors contributed equally to this research vides rich support for implementing complex correctness and performance checks, monitoring the validity and performance of the application executions.

To automate the execution of the individual ReFrame tests, we employed the Jenkins CI framework. Additionally, Jenkins provides a convenient interface and hooks to enable effective test failure triage. Jenkins enabled us to produce continuous submissions of the ReFrame tests, submitting a new execution of a given test after the completion of the previous execution. These features enabled the combined test harness of Jenkins and ReFrame components to produce a steady stream of diverse workloads for the system over the course of three weeks. The hooks in Jenkins provided tools for us to archive and notify on job failures. In the event of a ReFrame test producing a failure, Jenkins would produce a notification to a shared Slack space with a direct link to the failed test. Additionally on job failures Jenkins captured the job standard out and error to the web interface and saved the failed job stage to the Lustre file system. Easily accessing the standard out and error of the failed job through the web interface enabled quick triage of the failed tests for further investigation. Finally, Jenkins enabled the generation of various dashboards to provide a high-level view of the overall AT results. With the above Jenkins configuration, ALCF could trivially change the number of concurrent executions of the same test controlling which components of the system were stressed at any given time.

Over the course of a successful multi-week AT, a total of 99,381 jobs were executed with 146 failures. The system maintained 99.7% availability with 95.7% utilization including multiple 24-hour periods with no job failures due to system events. The test harness enabled ALCF to ensure the stability of the system for steady-state production operation, while performing as expected for scientific workloads. Throughout the Polaris AT process there were several lessons learned for successful system acceptance.

This configuration created a simple process for redirecting the archival of job output and failed job stages avoiding issues with file system availability. Having a hierarchical system for understanding failures streamlined the root cause analysis of issues. For Polaris, the first notification was through Slack from Jenkins which provided easy notification with minimally concise information. Next the Jenkins web interface provided the job standard output and error which allowed for quickly understanding most errors. Finally, the failed job ReFrame stages were archived on a Lustre filesystem and node logs (syslog, dmesg, etc.) were centrally gathered by the management stack for in-depth investigation. To enable the understanding of failed job underlying issues, generating test executions targeting specific system components enabled the isolation of the issue with a test to trigger it.

II. Related Work

When ALCF performed acceptance testing of its previous flagship supercomputer, Theta, in 2016, an internal application called the ALCF Test Harness was utilized [1]. The ALCF Test Harness provided automation for building, executing, validating results, as well as archiving artifacts during the acceptance testing process. These tasks are performed repeatedly during AT, and provide the metrics that determine if an acceptance test criteria is successful. By removing manual intervention with the ALCF Test Harness, mistakes caused by human error were reduced. The ALCF Test Harness was tightly integrated with the Cobalt [2] job scheduler.

River Compute Racks	40
Apollo Gen10+ Chassis	280
Compute Nodes	560
AMD EPYC 7543P CPUs	560
NVIDIA A100 GPUs	2240
Total GPU HBM2 Memory	87.5TB
Total CPU DDR4 Memory	280 TB
Total NVMe SSD Capacity	1.75 PB
Interconnect	HPE Slingshot
Slingshot Endpoints	1120
Rosetta Switches	80
Total Injection BW (w/ Cassini)	28 TB/s
Total GPU DP Tensor Core Flops	44 PF
Total Peak Power	1.8 MW

TABLE I: Polaris System Configuration

Jenkins is an open source tool for Continuous Integration and Continuous Deployment platform that is written in Java [3]. Around 2018, the ALCF began to utilize Jenkins for Continuous Integration in their HPC environment, namely on Theta, so that users can compile and execute their code [4]. Therefore it was a natural leap to utilize Jenkins for Polaris's AT as discussed in section VI-A.

The Swiss National Supercomputing Centre developed a regression framework for HPC systems written in Python called ReFrame that was presented at Cray Users Group in 2018 [5]. This framework interfaces with the scheduler and launcher for benchmarks that are executed on the HPC system. The output of the job execution is validated by user-defined sanity and performance functions.

HPE OneView is software that provides hardware error reporting and firmware information about HPE nodes within the system environment by communicating with the node's Lights-Out (ilo) card.

III. Hardware Details

The Polaris system consists of 560 compute nodes utilizing NVIDIA GPUs for acceleration. Each compute node has a single AMD EPYC "Milan" 7543P CPU with four NVIDIA HGX A100s. At deployment, Polaris achieved a peak performance of 23.8 PF on the November 2021 Top500 list [] and then 25.81 PF on the June 2022 list [] after the upgrade and tuning. Polaris utilizes the HPE Slingshot Interconnect network system for communication. The system has a total of 87.5 TB of HBM2 memory on the GPUs combined with 280 TB of DDR4 memory. The full Polaris system configuration is shown in Table I.

Each of the 560 Polaris nodes are heterogeneous, utilizing four NVIDIA A100 GPUs for acceleration. The node configuration is shown in Figure 1. The single AMD EPYC 7543P

CPU provides 32 Zen3 cores with 64 threads. Each of the four NVIDIA A100 GPUs includes 128 Streaming Multiprocessors with 64 FP32 CUDA cores per Streaming Multiprocessors. The CPU is connected to the GPUs using PCIe Gen 4 providing 64 GB/s bandwidth. The GPUs interconnect utilizes NVLink providing 600 GB/s. The single node specifications are show in Table II.

AMD EPYC 7543P CPUs	1
NVIDIA A100 GPUs	4
Total HBM2 Memory	160 GB
HBM2 Memory BW per GPU	1.6 TB/s
Total DDR4 Memory	512 GB
DDR4 Memory BW	204.8 GB/s
NVMe SSDs	2
Total NVMe SSD Capacity	3.2 TB
Slingshot NICs	2
Total Injection BW (w/ Cassini)	50 GB/s
PCIe Gen4 BW	64 GB/s
NVLink BW	600 GB/s
Total GPU DP Tensor Core Flops	78 TF

TABLE II: Polaris Single Node Configuration

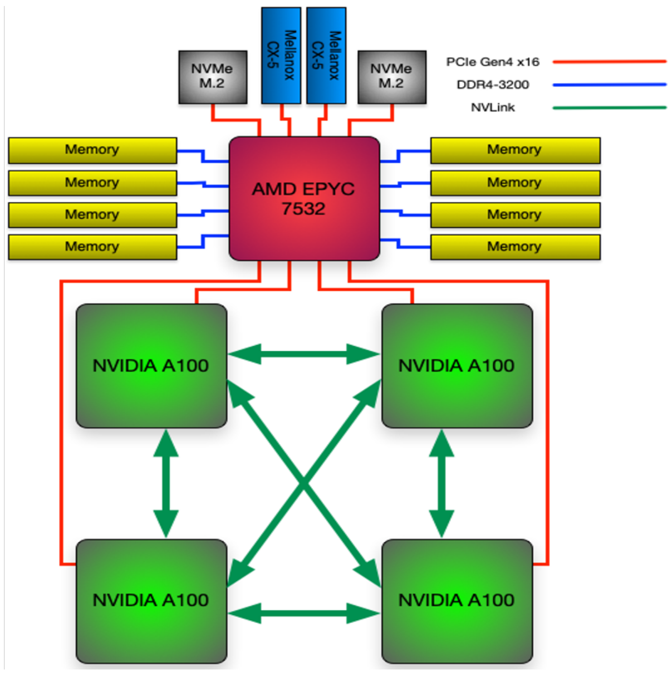


Fig. 1: Single Node Configuration.

