Balancing Load in More Ways than One

Verónica G. Melesse Vergara, Paul Peltz, Nick Hagerty, Christopher Zimmer,
Reuben D. Budiardja, Dan Dietz, Thomas Papatheodore,
Christopher Coffman, Benton Sparks
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: vergaravg@ornl.gov

***Abstract*—The newest system deployed by Oak Ridge National Laboratory (ORNL) as part of the National Climate-Computing Research Center (NCRC) strategic partnership between U.S. Department of Energy and the National Oceanic and Atmospheric Administration (NOAA), named C5, is a HPE/Cray EX 3000 supercomputer with 1,792 nodes interconnected with HPE's Slingshot 10 technology. Each node is comprised of two 64-core AMD EPYC 7H12 processors and has 256GB of DRAM memory. In this paper, we describe the process ORNL used to deploy C5 and discuss the challenges we encountered during execution of the acceptance test plan. These challenges include balancing of: (1) production workloads running in parallel on the Gaea collection of systems, (2) the mixture and distribution of tests executed on C5 against F2, the shared Lustre parallel file system, simultaneously, (3) compute and file system resources available, and (4) the schedule and resource constraints. Part of the work done to overcome these challenges included expanding monitoring capabilities in the OLCF Test Harness which are described here. Finally, we present benchmarking results from NOAA benchmarks and OLCF applications that were used in this study that could be useful for other centers deploying similar systems.**

***Index Terms*—system testing, acceptance testing, benchmarking**

I. Introduction

As a result of a recently renewed strategic partnership, the National Climate-Computing Research Center (NCRC) procured and deployed a new system for the National Oceanic and Atmospheric Administration (NOAA) research community in 2022. NCRC has been managed and operated by the National Center for Computational Sciences (NCCS) at Oak Ridge National Laboratory (ORNL) since 2009 [1]. NCCS systems and operations staff have deployed five separate systems as part of the Gaea project for NCRC since it was first established.

The newest system deployed, named C5, is a HPE/Cray EX 3000 supercomputer with 1,792 nodes that was deployed in 2022 and then expanded to 1,920 compute nodes in the first quarter of 2023. C5 shares a similar node architecture as that described in [12], however, there are key differences and improvements in the system software and configuration that are described in this work.

Although the C5 system is deployed and configured as an independent system, it is accessible to users via Slurm's multi-cluster environment alongside the two production systems, C3, a Cray XC40 with 1,496 dual Intel Haswell nodes, and C4, a Cray XC40 with 2,656 dual Intel Broadwell nodes. C5 which is a HPE Cray EX 3000 with 1,920 dual AMD Rome nodes. In addition, all NCRC systems, C3, C4, and C5, share a 37.5PB Lustre parallel file system named F2.

While the architecture of C5 is homogeneous, the nature of the NCRC multi-cluster environment posed unique challenges for this deployment. Production workloads were running concurrently on C3 and C4 while C5 was being deployed, which required us to carefully plan our testing in order to make progress towards system acceptance of C5 without disrupting user workloads.

Following ORNL's well-defined standard process for system acceptance, the C5 acceptance test plan includes four components: vendor, functionality, performance, and stability testing.

In addition to results from acceptance testing of C5, we include in this work benchmarking results and observations made while comparing the Intel Classic and Intel oneAPI programming environments using NOAA benchmarks and Oak Ridge Leadership Computing Facility (OLCF) applications. The lessons learned shared in this work can be useful to other centers interested in procuring similar systems, running applications used in this work, or those interested in transitioning to use the latest toolchains.

In this paper, we first describe the process used to install and deploy the HPE/Cray EX supercomputer, C5. Then, we discuss the challenges we encountered during execution of the acceptance test plan, which include balancing of: (1) production workloads running in parallel, (2) the mixture and distribution of tests executed on C5 against F2 simultaneously, (3) compute and file system resources available, and (4) the schedule and resource constraints. Part of the work done to overcome these challenges included expanding monitoring capabilities in the OLCF Test Harness which are briefly described here. Finally, we present benchmarking results from NOAA benchmarks and OLCF applications that were used in this study that could be useful for other centers deploying similar systems.

II. System Architecture

C5 was deployed in two separate phases as described in Table I. The first phase included 1,728 compute nodes that were installed in August 2022. Phase I acceptance testing included all components of the acceptance test, however, had a shorter stability test. Phase II expanded the system to 1,920 compute nodes in February of 2023 and its acceptance test included the full set of acceptance test components.

A. C5 compute system

The C5 compute system is comprised of 1,920 compute nodes which were delivered in two phases as described in Table I. Each node is comprised of two 64-core AMD EPYC 7H12 processors and has 256GB of DRAM memory.

C5 is configured with HPE/Cray's Slingshot 10 which consists of Mellanox ConnectX-5 interconnect with HPE/Cray Slingshot switches. Each compute node is comprised of two Mellanox ConnectX-5 network interface cards (NICs) to provide similar aggregate bandwidth (200GB/s) as a HPE/Cray Cassini NIC can provide. C5 shares a similar node architecture as that described in [12], however, there are key differences and improvements in the system software and configuration that are described in this work.

B. F2 file system

The F2 file system is a 37.5PB Lustre-based system consisting of 6 DDN SFA14KXs, each providing 12x 489TB declustered RAID targets. F2 has 36 object storage servers (OSSs), utilizing Dell R640s with 192GB of RAM, 2x Xeon 6126 CPUs, and Mellanox ConnectX-4 InfiniBand NICs. F2's metadata layer uses Lustre's distributed namespace phase 1 (DNE1), configured to use 4 metadata targets (MDTs). The F2 MDTs are provided by a single NetApp EF570; each MDT is exported to a dedicated metadata server (MDS), configured as a Dell R640 with 384GB of RAM, a Xeon 6146 CPU, and a Mellanox ConnectX-4 InfiniBand NIC. F2 uses ZFS backed Lustre and *lz4* compression.

The F2 file system serves several production systems in the NCRC including: C3, C4, and C5.

C3 is connected to F2 via 12 LNet routers and C4 is connected to F2 via 8 LNet routers, both with Mellanox ConnectX-4 InfiniBand interfaces. C5 is connected to F2 via 30 LNet routers configured as HPE ProLiant Gen10 servers with 256GB of RAM, an AMD EPYC 7302 CPU, a Mellanox ConnectX-6 InfiniBand interface to F2, and a Mellanox ConnectX-5 Ethernet interface to C5.

III. C5 Acceptance Test Components

The C5 acceptance test was divided in four different components that have been traditionally used by the Oak Ridge Leadership Computing Facility to evaluate the functionality, performance, and stability of a new system [11], [12]. These components include a vendor test (VT), a functionality test (FT), a performance test (PT), and a stability test (ST). For C5, each component was customized to evaluate the characteristics of the system and prepare for the unique workloads planned by the NCRC.

One advantage of utilizing a similar acceptance test procedure for each new system deployed is that we are able to leverage lessons learned from distinct deployments. For C5 acceptance testing, given that the Air Force Weather systems are very similar in architecture, we were able to leverage previous work including: updates to the OLCF Test Harness that enabled Slurm support, and the design and execution of functionality tests.

We describe the contents of each acceptance test component for C5 in more detail in this section.

