Early Application Experiences on Aurora at ALCF: Moving From Petascale to Exascale Systems

Colleen Bertoni, JaeHyuk Kwack, Thomas Applencourt, Abhishek Bagusetty, Yasaman Ghadar, Brian Homerding, Christopher Knight, Ye Luo, Mathialakan Thavappiragasam, John Tramm, Esteban Rangel, Umesh Unnikrishnan, Timothy J. Williams, Scott Parker
Leadership Computing Facility
Argonne National Laboratory
Lemont, IL, USA
Email: {bertoni, jkwack, tapplencourt, abagusetty, ghadar, bhomerding, knightc, yeluo, mthavappiragasam, jtramm, erangel, unnikrishnan, zippy, sparker}@anl.gov

***Abstract*—Aurora, installed in 2023, is the newest system being prepared for production at the Argonne Leadership Computing Facility (ALCF). Throughout multiple years of preparation, the ALCF has tracked the progress of over 40 applications from the Exascale Computing Project and ALCF's Early Science Project in terms of ability to run on Aurora and performance on Aurora compared to other systems. In addition, the ALCF has been tracking bugs and issues reported by application developers. This broad tracking of applications in a standardized way as well as tracking of over 1100 bugs and issues via source code reproducers has been essential to ensuring the usability of Aurora. It has also helped ensure a smoother transition for applications that can run on past or current production systems, like Polaris, the ALCF's current production system, to Aurora. To gain insight into the current state of applications which were ported to Aurora on both Aurora and Polaris, a set of applications are compared in terms of single GPU and single node performance on Aurora and Polaris. On average the Figure-of-Merit performance for the set of applications was 1.3x greater on a single GPU of Aurora than a single GPU of Polaris. The intra-node parallel efficiency of the set of applications was similar between Aurora and Polaris.**

***Index Terms*—Polaris, Aurora**

I. Introduction

The Argonne Leadership Computing Facility (ALCF) is a United States Department of Energy computing facility at Argonne National Laboratory. In the past, the ALCF has transitioned from one large-scale system to another roughly every 4 - 5 years. In the ideal scenario, users who can run on current production systems, like Polaris at the ALCF, would be able to smoothly transition their application to new systems, like Aurora, the next largest system at the ALCF, without too much effort and get reasonable performance.

This is difficult as vendors, hardware, and software stacks often change between different systems, and application developers often have to port their code to be able to run. For Aurora this was in particular challenging for application developers since the GPUs Aurora is based on, Intel Data Center Max Series GPUs, are the first of their kind from Intel. Although Intel has been developing integrated GPUs for many years [1], this is the first time Intel has developed discrete GPUs for high performance computing (HPC) systems. In addition to new hardware being developed, Intel also was developing a new software stack (Intel oneAPI) [2] to allow application developers to effectively target the hardware. In contrast to Aurora, Polaris, the current production resource at ALCF, is based on Nvidia A100 GPUs, which is a much more common GPU for HPC systems to be based on. For example, about 50% of the top 100 computers on the November 2023 Top500 list are Nvidia GPU-based, while only three are Intel GPU-based [3]. Although many open source and portable programming models are supported on Aurora, many applications had previously ported to CUDA, which is not supported by Intel since it is Nvidia's proprietary programming model.

In addition to newer hardware and software stacks, the scale of the new systems available often increases significantly. To give a sense of the scale of Aurora compared to Polaris, Polaris ranks 27th on the Top500 list while Aurora is currently 2nd, even with only part of the system [3]. The large increase in computing resources can enable application developers to implement methods which previously were not feasible with smaller computing resources. In particular, many of the application developers targeting Aurora were part of the Exascale Computing Project (ECP) [4] and the ALCF's Early Science Program (ESP) [5] and were implementing new algorithms and methods to be able to take advantage of exascale-level computing resources and run simulations that were not previously possible. Thus ensuring a smooth transition to Aurora from other systems like Polaris was complicated since not only was the software and hardware of the pre-production Aurora systems changing (since the GPU and software stack were new and being developed by Intel) but many application developers were also implementing exascale-ready algorithms and porting their code to new programming models.

One approach by the ALCF staff to ensure the progress of application development and address issues affecting applications was to create an Aurora Applications Working Group, which was formed with application developers and ALCF and Intel staff. This group tracked application progress in terms of functionality and performance on a quarterly

basis, starting in 2021 and continuing through 2024. The application tracking included over forty applications from ECP and ESP. Additionally, the ALCF staff maintained a growing test repository with bug reproducers reported by application developers. Tracking applications helped assess the the overall status and issues for the collective set of applications, and was essential for identifying wide-spread issues. Tracking bugs via a shared repository with the vendor, Intel, allowed for a clear determination of how well a version of the Software Development Kit (SDK) from the vendor would fix known bugs (or lead to regressions) which was essential for communication about how useful a new SDK was.

In addition to discussing application and bug tracking, to gain insight into the current state of how applications are running on pre-production Aurora and production-level Polaris, this work also evaluates the performance of 11 of the tracked applications on a single-GPU and single-node basis between Aurora and Polaris. Challenges, workarounds, or limitations encountered when porting to Aurora are discussed as well.

This paper is organized as follows: Section II presents information about two systems used. Section III provides the strategy used to track progress in science applications as well as early software from the vendor as the Aurora system was being prepared. Section IV presents the computational details and setup for a set of applications run on Aurora and Polaris, and Section V compares the results on Aurora and Polaris. Section VI discusses notable modifications or tuning that helped with successfully running on Aurora, and any workarounds or limitations that were encountered. Section VII summarizes this study.

II. System Architectures

The following sections give details on the two systems used, Polaris and Aurora.

A. Polaris

Polaris is composed of 560 nodes, where each node has one AMD EYPC 7532 32C 2.4GHz CPU and four NVIDIA A100 SXM4 40 GB GPUs [6], for a total of 560 CPUs and 2,240 GPUs. Each GPU has a peak theoretical performance of 19.5 Tflop/s in double precision (DP), giving the whole system a theoretical peak of 44 Pflop/s in DP.

Each node has 512 GiB DDR4 and 1.6TB of SSD local storage. The four GPUs on a node are connected via Nvlink.

Polaris is a petascale computer which is 27th on the Top500 list with a max LINPACK performance achieved of 25.81 PFlop/s as of November 2023.

A node-level diagram of Polaris is shown in Fig. 1a.

B. Aurora

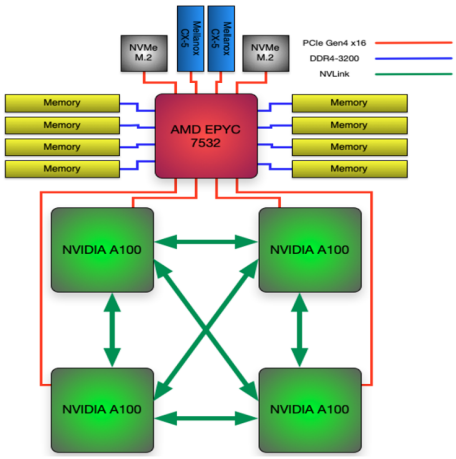
Aurora is composed of 10,624 nodes, where each node has 6 Intel Data Center Max Series GPUs [9] (also called Ponte Vecchios or PVCs) and and 2 Intel Xeon CPU Max Series CPUs with HBM.

Each CPU socket has 64 GB HBM and 512GB DDR5. Each GPU socket has 128GB HBM. The six GPUs on a node are connected via XeLink in an all-to-all topology. Each GPU has two stacks (or tiles), which can be targeted as independent GPUs.