A. Compute Characteristics

Each compute node has a single AMD EPYC "Milan" 7543P CPU. The base frequency is 2.8 GHz with a maximum boost frequency of 3.7 GHz. There are 32 Zen3 cores with 64 total threads per CPU. Each CPU is connected to four NVIDIA HGX A100s. The NVIDIA A100 is capable of 9.7 TF FP64 performance. Through the utilization of the tensor cores the FP64 performance increases to 19.5 TF. The BF16 and FP16 performance utilizing the tensor cores is 312 TF.

To measure the compute performance of the NVIDIA A100 GPUs, we utilized the CUDA implementation of the DGEMM kernel from the Parallel Research Kernels (PRK) [6]. PRK provides a variety of kernel operations implemented in many difference parallel programming models. We executed the cublas implementation to measure the compute performance across all Polaris GPUs. The results are shown in Table III.

Kernel	TFlops/sec
DGEMM	19.3
SGEMM	136.1

TABLE III: GEMM Average Performance Across All Polaris A100 GPUs

B. Memory Characteristics

Each GPU provides 40 GB of HBM2 memory, resulting in 160 GB per compute node. The HBM2 memory provides 1.6 TB/s bandwidth on the GPUs. Additionally, the CPU provides 512 GB of DDR4 memory through 8 memory channels. This results in 204.8 GB/s memory bandwidth when using the DDR4 memory.

The BabelStream [7] benchmark implements the STREAM [8] benchmark in various different programming models to provide support on a wider variety of platforms. We validated the performance of the NVIDIA GPUs by running the BabelStream benchmark on each GPU in the Polaris system. The CUDA implementation was utilized with a single thread offloading the kernels to a single GPU. The results are show in table IV.

Kernel	BW GBytes/sec
Copy	1387.64
Mul	1386.63
Add	1396.77
Triad	1397.36
Dot	1292.18

TABLE IV: BabelStream Average Performance Across All Polaris A100 GPUs

C. Communication Characteristics

Polaris utilizes the HPE Slingshot Interconnect network system. The Slingshot configuration is shown in Figure 2. Polaris has a total of 11 dragonfly groups, 10 of which are compute groups. The compute groups have 56 nodes each with 2 Slingshot NICs per node. Each compute group has 8 Rosetta switches with 4 links within each group. Each NIC has a single link to the Rosetta switches. There are 2 links between each compute group. The single non-compute group is a service group containing management nodes and I/O nodes.

Each link between and within the compute groups provide 200Gb/s. The Rosetta Switches provide multiple quality of service (QoS) levels. Additionally, the switches have aggressive adaptive routing with advanced congestion control and very low average and tail latency. Finally, the switches provide high performance multicast and reduction operations.

At the time of acceptance, Polaris utilized Slingshot 10, each node was equipped with 2 Mellanox ConnectX NICs with a 100Gb link. The injection bandwidth was ≈4TB/s

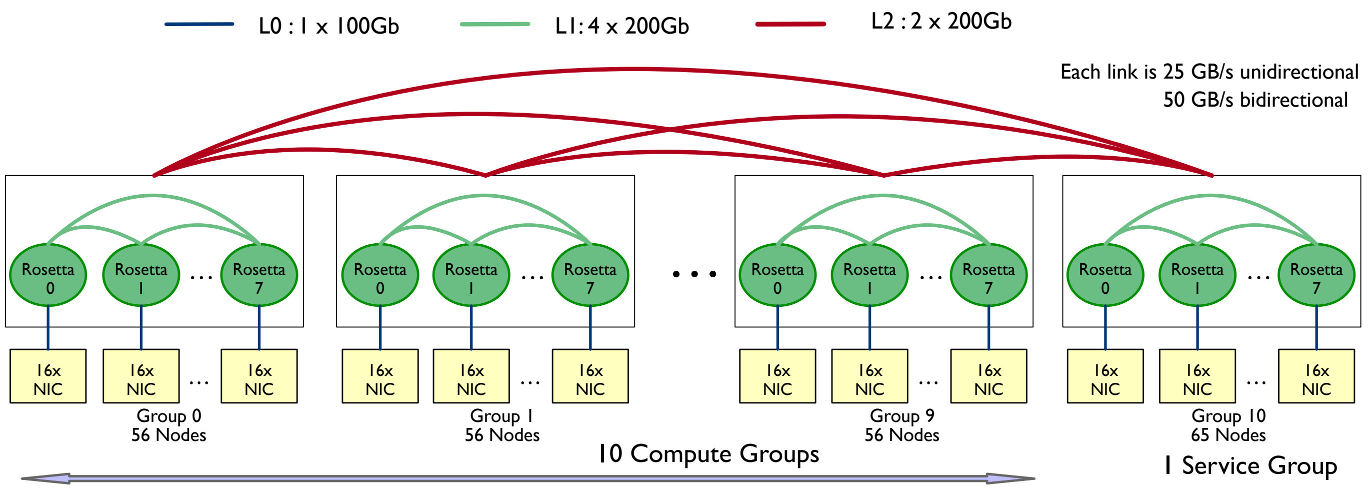


Fig. 2: Polaris Slingshot Configuration.

with Slingshot 10. Polaris will be upgraded to Slingshot 11 equipping it with 2 Cassini NICs per compute node, increasing the injection bandwifth to ≈28TB/s. In addition to doubling the injection bandwidth, Slingshot 11 provides MPI hardware tag matching and an independent MPI progress engine.

The ALCF MPI benchmarks [9] provide benchmarks to measure the performance of the communication on the system. The benchmarks were used to measure the bisection and injection bandwidth at the time of AT. The results are shown in Table V

Benchmark	TBytes/sec
Global Injection BW	13.7
Global Bisection BW	13.7

TABLE V: Bisection and Injection Bandwidth at time of AT.

D. *File Systems Used and Their Characteristics*

Polaris utilizes three of the ALCF's Lustre file systems in order to provide access to both large scale storage for project files, as well as provide access to the ALCF's Global home file system.

