

Ready for the Frontier: Preparing Applications for the World’s First Exascale System ^{*}

Reuben D. Budiardja^[0000-0003-0395-8532], Mark Berrill^[0000-0002-4525-3939], Markus Eisenbach^[0000-0001-8805-8327], Gustav R. Jansen^[0000-0003-3558-0968], Wayne Joubert^[0000-0003-4771-998X], Stephen Nichols^[0000-0003-3484-2735], David M. Rogers^[0000-0002-5187-1768], Arnold Tharrington^[0000-0002-2877-8768], and O. E. Bronson Messer^[0000-0002-5358-5415]

Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA
{reubendb^{***}, berrillma, eisenbachm, jansengr, joubert, nicholsss, rogersdm, arnoldt, bronson}@ornl.gov

Abstract. Frontier, a supercomputer at the Oak Ridge Leadership Computing Facility (OLCF), debuted atop the Top500 list of the world’s most powerful supercomputers in June 2022 as the very first computer to produce exascale performance. Making sure scientific applications are optimized on this architecture is the critical link necessary to translate the newly available computational power into scientific insight and solutions. To that goal, the OLCF developed the Center for Accelerated Application Readiness (CAAR) program to ensure that a suite of highly optimized applications is ready for scientific runs at the onset of production operations for Frontier. This paper describes our experience in porting and optimizing such suite of applications in the OLCF’s CAAR program.

1 Introduction and Background

Frontier debuted atop the Top500 list of the world’s most powerful supercomputers in June 2022 [5] as the very first computer to produce exascale performance. The machine also represents the latest iteration of performance for hybrid CPU-GPU supercomputers that was initiated with the arrival of Titan at the Oak Ridge Leadership Computing Facility (OLCF) in 2012. Since Titan’s debut, GPU-enabled scientific computing has moved from a somewhat exotic and perhaps niche methodology for a limited number of algorithms and codes to become the dominant method to achieve maximum performance on today’s most computationally demanding applications.

^{***} Corresponding author

^{*} **Notice:** This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

Making sure scientific applications are optimized on this architecture is the critical link necessary to translate the newly available computational power into scientific insight and solutions. Herein we provide the details of code porting and optimization work done to take maximum advantage of Frontier’s new exascale architecture.

It was recognized during the project to build Titan at the beginning of the last decade that substantial effort would be needed to bring scientific applications to the point of effective use of hybrid CPU-GPU platforms. The Center for Accelerated Application Readiness (CAAR) was then formed to carry out this program of work. CAAR is a development program designed to ensure that the OLCF’s hybrid CPU-GPU platforms deliver on their promise to advance scientific discovery. Following on from an initial small set of applications during the Titan project, CAAR has developed into a larger program with competitive proposals resulting in the formation of integrated teams of code stakeholders and developers, OLCF personnel, and vendor Center of Excellence collaborators brought together to work concertedly on a set of established application codes.

The experience of the OLCF and partners in previous instantiations of CAAR have been provided in [9] and [15]. Like these earlier experiences, the CAAR program for Frontier has a number of related, but distinct, aims. First is to have a suite of highly optimized applications ready for scientific runs at the onset of production operations for Frontier. Via close collaboration with teams dedicated to the development of the Frontier programming environment and tools, CAAR also serves to improve the environment for users and application developers in future, non-CAAR projects. Furthermore, CAAR serves as a laboratory for OLCF staff to develop platform-specific expertise, enabling a smooth transition to and effective support of user programs on Frontier. The experiences of CAAR teams are also translated into a robust training program and used to produce documentation of best practices on the machine.

An important ingredient in the optimization work undertaken by CAAR teams is the formulation of a “challenge problem” to be undertaken on Frontier at the close of development work. Although most modern scientific application codes are under near-constant development and may have a variety of specific simulation aims, the choice of a singular simulation target has proven invaluable to focus the optimization work. Nevertheless, the use of portable and maintainable approaches is a primary design criterion. All of these concerns are, of course, also highly dependent on the specifics of the Frontier platform. Another ingredient of a CAAR project is the formulation of a Figure of Merit (FOM) to allow us to have a quantitative measure of the development work. The FOM represents a performance of a CAAR application. At its simplest, a FOM is a measure of the amount of useful work the code can do per unit of time. A baseline FOM was taken on OLCF Summit at the beginning of the project, with the anticipation that a final FOM will be produced on Frontier at the end of the project. The final FOM captures both software- and hardware- improvements.

This paper is organized as follows. In the next section we briefly describe the pertinent OLCF systems for this work: Summit and Frontier. In Section 3 we describe the CAAR applications—their scientific domains, algorithmic motifs, performance characteristics—and the porting and optimization work for these applications for Frontier. We share some initial and early performance results from running these

applications on Frontier and Crusher, an early access system with identical hardware to Frontier. In Section 4 we share some lessons learned and optimization techniques common to the applications discussed here. We close by providing concluding remarks in Section 5.

2 Systems Overview

2.1 Summit

Summit is a production system at the OLCF. Debuted as the number one system in the June 2018 Top500 list, it currently sits as the fifth in the latest Top500 list with the theoretical peak speed at 200 Petaflops.

Summit consists of over 4608 IBM Power System AC922 compute nodes interconnected by a dual-rail EDR InfiniBand network in a non-blocking fat-tree network topology. Each of Summit compute node has two IBM Power9 CPUs and six NVIDIA Volta V100 GPUs. Three GPUs and one CPU are interconnected with NVLink, while the two CPUs are connected by an X-Bus. Each Power9 has 22 cores and is connected to 256 GB DDR4 DRAM, with one core reserved for operating system tasks. Each of the V100 GPUs is connected to a 16 GB High-Bandwidth Memory (HBM). More detailed descriptions on Summit architecture, including the measured speed-and-feeds of the system, can be found in the OLCF User Documentation [19].

Several programming models are supported on Summit: CUDA for GPU programming, OpenMP for multithreading on CPUs, OpenMP offload and OpenACC for directive-based GPU programming. Users have also built portability layers such as Kokkos and Raja on top of these programming models. Common numerical and I/O libraries are also provided as environment modules.

To aid with the development and porting work to Frontier, HIP is also provided on Summit. On systems with NVIDIA GPUs such as Summit, HIP acts as thin portability layers which then called the underlying CUDA compiler (e.g. `nvcc`). HIP-provided tools for porting are also available on Summit. Prior the deployment of Frontier and its early access systems, Summit was the primary development system for the CAAR program.