A. Vendor Test (VT)

The Vendor Test (VT) includes execution of diagnostics on all systems and subsystems. This component allows the vendor to assess the health of the hardware and demonstrate whether the system is ready for the next acceptance test component.

The tests in VT ensure that the installed hardware can complete power-on self-test (POST) successfully and is ready to be configured and localized by NCCS staff. The diagnostics include but are not limited to *High-Performance Linpack (HPL)*, *stream (CPU)* memory benchmark, and vendor-provided and vendor-executed network health checks.

In addition, during this component, the vendor executes the contractual applications on the system after they have been optimized for the architecture. For C5, VT included the five NCRC applications.

After the vendor has completed these tests, they provide a detailed report on results and issues identified during VT to the NCCS team.

B. Functionality Test (FT)

The Functionality Test (FT) component includes a broad set of functionality tests that cover: system administration, reliability and serviceability, network health, programming environment, and usability of the system.

During functionality testing, the individual components and characteristics of the system are evaluated using synthetic tests and well-known benchmarks.

In this component, we also ensure that all packages and libraries required by NCRC can be successfully built for the corresponding toolchains.

C. Performance Test (PT)

The Performance Test (PT) component ensures that the system provides the performance required in the contract. This component is usually uniquely tailored to each procurement. During PT, we verify that each contractual application can be executed on C5 successfully in isolation and can meet the expected performance targets. In addition, for C5, we executed multiple copies of each NOAA application to understand the performance impact when the system is fully loaded.

For C5 acceptance testing, the following NOAA applications were utilized:

Phase	Number of Nodes	Delivery	Installed	Total System Size
Phase I	1,792	July 2022	August 2022	1,792
Phase II	128	October 2022	February 2023	1,920

TABLE I
C5 Deployment Phases

- CM4: Coupled Model (CM4) is a coupled atmosphere-ocean general circulation model that includes atmosphere, ocean, sea ice, and land models.
- ESM4: Earth System Model (ESM4) includes atmosphere, ocean, sea ice, land, and ocean biogeochemical components as well as dust/iron cycling.
- SHiELD: System for High-resolution modeling for Earth-to-Local Domains (SHiELD) focuses on modeling for weather and subseasonal-to-seasonal forecasting.
- Spear: next-generation modeling system for seasonal to multidecadal prediction and projection.
- UFS: next-generation modeling system for weather prediction.

For this phase, we worked closely with HPE and NOAA staff to augment the NCRC test suite for the OLCF Test Harness to include the five contractual applications. While CM4 had been used previously for acceptance of C4, the remaining codes were new to NCCS staff.

D. Stability Test

The Stability Test (ST) component involves executing the expected workloads on the system for a predetermined number of days to measure the reliability, stability, and usability of the new system. During this component, we make heavy use of the OLCF Test Harness to maintain the system being deployed fully utilized and monitor status individual jobs. To successfully complete ST, all jobs on the system that complete must produce correct answers and perform within a predetermined threshold of expected runtime variability.

The same collection of benchmarks as was used in PT was then used during ST to generate a realistic workload continuously for a fixed number of days.

IV. C5 Acceptance Test Results

In this section, we provide a high level overview of the results obtained during acceptance testing of the C5 system.

A. Hardware and System Administration

The ORNL acceptance team decided to pare down some of the typical System Administration tests because many of the same tests were performed on an almost identical system previous to this system's acceptance. All of these tests passed as expected.

However, when it came time do the core of the functionality and stability tests, we encountered a number of issues with Lustre and the Slingshot fabric. One of the Lustre issues we encountered was that C5 had a significantly shorter LND timeout than F2 and the LNet routers at 50 seconds while F2 and the LNet routers were set to 120 seconds. This caused issues on boot where a subset of C5's compute nodes would timeout as LNet and F2 serviced all the requests. Once all the timeouts were set to 120 seconds most of C5 would consistently mount Lustre. A specific timeout formula must be set in order avoid some timeouts exceed others in the timeout hierarchy. An LND timeout >LNet transaction timeout >ptlrpc (Lustre) timeout. Also, the `at_min` timeout of 65 seconds was also set on the Lustre servers.

After these changes, some nodes were still having issues mounting Lustre and we would experience random slowness of connected nodes. These issues were tracked down to a mismatch between the setting `avoid_asym_router_failure` on F2 being set to 0 while everywhere else it was set to 1 while two of the LNet routers had asymmetric NIC failures. This caused a random set of nodes to fail to mount Lustre and was the cause of the intermittent performance issues observed as F2 would attempt to respond via the degraded paths. For the purposes of performing acceptance testing the two routers were removed from F2's routes and eventually `avoid_asym_router_failure` was disabled on C5 as F2 could not enable it due to other compute systems utilizing it.

C5's clients also needed these specific set of tuning parameters to fully address the issues described above: `peer_ credits` were reduced from 63 to 16, and `conns_per_peer` were reduced from 8 to 1. The `conns_ per_peer` setting was specifically needed due to workload issues causing locking on F2.

B. Network Testing

To test the Slingshot 10 network, we utilize several benchmarks that stress and measure the performance of the network under a variety of communication patterns and congestion scenarios. Our initial testing begins using MPIGraph [14] which exercises every endpoint in the network to every other endpoint over a series of tests. MPIGraph uses asynchronous point-to-point sends and receives using non-overlapping ring communication patterns. Initial testing on C5 using 1,800 nodes with 4MB messages demonstrates an average performance of approximately 8.6 GB/s/NIC with a maximum performance of approximately 11.6 GB/s/NIC. This is the expected performance on a Slingshot 10 network with 100 gbps Ethernet NICs.

GPCNet [8] is a tool for assessing the network's ability to perform under adversarial congestion scenarios. Using the GPCNet `network_load_test`, 80% of the nodes in the allocation will perform all-to-all congesting collectives repeatedly. The remaining nodes will perform a series of tests measuring the bandwidth and latency of several operations. GPCNet provides both uncongested and congested results for

TABLE II
GPCNet Congestion Performance Results

Name	Isolated Test	Congestion Test
RR Two-sided lat	2.6	2.8
RR Two-sided BW+Sync	1688	1693
Multiple Allreduce	42.3	43

comparison. While testing on C5, we ran with 10 processes per node across 1,800 nodes. The results, shown in Table II, show that the Slingshot 10 network in C5 handles adversarial congestion scenarios well. In particular, latency sensitive tests show virtually no impact when the all-to-all workload is running within the system. This is to be expected on a Slingshot 10 network where each NIC injection is 100 gbps but switch-to-switch links operate at 200 gbps.

During Phase II of the deployment, when the additional compute nodes and switches were added to the network, we had a few problems with the system's fabric.

Slingshot 1.7.3 was the version of the fabric manager that was used throughout the acceptance of C5. While that version was very stable, it lacked any useful diagnostics for problems in the network. The standard tools of `fmn_status` along with custom network debug scripts did not find any problems in the network either. The first issue we discovered was having two Slingshot groups disconnected from one another. This problem was due to a bug in the point-to-point file creation from the `slingshot-topology-tool`. It was incorrectly combining two Slingshot groups together and causing no routing to be sent to one of the Slingshot groups in the system. This bug does appear to be fixed in the Slingshot 2.x release.