The two project systems are named Grand and Eagle. Both file systems are identical in configuration and are based on HPE's E1000 Lustre Storage Systems. Each project file system is 100 PB in size, consisting of 160 Object Storage Targets (OST) each of which is based on 53 16TB HDD in a GridRAID de-clustered RAID array. The 160 OSTs are served by 40 Object Storage Servers (OSS) with each server responsible to serve 4 OSTs.

The Metadata for each project file system resides on 40 Meta Data Targets (MDT). Each MDT consists of NVMe Flash drives configured as a RAID-10 device. These 40 MDTs are served by 20 Meta Data Servers (MDS).

Eagle and Grand are ALCF facility resources and as such are available to other facility Compute Clusters including Theta and Cooley. In addition to being available for the Compute Clusters' use, the two project file systems also offer Globus support for data transfer into and out of the ALCF facility in support of distributed workflows.

In addition to the project file systems being made available to Polaris during acceptance the ALCF also made available a DDN AI-400X named Swift. Swift is a NVMe/Flash based Lustre file system which the ALCF used to host both a `/home` file system during acceptance and also a space called `/soft` for third party tools/libraries. Both `/home` and `/soft` were sub-directories in the Swift file system but were presented to the Polaris system as two separate file systems using Lustre fileset mounts. Using the fileset mounts with Swift will allow a seamless transition should the ALCF decide to re-host either `/home` or `/soft` to a new file system at a later time.

In order to provide access to the Lustre file systems from Polaris Compute Nodes (CN) a set of Lustre gateway nodes (GW) were utilized. The Lustre storage systems, both for projects and home, reside on the ACLF's HDR-200 Infiniband (IB) network. The Polaris CNs are on the Slingshot network. The Lustre gateway nodes, having network interfaces on both the IB and the Slingshot networks, allows Lustre file system traffic to bridge these two networks.

Due to supply constraints of IB cables during the time frame of the Polaris installation and acceptance, the ALCF accepted the Polaris system using a subset of the total number of GW nodes. 36 of the 50 GW nodes were used during acceptance. 4 of the 36 GW nodes were dedicated to serving the `/home` and `/soft` structures while the other 32 GW nodes supported the IO traffic to Grand and Eagle. This traffic isolation was done to separate small trivial IO as characterized by `/home` and `/soft` accesses from larger block IO to the project file systems. Due to not having the full set of GW nodes available at acceptance time, the acceptance bandwidth expectations were adjusted linearly with the available GW node count.

IV. System Software

Polaris is a leading-edge system for scientists and application developers. Polaris is utilizes the HPE Performance Cluster Manager (HPCM) to provide provisioning, management, and monitoring. Polaris offers excellent capabilities in simulation, data, and learning by using NVIDIA's existing HPC SDK. Additionally, Polaris provides support for HPE Cray MPI and MPICH via libfabric using the Slingshot provider.

A. PBS

Polaris uses Altair's PBS Professional (PBSPro) for workload scheduling and resource management, along with HPE's provided Parallel Application Launch Service (PALS) for user task execution via PALS's implementation of mpiexec. For system shakedown and acceptance testing, PBSPro was configured for FIFO with strict job ordering with backfilling enabled. This allowed for a mix of jobs at different sizes to run and kept the system saturated with work and efficiently allocated resources to maintain the utilization required for the stability phase of acceptance testing, while avoiding heel-toe starvation of large, capability-sized jobs by smaller jobs submitted to the system. Capability jobs are jobs that utilize greater than twenty percent of system resources and are a key metric for the ALCF. A mix of job sizes, all the way to full-system jobs were run via PBSPro during the acceptance test period, including during stability testing, representing a normal, expected production workload in the ALCF.

B. Programming Environments

Polaris provides the HPE Cray Programming Environment. HPE Cray MPI provides support for gpuDirect offload to A100 for multi-NIC and multi-GPU support. The NVIDIA HPC SDK provides the primary support for programming the NVIDIA A100. In addition to OpenMP and CUDA support, the DOE programming models Kokkos and RAJA are supported. Support for the SYCL/DPC++ programming model on NVIDIA GPUs is provided through the CodePlay computecpp compiler and through the Intel DPC++ opensource LLVM compiler.

The HPE Cray Programming Environment provides Python with builtin support for many modules, including numpy, scipy, pandas and mpi4py. Additionally, optimized data learning and analytics frameworks are provided which are optimized for the GPUs through use of the NVIDIA libraries. The NVIDIA Rapids library provides support for data science and analytics.

C. Debugging and Performance Analysis Tools

The Polaris system provides several debugging tools to assist developers. In addition to **gdb**, **CUDA-GDB** [10] provides support for debugging CUDA code. To support debugging at the scale needed for a system the scale of Polaris, **gdb4hpc** provides a parallelized gdb for HPC. Additionally, the Stack Trace Analysis Tool (**STAT**) [11] provides support for stack tracing at scale.

To assist developers in leveraging the performance available on the Polaris system, several performance analysis tools are supported. The Cray Performance Measurement and Analysis toolset (**CrayPAT**) [12] provides a large toolset to understand whole application performance. With the large portion of performance provided by the GPUs, extracting high application performance on the GPUs is necessary. The **NVIDIA Nsight** [13] tool provides for system-wide performance analysis enabling analysis of the application GPU performance.

V. Acceptance Testing and Phases

The acceptance process for Polaris was designed to ensure a productive and stable system for science to verify the functionality, performance, and stability of the system based on the contract requirements. The process draws on several past successful ALCF system acceptances, including Intrepid [14], Mira [15], and Theta [16]. An Acceptance Test Plan (ATP) was created to clearly define the phases of acceptance along with the entrance and exit criteria for each phase. Additionally, the ATP defined the roles, responsibilities, and activities performed during the acceptance. An Acceptance Test Checklist (ATC) was created to clearly define the specific tests with their expected results along with the owners of the tests. The expected performance projections were made using ThetaGPU [17]. ThetaGPU is a heterogeneous ALCF resource providing identical NVIDIA A100 40GB GPUs along with AMD EPYC 7742 CPUs. In contrast to Polaris, ThetaGPU provides 2 CPUs and 8 GPUs per node. Additionally, ThetaGPU uses a Fat Tree Infiniband network instead of Slingshot.

The 3 phases of the ATP which rigorously evaluate the system are the 1) Functional, 2) Performance, and 3) Stability phases. We chose to combine the Functional and Performance phases (ATP-FP) to run concurrently. The ATP-FP could take up to a week, but because many of the disruptive system functional tests were completed before starting this phase, it completed several days sooner. Before starting the ATP-FP 100% of the hardware had to be up and healthy. All application and benchmark tests that completed successfully must obtain a correct result and any defects had to be categorized and a root cause analysis completed. After ATP-FP completed, the Stability phase (ATP-S) could start, but before starting all hardware again had to be 100% up and healthy. ATP-S must run for 21 continuous calendar days achieving $\geq$ 95% availability while maintaining $\geq$ 90% utilization. Availability was defined as:

$$\frac{\sum i^N (S_i - D_i)}{\sum i^N S_i}$$

where S_i is the number of schedulable hours for node i (wall clock time minus downtime scheduled by Argonne)

D_i is the number of hours of downtime for node i.