2.2 Frontier

Frontier supercomputer consists of 9408 HPE/Cray EX compute nodes and several service nodes. Each compute node has a single 64-core AMD EPYC 7A53 “Optimized 3rd Gen EPYC” processor and four AMD Instinct™ MI250X Accelerator. Each of the MI250X Accelerators consists of two Graphics Compute Dies (GCDs) and are being presented to application as two devices (i.e. two GPUs). Therefore from an application perspective, eight GPUs (i.e. GCDs) are available to use. Each GCDs has 64 GB of high-bandwidth memory while the CPU is equipped with 512 GB DDR4 memory. The CPU and GPUs are interconnected with AMD Infinity Fabric, allowing peak bandwidths of 36 GB/s and 200 GB/s between host-and-device and device-to-device, respectively. A unique architectural feature of Frontier is that the four network interfaces are directly connected to the accelerators. The HPE Slingshot interconnect provides inter-node connectivity in a dragonfly topology providing 100 GB/s network bandwidth.

Frontier’s programming environment (PE) includes AMD ROCm and ROCm libraries [1] with HIP for GPU programming. HIP, similar to CUDA, is a C++ extension and runtime API to allow developers to write computational kernels for GPUs. HIP’s similarity to CUDA, other than pattern-discernible name changes to the API and library routine calls, allows for a straightforward porting of kernels written in CUDA. The HPE Cray provided PE also includes the Cray Compiling Environment (CCE) and AMD compilers capable of OpenMP for multithreading and offload to the GPUs. As with Summit, common numerical and I/O libraries are available via environment modules (see [19]).

The OLCF also provides an early-access system called Crusher. Crusher is virtually identical to Frontier except for its size: Crusher has 192 compute nodes total.

3 Applications

3.1 CoMet

The CoMet (Combinatorial Metrics) application [10] computes similarity metrics between vectors stored in large datasets for the purpose of solving clustering problems in areas such as genomics, climate, bioenergy and pandemics [13]. CoMet searches a very large combinatorial space in order to find clusters manifesting strong similarity relationships that indicate correlation characteristics of scientific interest. Its primary computational expense involves mixed precision GEMM operations comprising up to 90% or more of compute cycles for a typical run.

CoMet was initially ported to AMD GPUs using the HIP interface. Rather than using the HIP interoperability layer, the ported code uses `#ifdefs` in selected locations to allow compilation for either ROCm HIP or CUDA, thus allowing more controlled usage of the relevant libraries such as cuBLAS and rocBLAS. Subsequent development work focused on performance optimizations for the Frontier platform, primarily centered on the highly computationally intensive 3-way vector clustering code path. First, the formation of the matrix taken as input to the GEMMs was moved to the GPU, in keeping with the continuing theme of moving increasingly more computations to the GPU. Second, an algorithm change was made enabling the number of GEMMs required per result to be reduced from three to two with no change in the final answer, resulting in up to 50% performance improvement (see [13]). Finally, the thresholding process for the correlation metrics, which is used to discard all results except for the very small fraction of highly correlated values, was moved to the GPU. This not only made it possible to apply lossless compression (via the AMD rocPRIM library) to the results for much faster GPU-CPU transfer performance but also greatly reduced the CPU memory footprint for storing the metrics, making it possible to solve much larger problems.

Preliminary performance results on Frontier are shown in Figure 1 for the 3-way CCC method, showing near-perfect linear scaling. CoMet has achieved over **6.71 Exaflops** (FP16/FP32 mixed precision) at scale on Frontier. The FOM or figure of merit for CoMet is measured as the rate of science output in units of vector element comparisons per second of runtime. On Frontier, CoMet achieved 419.9 quadrillion comparisons/second on 9,074 compute nodes, a factor of **5.16X** faster than the Summit baseline

of 81.2 quadrillion comparisons per second, this speedup resulting from the combination of algorithmic improvements and increase in mixed precision flop rate of Frontier over Summit.

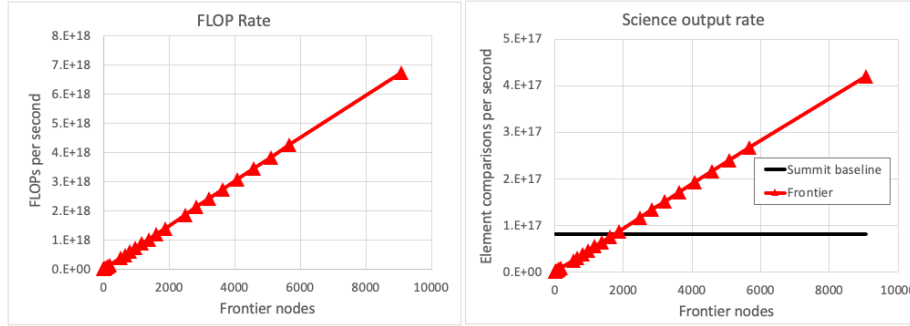


Fig. 1: CoMet Frontier scaling performance

3.2 Cholla: Computational Hydrodynamics on Parallel Architecture

Cholla is a hydrodynamics code with radiative cooling originally written to run natively on NVIDIA GPUs [25]. It has since evolved and is acquiring additional physics modules relevant for astrophysics and cosmological simulations such as self-gravity solver, particle tracking, and magneto-hydrodynamics. Cholla computational kernels are written in CUDA, with the CPU responsible only to manage inter-process communications via MPI and data movement between CPU and GPU memory. Prior to this project, Cholla has been productively used to perform scientific simulations with published results (see for example, [24] and references therein).

The challenge problem for this CAAR project is to perform simulations of a Milky Way-like galaxy that allow for self-consistent star formation and feedback within interstellar medium. Motivated by recent observational campaigns that have drastically increased the availability of exquisite data about the Milky Way, computational astrophysicists are eager to have simulation results that can reproduce the observational data at the requisite resolutions to be able to make detailed predictions of the galaxy evolution. Considering factors such as the size of the galaxy and star clusters, the target resolutions for such simulations are approximately 10000^3 covering about fifty parsecs of computational domain.

For this project, we define the FOM as $FOM = \frac{nCells \times nCycles}{Walltime}$, where $nCells$ is the total number of cells, $nCycles$ is the number of cycles, and $Walltime$ is the elapsed wall-clock time in seconds.

Porting Cholla to Frontier is relatively straightforward due to the similarity of CUDA and HIP. Every CUDA library routine used by Cholla maps to a corresponding HIP library routine, allowing simple name changes for porting. We started by using the Hipify tool to transition to HIP. The following considerations, however, urged us to come up

with an alternative approach. If we had ported everything to HIP as a separate code base, we will be burdened with maintaining two codes: HIP-based and CUDA-based. If we had ported to HIP in-place, Cholla’s users on CUDA systems will have the additional burden of installing HIP¹. Even if performance impact is minimal, we felt that this is a significant burden to users on CUDA system.