The second major issue that was not fully understood was a fabric routing issue identified above. It manifested in several different ways with applications, e.g., job timeouts, Lustre timeouts, application failures. Due to these issues, we decided to revert back to the fabric configuration that was in place prior to the Phase II expansion of nodes and switches. Before that was done, however, the fabric was reset one last time, which seemed to fix the routing issue. The team can only speculate as to why this final fabric reset fixed the issues observed, but since then, similar timeouts have not been observed in application jobs. Resetting the fabric has been previously used to fix a variety of problems with the Slingshot network when a specific root cause for an issue cannot be identified.

C. Programming Environment Testing

C5 utilizes the full HPE/Cray Programming Environment (PE) which includes separate sets of programming environments for three different toolchains: Intel (default), GNU Compiler Collection (GCC), and the Cray Compiler Environment (CCE). Unlike previous systems, C5 was the first system deployed by NCCS that offered two flavors of the Intel programming environment: Intel Classic and Intel oneAPI. The `intel-classic` modulefile enables use of Intel's traditional compilers: `icc`, `icpc`, and `ifort`. In contrast, the `intel-oneapi` modulefile enables use of Intel's oneAPI compilers that provide: `icx`, `icpx`, and `ifx` which support both CPU and GPU architectures.

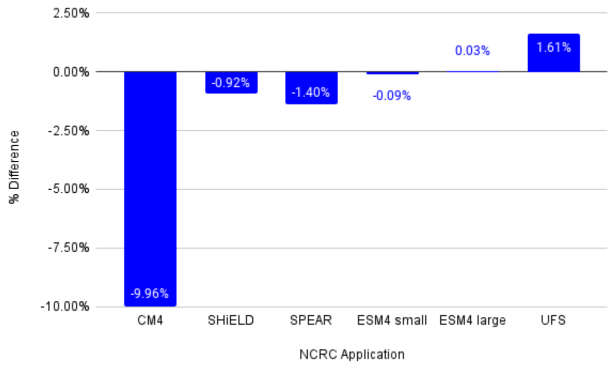


Fig. 1. Performance results obtained during the PT component of C5's acceptance test (positive % is better).

Due to the fact that NOAA relies heavily on the Intel Classic compiler, it was set as default on C5 and was the primary toolchain exercised during acceptance testing.

The NCRC contractual benchmarks were compiled using the `intel-classic/2022.2.1` compilers provided by the HPE/Cray PE. In addition, we attempted to build these benchmarks using the `intel-oneapi` toolchain and found incompatibilities with the stack, likely due to newer standards implementations. The codes will require updates by the NCRC developers as the end-of-life approaches for the `intel-classic` compilers.

D. Performance test results

As part of acceptance testing, we executed the NCRC benchmarks individually to evaluate the performance capabilities of the system. This set of tests are executed in two different modes. First, we run a single copy of the benchmark in isolation on the system. Then, we run many copies of the same benchmark to fill the entire system and compare the difference in performance observed between the two scenarios.

Due to the challenges described in Section VI, the second scenario had to be modified from the initial intended full-system scale to prevent potential impacts to production workloads on C3 and C4 running against the F2 file system.

The results from the performance test demonstrate the system is capable of running a realistic number of simultaneously executed copies for each of the NCRC benchmarks. The performance differences observed between the isolated single-copy tests reported by HPE during VT and our reproduced results from PT are summarized in Figure 1.

E. Stability test results

As part of stability testing (ST), we executed a combination of all NCRC benchmarks distributed across different metadata targets (MDTs). We used the OLCF Test Harness to launch four independent sets of tests that included the benchmarks and job sizes as shown in Table III. The Phase II ST ran for

TABLE III
STABILITY BENCHMARK TESTS

Test	Type	Job Size (nodes)
CM4	NOAA	48
ESM4 (small)	NOAA	18
ESM4 (large)	NOAA	26
Spear	NOAA	22
SHiELD	NOAA	96
UFS	NOAA	24

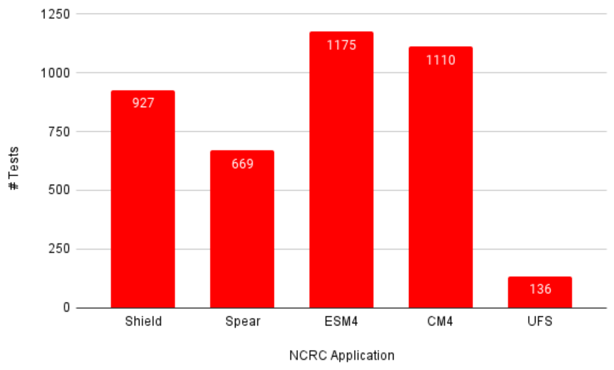


Fig. 2. Number of jobs executed per NCRC application during the 10-day ST component.

10 continuous days and in total executed 4,017 jobs across all NCRC applications as shown in Figure **??**. The breakdown of failures by application is shown in Figure 3

ST jobs were also analyzed for runtime variation. Figures 4, 5, 6, 7, 8 and 9 show the elapsed runtime of each job as a function of time, by test. These results are summarized in Table IV. Jobs observed at the very top of the plot are jobs that ran until walltime. Jobs falling below the cluster of average points suffered from file system errors, MPI errors, hardware failures, etc.

Figure 4 shows the elapsed time of each CM4 job during ST. There are several points around *x=120* and *x=160* hours that are nearly double the expected run time. These jobs are classified as performance failures.

Figures 5 and 6 show the elapsed time of each ESM4 small and large job during ST.

Figure 7 shows the elapsed time of each SHiELD job run during ST.

Figure 8 shows the elapsed time of each Spear job during ST. The highest data points (approx. 14,500 seconds) correspond to jobs that ran until walltime. Jobs far below the average elapsed time likely failed with errors ranging from file

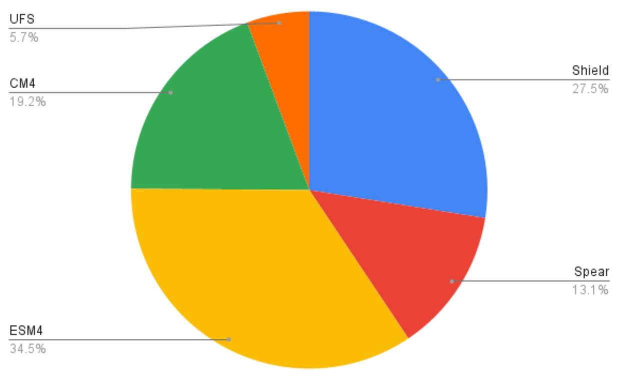


Fig. 3. Percentage of job failures by application observed during the 10-day ST component.

TABLE IV
STABILITY BENCHMARK RUNTIME VARIABILITY

Test	Avg Elapsed (s)	Standard Deviation	# Jobs
CM4	4148.1	11.3	1050
ESM4 (small)	6016.8	8.0	416
ESM4 (large)	4565.7	8.8	660
Spear	5808.1	13.2	647
SHiELD	4201.2	7.2	861
UFS	6049.5	4.0	123

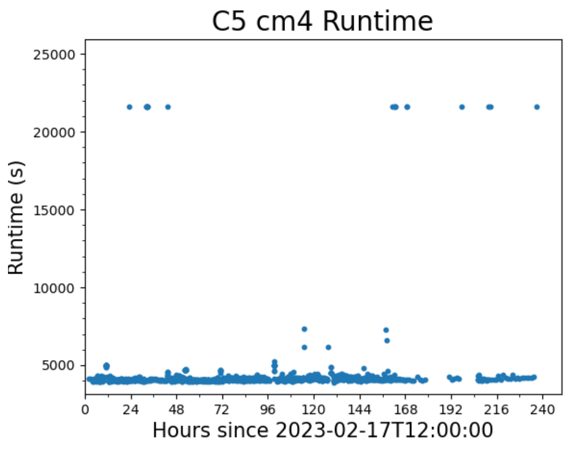


Fig. 4. The elapsed time of each CM4 job as a function of the start time of the job.