Also during ATP-S, there needed to be at least 24 continuous hours with no failures due to system defects and at least every application that could, had to have a full system (560 nodes) job complete successfully.

A. Functional and Performance Tests

1) Applications and Benchmarks: To ensure that the Polaris system successfully supports the range of ALCF user applications, 6 applications were selected to be included in the acceptance testing. These applications were selected to cover the breadth of simulation, data, and learning application needs. Additionally, these applications were chosen because they have a large user base, provide a high level of scalability, and are motivated to collaborate with the ALCF team. The following are the applications that were utilized during the AT of Polaris:

- *QMCPACK* [18]: Open source quantum Monte Carlo package for ab initio electronic structure calculations Weak scaling from 1 to 560 nodes, MPI + OpenMP, C++
- *LAMMPS* [19]: Classical molecular dynamics code with a focus on materials modeling Weak scaling from 1 to 560 nodes, MPI + Kokkos, C++
- *NekBench* [20]: Proxy application for NekRS (based on NEK5000) Navier Stokes solver based on the spectral element method Strong scaling from 32 to 512 nodes, MPI + OCCA, C++
- *HACC* [21]: Extreme-scale cosmological simulation code Weak scaling from 1 to 560 nodes, MPI + CUDA, C++
- *Cosmic Tagger* [22]: Removes background particles by applying semantic segmentation on full detector images from the SBND detector via deep learning Weak scaling from 1 to 512, Python + PyTorch + mpi4py, using NVIDIA container
- *Uno* [23]: Predict drug response to fight cancer cells via machine/deep learning Weak scaling on 1 node with 1 and 7 MIG instances on a single GPU, Python + Keras + Tensorflow

In addition to the 6 applications chosen, the following benchmarks were included to thoroughly test the system components:

- *OvO* [24]: Collection of OpenMP Offloading test functions for C++ and Fortran
- *SOLLVE VV* [25]: OpenMP Validation and Verification project is a suite of test cases to validate conformance and correctness for OpenMP 4.5/5.0 C/C++
- *HPL* [26]: Portable implementation of High-Performance LINPACK Benchmark
- *DGEMM* [6]: Benchmark to measure sustained floating-point rate
- *STREAM* [8] [7]: Benchmark to measure sustainable memory bandwidth, both GPU HBM and DDR4
- *IOR* [27]: Tests the performance of parallel file systems
- *MPI* [9]: Measures the performance of the communication
- *HPL-AI* [28]: High Performance LINPACK highlighting the convergence of HPC and AI workloads

2) System: In addition to the functional and performance application/benchmark tests discussed. A large set of functional system tests are run to ensure a productive system for science. Many of these tests ran outside of the Polaris Test Harness as human interaction was required.

- *Bill of Materials (BOM) Validation*: Validate that all hardware and software listed in the BOM is present, delivered, and installed
- *Cold and Warm Reboot*: Reboot all components - CNs, GWs, etc. - up and including the entire Polaris system from both warm and cold boot conditions to a correctly operational state
- *Software Environment*: Validate that the software environment - e.g., HPE PE, NVIDIA SDK, modules, etc. - is installed as described in Section IV.
- *Workload Manager and Batch Jobs*: Verify that the PBSPro workload manager can be configured and run workloads of many sizes and types and the job logs are captured and available
- *Compute Node Operating System (CNOS)*: Verify the configuration and functionality of the CNOS - e.g., system logs are captured; access to filesystems; process accounting; basic authentication, authorization, and accounting, etc
- *Reliability, Availability, and Serviceability (RAS)*: Set of tests to ensure RAS - e.g., system diagnostics present and functioning, component availablity/status is logged, power (AC and DC) events are handled gracefully and correctly (both for redundant and non-redundant fed), iLO is configured and accessible, etc
- *Network*: Test all components of the network are running the expected software and/or firmware, are remotely manageable, and handle various types of failures
- *System Security*: Perform security scans to ensure that the system does not present any known security exploits with a CVE rating of high or critical; this includes system software as well as firmware on all nodes. We used OneView for firmware verification by having it compare the firmware versions provided by HPE against what was installed on the nodes

B. Stability

During the stability portion of the acceptance testing all of the applications and some of the benchmarks tested in the functional and performance phases were utilized. Across the six applications, 71 workload configurations were utilized. These configurations ranged from single node to full machine jobs (560 nodes) and included several long running, multi-hour configurations. In addition to the application tests, two single-node benchmarks were included with seven different problem configurations. The full breakdown of workload configurations and scale are listed in table VI.

Application/Benchmark	Configurations	Scale (Nodes)
CosmicTagger	8	1 – 128
HACC	20	1 – 560
LAMMPS	22	1 – 560
NekBench	10	32 – 512
QMCPack	10	2 – 560
Uno	1	1
BabelStream	5	1
D/SGEMM	2	1

TABLE VI: Stability Phase Tests

VI. Polaris Test Harness

The Polaris Test Harness leverages two open-source technologies, ReFrame [5] and Jenkins [3]. These two technologies provide features to create an automatic tool for test control, execution and validation.

A. Jenkins

As discussed in Section II, the ALCF has utilized Jenkins within the HPC environment in the past for Theta; so when the deployment of a new supercomputer arose, there was the option update the custom built Test Harness used for Theta's AT, or use Jenkins similar to how we used it for Continuous Integration & Continuous Deployment (CICD) on Theta.

We deployed a dedicated Jenkins instance for AT for the ALCF; while Polaris AT was the only AT occurring at the time, it was deployed with multitenancy in mind. We connected Jenkins to the ALCF centralized LDAP environment which provides authentication as well as Linux groups. During the AT of Polaris it is important to point out that the vendors, HPE and NVIDIA, had read-only access to Jenkins while the ALCF staff driving and managing AT had read-write access. Within this instance of Jenkins, folders were utilized to isolate different systems (i.e., Polaris and future systems such as Aurora), if needed, as well as access controls.

For each of the different applications used for AT and described in Section II, there was a range of different parameters which constituted its own test. For example, Uno had one configuration whereas LAMMPS had twenty-two different configurations (see table VI), and as a result twenty-two different Jenkins jobs.

The Jenkins application was permitted to SSH into the Polaris login nodes, as a service account, so that it could spawn its own executor on the server. The only requirement for Jenkins besides SSH access was Java. Jenkins provides the ability to control the number of concurrent job submissions on a node as well as the concurrent executions for a given job. In other words, we could define that Login1 can have 16 Jenkins jobs executing concurrently, and of the 16, only 2 STREAM jobs of a given configuration could be simultaneously running.