Our approach is to instead have a simple conditional compilation file—using C preprocessor—that does the translation from CUDA to HIP calls based on a compilation flag. The small number of CUDA library calls used in Cholla made this a relatively simple process².

When Cholla was originally written, the GPU high-bandwidth memory size was typically only a fraction of the available host memory. It therefore utilized the “subgrid splitting” technique—where the grid is split into multiple blocks, copied into GPU memory on which computations are performed, and copied back to the CPU consecutively—to fit larger volume than what was possible on GPU memory only. This assumption is no longer true on Frontier, where the size of GPU memory is equal to that of host memory. By removing the subgrid splitting feature, we actually introduced a large speed up by virtue of removing data movement to and from GPU memory while at the same time simplifying the code.

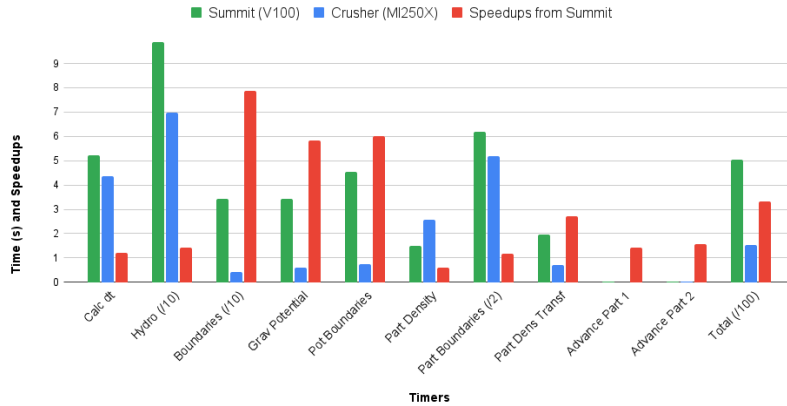


Fig. 2: Timing (green and blue bars) and Speedups (red bars) of computational portions for Cholla on Summit’s Nvidia V100 (green) and Crusher’s AMD’s MI250X GCD (blue). The timing for Hydro, Boundaries, Gravitational potential, Particle boundaries are scaled down by factor of 10, 10, 100, 2, and 100 respectively to fit the scale of the chart.

Keeping the data persistently on GPU memory is also motivated by another architectural feature of Frontier: hardware- and software-supported GPU-aware MPI. Since

¹ On CUDA systems, HIP acts as a thin portability layers that then calls the CUDA compiler `nvcc`.

² How we do this can be observed from the file available from the Cholla public repository: <https://github.com/cholla-hydro/cholla/blob/main/src/utils/gpu.hpp>

Frontier’s network interfaces are connected directly to the GPU memory, it is more efficient for communications to be done on data residing on GPU memory. We modified Cholla to utilize GPU-aware MPI for the data already residing on GPUs.

Figure 2 shows timings and speedups of computational portions of Cholla for the FOM problem. This test problem was run with 64 GPUs on Summit NVIDIA V100 and 64 GCDs on Crusher’s AMD MI250X. This comparison was performed after all the software changes discussed above. As we can see from the plot, most of the computational portions show some speedup on AMD MI250X GCD, with some achieving more than 5X speedup. The total speedup is more than a factor of 3X. We attribute this to mostly two factors. First is the fact that the network interfaces are directly attached to the GPUs. The computational portions that get the most speedups are communication-heavy. The second contributing factor is the higher memory-bandwidth available on Frontier’s hardware.

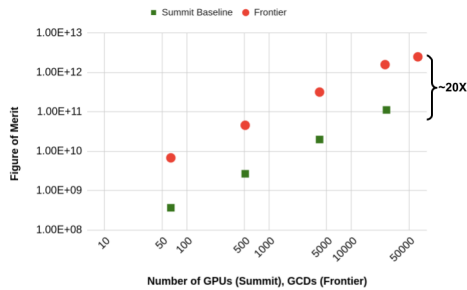


Fig. 3: Cholla FOM as a function of number of GPUs (Summit) or GCDs (Frontier).

Figure 3 shows the total speedup on Frontier over baseline runs on Summit. Note that the baseline runs were performed with the version of the code prior to the developments described above. The FOM speedups on Frontier’s 64,000 AMD MI250X GCDs is 20X versus the baseline code on Summit with 26,624 NVIDIA V100 GPUs. From the results shown in Figure 2, we can infer that software developments contribute a speedup over $\sim 4X$, while hardware improvements contribute at least $\sim 4X$ to make up the total 20X speedups.

3.3 GESTS: GPUs for Extreme-Scale Turbulence Simulations

The GESTS project performs direct numerical simulations (DNS) of turbulent fluid flows across a wide range of scales according to the Navier-Stokes equations. The algorithms developed by GESTS use a Runge-Kutta scheme in time and a Fourier-spectral representation in space to create a pseudo-spectral method that computes the nonlinear terms in physical space and performs all other computations in wave number space. The original GPU-enabled algorithm along with previous results from Summit are presented in [23].

The basic elements of the GESTS algorithm include a domain decomposition among the MPI processes, transposes of the solution domain to allow FFTs to be taken in each direction, local data movements arising from the non-contiguous nature of messages for the required all-to-all communication, as well as data movements between the host and device memory. This 3D FFT problem is in fact well known for being communication intensive, which can limit scalability at large problem sizes. The challenge—which is shared by many other user application codes—is to allow a communication-intensive algorithm to benefit fully from heterogeneous platforms whose principal advantage is fast computation.

The GESTS codes are written in modern Fortran using MPI for communication and OpenMP for CPU multi-threading and GPU offloading. The codes are formed around a custom-built 3D FFT algorithm that computes the FFTs on both CPUs and GPUs via FFTW (for CPUs), CUDA cuFFT (for NVIDIA GPUs), or ROCm rocFFT (for AMD GPUs). OpenMP offloading functionality is used to manage data movement between the host and device, to enable GPU-Direct MPI communications, and to accelerate a variety of array operations on the GPUs.

Two variants of codes have been developed: a “Slabs-” and a “Pencils-” decomposition (see Figure 4). For a N^3 problem across P MPI processes, the “Slabs” decomposition cuts the domain in only one of the three dimensions at a time to form P partitions. This decomposition allows the code to perform the FFT computations for two of the three dimensions before requiring expensive MPI communications to form contiguous data for the FFT computations in the third dimension. Since each slab must contain a full 2D slice of the domain, the “Slabs” code can have a maximum of $P = N$ MPI processes, which can be a critical limitation for very large problems.