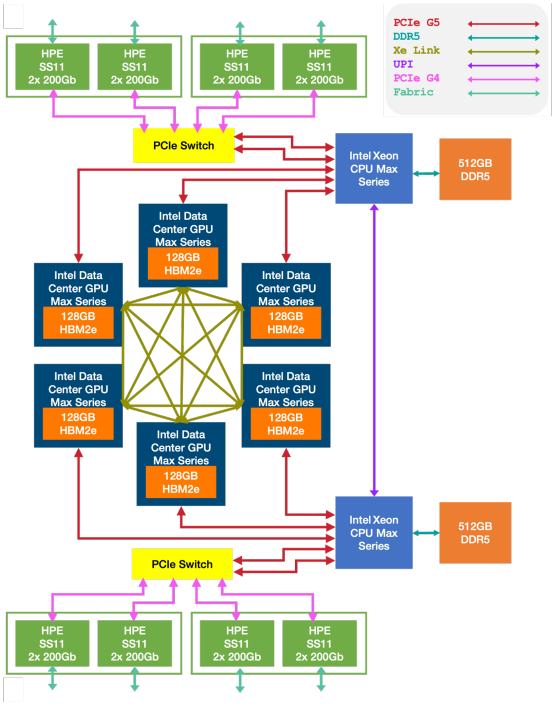
Aurora is currently 2nd on the Top500 list with a max LINPACK performance achieved of 585.34 PFlop/s. However, only part of the Aurora system was used for this Top500 entry, as it is expected to be updated with the entire system before its production.

A node-level diagram of Aurora is in Fig. 1b.

Additionally, Aurora has a Test and Development System called Sunspot which consists of 128 nodes. The hardware and SDKs on the nodes are identical to what is on Aurora.



(a) Polaris Node Diagram [7]



(b) Aurora Node Diagram [8]

Fig. 1: Node Diagrams

III. Tracking Application and pre-production Software Progress on Aurora

A multi-year effort was undertaken to prepare over forty applications for Aurora from the Argonne Early Science Program (ESP) and the Exascale Computing Project (ECP). This work generally consisted of several activities:

- Implementation of new algorithms to enable new types of scientific simulations.
- Porting to a programming model available on Aurora
- Resolution of all issues preventing an application from running in the Aurora development environment
- Tuning to achieve good performance on one, or a small number of nodes
- Scaling to a significant fraction of the Aurora nodes with good performance

The applications work for Aurora was coordinated through the Aurora Applications Working Group which brought together Aurora application developers and Argonne and Intel staff. The group met regularly to review the progress of individual applications and to assess the overall status of the applications development effort with an eye to understanding and mitigating the overarching problems in readying applications. On a quarterly basis Aurora application development teams were surveyed to quantify the status of their efforts and the results of these surveys are presented here to provide an overview on the progress through the above steps over time.

Exascale computing represents a significant increase in computing power over previous generations of large scale systems. To take advantage of this new capability many research teams sought to develop and implement more complex and novel algorithms to enable the simulation of physics that was previously not possible. An initial challenge was therefore the development and implementation of these algorithms. Figure 2 shows the status of applications science implementations from the fourth quarter (Q4) of 2020 to the first quarter (Q1) of 2024. Some of this work may have been performed on platforms other than Aurora, and a complete implementation does not necessarily indicate that an application has been ported or was running on the Aurora development platforms, only that the code was running somewhere. While many applications reported a complete implementation of their algorithms in the response to the first survey it still took several years before virtually all applications had completed this step. In the early quarters many applications were still refining, altering, or expanding their plans which in some quarters resulted in the number of applications reporting a complete implementation dropping from the previous.

The Aurora system contains Intel GPUs which represent a new variant of GPU accelerated heterogenous hardware. While the hardware is similar in many ways to what is seen in NVIDIA based systems, such as Polaris, enabling applications to utilize a programming model supported on Aurora was another significant task that many applications had to address. In many cases this work was overlapped with other work, such as implementing new algorithms, or tuning other already ported kernels for performance. While Aurora does not support CUDA, which is an NVIDIA proprietary programming model, however it does provide a number of open and portable programming models including SYCL, OpenMP, Kokkos, RAJA, and HIP. Figure 3 show the progress in porting applications over time to one of the programming model available on Aurora. Initially only approximately a quarter of the applications were implemented in a supported programming mode (typically via Kokkos or OpenMP). A large increase in the number of fully ported applications, from 30 to 37, was observed in the second quarter (Q2) of 2023 which was same quarter in which a larger scale test and development system (Sunspot) which had the same hardware as in Aurora became available. This likely catalyzed the final porting steps for applications which were close to being fully ported. While most applications made steady progress some initial backwards progress was reported by some applications as they altered their plans, and for most applications porting was a multi-year effort.

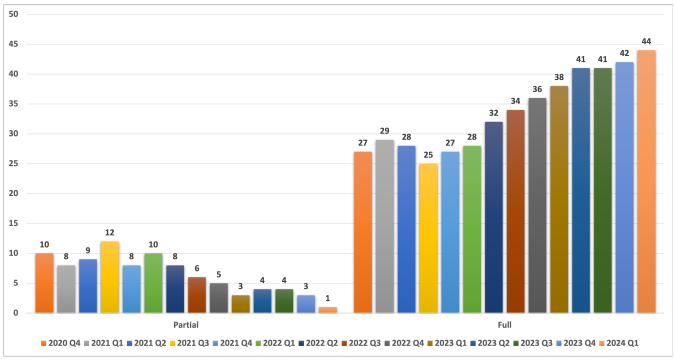


Fig. 2: Application Science Implementation Status

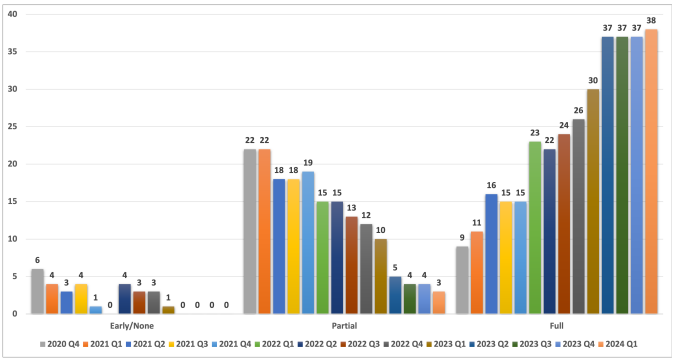


Fig. 3: Application Porting Status

Given the novelty of the Intel GPU hardware and associated software environment, enabling applications to run and achieve good performance involved a close multi-year collaboration between Intel, Argonne, and application developers to resolve issues and improve performance. Progress in these areas at the node level is shown in Figure 4, which covers from Q2 of 2022, when developers generally obtained initial access to the Intel GPUs used on Aurora, to the most recent assessment at

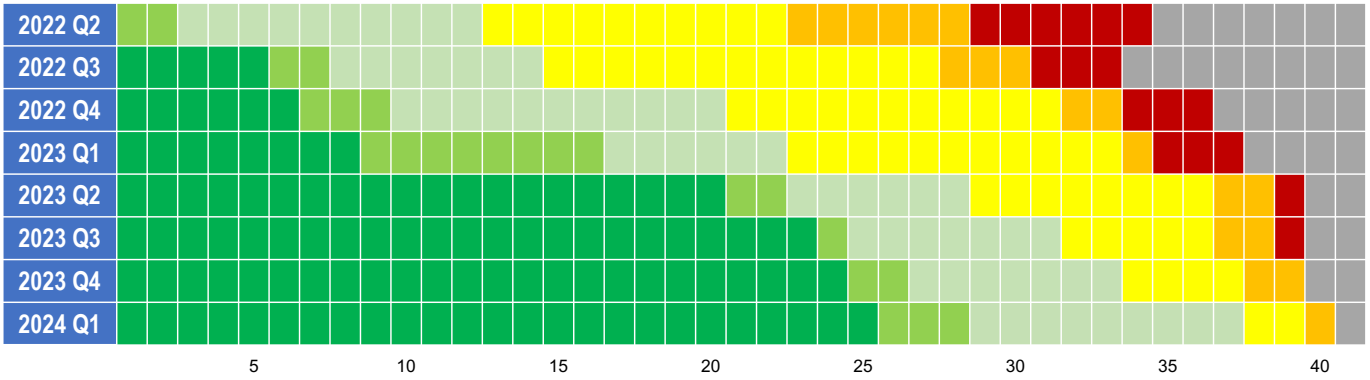


Fig. 4: Status of Applications on Aurora Over Time

the end of Q1 of 2024. Figure 4 shows a high level assessment of application functionality and performance, which each box represents an application and successfully running applications colored in differing shades of green depending on the level of performance achieved. Dark green denotes applications running with a high degree of performance, where only incremental performance gains likely remain. Medium green represents applications where reasonable performance has been achieved but significant performance increases may still be achievable, and light green indicates application where performance is low or where performance has not been evaluated. Yellow denotes applications that are partially running, which may represent some but not all codes paths working, or that the code is only partially ported. Orange denotes application running to a lesser degree with only select kernels ported or functional. Red denotes applications that have not run successfully to any degree, and Grey indicates the applications that were not yet tested on the Aurora Intel GPUs. Progress on scaling applications to run at larger scale on Aurora is ongoing with multiple applications having successfully scaled up to two thousand nodes.