Within a Jenkins job, regardless of ReFrame, a failure is detected when a Return Code (RC) for a command in the job's script returns a non-zero and is discussed further below. Regardless of the RC, Jenkins will capture `stdout` and `stderr` gathered for its agent.

One of the useful Jenkins features that was utilized was integration with Slack for notifcations in addition to standard email notifications. We created a dedicated Slack channel for Jenkins job failures which the AT team monitored. The notifications included a URL to the failed job's output making it easy to do initial triage (see 4). While we used a centralized Slack Channel, it is also possible to notify individual users within Slack as well.

Jenkins also has the ability to execute conditional steps during a job's execution, and we leveraged this functionality to capture artifacts when a job failed so that an investigation could take place.

B. ReFrame

ReFrame [5] is a python framework for the development of HPC system tests. It provides built-in support for HPC job schedulers, including Polaris's job scheduler, PBSPro, allowing for the easy submission of jobs and tracking of job progress. Additionally, ReFrame provides rich support for implementing complex correctness and performance checks.

ReFrame was configured to recognize Polaris's nodes, scheduler, launcher, module system, and environments. The ALCF test files were then created with the assistance of the test owners to create one or more sanity functions to validate the execution. For the performance checks, the mutually agreed to projected performance metrics in the ATP were encoded and compared against. Each job customized the job options and launcher options to control the resources and affinity.

The execution of the ReFrame tests followed the same pattern. For each instance of a test execution, ReFrame creates a directory and copies or links to all needed files/executables for the test. For stability testing, a unique test stage directory is created to enable safe concurrent executions of multiple instances of a test. ReFrame provides builtin support for PBS to submit jobs as defined by the python test files. The Polaris instance of the ReFrame PBS support is customized to match the PBSPro support on Polaris. Upon job completion the sanity functions encoded in the python test files are run to verify the correctness of the test. The sanity functions range from simple checks for specific output values to validating the results of different aspects of the scientific simulation is within expected α. The test instances check the achieved performance against figure of merits used during the performance test phase triggering failures if performance is below $\sim$ 5% of target.

C. Jenkins and ReFrame Pipeline

Each of the individual test configurations is a single Jenkins job, each of which can be enabled or disabled. Through this mechanism, we controlled the jobs which were active on the system at any given time. With this control we initiated the stability test with full machine jobs. After allowing only full machine jobs to run continuously on the system for hours, the smaller configurations were enabled.

Jenkins and ReFrame handle different aspects of the AT process and together they compose the pipeline utilized for Polaris. Figure 3 illustrates the pipeline as described below. Jenkins first submits a test, or Jenkins Job, which is then dispatched to the node where Jenkins has SSH access. The Jenkins job launches the ReFrame test. ReFrame takes over staging the test, submitting it to PBS, and validating the output as well as the performance metrics. If the test passed the check for completeness and performance, ReFrame exited with a RC of zero, which signaled to Jenkins that the test was successful. During AT, Jenkins continuously submitted jobs. Regardless of job exit status, the test is requeued to run again.

When a test returned a non-zero to Jenkins, Jenkins sent a failure message to a specific Slack channel along with the test's execution link for easy access. Jenkins then preserved the artifacts of the failed test by copying the stage test to

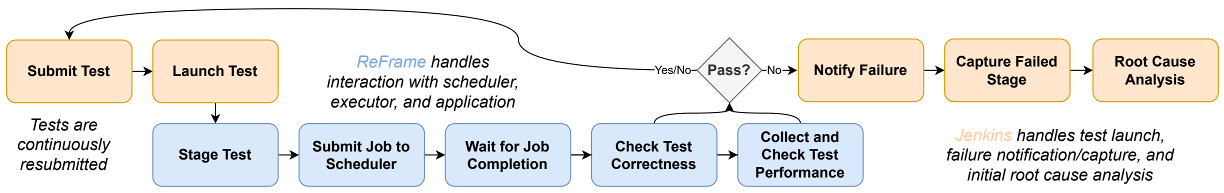
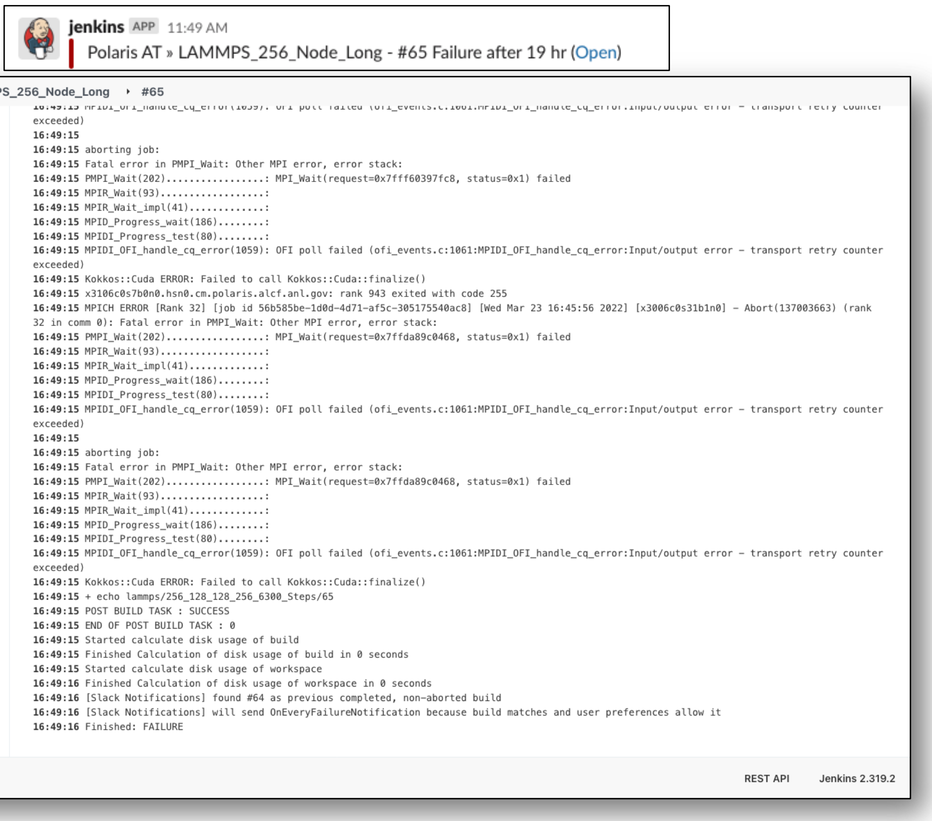

Submit Test
Launch Test
Tests are continuously resubmitted
Stage Test
ReFrame handles interaction with scheduler, executor, and application
Submit Job to Scheduler
Wait for Job Completion
Check Test Correctness
Collect and Check Test Performance
Yes/No
Pass?
No
Notify Failure
Capture Failed Stage
Root Cause Analysis
Jenkins handles test launch, failure notification/capture, and initial root cause analysis

Fig. 3: Polaris Test Harness Pipeline.