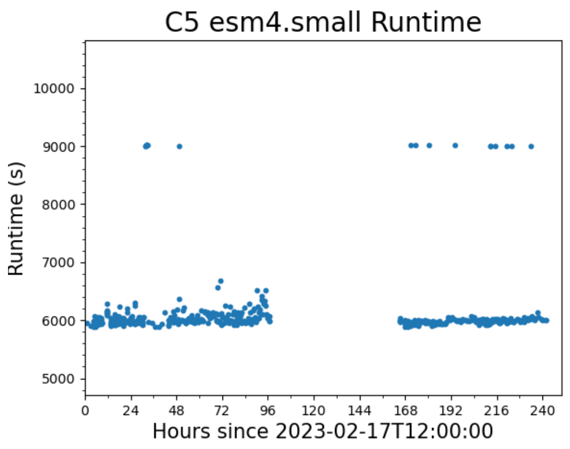


Fig. 5. The elapsed time of each ESM4-small job as a function of the start time of the job.

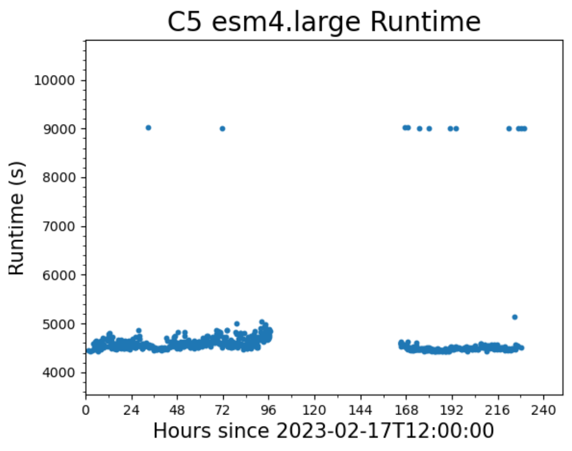


Fig. 6. The elapsed time of each ESM4-large job as a function of the start time of the job.

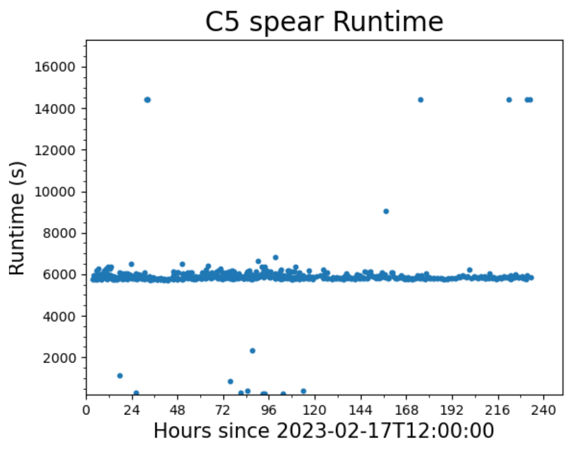


Fig. 8. The elapsed time of each Spear job as a function of the start time of the job.

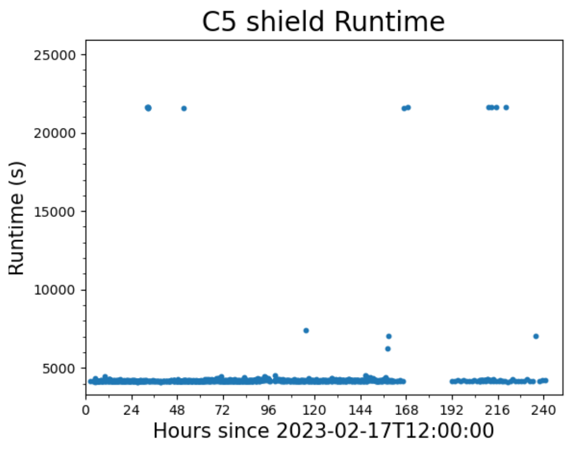


Fig. 7. The elapsed time of each SHiELD job as a function of the start time of the job.

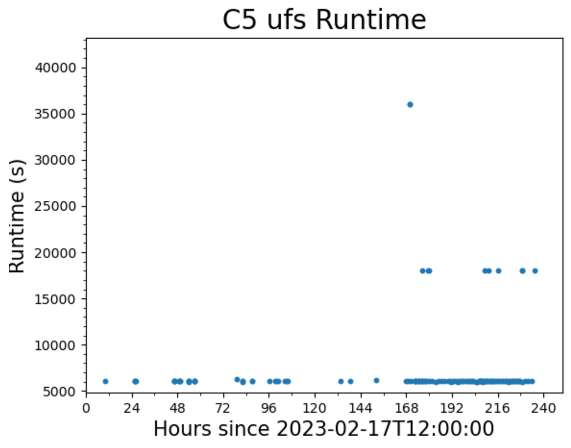


Fig. 9. The elapsed time of each UFS job as a function of the start time of the job.

system hiccups to node failures.

Figure 9 shows the elapsed time for each UFS job. At the beginning of ST, UFS was causing jobs to walltime while building the next job, using the OTH's resubmit capability. This issue caused the sparse data at the beginning of ST, but has since been resolved.

For all five NCRC applications, we observed a non-trivial number of outliers that occurred during ST. Given that F2 was not dedicated exclusively to C5 acceptance efforts, it is, unfortunately, difficult to isolate the root cause for many of these performance failures and timeouts. It is worth noting, however, that since the fabric was reset as described in IV-B, we have not seen similar issues.

V. OLCF Test Harness Updates

We used our in-house developed OLCF Test Harness system (OTH) [15], [17] to orchestrate launching tests throughout acceptance. Previously, we would monitor job output and harness status output through standard UNIX text and file manipulation commands. For this acceptance test, we extended the OTH to output harness events (e.g., "build start", "job end") and status codes to an InfluxDB database. We then built a series of dashboards in Grafana to organize and display these data in real-time. These dashboards allowed us to more quickly and easily monitor testing progress as jobs were running as well as filter OTH jobs from other workloads running on the system.

The OTH Monitor lives in a local deployment of OpenShift. The architecture is fairly straightforward and consists of several pods:

1) InfluxDB: is the central collection location of all the test data points and is backed by a persistent volume claim.
2) Grafana: holds the Grafana instance for displaying data.
3) MariaDB: holds static system information queried by Grafana to build certain dashboards.

We maintain development and production versions of each of these to facilitate testing and development as needed.

OTH tests log events and results to the InfluxDB database in real-time. In the event of hardware or communication faults, a Python script ran hourly to find any OTH events that were not posted to InfluxDB. Common failure modes have been assigned unique status codes to enhance failure visualization on Grafana dashboards.