The “Pencils” decomposition divides the domain in two dimensions at a time. This decomposition allows the use of up to N^2 MPI processes, but requires an expensive MPI communication between each FFT computation since only one direction is contiguous at a time. Specifically, for an N^3 problem across P MPI processes, one dimension is divided into P_r pieces and a second dimension is divided into P_c pieces such that $P = P_r \times P_c$. Figure 4 demonstrates both decomposition approaches.

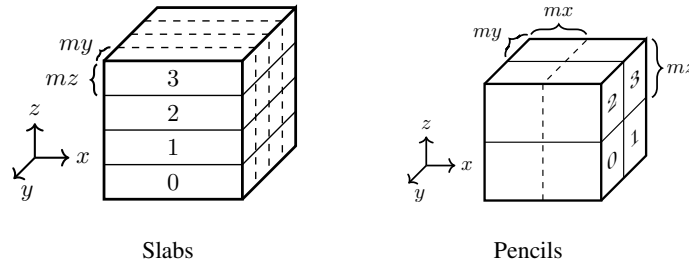


Fig. 4: Slab vs Pencil Decomposition

As seen in Tables 1 and 2, the MPI communications consume approximately 70% of the runtime for the multi-node jobs while the forward and inverse FFT transforms in the three coordinate directions account for roughly 20% of the runtime. Virtually all of the MPI communications are used to transpose the arrays into contiguous pencils for the FFT computations. As a result, the 3D FFT algorithm consumes upwards of 90% of the runtime for the DNS codes. This demonstrates clearly how efficient DNS computations depend critically on a highly performant 3D FFT algorithm.

The Figure of Merit (FOM) chosen for GESTS is defined as the total number of grid points in the simulation divided by the average time to compute each time step and is presented in Equation 1

$$FOM = \frac{N^3}{\Delta t} \quad (1)$$

N^3	Nodes	Ranks	FFT(s)	Pack+Unpack(s)	MPI(s)	Other(s)	Total(s)
2048 ³	1	8	1.364	0.243	1.841	0.635	4.083
4096 ³	8	64	1.432	0.246	6.930	0.658	9.266
8192 ³	64	512	1.583	0.255	8.110	0.672	10.620
16384 ³	512	4096	2.200	0.269	8.427	0.675	11.571
32768 ³	4096	32768	3.420	0.276	9.220	0.704	13.62

Table 1: Timings for "Slabs" code on Frontier

N^3	Nodes	P_r	P_c	FFT(s)	Pack+Unpack(s)	MPI(s)	Other(s)	Total(s)
2048 ³	1	2	4	1.820	1.092	2.333	0.590	5.835
4096 ³	8	2	32	1.643	0.786	8.070	0.791	11.290
8192 ³	64	2	256	1.454	0.610	8.760	0.686	11.510
16384 ³	512	2	2048	2.340	0.600	9.460	0.730	13.130
32768 ³	4096	4	8192	3.385	0.608	10.470	1.337	15.800

Table 2: Timings for "Pencils" code on Frontier

The reference FOM was computed on Summit as part of an INCITE 2019 project [23] and is provided in Table 3 along with timings from Frontier (see Tables 1 and 2). As Table 3 shows, GESTS sees a $> 5x$ speedup on Frontier on 4096 nodes for both decomposition strategies which exceeds the CAAR project goal of a $4x$ speedup. Tests are ongoing for $N = 32768$ on 8192 Frontier nodes with the "Pencils" code, and full production simulations are expected in the near future. These 32768³ cases are the largest known DNS computations to date with a point total in excess of 35 trillion grid points. Frontier is the only machine in the world with the memory capacity to complete these simulations.

Machine	Decomposition	#GPUs	N^3 (points)	$\Delta\tilde{t}$ (sec)	FOM (points/sec)	Speedup
Summit	Slab	18432	18432 ³	14.24	4.398×10^{11}	—
Frontier	Slab	32768	32768 ³	13.62	2.583×10^{12}	5.87x
Frontier	Pencil	32768	32768 ³	15.8	2.227×10^{12}	5.06x

Table 3: Comparison of FOM on Summit and Frontier

3.4 LBPM: Lattice Boltzmann Methods for Porous Media

The LBPM software package relies on lattice Boltzmann methods to model transport processes in systems with complex microstructure, including flows through porous media, multiphase flow, and membrane transport processes. Microscope image data—such as 3D data from x-ray micro-computed tomography that reveals the internal structure of complex materials—is commonly used to specify input geometries for LBPM. Digital rock physics is a core capability of LBPM, particularly direct pore-scale simulations of

two-fluid flow through geological materials such as rock or soil. LBPM is freely available through the Open Porous Media project [7; 17]. The LBPM multiphase flow solver uses a color Lattice-Boltzmann model that is defined by a set of three lattice Boltzmann equations (LBEs), corresponding to one momentum transport equation and two mass transport equations. The LBEs are defined based on a quadrature scheme to discretize the velocity space in the continuous Boltzmann transport equation. LBPM includes a thread-based framework to carry out in situ analysis of the flow behavior. The analysis framework is configured to compute integral measures from the flow that capture essential aspects of the behavior. In addition, the analysis framework can identify and track connected and disconnected parts of the fluid performing the integral analysis over these individualized sub-regions [18].

The LBPM code is written in object-oriented C++ using MPI for communication. Analysis routines and I/O are handled in separate CPU threads using the native C++11 thread capabilities. The code used CUDA for the initial GPU capabilities. While the GPU routines represent the computationally expensive portion of the physics, analysis and IO routines are performed on the CPU. The porting effort focused on updating GPU routines to HIP to target the AMD GPUs. The majority of the simulation data remains on the GPU during the calculations over all timesteps with communication from the GPU to the host occurring every n-th timestep for analysis or IO needs. The communication between nodes utilizes pack/unpack routines that are implemented on the GPU to avoid copying the entire domain to the host and the communication can take advantage of the GPU direct MPI communication. The code includes a number of unit tests that were leveraged throughout the porting process to ensure correctness and to test the performance of individual routines such as communication.

To compare performance across multiple architectures and scale the project defined a FOM (Figure of Merit) defined as the number of Millions of Lattice Updates Per Second (MLUPS). Table 4 shows the weak scaling performance data obtained on Summit. Weak scaling demonstrated 80% scaling efficiency on Summit at full machine using the original version of the code. Table 5 shows the weak scaling performance data obtained on Crusher after porting to AMD GPUs.