The development of a novel HPC software stack requires long term effort and in many cases application development work is gated by the functionality of this software environment. To facilitate the resolution of issues in the Aurora software stack Argonne staff constructed an easy-to-use bash based testing framework based on bats-core [8]. The basic bats-core functionality was augmented with additional functionality and features, such as robust test timeouts and device resetting when tests hung, utility functions for figure-of-merit performance recording, and archiving of test run artifacts. The simplicity of the test framework presented a low barrier to the development of tests and reproducers, which was essential to enabling application developers to add reproducers. The testing framework began with only a few initial tests and grew to contain over 1100 tests and reproducers. In order to add a reproducer, the reporter would need to make a new subdirectory in the test set with the files needed to reproduce the issue and add bats file with commands to reproduce the issue. An example of a bats file is shown in Fig. 5.

```
# User code to be added by the bug reporter.
# This bash function contains the commands to test.
@test "compile_run_test" {
  # test_init creates a working directory
  # and copies test files inside.
   test_init

  # Commands from a user to reproduce the bug.
  # For example here, a file is compiled and run.
  # If both commands exit with a zero exit code,
  # the test passes.
  # Otherwise the test will fail on the first command
  # that exits with a non-zero exit code.
  # Thus, it will fail if either compile or run fail.

  icpx ... main.cpp

  ./a.out
}
```

Fig. 5: Example of a simple `bats` file used in the test set

The test set is in a private git repository which is shared with Intel and run by their engineers internally. Additionally, a portion of the tests were incorporated into the internal compiler testing at Intel. The testing framework was run with each new pre- production Aurora software drop which allowed Argonne staff to monitor bug fixes and regressions, and quickly identify if a bug was fixed or not. Tests were kept in the test set after a fix was delivered in order to monitor for possible regressions.

For Aurora the development of a common testing framework that was used to consistently report and track issues proved to be highly beneficial in maturing the software stack, which in turn was critical to enabling applications to run successfully on the Aurora testbeds. The use of the test framework was found to help reduce the time from bug identification to resolution, which was critical to enabling application progress. In particular, the use of a shared standard test framework allowed Intel to quickly access issue reproducers packaged in a form that was convenient for compiler developers to work with. The test set was also found to facilitate communication and discussions about bugs with the vendor, since concrete

data from the same test and source code could be referred to when resolving differences between what the vendor and the facility were seeing. Once a sufficient number of tests were in the framework it became a useful tool for validating systemic underlying changes in the environment, such as firmware changes, or software environment, such as the switch from OpenCL to Level Zero for the underlying runtime. Fig. 6 shows the number of total bugs in the test set from 2019 to 2023 for a subset of the software drops from Intel and an increase from an approximately 30% pass rate to a close to 90% pass rate.

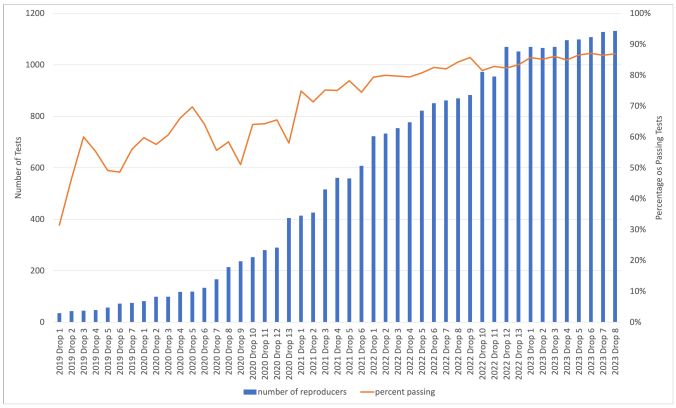


Fig. 6: Reproducers In the Test Set Over Time (2019 to 2023)

IV. Configurations for Application Benchmarking

In this section, details of the applications studied are presented. The eleven applications here are a subset of applications from the ECP and ESP projects which were tracked by the ALCF in the Aurora Applications Working Group, discussed in Section III. Table I shows the employed applications in this study and their science domain, base language, programming models/portability layers, and Figure-of-Merit (FOM) definition.

A brief description of each code as well as details on the build/runtime environment is:

A. AMR-Wind

AMR-Wind [10] is a massively parallel, block-structured adaptive-mesh, incompressible flow solver for wind turbine and wind farm simulations built on top of the AMReX [11] framework. The version of AMR-Wind used in this study is v0.9.0-4-g885f4137 with the AMReX version 23.11-5-gd36463103dae, and it was built with `Intel oneAPI DPC++/C++ Compiler 2024.0.0` on Aurora and `NVIDIA CUDA compiler release 11.8` on Polaris. `abl_godunov.i` input is used to solve atmospheric boundary layer flows in a cubic box with periodic boundary condition in yz and xz planes, wall boundary conditions on the bottom xy plane, and slip boundary conditions with a constant temperature gradient on the top xy plane. Each GPU solves $512*256^2$ cells for $2000*1000^2$ geometry with the maximum grid size of 256 for 3 time steps. On Aurora, 2 MPI ranks (1 MPI rank per stack) were assigned to each GPU while a single MPI rank per GPU was used on Polaris.

B. CNS-libParanumal

CNS is a high-order, discontinuous Galerkin-based, compressible flow solver, and is one of the components built on top of the libParanumal library [12]. The CNS application uses the OCCA portability library [13] that enables runtime translation and compilation of targeted kernels to programming model-specific backend codes (SYCL/CUDA) for execution across different hardware architectures. The version of libParanumal used for this study is v0.5.0 with OCCA version v1.6.0. The input file `setupBoxHex3D.rc` is used for the study which sets up a cubic box with hexahedral elements with a polynomial order of 5, with an initial Gaussian pressure wave in the flowfield, and uses the low-storage Runge-Kutta time-integrator to solve for 10000 time steps. The total runtime for the time integration phase is measured. For the weak scaling study, each GPU is setup to solve for $64*32*32$ elements. On Aurora, CNS was built with the `Intel oneAPI 2024.0` SDK, and run with one MPI rank assigned to each PVC tile, i.e. 2 MPI ranks per PVC GPU using the SYCL backend of OCCA. On Polaris, CNS was built with the `GNU 11.2.0 SDK` along with the `CUDA 11.8.0` standalone toolkit and run with one MPI rank assigned to each A100 GPU using the OCCA CUDA backend.