Polaris AT » LAMMPS_256_Node_Long - #65 Failure after 19 hr (Open)

```
exceeded)
16:49:15
16:49:15 aborting job:
16:49:15 Fatal error in PMPI_Wait: Other MPI error, error stack:
16:49:15 PMPI_Wait(202).................: MPI_Wait(request=0x7fff60397fc8, status=0x1) failed
16:49:15 MPIR_Wait(93)..................:
16:49:15 MPIR_Wait_impl(41).............:
16:49:15 MPID_Progress_wait(186)........:
16:49:15 MPIDI_Progress_test(80)........:
16:49:15 MPIDI_OFI_handle_cq_error(1059): OFI poll failed (ofi_events.c:1061:MPIDI_OFI_handle_cq_error:Input/output error - transport retry counter
exceeded)
16:49:15 Kokkos::Cuda ERROR: Failed to call Kokkos::Cuda::finalize()
16:49:15 x3106c0s7b0n0.hsn0.cm.polaris.alcf.anl.gov: rank 943 exited with code 255
16:49:15 MPICH ERROR [Rank 32] [job id 56b585be-1d0d-4d71-af5c-305175540ac8] [Wed Mar 23 16:45:56 2022] [x3006c0s31b1n0] - Abort(137003663) (rank
32 in comm 0): Fatal error in PMPI_Wait: Other MPI error, error stack:
16:49:15 PMPI_Wait(202).................: MPI_Wait(request=0x7ffda89c0468, status=0x1) failed
16:49:15 MPIR_Wait(93)..................:
16:49:15 MPIR_Wait_impl(41).............:
16:49:15 MPID_Progress_wait(186)........:
16:49:15 MPIDI_Progress_test(80)........:
16:49:15 MPIDI_OFI_handle_cq_error(1059): OFI poll failed (ofi_events.c:1061:MPIDI_OFI_handle_cq_error:Input/output error - transport retry counter
exceeded)
16:49:15
16:49:15 aborting job:
16:49:15 Fatal error in PMPI_Wait: Other MPI error, error stack:
16:49:15 PMPI_Wait(202).................: MPI_Wait(request=0x7ffda89c0468, status=0x1) failed
16:49:15 MPIR_Wait(93)..................:
16:49:15 MPIR_Wait_impl(41).............:
16:49:15 MPID_Progress_wait(186)........:
16:49:15 MPIDI_Progress_test(80)........:
16:49:15 MPIDI_OFI_handle_cq_error(1059): OFI poll failed (ofi_events.c:1061:MPIDI_OFI_handle_cq_error:Input/output error - transport retry counter
exceeded)
16:49:15 Kokkos::Cuda ERROR: Failed to call Kokkos::Cuda::finalize()
16:49:15 + echo lammps/256_128_128_256_6300_Steps/65
16:49:15 POST BUILD TASK : SUCCESS
16:49:15 END OF POST BUILD TASK : 0
16:49:15 Started calculate disk usage of build
16:49:15 Finished Calculation of disk usage of build in 0 seconds
16:49:15 Started calculate disk usage of workspace
16:49:16 Finished Calculation of disk usage of workspace in 0 seconds
16:49:16 [Slack Notifications] found #64 as previous completed, non-aborted build
16:49:16 [Slack Notifications] will send OnEveryFailureNotification because build matches and user preferences allow it
16:49:16 Finished: FAILURE
```

Fig. 4: Initial root cause analysis through Slack notification and Jenkins.

another location. Once these steps were completed, a manual Root Cause Analysis was initiated by one of the AT team.

D. Root Cause Analysis

A root cause analysis was performed on every job that failed during all 3 AT phases to determine whether the failure was caused by the system - e.g., system software defect, hardware failure, etc. - or not - e.g., application defect, workload manager issue, full filesystem, etc.

In the event of a test failure Jenkins sent a notification via Slack including a direct link to the failed test output in the Jenkins GUI. The `stdout` and `stderr` from the ReFrame submission was captured in the Jenkins console log of all failed Jenkins Jobs. An example of the Jenkins job console log interface is shown in Figure 4. This easy access to the console log through the Jenkins web GUI simplified the triage of job failures and was the initial step in every root cause analysis. Once some common failure signatures were identified, it was fast and easy to classify these failures using the Jenkins console log. Some failures could not be positively classified using only the Jenkins console log. In these cases, further analysis was conducted by investigating node system logs, BMC/iLO logs and metrics, etc.

When the root cause of the failure was unclear through system logs for job artifacts, steps were taken to reproduce the issue. To assist with the reproduction of issues, we utilized features of Jenkins and ReFrame to control the test execution. In some cases the exact group of nodes involved with the original failure were utilized to reproduce the failure. In these cases, the ReFrame test configuration was altered or duplicated to submit to the suspect nodes or a subset thereof. In other cases it required running a specific application workload repeatedly on many different nodes. In these cases, Jenkins

Jobs were replicate to increase the number of concurrent executions of the test to give the job more presence during the testing phase. Jenkins and ReFrame allowed the AT team to control the exact workload mixture at any given time to stress the system in the desired way.

VII. Results

The Polaris system was successfully accepted after completing each phase of the ATP. The Functional & Performance phases were completed successfully in less than the planned 7 day period. Following the Functional & Performance phases, the Stability phase was completed over the course of 21 days. After a successful acceptance the Polaris system was transitioned to production. The individual results for tests for each of the phases of the ATP follow.

A. Functional & Performance Results

1) QMCPACK: QMCPack returned correct results across the tested problems. The performance achieved was 10% below the projected target. This performance loss was due to changing one of the solver methods to run on the CPU rather than the GPU. A deprecation in the `cusolver` method used by QMCPack for this particular NVIDIA driver version motivated this change during AT.

2) LAMMPS: LAMMPS returned correct results across all test problems. The performance exceeded the projected performance for all targets except for the single node run, which was still within performance tolerances.

3) NekBench: NekBench correctly completed all test cases and exceeded the performance projections in all test cases.

4) HACC: HACC correctly completed all test cases and exceeded the performance projections in all test cases.

5) Cosmic Tagger: Cosmic Tagger completed correctly for all test cases with the exception of the 512-node scale. The performance failed to meet the performance projections on all correct test cases.

6) Uno: Uno successfully executed on a single node with NVIDIA Multi-Instance GPU mode (MIG) [29] enabled for a single instance as well as the maximum seven instances.