We constructed and used several dashboards throughout acceptance. These dashboards were developed using the Flux query language [2]. The first dashboard shows the most recent failed tests in chronological order. This dashboard provides the name of the test, the scheduler job identifier, the status code, and the path to the launch directory on the file system. An example screenshot of this dashboard is shown in Figure 10. The user can fully customize the fields to be displayed using a drop-down list of check boxes. A small modification to this dashboard yields one that shows all jobs, listed in chronological order. Another dashboard computes the number of successful, currently running, currently building, and failed tests within a time window. An example of this dashboard is shown in Figure 11.

VI. C5 Acceptance Challenges

In this section, we highlight a few of the challenges that we encountered as well as discuss some of the workarounds implemented.

A. *Concurrent application execution on F2*

The first major hurdle that we faced during system acceptance testing of C5 happened while executing PT using NOAA-provided applications. The acceptance test plan required C5 to be able to compute a well-defined volume of work as part of the contract. To meet this requirement, we needed to load the system with as many copies of each individual benchmark as could fit on the system. However, due to the heavy I/O load that each benchmark generated on F2, in order to avoid impacting production workloads, we had to find a balance between the number of benchmarks actively running on C5 and the load observed on F2 from all NCRC systems.

During initial attempts to launch the PT and ST, we observed DNE1 lock contention. The NCRC test suite running on C5 ended up exacerbating an existing user workload issue to the point where jobs were failing to run in a timely manner. The OTH would launch multiple jobs using the same input deck and thus the metadata server (MDS) servicing that load. We later determined that the core issue was due to the fact that jobs were structured to do millions of file opens and hold them open for much of the job, a single job causes some lock contention and multiple jobs attempted to juggle locking for the same set of files. At the time of acceptance testing, this was not understood but the issue manifested as extensive locking.

After careful tuning and monitoring of F2, we decided to both spread the workloads running on C5 across different metadata targets (MDTs) in order to avoid overloading a single one. We also chose to split acceptance into smaller sub-phases that could tolerate interruptions and would still help us identify issues.

The original workaround for this issue causing Lustre to crash was to spread out the start of jobs and eventually creating a set of data per MDS. Since then, the user workload was modified to create a copy of the input deck per run which has effectively solved this in production.

B. *Impact to NCRC production workloads*

Due to the nature of NOAA's workloads, the load on the queue on C3 and C4 can vary greatly between weeks. During times when critical work was being executed, we had to pivot our testing to use benchmarks that would introduce minimal I/O load on the file system but would still exercise the new hardware sufficiently. To accomplish this, in addition to the five NOAA codes, we augmented the NCRC test suite with three applications that are used regularly for testing on the OLCF's flagship systems, Summit and Frontier, namely minisweep, GenASiS, and LSMS. These three applications were chosen not only because they were developed at ORNL, but also because they have helped us identify system issues during past deployments.

C. *New compiler toolchains*

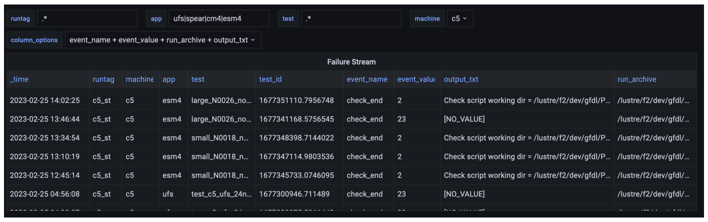
The NCRC multicluster environment has the Intel compiler as the default toolchain used by NOAA users. Initially, we intended to run all acceptance test applications using the latest available compiler toolchains which included the relatively new Intel oneAPI compilers. However, due to issues identified building the NOAA applications with Intel oneAPI, we decided to revert to the Intel Classic compiler. This choice was made to provide a smoother transition to operations for NOAA users and give the NCCS team sufficient time to evaluate the Intel oneAPI toolchain.

D. *High-utilization with smaller-sized jobs*

During both the PT and ST components, the acceptance test plan required us to fill the system with NCRC applications. Because all the NCRC application jobs provided used fewer than 100 nodes, this meant we needed to maintain at a minimum approximately 20 applications running simultaneously. Although this normally would not be an issue, because acceptance is run using the same bot user, the directory on F2 where the tests are running is tied to same MDT. As a result, due to the F2 issues described in Section VI-A, running with the maximum number of copies resulted in us overloading that single MDT which caused widespread file system issues on the NCRC environment. As a result of these obstacles, we artificially capped the maximum number of copies run for each application.

E. *Long build times for NCRC applications*

The build time for each NCRC application can be fairly lengthy ranging from tens of minutes to over an hour. This presented a challenge as the OLCF Test Harness requires that each application submitted during ST as a batch job is built immediately prior to submission. The lengthy builds resulted in fewer jobs executed simultaneously which impacted



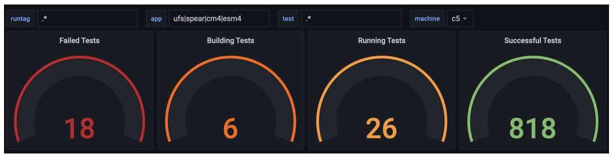
runtag .* app ufs|spear|cm4|esm4 test .* machine c5

column_options event_name + event_value + run_archive + output_txt

Failure Stream

_time	runtag	machine	app	test	test_id	event_name	event_value	output_txt	run_archive
2023-02-25 14:02:25	c5_st	c5	esm4	large_N0026_no...	1677351110.7956748	check_end	2	Check script working dir = /lustre/f2/dev/gfdl/P...	/lustre/f2/dev/gfdl/...
2023-02-25 13:46:44	c5_st	c5	esm4	large_N0026_no...	1677341168.5756545	check_end	23	[NO_VALUE]	/lustre/f2/dev/gfdl/...
2023-02-25 13:34:54	c5_st	c5	esm4	small_N0018_n...	1677348398.7144022	check_end	2	Check script working dir = /lustre/f2/dev/gfdl/P...	/lustre/f2/dev/gfdl/...
2023-02-25 13:10:19	c5_st	c5	esm4	small_N0018_n...	1677347114.9803536	check_end	2	Check script working dir = /lustre/f2/dev/gfdl/P...	/lustre/f2/dev/gfdl/...
2023-02-25 12:45:14	c5_st	c5	esm4	small_N0018_n...	1677345733.0746095	check_end	2	Check script working dir = /lustre/f2/dev/gfdl/P...	/lustre/f2/dev/gfdl/...
2023-02-25 04:56:08	c5_st	c5	ufs	test_c5_ufs_24n...	1677300946.711489	check_end	23	[NO_VALUE]	/lustre/f2/dev/gfdl/...

Fig. 10. Grafana failure stream dashboard for the OTH.



runtag .* app ufs|spear|cm4|esm4 test .* machine c5

Failed Tests	Building Tests	Running Tests	Successful Tests
18	6	26	818

Fig. 11. Grafana summary dashboard for the OTH.

overall utilization. To overcome this challenge, we designed variations of each application that would repeatedly submit a precompiled binary instead.

VII. Scaling study of OLCF applications

As described in Section III, the acceptance test plan already includes an exhaustive set of tests that are executed to understand the readiness of a system for production workloads including the specific contractual benchmarks. However, to evaluate a system as broadly as possible, it is often preferable to work with applications that are well understood internally at the OLCF. To that end, in this work we include scaling studies conducted on C5 for minisweep, GenASiS, LAMMPS, and LSMS leveraging the different compiler toolchains available on C5: Intel (both `intel-classic` and `intel-oneapi`), CCE, and GNU.