Ranks	Nodes	MFLUPS (per rank)	MLUPS (total)
6,144	1024	384	2.36e6
12,288	2048	369	4.54e6
24,576	4096	313	7.71e6

Table 4: Performance on Summit

3.5 LSMS

The Locally-Selfconsistent Multiple Scattering (LSMS) code implements a scalable approach for first principles all-electron calculations of alloys and magnetic solid state

Ranks	Nodes	MLUPS (per rank)	MLUPS (total)
1	1	640	640
8	1	610	4880
64	8	594	37,952
512	64	595	304,640

Table 5: Performance on Crusher

systems. It is available under an open source license [14]. LSMS solves the Schrödinger equation for the electrons inside a solid using the Kohn-Sham density functional theory (DFT) formalism [12]. This transforms the interacting many electron problem into a tractable effective one-electron problem. In the transformed problem, the many body effects are captured by the exchange-correlation functional, for which various approximations are available. While most widely used DFT codes diagonalize the Kohn-Sham Hamiltonian, LSMS is inspired by the Korringa-Kohn-Rostoker method [20] and it employs a real space multiple scattering formalism to calculate the electronic Green’s function.

LSMS calculates the local spin density approximation to the diagonal part of the electron Green’s function. The electron and spin densities and energy are readily determined once the Green’s function is known. Linear scaling with system size is achieved in LSMS by using several unique properties of the real space multiple scattering approach to the Green’s function, namely: 1) the Green’s function is “nearsighted”, therefore, each domain, *i.e.* atom, requires only information from nearby atoms in order to calculate the local value of the Green’s function. 2) the Green’s function is analytic everywhere away from the real axis, therefore, the required integral over electron energy levels can be analytically continued onto a contour in the complex plane where the imaginary part of the energy further restricts its range; and 3) to generate the local electron/spin density an atom needs only a small amount of information(phase shifts) from those atoms within the range of the Green’s function. The very compact nature of the information that needs to be passed between processors and the high efficiency of the dense linear algebra algorithms employed to calculate the Green’s function are responsible for the scaling capability of the LSMS code. The LSMS code is written in C++ with a few remaining legacy routines written in Fortran and it utilizes MPI for communication as well as OpenMP for multi-threaded CPU execution. GPU acceleration is achieved through CUDA or HIP kernels as well as through the use of dense linear algebra libraries.

For LSMS we defined a figure of merit that captures both strong and weak scaling opportunities as well as increases in the physical accuracy captured during the calculations. Thus FOM_{LSMS} combines the number of atoms in the simulation N_{atom} , the number of energies on the integration contour $N_{energies}$, the number of sites in the local interaction zone N_{LIZ} , the maximum angular momentum l_{max} and time per selfconsistency iteration in seconds $t_{iteration}$ as $FOM_{LSMS} = N_{atom}N_{energies} ((l_{max} + 1)^2N_{LIZ})^3 / t_{iteration}$. The results for scaling runs on Frontier are shown in table 6.

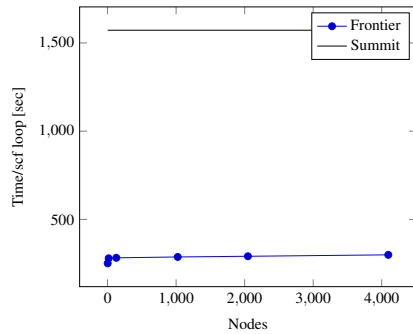
The port of LSMS to the AMD GPU architecture of Frontier builds on our GPU implementation for the Titan system at OLCF [9]. The main kernel that accounts for more

Atoms	Nodes	FOM	FOM/Node	FOM(Frontier)/FOM(Summit)
128	2	2.11E+12	1.05402E+12	6.270912661
1024	16	1.51E+13	9.464E+11	5.63065207
8192	128	1.20E+14	9.37336E+11	5.576724997
65532	1024	9.43E+14	9.21091E+11	5.480073895
65532	2048	1.81E+15	8.85586E+11	5.268835897
65532	4096	3.49E+15	8.52815E+11	5.073863288
131072	2048	1.86E+15	9.09072E+11	5.408568929
131072	4096	3.62E+15	8.84375E+11	5.261631366

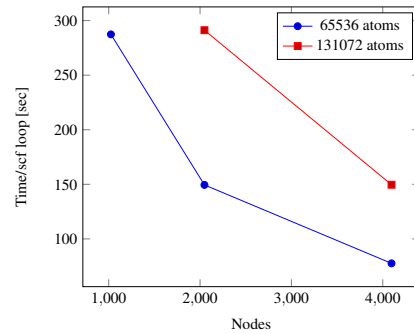
Table 6: FOM results for LSMS on Frontier and comparison to Summit.

than 95% of the floating point operations in a typical LSMS run calculates the block diagonal parts of the Green’s function in the local interaction zone approximation. This requires the calculation of a small block of the inverse of a dense non-Hermitian complex matrix. This task can be readily solved using the dense matrix multiplication, LU factorization and solver routines in the rocBLAS and rocSolver libraries. Additional kernels that construct the matrices that enter the solvers are written in HIP. While these follow the same structure as the equivalent CUDA kernels, the performance was significantly improved by rearranging the matrix index calculations to alleviate the contention between integer and floating point operations on the AMD accelerators.

With the ports to AMD GPUs described above for LSMS we have obtained early scaling and performance measurements on Frontier. In figure 5a we show the weak scaling of FePt calculations from 2 nodes to 4096 nodes with 64 atoms per node (8 atoms per GPU). For comparison we show the time on Summit for a single node with the same number of atoms per GPU. Additionally we also have early strong scaling results for 65,536 and 131,072 atom systems on Frontier that are shown in figure 5b. These results indicate that, together with the CAAR improvements in LSMS, we can, at present, expect an $\approx 5.5\times$ per node speedup on Frontier when compared to Summit.



(a) Weak scaling of LSMS for FePt ($l_{max} = 7$) with 64 atoms per node from 2 to 4096 Frontier nodes.



(b) Strong scaling of LSMS for FePt ($l_{max} = 7$) from 1024 to 4096 Frontier nodes.

Fig. 5: Weak- and strong-scaling of LSMS.

3.6 NUCCOR/NTCL

NUCCOR (Nuclear Coupled-Cluster Oak Ridge) is a nuclear physics application designed to compute properties of atomic nuclei from first principles using high-performance computing resources at Oak Ridge National Laboratory (ORNL). From its inception at the start of the millennium [3], it has changed the perception of ab-initio nuclear physics from impractical and too computationally expensive to practical and state-of-the-art. Since NUCCOR uses the coupled-cluster method, a method that scales polynomially with the number of particles present in the atomic nucleus, it has a significant advantage over other competing methods that only scale exponentially. With the exponential growth in the availability of high-performance computing resources at ORNL over the past two decades, NUCCOR has gone from computing properties of oxygen-16, a nucleus with only eight protons and eight neutrons, in 2004 [3], to lead-208, a nucleus with 82 protons and 126 neutrons, in 2022. Frontier will increase the reach of ab-initio nuclear theory to encompass all atomic nuclei and test the theoretical foundations that define low-energy nuclear physics.