7) OvO: OvO completed with the NVIDIA compilers and had a 79% success rate. This matches the results on ThetaGPU and was expected for the compiler.

8) SOLLVE VV: The SOLLVE suite ran successfully with the NVIDIA compilers. The results matched the expected results for the compiler from ThetaGPU. The breakdown of the results are shown in Table VII.

OpenMP Version	C++	C	F90
4.5	100%	92%	91%
5.0	53%	23%	38%

TABLE VII: Pass Rates for SOLLVE suite.

9) HPL: HPL was run successfully and submitted to the Top500. The November 2021 list shows Polaris ranked at #12 with 23,840 TFlop/s

Test Case	Runs	Passed
BOM Validation	2	Yes
Cold and Warm Reboot	3	No
System Environment	10	Yes
PBS and Batch Jobs	21	Yes
Compute Node OS	7	Yes
RAS and Power Redundancy	11	Yes
Network	6	Yes

TABLE VIII: Functional System Results.

10) DGEMM: The results for DGEMM are shown in III. All nodes obtained greater than 95% of the peak for both DGEMM and TF32.

11) STREAM: The results for STREAM are shown in III. All nodes obtained greater than 95% of the expected peak.

12) IOR/mdtest: The IOR (IO throughput) and mdtest (meta data performance) results measured were within expectation for the less then full set of Lustre GW nodes that were available at the time of acceptance.

13) MPI: The results for the ALCF MPI benchmarks are shown in III. Additionally, the OSU functional benchmarks were successfully executed.

14) HPL-AI: HPL-AI was successfully run and was submitted to the HPL-AI list. The November 2021 list shows Polaris ranked #8 with a score of 0.114.

B. Functional System Results

All functional tests passed or were deferred at Argonne's discretion as seen in Table VIII. The cold and warm reboot test did not pass due to the full system reboot exceeding the time required by the test.

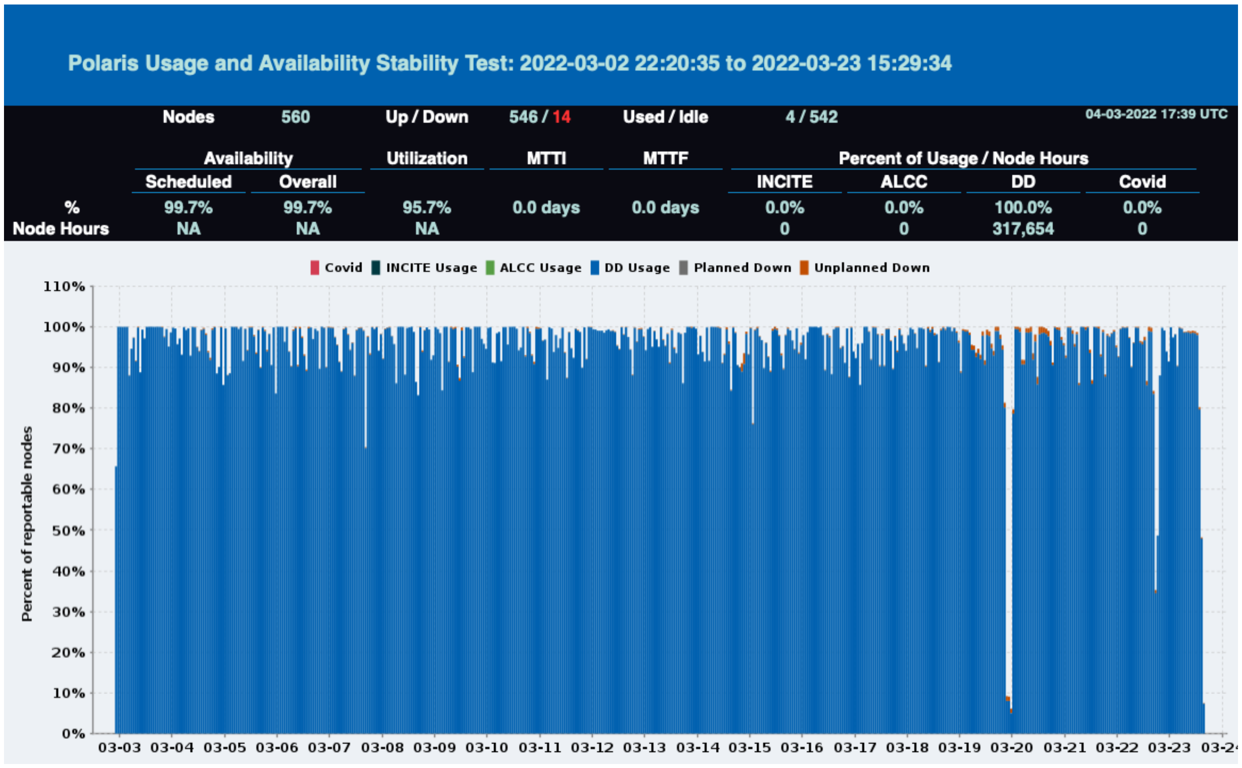
C. Stability Results

The stability phase of the Polaris acceptance imposed the following requirements which were met:

- 21 contiguous days
- 95% availability
- 90% load
- 24 hours of no job failures related to system software or hardware

In total, 99,381 total jobs were run, of which 146 were failures. 99.7% availability was achieved with 95.7% utilization. During the stability period, there were 6 distinct 24+ hour periods without job failures due to system software or hardware. The full breakdown of test execution and failures is shown in table IX. The type of failures are shown in table X. NekBench error was a known failure which occurred occasionally in the MLX5 ethernet driver. The environment for the NekBench tests was setup to minimize the occurrences of this error during stability.

Over the course of the Stability phase, the test harness notified and assisted with the root cause analysis of the failures experienced. Additionally, the availability and utilization metrics were tracked and accessible through a dashboard. The full stability run availability and utilization is shown in Figure 5.


Polaris Usage and Availability Stability Test: 2022-03-02 22:20:35 to 2022-03-23 15:29:34

Nodes 560 Up / Down 546 / 14 Used / Idle 4 / 542 04-03-2022 17:39 UTC

	Availability		Utilization	MTTI	MTTF	Percent of Usage / Node Hours			
	Scheduled	Overall				INCITE	ALCC	DD	Covid
%	99.7%	99.7%	95.7%	0.0 days	0.0 days	0.0%	0.0%	100.0%	0.0%
Node Hours	NA	NA	NA			0	0	317,654	0

Fig. 5: Polaris Availability and Utilization during Stability.

App/Benchmark	Runs	Failures
BabelStream	45903	3
CosmicTagger	6359	1
D/SGEMM	16404	0
HACC	6989	13
LAMMPS	9598	36
NekBench	1837	12
QMCPack	2947	21
Uno	374	1

TABLE IX: Stability Application Runs and Failures.