C. DCMesh

DC-MESH (divide-and-conquer Maxwell-Ehrenfest-Surface Hopping) combines non-adiabatic quantum molecular dynamics (NAQMD) with Maxwell equations for light with a divide-and-conquer scheme [14]. The commit `ad381f6` of DC-MESH is used in this study on both Aurora and Polaris. The benchmark uses the Local Field Dynamics (LFD) miniapp simulating PbTiO3 material consisting of 40 atoms. Each MPI rank, 288 Kohn-Sham wave functions are represented using the plane-wave basis, while each orbital in LFD is represented on 70×70×72 finite-difference mesh. On Polaris, `LLVM 17` compiler was used for OpenMP offload features and one MPI rank per A100 GPU was used. On Aurora/Sunspot, `Intel oneAPI 2024.0` release was used and one MPI rank per GPU tile, namely two MPI ranks per PVC GPU was used.

D. FlashX

Flash-X [15] is a multi-physics simulation software instrument written in Fortran, and uses parallel programming model OpenMP as well as OpenACC to port the code on heterogeneous systems. In this study, Flash-X is used underneath Thornado which computes transport of neutrinos and their interaction with matter. Thornado approximates the neutrino radiation field by solving for angular moments of the kinetic distribution function in a multi-species spectral two-moment approach [16], [17]. The benchmark uses streaming sine wave (SSW) that models the free-streaming of neutrinos in phase-space with 16 neutrino energy elements, 4 moments, 2 neutrino species, and 2 nodes. The workload for each GPU

is set to a domain size of 100 x 100 x 50 and 24 blocks with 512 cells per block. On Aurora, Flash-X using OpenMP target-offloading was built with the `Intel oneAPI 2024.0` and `magma-2.7.0` and on Polaris, Flash-X using OpenACC was built with `nvhpc/23.3`. For both systems, the commit `623c452973da110b4917f2cc80d9fe440881b51f` was used for this build. One MPI rank is set per tile on PVC and per GPU on A100.

E. GAMESS

GAMESS [18], [19] is an electronic structure code with many different quantum chemistry methods implemented. GAMESS is mainly written in Fortran, with some C++ libraries. To target GPUs from the main Fortran code, GAMESS uses OpenMP offload. The port from CPU OpenMP to GPU OpenMP is discussed in Ref. [20]. The problem chosen for this study is to compute the RI-MP2 energy of a cluster of 2,880 water molecules with the EFMO/RI-MP2 method [21]. The EFMO/RI-MP2 method use multi-level parallelism (coarse-grained parallelism via MPI groups and OpenMP CPU threads and OpenMP offload to GPUs inside each MPI group) to achieve a high level of parallelism. The input file studied here is a water cluster: `water.inp` and can be found in Ref. [22].

On Aurora, GAMESS was built using commit `73a76205b` from branch 'tsatta/w2-efp-offload-TF_intel' with the `Intel oneAPI 2024.0` SDK. GAMESS is run with 12 compute ranks (one per PVC tile) and 8 OpenMP CPU threads.

On Polaris, GAMESS is built using commit `7f2ba73` from branch 'buu/w2-polaris' with `Nvidia HPC SDK 23.3`. GAMESS is run with 4 compute ranks (one per A100 GPU) and 8 OpenMP CPU threads.

F. CRK-HACC

HACC (Hardware/Hybrid Accelerated Cosmology Code) is a cosmological simulation code used for studying the formation of large-scale structure in the universe. Extended with a modern smoothed particle hydrodynamics (SPH) approach called CRKSPH [23], CRK-HACC [24] adds the ability to resolve gas physics to HACC's N-body gravitational solver. The long-range gravity force solver is written in C++ and MPI. The short-range gravity and hydro solvers are written using programming models for GPU acceleration, currently having CUDA, HIP, and SYCL implementations.

The configuration for this study is a cosmological adiabatic (gravity+hydrodynamics) simulation with two species (dark matter and baryon) of particles, $\sim 256^3$ per GPU, respectively. Due to application configuration constraints, a fixed number of MPI ranks was used for the test problems on each system: 12 MPI ranks on Sunspot/Aurora and 8 MPI ranks on Polaris. Scaling was achieved by increasing the simulation volume and maintaining a fixed particle density.

On Sunspot/Aurora, the SYCL implementation of CRK-HACC was built using the `Intel oneAPI DPC++/C++ Compiler 2024.1.0`. On Polaris, the CUDA implementation of CRK-HACC was built using the NVIDIA `cuda_11.8.r11.8/compiler.31833905_0`.

G. LAMMPS

The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a general-purpose molecular simulation code with a focus on molecular dynamics [25] . LAMMPS supports potentials for a wide range of systems, including solid-state materials, soft matter, liquids, and coarse-grained or mesoscopic systems.

LAMMPS runs on a single processor or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain. LAMMPS is written in C/C++ and includes support for OpenMP on CPUs, a GPU package with CUDA, HIP, and OpenCL backends, and a Kokkos package. Kokkos is the primary programming model for this work with SYCL backend on Aurora and CUDA backend on Polaris.

This work targets molecular dynamics simulation of materials related to nuclear fusion and fission using SNAP potential which stands for Spectral Neighbor Analysis Potential. In this work 26 neighbors were used for each atom and 9 descriptors used to describe the energy of the atom with respect to its surrounding, using LAMMPS version Nov-2023. To make sure each GPU on A100 or tile on PVC has enough work to do, 16,000 atoms per tile or 32,000 atoms per GPU were used. The run time was 100 time steps. 1 MPI rank per PVC tile was used on Aurora, and 1 MPI rank per A100 was used on Polaris. On Aurora LAMMPS was built with `Intel OneAPI 2023.05.15.006` and on Polaris, LAMMPS was built with `cuda_11.8.r11.8/compiler.31833905_0`.

H. NWChemEx

NWChemEx's [26] component TAMM [27] focuses on tensor algebra operations such as contractions in computational chemistry which contributes to a significant fraction of the computing time. The widespread use of tensor contractions between large multi-dimensional tensors in describing electronic structure theory has motivated the development of multiple tensor algebra frameworks targeting heterogeneous computing platforms. In addition, a single-source, cross-platform C++ abstraction layer programming model, SYCL backend in TAMM was already tested for this computational chemistry methods such as CCSD(T) coupled-cluster formalism. [28]

A primitive runtime launch configuration involves launching number of MPI ranks per node same as the number of Intel PVC tiles per node plus one and Nvidia A100 GPUs per node plus one for Aurora and Polaris respectively. This additional MPI rank was used to support PGAS configurations from Global Arrays project. [29] Benchmarks were performed with NWChemEx configured with `GCC 11.2`, `CUDA 11.8` and `cray-mpich-8.1.25` for Polaris and `Intel oneAPI compiler version 2023.12.15.002` for Aurora.

An interesting software formalism for Aurora is to explicitly use High Bandwidth Memory (HBM) component via numactl APIs [30] from the CPUs on node as a staging buffer for faster host-device data transfer and standard DDR component for large-scale distributed memory operations.

I. OpenMC

OpenMC [31] is an open source Monte Carlo neutral particle transport application. It is used extensively for the simulation of nuclear fission reactors and fusion energy devices, as well as for more general radiation transport analysis. An exascale-focused GPU version of OpenMC has recently been developed [32] and has demonstrated highly efficient performance at scale on NVIDIA, Intel, and AMD GPU supercomputers [33] via C++ OpenMP target offloading. On Intel GPUs, OpenMC was launched in 4 CCS mode with 4 MPI ranks per tile (8 MPI ranks per GPU), while on NVIDIA GPUs OpenMC used 4 MPI ranks per GPU with the NVIDIA multi-process service (MPS) enabled.

The benchmark simulation problem used in this study represents a realistic small modular reactor with around 195,000 unique depleted fuel regions. The figure of merit for performance is the active batch particle tracking rate in terms of millions of particle histories per second. Notably, this figure includes costs for reaction rate tallies for each uniquely depleted fuel region. For offloading to Intel GPUs, OpenMC utilized the `Intel oneAPI compiler version 2023.12.15.002`. For offloading to NVIDIA GPUs, OpenMC utilized the `LLVM 17.0.6` compiler with `CUDA 11.8.0`.