NUCCOR solves the time-independent Schrödinger equation for many interacting protons and neutrons, collectively called nucleons, using the coupled-cluster method. The coupled-cluster method rewrites the Schrödinger equation as an eigenvalue problem in a finite basis set. By constructing a similarity transformation for the Hamiltonian matrix using a fixed-point iteration, the eigenvalue problem can be truncated to a smaller basis set and solved at a lower computational cost. However, the basis set is still too large for the eigenvalue problem to be solved exactly, so Krylov sub-space methods, like Arnoldi and non-symmetric Lanczos, are used to extract the lowest eigenpairs to a specified precision.

Due to the inherent symmetries in an atomic nucleus, block-sparse tensor contractions dominate the computational cost of the NUCCOR application. Each iteration consists of multiple tensor contraction terms, where each term is block-sparse. NUCCOR exploits the sparsity patterns to convert them into a series of local, dense tensor contractions of different dimensions. The number of tensor contractions and the dimensions of each contraction depend on the nucleus and the size of the chosen basis set. Typically, the size of the basis set is on the order of thousands, while the dimensions of the tensor contractions vary from less than ten to several million. A balanced distribution scheme among MPI ranks is critical for excellent performance on Summit and Frontier, as well as a highly tuned tensor contraction library.

NUCCOR uses the Nuclear Tensor Contraction Library (NTCL) [8] to perform distributed block-sparse tensor contractions and dense local tensor contractions. NTCL presents an architecture-independent API to the user and supports multiple hardware backends using a plugin structure. For example, the HIP backend supports Crusher and Frontier, while the CUDA backend supports Summit. For the CAAR project, we ported

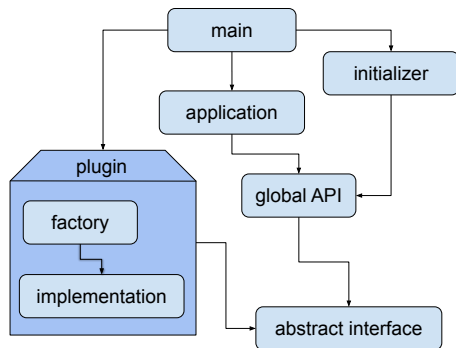


Fig. 6: NTCL implements a plugin structure using the strategy and abstract factory patterns. The application knows only the architecture-independent API, while the main program unit initializes the architecture-dependent API.

the CUDA plugin to HIP, which was very straightforward. After a search-and-replace operation, we only had to perform minor adjustments, primarily to accommodate 64 work items in a wavefront, compared to 32 threads in a warp using CUDA.

NTCL and NUCCOR are written in modern Fortran and use features from the Fortran 2018 standard. Internally, NTCL implements the plugin structure using a standard strategy pattern to encapsulate memory, dependencies, and algorithms. In addition, an abstract factory pattern enables the user to write plugin-independent code for maximum efficiency. Figure 6 shows the overall structure.

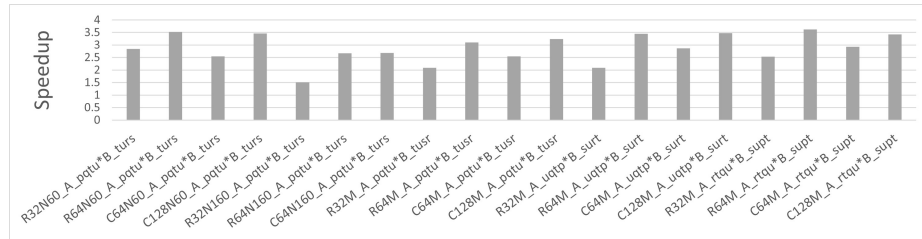


Fig. 7: Tensor contraction speedups in NTCL using a single GCD of the MI250X on Crusher, compared to a single V100 on Summit for different tensor shapes and sizes. The column label denotes the datatype, the tensor dimensions, and the tensor contraction type in Einstein summation notation.

Figure 7 shows preliminary performance results from Crusher. Each column identifies the speedup from an NV100 on Summit to a single GCD on the MI250X on Crusher. We need Frontier to run a representative use case for NUCCOR. If we run a use case appropriate for Crusher, it will result in many small tensor contractions and skew the performance results. Instead, we show a selection of tensor contractions that are important for the figure of merit – the number of tensor contraction operations per second.

3.7 NAMD

NAMD [22] is a C++ molecular dynamics (MD) code that simulates atomistic biological systems on CPU and GPU architectures. It has demonstrated scalable performance on systems consisting of millions of atoms over several hundred thousand cores [21]. NAMD is implemented with Charm++ [11], a message passing parallel programming framework that provides migratable objects, asynchronous methods, and an adaptive runtime system.

As is typical for molecular dynamics codes, the calculation of forces, pairwise 2-body non-bonded, bonded, angle, and dihedral interactions are the computational bottleneck. To reduce the time-to-solution, NAMD uses a spatial-domain decomposition to partition the force calculations over the processor. This had been ported to CUDA.

The methodology of porting NAMD to Frontier was to use HIP. Because HIP and CUDA kernels are syntactically the same, we used a common header file that defines C preprocessor macros to redefine CUDA calls to HIP calls such as the ones shown below:

```
#define cudaGetDevice hipGetDevice
#define cudaStream hipStream
```

HIP does have limitations in adapting to CUDA's API (these limitations are listed in the HIP documentation). Consequently, it is advisable to keep the CUDA code simple to avoid these limitations.

Another difference between HIP and CUDA kernels is that the warp size in CUDA is 32 while that for HIP is 64. Many NAMD kernels had hard-coded the warp size which resulted in deleterious performance effects when run on the AMD GPUs. As part of this project, we improved the code portability on different GPUs by using a variable to represent warp size.

With AMD devices having larger warp sizes than NVIDIA devices, it is important to carefully consider and benchmark performance effects of problem decomposition over the GPUs. In NAMD, the atom's tiles sized was tightly coupled and set equal to the device warp size. This had the side effect of increasing the number of neighboring atoms in which lowered the percentage of effective interacting pairs. This problem was addressed by decoupling the tile sizes from the device's warp size.