A. *minisweep*

Minisweep is a C++ open-source mini-application that captures the compute intensive portion of the Sn radiation transport code Denovo [13]. Minisweep has been used previously to evaluate performance of new architectures as well as functionality of compiler toolchains [16]. Minisweep supports MPI, OpenMP (CPU), CUDA, OpenACC, OpenMP (GPU offload), and more recently HIP. In this work, we focus on the MPI-only version to understand its scalability on C5. Figure 12 shows a the execution time of a simulation with 64^3 cells and an increasing number of ranks. For all cases 4 MPI ranks are used per node. Results show that above 512-ranks, the performance begins to decrease. Further investigation would be needed to better understand the sharp drop in performance observed at a relatively modest job size.

B. GenASiS

GenASiS (*Gen*eral *A*strophysics *Si*mulation *S*ystem) is a multiphysics simulation framework aimed at performing the simulations of astrophysical phenomena [7]. Written in modern Fortran, it is an extensible, modular code by exploiting the object-oriented features of the language standard while

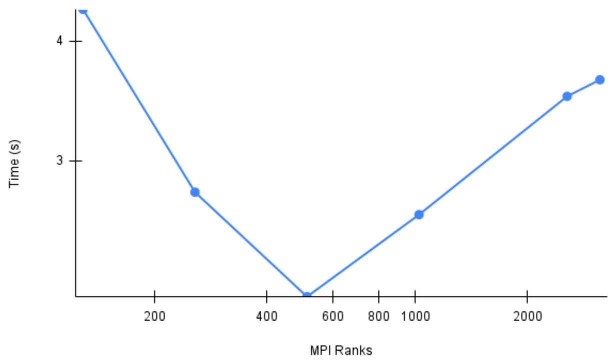


Fig. 12. Minisweep multi-node scaling on C5 for a system of size 64^3 cells.

maintaining high-performant computational kernels. GenASiS uses OpenMP for multithreading on CPU and to offload its computational kernels to accelerators such as GPUs [4]. For this work, we use the fluid dynamic benchmark problem `RiemannProblem` as implemented by GenASiS Basics, a subdivision of GenASiS [5]. Since C5 does not have accelerators, we build GenASiS with OpenMP CPU multithreading only. Figure 13 plots the strong-scaling of `RiemannProblem` in GenASiS Basics as we increase the number of OpenMP threads. With all four compilers available on the system (CCE, GCC, Intel Classic, and Intel oneAPI), the scalability suffers beyond four threads. This is in contrast to previous results on different systems where we have demonstrated strong-scaling with up to 12 OpenMP threads (for example, Figure 8 in [6]). At this point we continue to investigate the cause of this poor scalability. One finding identified on C5 when using the Intel oneAPI Fortran compiler (IFX) version 2022.2.1 20221020 was an internal compiler error triggered by the `-fast` flag. As a workaround, the results presented here used `-O2` instead.

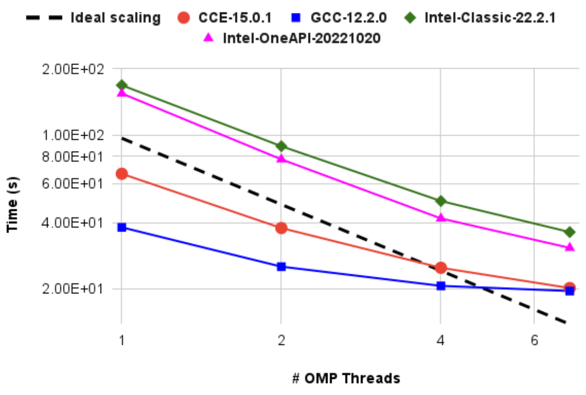


Fig. 13. Strong-scaling of GENASISBASICS `RiemannProblem` with 64^3 cells for 100 timesteps. The dashed black line shows ideal scaling as a reference. Scaling with GCC, CCE, Intel-Classic, and Intel-oneAPI compilers are plotted as blue squares, red circles, green diamonds, and magenta triangles, respectively.

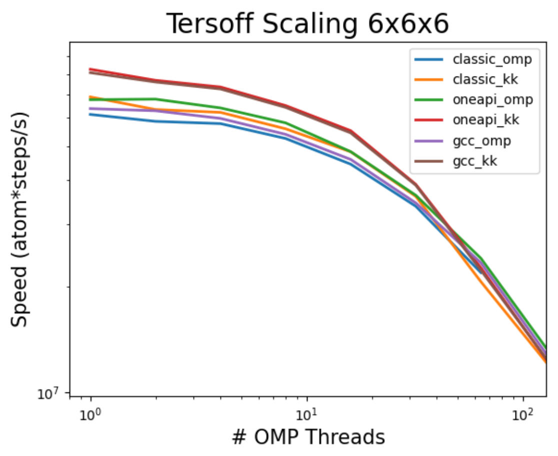


Fig. 14. Achieved speed of a 6.9 million atom Tersoff simulation as a function of the number of OpenMP threads per MPI rank.

C. LAMMPS

The molecular dynamics software, LAMMPS [3], was used to test the MPI+OpenMP programming model. The OpenMP and Kokkos packages were used to utilize OpenMP multi-threading. The Intel Classic, Intel oneAPI, and GCC compiler toolchains were tested using the Tersoff and ReaxFF benchmarks provided in LAMMPS. A "replicate" command was placed after system initialization in each of the input files to allow for easily scaling the system size.

In this work, we examine the performance impact of the balance of OpenMP threads to MPI ranks. In all experiments, the total number of threads (the product of the number of MPI ranks and OpenMP threads) is 128, which fully consumes the 2 CPUs on each compute node, with hyper-threading disabled. The Slurm *cpu-bind=mask_cpu* flag was utilized to provide an explicit set of sequential cores for OpenMP threading to each MPI task. Experiments were performed using GCC 12.2.0, Intel oneAPI 2022.0.2, and Intel Classic 2021.5.0 compilers. In each figure legend, the *omp* suffix is the OpenMP package of LAMMPS, while the *kk* suffix is the Kokkos package of LAMMPS, compiled using the OpenMP backend of Kokkos.

Figures 14, 15, and 16 show the scaling behavior of 6.9-million atom, 16.3-million atom, and 32-million atom replicates of the LAMMPS Tersoff benchmark, respectively. All compiler toolchains demonstrated similar scaling behavior on all 3 system sizes. At low thread counts using the Kokkos package, the GCC and Intel oneAPI compilers achieve about 10% better performance than Intel Classic. but by 64 OpenMP threads, this performance gap is closed in the smallest and largest systems. The middle size, the 16.3-million atom system, displayed a sharp drop in speedup above 8 OpenMP threads when using the Kokkos package for all 3 compilers. This behavior is under further investigation. Run-time errors were encountered near the 1 MPI rank / 128 OpenMP thread data point. Run-time errors were typically associated with memory allocation or integer overflow errors.