J. SW4

SW4 (Seismic Waves, 4th Order) is a summation-by-parts, explicit time-stepping, fourth order finite difference program for simulating seismic wave propagation [34]. The SW4 code is written in C++ and utilizes RAJA [35] to enable parallel kernel execution on GPUs. This work utilizes the 'berkeley-r.in' test input with nx=701. For both Aurora and Polaris, SW4 was built using RAJA (`v2023.06.01`).

On Aurora, SW4 targets the SYCL backend of RAJA and was built using the `Intel oneAPI 2024.0 SDK`. This work executes SW4 with a single MPI rank per PVC tile on Aurora.

On Polaris, SW4 was built using the `cuda 11.8.0` toolkit with the `GNU 11.2.0` compiler to target the RAJA CUDA backend. This work execute SW4 with a single MPI rank per A100 GPU on Polaris.

K. XGC

XGC is a gyrokinetic particle-in-cell code for simulating magnetically confined fusion plasmas in toroidal devices [36]. The primary language is C++. XGC uses Kokkos [37] for GPU offloading and performance portability in general. XGC also uses parts of the Cabana [38] library. The test problem is XGC1Example [39] (electrostatic, full-f, with drift kinetic electrons), keeping 9397 particles per grid vertex in each radial-poloidal plane. Weak scaling increases the number of radial-poloidal planes (toroidal grid points) proportional to the number of MPI ranks.

On Aurora, XGC was built from commit `090639c8` from the main branch using the `Intel oneAPI 2024.0` SDK. It was run using 1 MPI rank per PVC tile and 16 OpenMP CPU threads per rank.

On Polaris, XGC was built from commit `090639c8` from the main branch using the `GNU 11.2.0` compiler with `cudatoolkit 11.8.0`. It was run using 2 MPI ranks per A100 GPU and 16 OpenMP CPU threads per rank.

V. Performance Comparison of Applications on Aurora and Polaris

To evaluate performance of the applications on Aurora and Polaris, we considered the FOM of the applications running on a single PVC vs. a single A100, as well as the parallel efficiency on a single node of Aurora (1,3,6 PVCs) and Polaris (1,2,4 A100s).

Figure 7 shows the FOM performance of the applications on a PVC on Aurora relative to the FOM performance on an A100 on Polaris. Most of the applications are over a ratio of 1, meaning that they perform better on PVC than A100. The change in performance on A100 vs. PVC will depend on algorithm, as different algorithms will expect different bottlenecks (compute, bandwidth, cache, latency-bound, etc.). Further investigation of what performance is expected for each application will be done in the future. The pre-production Aurora GPU shows approximately 1x to up to 2x performance better than the production-level Nvidia GPUs at ALCF, and on average 1.3x speedup.

NWChemEx has the smallest ratio of PVC performance to A100. This is expected due to a limitation of the current version of SYCL which resulted in needing to add in extra synchronization to the Aurora version which was not needed on Polaris. This is discussed in more detail in Section VI.

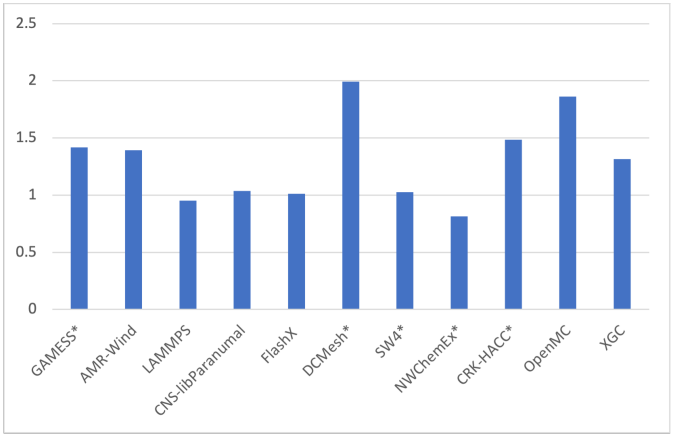


Fig. 7: FOM Performance on a PVC, Relative to A100. The asterisk denotes that the runs were done on a slightly earlier version of the firmware and driver than the results without the asterisk.

A comparison of the FOMs for each application on Polaris and Aurora is shown in Table II. Note that the FOM from different applications have different units, so the comparing between applications is not meaningful.

TABLE I: Applications tested across Polaris and Aurora

Application	Science Domain	Base language	Programming model or Portability layer	FOM definition	Scaling
AMR-Wind	Computational Fluid Dynamics	C++	AMReX (SYCL,CUDA)	Ncell/WT-per-step/10^6	Weak scaling
CNS-libParanumal	Computational Fluid Dynamics	C++	OCCA	NDoFs*Nsteps/WT/10^6	Weak scaling
DCMesh	Electronic Structure	C++	OpenMP Offload	MPI ranks/Total_time	Weak scaling
FlashX	Steller Explosion	Fortran	OpenMP Offload/OpenACC	DOFs*Ntsteps/Evolution_Time/10^9	Weak scaling
GAMESS	Electronic Structure	Fortran	OpenMP Offload	1/(Wall Time)/10^5	Strong scaling
CRK-HACC	Cosmology	C++	SYCL	particles*steps*subcycles /time /1e6	Weak scaling
LAMMPS	Chemistry/Material Science	C++	Kokkos	(#of atoms)*(# of steps) / (Loop time (sec))/1e6	Weak scaling
NWChemEx	Electronic Structure	C++	SYCL	1/(Wall Time)	Strong scaling
OpenMC	Particle Transport	C++	OpenMP Offload	million neutrons/second	Weak scaling
SW4	Earth Science	C++	RAJA	(f_max$\hat{4}$)/(solver time × 7.6)	Strong scaling
XGC	Tokamak Plasma Kinetics	C++	Kokkos	(1/(Wall Time per particle per ion timestep))/1e6	Weak scaling

TABLE II: FOM Within a Node on Aurora and Polaris. The asterisk denotes that the runs were done on a slightly earlier version of the firmware and driver than the results without the asterisk.

	1 PVC	1 A100	Half-node (3 PVC)	Half-node (2 A100)	6 PVC	4 A100
AMR-Wind	33.88	24.35	92.61	43.05	178.30	73.54
CNS-libParanumal	162.93	157.21	459.53	308.31	938.81	603.71
DCMesh*	2.08	1.05	6.28	2.09	12.37	4.14
FlashX	0.50	0.49	1.40	0.91	2.70	1.97
GAMESS*	13.34	9.41	39.31	18.66	79.52	36.82
CRK-HACC*	3.79	2.55	10.94	5.22	21.54	10.23
LAMMPS	2.04	2.14	5.42	4.24	10.60	8.43
NWChemEx*	0.83	1.02	2.43	2.05	4.78	4.03
OpenMC	0.35	0.19	1.04	0.37	2.03	0.73
SW4*	0.0015	0.0014	0.0036	0.0027	0.0064	0.0050
XGC	4.65	3.54	13.61	7.10	25.92	12.84

To investigate the single-node scaling, the parallel efficiency inside a single node of Aurora is shown in Fig. 8 and the parallel efficiency inside a single node of Polaris is shown in Fig. 9. The parallel efficiency is similar on both systems for most of the applications. However, there several differences to discuss.

AMR-Wind on Polaris loses its parallel efficiency quickly via weak scaling with 4 GPUs (i.e., 75.5%) compared to its efficiency with 6 GPUs (i.e., 87.7%) on an Aurora node. According to performance profiling with tools (e.g., NVIDIA Nsight System on Polaris, and Intel VTune on Aurora), the PCIe traffic behavior with multiple GPUs differs between Aurora and Polaris. On an Aurora node, PCIe bandwidth increased linearly with 3 GPUs and 6 GPUs compared to a single GPU. However, on a Polaris node, PCIe traffic took 2.3 times longer with 4 GPUs than with a single GPU, while traffic with 2 GPUs took only 7% longer than with a single GPU. Further investigation will be carried out to identify the root cause of the intra-node efficiency drop on Polaris.