Failure Type	Count
Hardware	32
Human Error	2
Performance	8
Walltime	43
Software	1
NekBench Error	1

TABLE X: Failure Types

D. *Results Reporting*

These results were visualized by our Business Intelligence (BI) team. The metrics were calculated from data collected using the following methods:

- *Polling*: PBS command line commands were executed every 60 seconds to check the status of each node. The output was parsed and stored in a database table.
- *ETL of log files*: Our Extract Translate and Load (ETL) processing extracted data from the log files created by PBS and loaded them into our raw database tables. Each line is a record with record types that define the Queued, Start, End, Abort events of each job.
- *Back end processing*: Back end processes read the raw database data and inserted records into our Dimension and Fact database tables in our data warehouse. Further processing aggregated the dimension and fact records into summary records which were used by graphing software to generate graphs and data images.

Jenkins launched the software for all of the above resulting in a web page that updated every 3 minutes displaying the near real time metric values and graphs. Figure 5 was generated by the BI team showing availability and usage of the system during various phases of AT.

VIII. Polaris Upgrade

Polaris began life with compute nodes being delivered with AMD ”Rome” CPUs with the same 32-cores but Zen2 instead of the Zen3 micro-architecture. The original design of Polaris was to use the AMD ”Milan” processors described throughout, but in order to speed up initial delivery and acceptance, the Rome CPUs were used in the interim. The effort in development of the fully automated test harness paid off many times over. HPE performed the CPU upgrade from Rome to Milan CPUs onsite. The automated test harness was enabled once the compute node upgrades were complete in order to shakeout nodes. The test harness quickly helped to identify failing nodes. These could have been bad CPUs, or simple component issues resulting from jostling of NICs, DRAM or cables which were easily fixed. Once this shakeout was complete, a three data stability test was completed using the same conventions as defined in section V-B. Passing this three day stability proved that the Polaris CPU upgrade had been completed successfully and the system was ready for production once again.

Acknowledgment

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.

We would like to recognize all the ALCF staff who contributed on the integration efforts and acceptance testing.

We would like to thank the following HPE and NVIDIA personnel for their extraordinary efforts: Jon Bouvet, Carrie Breuer, Greg Cross, Lisa Giacchetti, Max Katz, Mark Juaire, Todd Letsche, and many others.

References

[1] K. Harms, T. Leggett, B. Allen, S. Coghlan, M. Fahey, C. Holohan, G. McPheeters, and P. Rich, "Theta: Rapid installation and acceptance of an xc40 knl system," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, 2017.

[2] "Cobalt: Component-based lightweight toolkit." [Online]. Available: https://trac.mcs.anl.gov/projects/cobalt

[3] [Online]. Available: https://www.jenkins.io/

[4] B. Lenard and T. Jackson. Continuous integration in a cray multiuser environment. [Online]. Available: https://cug.org/proceedings/cug2018_proceedings/includes/files/pap171s2-file1.pdf

[5] V. Karakasis, V. Rusu, A. Jocksch, and J.-G. Piccinali, "A regression framework for checking the health of large hpc systems - cug," 2018. [Online]. Available: https://cug.org/proceedings/cug2017_proceedings/includes/files/pap122s2-file1.pdf

[6] R. F. Van der Wijngaart and T. G. Mattson, "The parallel research kernels," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, 2014, pp. 1–6.

[7] "Evaluating attainable memory bandwidth of parallel programming models via babelstream," *Int. J. Comput. Sci. Eng.*, vol. 17, no. 3, p. 247–262, jan 2018.

[8] J. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," *http://www. cs. virginia. edu/stream/*, 2006.

[9] V. Morozov, J. Meng, V. Vishwanath, J. R. Hammond, K. Kumaran, and M. E. Papka, "Alcf mpi benchmarks: Understanding machine-specific communication behavior," in *2012 41st International Conference on Parallel Processing Workshops*, 2012, pp. 19–28.

[10] "Cuda-gdb." [Online]. Available: https://docs.nvidia.com/cuda/cuda-gdb/index.html

[11] D. C. Arnold, D. H. Ahn, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Stack trace analysis for large scale debugging," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–10.

[12] S. Kaufmann and B. Homer, "Craypat-cray x1 performance analysis tool," *Cray User Group (May 2003)*, 2003.

[13] "Nvidia nsight." [Online]. Available: https://developer.nvidia.com/nsight-systems

[14] "Intrepid supercomputer." [Online]. Available: https://www.anl.gov/article/alcf-intrepid-100tf-bgp-system-goes-live

[15] "Mira." [Online]. Available: https://www.alcf.anl.gov/alcf-resources/mira

[16] T. Leggett, K. Harms, B. Allen, S. Coghlan, M. Fahey, E. Holohan, G. McPheeters, and P. Rich, "Theta: Rapid installation and acceptance of an xc40 knl system," 1 2018. [Online]. Available: https://www.osti.gov/biblio/1432488

[17] "Thetagpu." [Online]. Available: https://www.alcf.anl.gov/alcf-resources/theta

[18] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley *et al.*, "Qmcpack: an open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids," *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, 2018.

[19] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen *et al.*, "Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, vol. 271, p. 108171, 2022.

[20] "Nekbench." [Online]. Available: https://github.com/Nek5000/nekBench

[21] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka *et al.*, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016.

[22] R. Acciarri, C. Adams, C. Andreopoulos, J. Asaadi, M. Babicz, C. Backhouse, W. Badgett, L. Bagby, D. Barker, V. Basque *et al.*, "Cosmic ray background removal with deep neural networks in sbnd," *Frontiers in artificial intelligence*, vol. 4, p. 649917, 2021.

[23] P. Balaprakash, R. Egele, M. Salim, S. Wild, V. Vishwanath, F. Xia, T. Brettin, and R. Stevens, "Scalable reinforcement-learning-based neural architecture search for cancer deep learning research," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2019, pp. 1–33.

[24] "Ovo: Openmp vs offload." [Online]. Available: https://github.com/TApplencourt/OvO

[25] T. Huber, S. Pophale, N. Baker, M. Carr, N. Rao, J. Reap, K. Holsapple, J. H. Davis, T. Burnus, S. Lee, D. E. Bernholdt, and S. Chandrasekaran, "Ecp sollve: Validation and verification testsuite status update and compiler insight for openmp," in *2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2022, pp. 123–135.

[26] J. J. Dongarra, "Performance of various computers using standard linear equations software," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 3, pp. 22–44, 1992.

[27] "Ior." [Online]. Available: https://ior.readthedocs.io/en/latest/

[28] "Hpl-ai." [Online]. Available: https://icl.utk.edu/hpl-ai/

[29] "Multi-instance gpu." [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/mig/mig.html