For biological MD simulations long-ranged electrostatics effects are significant and are typically calculated with methods that involve Fast Fourier Transforms (FFTs). We initially attempted to use rocFFT to do this but found out that it did not deliver superior performance to the VkFFT library.

NAMD's use of the Charm++ parallel programming framework requires the Open Fabrics Interfaces (OFI) target on Frontier to perform optimally. Unfortunately, we encountered functionality issue with using OFI on the current Slingshot 11 on Frontier. While we continue to work with the vendor to resolve this issue, we had to resort to using the Charm++ MPI target which is not nearly as performant as the OFI target.

3.8 PIConGPU

PIConGPU is a particle-in-cell code that solves the Maxwell electromagnetic equations simultaneously with the motion of a plasma of electrons [2]. Internally, electric and magnetic fields are stored on a staggered, Yee-grid and updated using finite-difference time-domain methods. Electrons are propagated in time by moving 'macroparticles'. Each macroparticle represents a group of electrons within a deformable spatial shape.

This explicit representation of particles and fields is necessary to create high fidelity models of high-energy plasmas because it contains the full relativistic physics of particle-field interactions. Results from these simulations can be used to parameterize

continuum, density-based models like the Vlasov-Poisson equation. As a time-domain code, PIConGPU’s scaling is governed by a Courant-Friedrichs-Lewy (CFL) condition for the electromagnetic wave velocity.

In the traveling wave electron acceleration experiment (TWEAC)[4], a short, 10-femtosecond burst from two crossed beams of ultraviolet light accelerate electrons to nearly the speed of light. The accelerated electrons leave behind a cavity that can extend 10-s of micrometers. A full simulation of these events requires observing a length scale of 100-s of micrometers over hundreds of femtoseconds, a volume which contains, at minimum, around 10^{14} electrons and 10^{12} grid cells simulated for thousands of time-steps.

For systematically exploring the design of these laser-accelerated plasma setups, it is essential to have a high-performance, parallel application with good weak scaling to large simulation volumes. PIConGPU has shown extremely good weak scaling efficiency, even up to full Frontier scale (Fig. 8). Its strong scaling is essentially limited by needing at least 200^3 cells per GPU in order to saturate the GPU’s compute speed. PIConGPU’s FOM thus measures number of time-steps completed per second times a weighted average of particles (90%) and grid cells (10%).

PIConGPU faced unique challenges increasing its throughput because of several factors. Primarily, the application had already been heavily optimized for GPU computations: GPU-resident data, use of 32-bit floats in most places, and hand-optimization of kernels ranked slowest on existing hardware. On the other hand, more than 90% of its run-time is already spent in kernels, directly enabling the code able to utilize faster GPUs. Frontier’s MI250x GPUs have the same FLOP-rate for 32-bit as 64-bit floats, a factor which doesn’t directly benefit existing use cases. Instead, faster double-precision arithmetic enables simulating previously inaccessible laser setups that require high-fidelity.

Fig. 8 shows PIConGPU’s achieved FOM during a full-Frontier simulation of the TWEAC system. In this particular configuration, the simulation size included $2.7 \cdot 10^{13}$ macroparticles in 10^{12} grid cells. One thousand time-steps completed in a mere 6 and a half minutes. The right panel shows significant variations in the wall-time per simulated time-step. These may potentially be a symptom of network congestion effects. If confirmed, performance will increase with network upgrades.

The average FOM in the run above (65.7 TUpdates / sec) was a factor of 3.9 higher than the Summit benchmark run completed at the start of the project (16.8 TUpdates/sec). Achieving this progress happened in several steps. First, the team added ROCm support to their performance-portability libraries, alpaka and cupla [16]. The team also compiled and tested the I/O backend library, openPMD [6]. The initial test run was 20% slower on a single AMD MI100 GPU³ as it would have on a NVIDIA V100. There were also program crashes observed during development that originated from within the GPU and MPI library and device driver stacks. Once found, these issues were raised with the compiler and hardware vendors, which lead to help troubleshooting, improved libraries, and issue resolution.

Timing GPU kernels was done without any code changes using nvprof and rocprof. It showed that about 92% of the execution time was spent running GPU kernels on

³ MI100 is a previous generation of AMD accelerators.

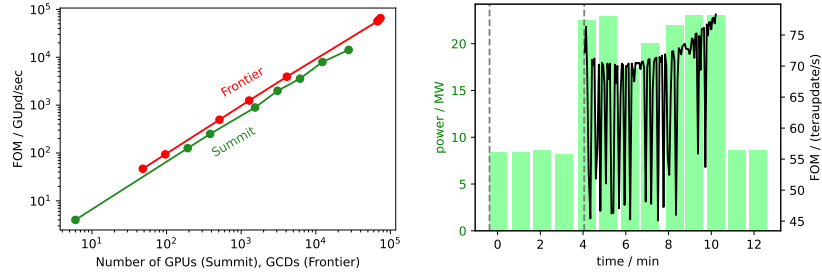


Fig. 8: TWEAC laser simulation weak scaling at 14.6×10^6 cells per GCD (left panel) and (right panel) 9216-node Frontier run showing system power utilization (left scale) and instantaneous figure of merit (units of 10^{12} updates per second, right scale).

both platforms. Individually, all the kernels were faster on MI100 except for the kernel computing the driving ‘background’ laser field. This kernel evaluates sin and cos functions, and requires significant GPU register space. After gaining some a few percent speedup by trying some mathematical reformulations, the developers eventually decided to use time-propagation instead of direct calculation for this field. The result was a 25% speedup in the single MI250x GCD vs. V100 comparison. Scaling this up by the number of GPUs available on Frontier, the overall increase in throughput is 4 times higher than Summit.

4 Lessons Learned

We summarize in this section the common insights and lessons learned from porting and optimizing the CAAR applications on Frontier.

We first note that all of the CAAR applications described in this paper had been “production-ready” applications at the beginning of the program. In other words, these applications have been used to perform simulations with published results and have been fairly well optimized on pre-exascale systems such as Summit. Almost by necessity, this means that these applications were already GPU-accelerated at the beginning of the CAAR program.

For porting from CUDA to HIP, the resemblance of the two language extensions tremendously helped the porting efforts. Although the HIP-ify tool was initially helpful, applications chose to instead use a simpler translation layer via C-preprocessor or header include file. This header file provides a name-translation layer between CUDA and HIP routines (for example, see 3.7 and 3.2). In other cases, developers used OpenMP offloading (GESTS) or reliance on cross-platform libraries (NUCCOR, NAMD, CoMet) to achieve performance portability.