SW4 shows lower parallel efficiency running on Aurora compared to Polaris. From looking at a timing breakdown, this is from a difference in performance for the communication routines in SW4. For single node execution, there is additional communication between the 12 MPI ranks per node on Aurora compared to the 4 ranks per node on Polaris. There is ongoing work to optimize the communication routines for SW4 on Aurora.

CRK-HACC shows super-linear scaling on Polaris. The code is weak-scaled by keeping the amount of work per GPU constant but also keeping the number of MPI ranks constant (12 for Aurora and 8 for Polaris). Thus, the overhead of oversubscribing MPI ranks to GPUs is greatest on the reference case of 1 GPU. For multi-node scaling, this would not occur since the number of ranks per GPU is constant in that case.

Note that the two results from OpenMC on 6 PVC and 4 A100 were taken from Ref. [33].

VI. Discussion of Porting To Aurora

This section discusses any challenges to porting and achieving performance on Aurora. Additionally, any specific tuning or modifications that the applications needed to make to be able to run and perform well on Aurora are discussed.

A. AMR-Wind

AMReX framework has been ported to Intel GPUs with the SYCL programming model, and AMR-Wind uses the AMReX framework as the portability layer. At the beginning, Intel IRIS Gen9 integrated GPU was the target platform, and AMR-Wind performance was around a half million cells per second (i.e., FOM was around 0.5). Once an early Intel discrete GPU was available for the development, the performance increased around 6 times (i.e., FOM was above 3.0). The first silicon of PVC doubled AMR-Wind performance (i.e., FOM was around 7.5). With the dual stack design of the latest PVC on Aurora

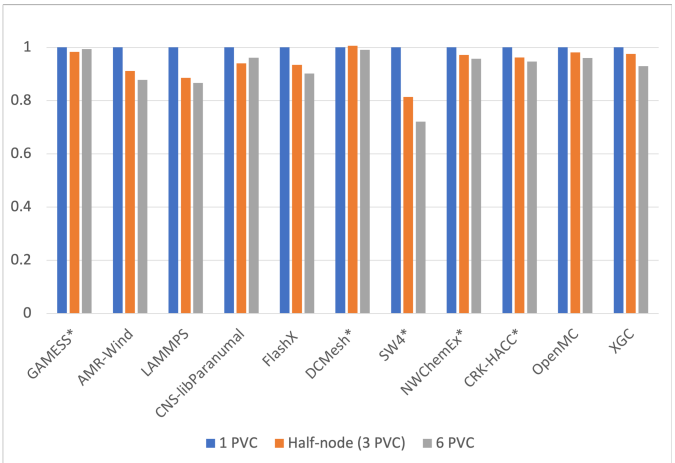


Fig. 8: Parallel Efficiency on a Single Node of Aurora. The asterisk denotes that the were done on a slightly earlier version of the firmware and driver than the results without the asterisk.

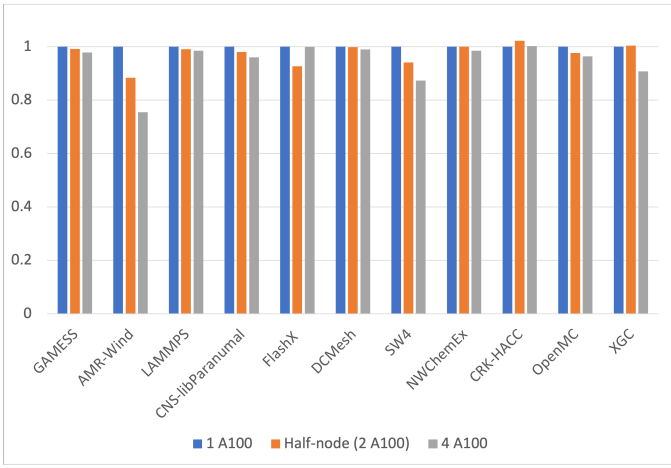


Fig. 9: Parallel Efficiency on a Single Node of Polaris

and a myriad of progress in the oneAPI software over the last few of years, AMR-Wind performance on a single PVC increased more than 4 times from the earlier version of PVC and corresponding oneAPI software.

The oneAPI software has several knobs for obtaining better application performance, and the use of immediate command lists [40] is beneficial to improve AMR-Wind performance on Aurora. With this feature, multiple command lists can run concurrently on a single hardware queue and batching of kernels on the GPU is allowed, while it generates more host overhead on appending an operations to the command list. It has been the default submission mode on Aurora since `oneAPI/2023.2 release`, and AMR-Wind gains around 10% more performance with the same workloads on an Aurora node with the feature.

B. CNS-libParanumal

The first step involved in porting CNS to Aurora was to implement the SYCL (DPC++) backend in the OCCA portability library. This was a major effort undertaken as part of the ECP project in order to port other downstream applications such as nekRS as well. Since all the core kernels in the CNS application and libParanumal libraries are ported using OCCA, porting and running it on Aurora did not require any major code modifications to the application itself. There were some code modifications required to remove the use of variable length arrays (VLA) that are not allowed in C++11 standard as per the Intel SDK. Through the implementation of new test cases and validation efforts, some bugs were also found and fixed with the help of the developers. In order to tune the performance, there were several efforts undertaken. In particular, enforcing SIMD16 instructions using the `IGC_ForceOCLSIMDWidth` environment variable, along with the compiler flag to automatically choose the register file size (128kB or 256kB), enabled a larger register file per thread for some of the kernels with a large register pressure and thereby avoid register spilling.

C. DCMesh

Unlike a lot of applications having NVIDIA GPU port far ahead of the support on Intel GPUs, DCMesh adopted the portable OpenMP programming model and made its GPU porting almost fully vendor agnostic. Some calculations involving matrix-matrix multiplications still rely on calling vendor optimized libraries with minimal source codes using vendor specific programming models. This porting strategy maximized portability without sacrificing performance on either GPUs.

D. FlashX

During the porting of Flash-X on Intel-based HPC systems, initially, Thornado was identified as a computing intensive module that consumes 80% of FLOPS. Hence, Thornado was the target for performance optimization and enhancement on PVC. Several compiler-related bugs were reported and they have been resolved timely by Intel. Controlling device memory via `Level Zero` specifications by tuning an OpenMP offload environment variable, `LIBOMPTARGET_LEVEL_ZERO_MEMORY_POOL`, helped to obtain significant performance improvement (2x on SSW). Further, switching from JIT to AOT compilation reduced total time of Thornado by 2.2x on SSW. One kernel in Thornado initially failed to compile due to needing to use more bytes than allowed to pass parameters to a kernel. However, this was overcome by increasing the limit allowed by setting an environmental variable `IGC_OverrideOCLMaxParamSize`.

E. GAMESS

A large effort was carried out to port GAMESS to OpenMP Offload for GPUs during ECP. Compared to other vendors, there were no large structural changes needed to be able to run on Aurora. However, there were several parameters we needed to tune to improve performance on Aurora, as well as multiple bugs in the early software.

The GAMESS team found that tuning the number of OpenMP threads, OpenMP teams, and using compiler flags to decrease register spilling was essential to getting performance on Aurora. For example, the tuned options can be up to 4x faster than the default options.

One of the biggest performance issues early on was that the routines that called 'omp atomic' were significantly slower than on other architectures. This was isolated and added as a stand-alone test in the bug repository, and it was fixed by Intel.

F. CRK-HACC

The CRK-HACC codebase was, and is currently, under active development using CUDA. To prepare for Aurora, effort was made on automating the migration from CUDA source files to SYCL using SYCLomatic [41], [42] and developing customized Clang-based tools for fine tuning the source-to-source translation. Five of the most intensive kernels were identified and the focus of a performance optimization effort [43], where SYCL sub-group "shuffle" operations were identified to be a performance bottleneck. Solutions using shared local memory and inline assembly were developed for the Intel PVC GPU.