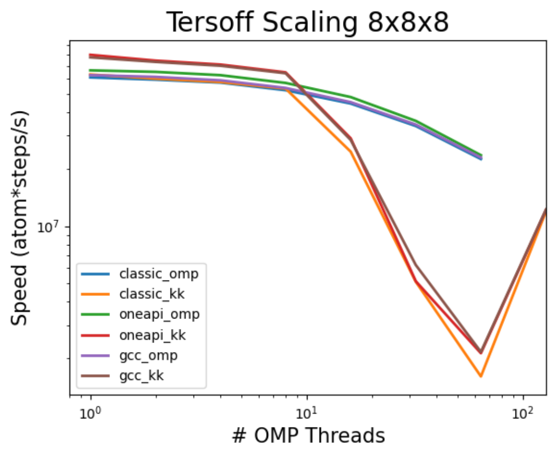


Fig. 15. Achieved speed of a 16.3 million atom Tersoff simulation as a function of the number of OpenMP threads per MPI rank.

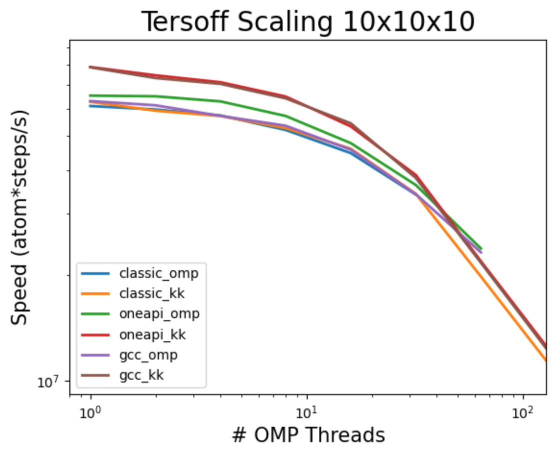


Fig. 16. Achieved speed of a 32 million atom Tersoff simulation as a function of the number of OpenMP threads per MPI rank.

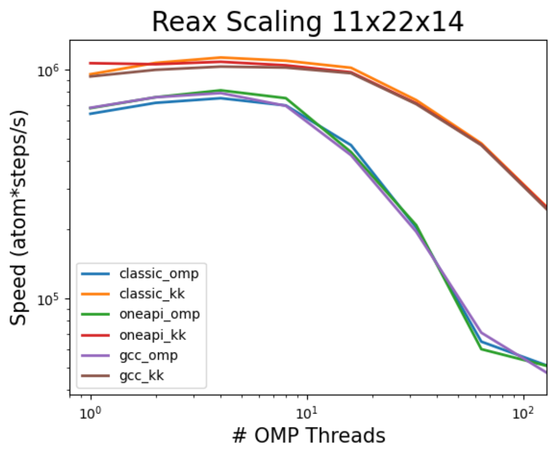


Fig. 17. Achieved speed of a 196 thousand atom ReaxFF simulation as a function of the number of OpenMP threads per MPI rank.

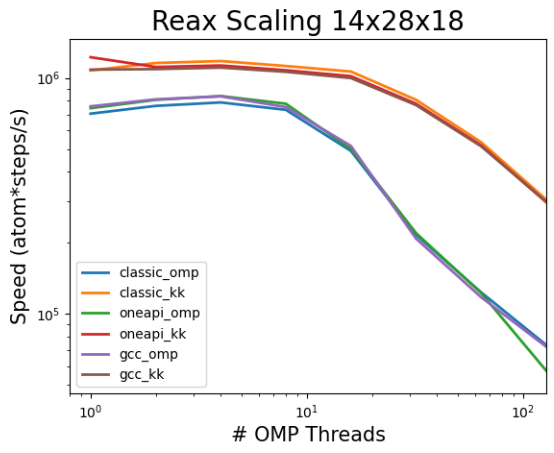


Fig. 18. Achieved speed of a 409 thousand atom ReaxFF simulation as a function of the number of OpenMP threads per MPI rank.

Figures 17 and 18 show the scaling behavior of 196-thousand atom and 409-thousand atom replicates of the LAMMPS ReaxFF benchmark, respectively. All compiler toolchains demonstrated similar scaling behavior on both system sizes. In contrast to the Tersoff benchmark, all 3 compilers achieve similar performance when using the Kokkos package at all OpenMP thread counts. The Kokkos package achieves better performance than the OpenMP package at all thread counts for all compiler toolchains. One contributing factor may be the active development on the Kokkos package of the ReaxFF potential of LAMMPS. The improved performance demonstrated using the ReaxFF benchmark with the Kokkos acceleration package between the September 2021 and February 2022 versions of LAMMPS has been shown in [10]. The same optimizations used to achieve this improved performance likely benefit the OpenMP performance as well, since the specific backend is abstracted away by Kokkos.

Figures 19 and 20 show the achieved performance of the ReaxFF potential when using the Kokkos package from the

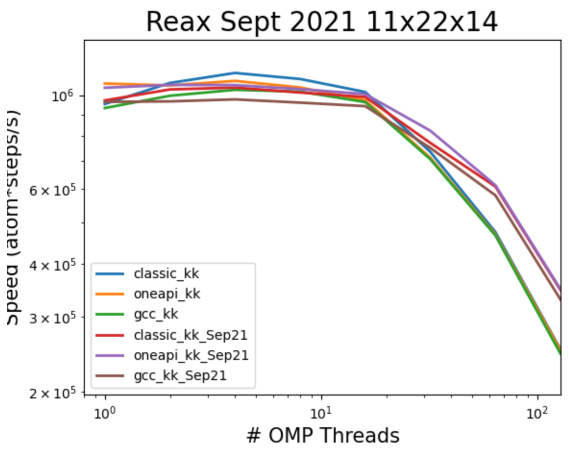


Fig. 19. Achieved speed of a 196 thousand atom ReaxFF simulation as a function of the number of OpenMP threads per MPI rank for September 2021 release of LAMMPS.

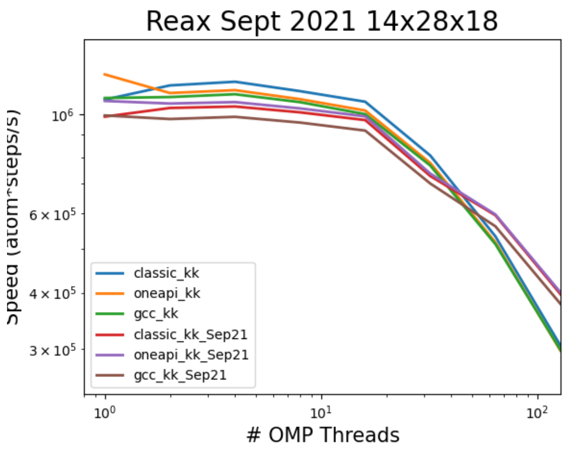


Fig. 20. Achieved speed of a 409 thousand atom ReaxFF simulation as a function of the number of OpenMP threads per MPI rank for September 2021 release of LAMMPS.

September 2021 version of LAMMPS, compared to the current version of LAMMPS. The *classic_kk*, *oneapi_kk*, and *gcc_kk* series are the same data presented in Figures 17 and 18. At the larger system size, the LAMMPS source code used in this study out-performs the September 2021 source code by about 10%. However, at 1 MPI rank / 128 OpenMP threads, the September 2021 release of LAMMPS out-performs the current source code.

D. LSMS

LSMS is an open source application developed at Oak Ridge National Laboratory [9], [18] specifically designed for scalable first principles calculations of materials. LSMS supports different architectures and has been utilized to stress various systems during acceptance testing at the OLCF including Titan, Summit, and Frontier.