G. LAMMPS

In this work LAMMPS with Kokkos using SYCL backend was used, which resulted in having to manage multiple dependencies and the combination of early versions of KOKKOS, Intel software and tools, and early hardware. Most of the work was done on a proxy application that executed the main kernels. The first published work [44] focused on a direct OpenMP 4.5 implementation of the mini app and understanding how source code modifications impacted the performance across three vendor's GPUs at the time. Testing the full LAMMPS application on early hardware was enabled in 2022 and that brought new challenges with respect to evolving software stack and understanding performance relative to other architectures. This work was a collaborative effort with Intel, Kokkos, and LAMMPS developers to understand observed vs. expected performance aided by tools, such as VTune. Some changes that were done to improve the performance are as follow: a) Manually casting shared memory pointer from global to local address space b)Tuning of workgroup size for different kernels c)Experimental Kokkos interface to set workgroup size on per-kernel basis.

H. NWChemEx

NWChemEx is one of the co-design projects of ECP to be written using modern C++ and Python to take advantage of the potential of exascale computing era. The code was initially developed for Nvidia GPUs using CUDA and then ported to HIP and SYCL for AMD and Intel GPUs respectively. The porting strategy involved using SYCLomatic [41] to port CUDA kernels to SYCL.

A current issue related to SYCL's `host_task` functionality limits the performance on Intel GPUs. As a workaround, the application enforces a synchronization of the SYCL event returned by the `host_task` for the Level-Zero plugin associated with Intel GPUs.

I. OpenMC

On both Intel and Nvidia GPUs, a "unity" build with cmake was used, wherein all source code was treated as a single compilation unit to reduce the need for link time optimization. Full details of the OpenMC porting and optimization effort are available in [32], [33]. To improve performance on the Intel system, the Intel oneAPI compiler's `IGC_ForceOCLSIMDWidth` environment variable was set to `16`.

J. SW4

SW4 utilizes RAJA for GPU acceleration by implementing RAJA execution policies targeting a compatible backend. The porting of SW4 for Aurora was developed alongside the development of the SYCL backend of RAJA. The existing RAJA execution policies for the CUDA backend were ported to target the SYCL backend. Beyond the development of the RAJA SYCL backend support, a workaround was implemented to remove function pointer usage on the device since it wasn't supported at the time of porting (although it is now).

SW4 required additional optimization to improve the performance for Aurora. Several of the most computationally expensive kernels in SW4 create significant register pressure. For SW4, compiling with use of the large register file option provides significantly improved performance by reducing spilling. Additionally, SW4 performs a 125-point stencil computation resulting in performance sensitivity to the default 3-D SYCL kernel work group size selection. RAJA is designed to expose these tuning knobs and additional performance improvements were gained by optimizing these RAJA execution policies.

K. XGC

XGC uses Kokkos [37] and some parts of Cabana [38] for performance portability. The earliest available usable port of Kokkos for Intel GPUs was implemented with the `OpenMPTarget` execution space, so that was used with early integrated graphics and discrete GPU testbed hardware. Once the `SYCL` execution space implementation of Kokkos matured, the Kokkos development team recommended switching to that. Working with the developers implementing Kokkos for Intel GPU hardware was invaluable. In benchmarking PVC, it became clear early on that the standard single-GPU test case was too small to make good use of the amount of concurrency supported by a PVC GPU. Increasing the number of particles per mesh vertex by a factor of two or more improved the relative performance w.r.t. an A100 GPU significantly. There are still some performance knobs to tune for optimizing XGC on PVC, such as adjusting the SIMD width and finding ways to reduce register spills; this is ongoing work.

VII. Conclusions

The ALCF has been preparing for Aurora for the last several years. As part of the preparation, one of the approaches the

ALCF took was tracking application status via a working group with ALCF, Intel, and application developers, as well as quarterly surveys sent to ECP and ESP application development teams. Additionally, bugs and issues were tracked via a shared git repository with source code reproducers. These tracking strategies were essential to ensuring the usability of Aurora by a wider range of applications, by helping the ALCF identify wide-spread issues and communicate with the vendor efficiently about bugs. To evaluate the current state of the tracked applications on Aurora and Polaris, the single-GPU and intra-node scaling performance for a subset of the tracked applications was discussed. The intra-node scaling was similar on both Polaris and Aurora, and on average the single GPU FOM performance on pre-production Aurora was 1.3x that on production-level Polaris.

Future work will also include applications from a wider range of programming models and domains, like machine learning. Additionally, the application performance differences seen between Aurora and Polaris will be investigated in more detail, as well as power measurement comparisons between Aurora and Polaris.

ACKNOWLEDGMENT

This work was done on a pre-production supercomputer with early versions of the Aurora software development kit.

This work was supported by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration).

REFERENCES

[1] "The compute architecture of intel processor graphics gen9." [Online]. Available: https://cdrdv2-public.intel.com/774710/the-compute-architecture-of-intel-processor-graphics-gen9-v1d0-166010.pdf

[2] "Inlte oneapi." [Online]. Available: https://www.intel.com/oneapi/

[3] T. Lists, "Doe/sc/argonne national laboratory systems on the top500 list," 2023. [Online]. Available: https://www.top500.org/site/47347/

[4] [Online]. Available: https://www.exascaleproject.org/

[5] [Online]. Available: https://www.alcf.anl.gov/science/early-science-program

[6] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[7] B. Homerding, B. Lenard, C. Blackworth, C. Holohan, A. Kulyavtsev, G. McPheeters, E. Pershy, P. Rich, D. Waldron, M. Zhang, K. Harms, T. Leggett, and W. Allcock, "Polaris and acceptance testing." CUG2023 Proceedings, 2023.

[8] J. Kwack, "The path to Aurora, GPU for Science Day." [Online]. Available: https://www.nersc.gov/users/training/past-training-events/2023/gpus-for-science-day-2023/

[9] W. Gomes, A. Koker, P. Stover, D. Ingerly, S. Siers, S. Venkataraman, C. Pelto, T. Shah, A. Rao, F. O'Mahony, E. Karl, L. Cheney, I. Rajwani, H. Jain, R. Cortez, A. Chandrasekhar, B. Kanthi, and R. Koduri, "Ponte vecchio: A multi-tile 3d stacked processor for exascale computing," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 42–44.

[10] AMR-Wind Webpage. [Online]. Available: https://amr-wind.readthedocs.io/en/latest/

[11] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, and M. Zingale, "AMReX: a framework for block-structured adaptive mesh refinement," *Journal of Open Source Software*, vol. 4, no. 37, p. 1370, May 2019. [Online]. Available: https://doi.org/10.21105/joss.01370

[12] N. Chalmers, A. Karakus, A. P. Austin, K. Swirydowicz, and T. Warburton, "libParanumal: a performance portable high-order finite element library," 2022, release 0.5.0. [Online]. Available: https://github.com/paranumal/libparanumal

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

[14] T. Linker, K. ichi Nomura, A. Aditya, S. Fukshima, R. K. Kalia, A. Krishnamoorthy, A. Nakano, P. Rajak, K. Shimmura, F. Shimojo, and P. Vashishta, "Exploring far-from-equilibrium ultrafast polarization control in ferroelectric oxides with excited-state neural network quantum molecular dynamics," *Science Advances*, vol. 8, no. 12, p. eabk2625, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/sciadv.abk2625

[15] A. Dubey, K. Weide, J. O'Neal, A. Dhruv, S. Couch, J. A. Harris, T. Klosterman, R. Jain, J. Rudi, B. Messer *et al.*, "Flash-x: A multiphysics simulation software instrument," *SoftwareX*, vol. 19, p. 101168, 2022.

[16] M. Shibata, K. Kiuchi, Y. Sekiguchi, and Y. Suwa, "Truncated Moment Formalism for Radiation Hydrodynamics in Numerical Relativity," *Progress of Theoretical Physics*, vol. 125, pp. 1255–1287, 2011.

[17] C. Y. Cardall, E. Endeve, and A. Mezzacappa, "Conservative 3+1 general relativistic variable Eddington tensor radiation transport equations," *Physical Review D*, vol. 87, p. 103004, 2013.

[18] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, "General atomic and molecular electronic structure system," *Journal of Computational Chemistry*, vol. 14, no. 11, pp. 1347–1363, 1993.

[19] G. M. J. Barca, C. Bertoni, L. Carrington, D. Datta, N. De Silva, J. E. Deustua, D. G. Fedorov, J. R. Gour, A. O. Gunina, E. Guidez, T. Harville, S. Irle, J. Ivanic, K. Kowalski, S. S. Leang, H. Li, W. Li, J. J. Lutz, I. Magoulas, J. Mato, V. Mironov, H. Nakata, B. Q. Pham, P. Piecuch, D. Poole, S. R. Pruitt, A. P. Rendell, L. B. Roskop, K. Ruedenberg, T. Sattasathuchana, M. W. Schmidt, J. Shen, L. Slipchenko, M. Sosonkina, V. Sundriyal, A. Tiwari, J. L. Galvez Vallejo, B. Westheimer, M. Włoch, P. Xu, F. Zahariev, and M. S. Gordon, "Recent developments in the general atomic and molecular electronic structure system," *The Journal of Chemical Physics*, vol. 152, no. 15, p. 154102, 2020. [Online]. Available: https://doi.org/10.1063/5.0005188

[20] F. Zahariev, P. Xu, B. Westheimer, S. Webb, J. Vallejo, A. Tiwari, V. Sundriyal, M. Sosonkina, J. Shen, G. Schoendorff, M. Schlinsog, T. Sattasatuchana, K. Ruedenberg, L. Roskop, A. Rendell, D. Poole, P. Piecuch, B. Pham, V. Mironov, and M. Gordon, "The general atomic and molecular electronic structure system (gamess): Novel methods on novel architectures," *Journal of Chemical Theory and Computation*, vol. 19, 10 2023.

[21] B. Q. Pham, L. Carrington, A. Tiwari, S. S. Leang, M. Alkan, C. Bertoni, D. Datta, T. Sattasathuchana, P. Xu, and M. S. Gordon, "Porting fragmentation methods to gpus using an openmp api: Offloading the resolution-of-the-identity second-order møller–plesset perturbation method," *The Journal of Chemical Physics*, vol. 158, no. 16, 2023.

[22] [Online]. Available: https://github.com/colleeneb/cug2024

[23] N. Frontiere, C. D. Raskin, and J. M. Owen, "Crksph–a conservative reproducing kernel smoothed particle hydrodynamics scheme," *Journal of Computational Physics*, vol. 332, pp. 160–209, 2017.

[24] N. Frontiere, J. Emberson, M. Buehlmann, J. Adamo, S. Habib, K. Heitmann, and C.-A. Faucher-Giguère, "Simulating hydrodynamics in cosmology with crk-hacc," *The Astrophysical Journal Supplement Series*, vol. 264, no. 2, p. 34, 2023.

[25] [Online]. Available: https://github.com/lammps/lammps

[26] K. Kowalski, R. Bair, N. P. Bauman, J. S. Boschen, E. J. Bylaska, J. Daily, W. A. de Jong, T. Dunning Jr, N. Govind, R. J. Harrison *et al.*, "From nwchem to nwchemex: Evolving with the computational chemistry landscape," *Chemical reviews*, vol. 121, no. 8, pp. 4962–4998, 2021.

[27] E. Mutlu, A. Panyala, N. Gawande, A. Bagusetty, J. Glabe, J. Kim, K. Kowalski, N. P. Bauman, B. Peng, H. Pathak *et al.*, "Tamm: Tensor

algebra for many-body methods," *The Journal of Chemical Physics*, vol. 159, no. 2, 2023.

[28] A. Bagusetty, A. Panyala, G. Brown, and J. Kirk, "Towards cross-platform portability of coupled-cluster methods with perturbative triples using sycl," in *2022 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, 2022, pp. 81–88.

[29] J. Dinan, P. Balaji, J. R. Hammond, S. Krishnamoorthy, and V. Tipparaju, "Supporting the global arrays pgas model using mpi one-sided communication," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 739–750.

[30] A. Kleen, "A numa api for linux," *Novel Inc*, 2005.

[31] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith, "OpenMC: A state-of-the-art Monte Carlo code for research and development," *Annals of Nuclear Energy*, vol. 82, pp. 90–97, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030645491400379X

[32] J. R. Tramm, P. K. Romano, J. Doerfert, A. L. Lund, P. C. Shriwise, A. R. Siegel, G. Ridley, and A. Pastrello, "Toward portable GPU acceleration of the OpenMC Monte Carlo particle transport code," in *PHYSOR 2022 - International Conference on Physics of Reactors*, May 2022.

[33] J. R. Tramm, P. K. Romano, P. C. Shriwise, A. L. Lund, J. Doerfert, P. Steinbrecher, A. R. Siegel, and G. Ridley, "Performance portable Monte Carlo particle transport on Intel, NVIDIA, and AMD GPUs," in *SNA + MC 2024: Joint International Conference on Supercomputing in Nuclear Applications + Monte Carlo*, Paris, France, Oct. 2024, submitted. Preprint available at https://arxiv.org/abs/2403.12345.

[34] N. A. Petersson, B. Sjögreen, H. Tang, and R. Pankajakshan, "Wave propagation in anisotropic elastic materials and curvilinear coordinates using a summation-by-parts finite difference method," *Journal of Computational Physics*, vol. 299, p. 820–841, Oct. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2015.07.023

[35] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, and T. R. Scogland, "Raja: Portable performance for large-scale scientific applications," in *2019 ieee/acm international workshop on performance, portability and productivity in hpc (p3hpc)*. IEEE, 2019, pp. 71–81.

[36] S. Ku, C. Chang, and P. Diamond, "Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry," *Nuclear Fusion*, vol. 49, no. 11, p. 115021, sep 2009. [Online]. Available: https://dx.doi.org/10.1088/0029-5515/49/11/115021

[37] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke, "Kokkos 3: Programming model extensions for the exascale era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 805–817, 2022.

[38] S. Slattery, S. T. Reeve, C. Junghans, D. Lebrun-Grandié, R. Bird, G. Chen, S. Fogerty, Y. Qiu, S. Schulz, A. Scheinberg, A. Isner, K. Chong, S. Moore, T. Germann, J. Belak, and S. Mniszewski, "Cabana: A performance portable library for particle-based simulations," *Journal of Open Source Software*, vol. 7, no. 72, p. 4115, 2022. [Online]. Available: https://doi.org/10.21105/joss.04115

[39] Testing xgc. [Online]. Available: https://xgc.pppl.gov/html/kernels_and_tests.html#testing-xgc

[40] Level Zero Immediate Command Lists Webpage. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/guide/level-zero-immediate-command-lists.html

[41] A. Huang, "Syclomatic compatibility library: making migration to sycl easier," in *Proceedings of the 2023 International Workshop on OpenCL*, 2023, pp. 1–2.

[42] Z. Wang, Y. Plyakhin, C. Sun, Z. Zhang, Z. Jiang, A. Huang, and H. Wang, "A source-to-source cuda to sycl code migration tool: Intel® dpc++ compatibility tool," in *International Workshop on OpenCL*, 2022, pp. 1–2.

[43] E. M. Rangel, S. J. Pennycook, A. Pope, N. Frontiere, Z. Ma, and V. Madananth, "A performance-portable sycl implementation of crk-hacc for exascale," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1114–1125.

[44] N. A. Mehta, R. Gayatri, Y. Ghadar, C. Knight, and J. Deslippe., "Evaluating perfor-mance portability of openmp for snap on nvidia, intel, and amd gpu's using the roofline methodology," in *IEEE International Workshop on Accelerator Pro-gramming Using Directives*, 2020, pp. 3–24.