In this work, we utilize the CPU-only version of LSMS with MPI and OpenMP support enabled to better understand

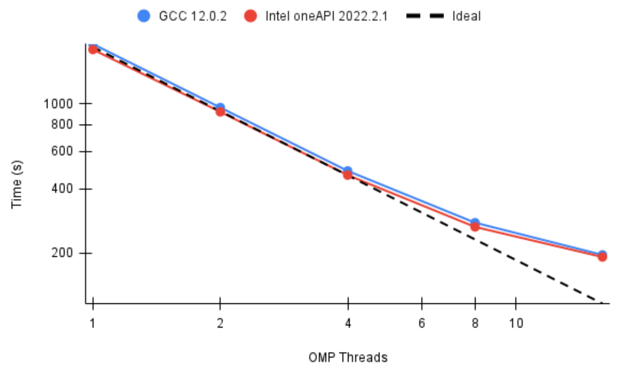


Fig. 21. Execution time of 128-atom simulation using LSMS on a single-node of C5 using 8 MPI ranks and varying number of OpenMP threads.

the scalability of the application on the C5 system. The results shown in Figure 21 use GCC 12.2.0 and Intel oneAPI 2022.2.1 and demonstrate that for both toolchains the performance is comparable and the application is able to scale fairly well up to 4 OMP threads.

VIII. Conclusions

In this work, we present an overview of the installation and deployment of the NCRC's latest supercomputer, C5. C5 is a 1,920-node HPE/Cray EX 3000 supercomputer powered by two 64-core AMD EPYC 7H12 processors per node that will be used in production to support workloads from the NCRC.

The unique circumstances surrounding C5's deployment resulted in the design of new tools and in modifications to procedures that gave us the flexibility to accommodate production workloads executing on the C3 and C4 systems also part of the NCRC multi-cluster architecture. The monitoring tools customized for C5 acceptance testing helped the NCCS team make progress even when test components had to be split across time periods or deferred to a later time.

By working closely with HPE and NOAA, NCCS developed a strategy that proved successful to address critical issues identified preventing full simultaneous utilization of C3, C4, and C5 systems for production.

Through C5's acceptance testing, we were able to identify specific gaps in the vendor-provided tools as demonstrated by the network health issues identified using GPCNet and mpiGraph that were undetected via network diagnostics. Being able to have more robust tools to diagnose issues on Slingshot-based networks will be critical to ensure a smooth transition to operations of the system.

As shown in Sections VII and IV, the C5 system is capable of supporting expected NCRC workloads executing the five primary NOAA applications in addition to a broader range of applications selected from the OLCF portfolio. The results presented here provide early experiences using the Intel oneAPI toolchain on a HPE/Cray EX 3000 supercomputer and highlight a few differences that were observed both in functionality as well as performance when comparing to the other available toolchains on the system. Although in some cases, Intel oneAPI demonstrated better performance than Intel Classic, for example with GenASiS, its performance was still below that observed with CCE and GNU. Furthermore, NOAA applications are heavily dependent on not only the Intel Classic compiler but also a specific version. Given that NOAA requires bit-wise reproducibility, upgrades to default versions of any toolchain must be considered carefully in order to evaluate the impact.

The initial scaling studies presented here highlight a couple of areas worth exploring further including the poor scaling observed with OpenMP in GenASiS and LAMMPS, as well as the limited scaling with minisweep.

We hope that the experiences shared here will be of interest and helpful to centers that have users with similar requirements and are exploring transitioning to systems with similar architectures and compiler toolchains.

Acknowledgment

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] DOE and NOAA extend strategic partnership.

[2] Influxdb and flux. https://www.influxdata.com/products/flux/.

[3] LAMMPS, 2018.

[4] Budiardja, R. D., and Cardall, C. Y. Targeting gpus with openmp directives on summit: A simple and effective fortran experience. *Parallel Computing 88* (2019), 102544.

[5] Budiardja, R. D., and Cardall, C. Y. Genasis basics: Object-oriented utilitarian functionality for large-scale physics simulations (version 4). *Computer Physics Communications 281* (2022), 108505.

[6] Cardall, C. Y., and Budiardja, R. D. Genasis mathematics : Object-oriented manifolds, operations, and solvers for large-scale physics simulations. *Computer Physics Communications 222* (2018), 384–412.

[7] Cardall, C. Y., Budiardja, R. D., Endeve, E., and Mezzacappa, A. GENASIS: GENERAL ASTROPHYSICAL SIMULATION SYSTEM. i. REFINABLE MESH AND NONRELATIVISTIC HYDRODYNAMICS. *The Astrophysical Journal Supplement Series 210*, 2 (jan 2014), 17.

[8] Chunduri, S., Groves, T., Mendygral, P., Austin, B., Balma, J., Kandalla, K., Kumaran, K., Lockwood, G., Parker, S., Warren, S., Wichmann, N., and Wright, N. GPCNeT: Designing a Benchmark Suite for Inducing and Measuring Contention in HPC Networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2019), SC '19, Association for Computing Machinery.

[9] Eisenbach, M., Zhou, C.-G., Nicholson, D., Brown, G., Larkin, J., and C Schulthess, T. Thermodynamics of magnetic systems from first principles: Wl-lsms. In *Proceedings of the 2010 SciDAC conference* (04 2010).

[10] Hagerty, N., Melesse Vergara, V., and Tharrington, A. Studying performance portability of LAMMPS across diverse gpu-based platforms. Tech. rep., Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2022.

[11] Larrea, V. G. V., Joubert, W., Brim, M. J., Budiardja, R. D., Maxwell, D., Ezell, M., Zimmer, C., Boehm, S., Elwasif, W., Oral, S., et al. Scaling the summit: deploying the world's fastest supercomputer. In *International Conference on High Performance Computing* (2019), Springer, pp. 330–351.

[12] Melesse Vergara, V., Budiardja, R., Peltz, P., Niles Jr, J., Zimmer, C., Dietz, D., Fuson, C., Liu, H., Newman III, P., Simmons, J., et al. A step towards the final frontier: Lessons learned from acceptance testing of the first hpe/cray ex 3000 system at ornl.

[13] Messer, O. B., D'Azevedo, E., Hill, J., Joubert, W., Berrill, M., and Zimmer, C. Miniapps derived from production hpc applications using multiple programing models. *The International Journal of High Performance Computing Applications 32*, 4 (2018), 582–593.

[14] Moody, A. Contention-free Routing for Shift-based Communication in MPI Applications on Large-scale Infiniband Clusters, 10 2009.

[15] Tharrington, A. N. Nccs regression test harness, version 00, 9 2015.

[16] Vergara Larrea, V. G., Budiardja, R. D., Gayatri, R., Daley, C., Hernandez, O., and Joubert, W. Experiences in porting mini-applications to openacc and openmp on heterogeneous systems. *Concurrency and Computation: Practice and Experience 32*, 20 (2020), e5780.

[17] Veronica G. Vergara Larrea, Michael J. Brim, A. T. R. B., and Joubert, W. Towards acceptance testing at the exascale frontier. *Cray User Group* (2020).

[18] Wang, Y., Stocks, G. M., Shelton, W. A., Nicholson, D. M. C., Szotek, Z., and Temmerman, W. M. Order-n multiple scattering approach to electronic structure calculations. *Phys. Rev. Lett. 75* (Oct 1995), 2867–2870.