After the initial porting, further optimizations were then obtained by the following techniques common to these applications:

- Abstracting out hardware-dependent limits: GPU thread size, warp size, or work-groups should not be hardwired to the kernel or kernel launch parameters so that they can be changed depending on the system on which the application is being run

- Exploiting larger HBM size: since Frontier has more HBM capacity per GCD, sizing the problem appropriately increases FOM speedups by reducing or eliminating data movement to/from the host DRAM
- Using GPU-aware MPI: with the network interfaces directly connected to the GPUs, MPI communications with data residing on HBM are much more efficient.
- Making data on the GPU as persistent as possible to reduce transfer costs.
- Performing pack/unpack of data on the GPU when appropriate to minimize transfer costs.
- Using OpenMP threads to speed up computations that remain on the CPU.

Beyond this list, specific optimizations due to hardware and algorithmic differences may still be unavoidable (as discussed in §3.1, §3.5, and §3.8).

Some challenges we faced for this work can be attributed to the fact that the programming environments and profiling tools for Frontier were much less mature compared to the ones we had been accustomed to. Through this work and close collaborations with our vendor partners, the programming environments (compilers and software libraries) and profiling tools are now much improved.

5 Conclusions

In this paper, we described the porting and optimization of a suite of applications to be ready for Frontier. Via the CAAR program, this work ensures that these applications are ready to capitalize Frontier’s computational power for new scientific insights on the first day of Frontier’s production period.

With this suite of applications, we show that Frontier delivers on its performance potential. Most applications described here achieve significant FOM speedups compared to their baseline on Summit. While the initial code porting is generally straightforward, architecture-specific optimizations and tuning are also crucial in achieving those speedups. We have described those optimizations in this paper which would serve as guides for future oncoming applications.

The CAAR program has also served its purpose in improving Frontier’s environment by uncovering many initial issues not atypical to a maiden system. The resolutions of these issues benefit applications that will be and are currently being onboarded to Frontier. Meanwhile, we continue to work with our vendor partners to resolve remaining open issues.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

The work described here were collaboratively performed by members of the respective project teams and Frontier’s Center of Excellence. We acknowledge their significant contributions to the success of the Frontier’s CAAR program. They include

P.K. Yeung (Georgia Tech), Rohini Uma-Vaideswaran (Georgia Tech), Kiran Ravikumar (HPE), Steve Abbott (HPE), Matt Turner (HPE), Alessandro Fanfarillo (AMD), Swarnava Ghosh (ORNL), Kariia Karabin (ORNL), Yang Wang (Carnegie Mellon University), Vishnu Raghuraman (Carnegie Mellon University), Franco Moitzi (University of Leoben), Alessandro Fanfarillo (AMD), David Hardy (University of Illinois Urbana-Champaign), Julio Maia (AMD), Josh Vermass (Michigan State University), Tim Mattox (HPE), Morten Hjorth-Jensen (MSU), Gaute Hagen (ORNL), Justin Lietz (ORNL), Rene Widera (Helmholtz-Zentrum Dresden-Rossendorf - HZDR), Klaus Steiniger (HZDR), Sergei Bastrakov (HZDR), Michael Bussmann (HZDR), Fabian Mora (U. Delaware), Richard Pausch (HZDR), Guido Juckeland (HZDR), Jeffrey Kelling (HZDR), Matthew Leinhauser (U. Delaware), Jeffery Young (Georgia Tech.), Franz Pöschl (HZDR), Alexander Debus (HZDR), Sunita Chandrasekaran (U. Delaware), Evan Schneider (U. Pittsburgh), Bruno Villasenor (U. California Santa Cruz, AMD), Brant Robertson (U. California Santa Cruz), Robert Caddy (U. Pittsburgh), Alwin Mao (U. Pittsburgh), Trey White (HPE), Dan Jacobson (ORNL), Jakub Kurzak (AMD).

Bibliography

- [1] AMD: New amd rocm information portal. <https://rocm-docs.amd.com/en/latest/> (2022), accessed: 2022-06-01
- [2] Bussmann, M., Burau, H., Cowan, T.E., Debus, A., Huebl, A., Juckeland, G., Kluge, T., Nagel, W.E., Pausch, R., Schmitt, F., Schramm, U., Schuchart, J., Widera, R.: Radiative signatures of the relativistic kelvin-helmholtz instability pp. 5:1–5:12 (2013). <https://doi.org/10.1145/2503210.2504564>
- [3] Dean, D.J., Hjorth-Jensen, M.: Coupled-cluster approach to nuclear physics. *Phys. Rev. C* **69**, 054320 (May 2004). <https://doi.org/10.1103/PhysRevC.69.054320>
- [4] Debus, A., Pausch, R., Huebl, A., Steiniger, K., Widera, R., Cowan, T.E., Schramm, U., Bussmann, M.: Circumventing the dephasing and depletion limits of laser-wakefield acceleration. *Phys. Rev. X* **9**, 031044 (Sep 2019). <https://doi.org/10.1103/PhysRevX.9.031044>
- [5] Dongarra, J., Strohmaier, E., Simon, H., Meuer, M.: *TOP500*. <https://www.top500.org/lists/top500/2022/11/> (2022)
- [6] Huebl, A., Lehe, R., Vay, J.L., Grote, D.P., Sbalzarini, I.F., Kuschel, S., Sagan, D., Mayes, C., Perez, F., Koller, F., Bussmann, M.: openPMD: A meta data standard for particle and mesh based data (2015). <https://doi.org/10.5281/zenodo.591699>, DOI:10.5281/zenodo.591699
- [7] Initiative, T.O.P.M.: Open porous media project. <https://opm-project.org/> (2022), accessed: 12-30-2022
- [8] Jansen, G.R.: NTCL – nuclear tensor contraction library. <https://gitlab.com/ntcl/ntcl> (2022)
- [9] Joubert, W., et al.: Accelerated application development: The ornl titan experience. *Computers & Electrical Engineering* **46**, 123–138 (2015). <https://doi.org/https://doi.org/10.1016/j.compeleceng.2015.04.008>

- [10] Joubert, W., Weighill, D., Kainer, D., Climer, S., Justice, A., Fagnan, K., Jacobson, D.: Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 717–730 (2018). <https://doi.org/10.1109/SC.2018.00060>
- [11] Kalé, L.V.: Charm++, pp. 256–264. Springer US, Boston, MA (2011)
- [12] Kohn, W., Sham, L.J.: Self-consistent equations including exchange and correlation effects. *Phys. Rev.* **140**, A1133–A1138 (Nov 1965)
- [13] Lagergren, J., Cashman, M., Melesse Vergara, V., Eller, P., Gazolla, J.G.F.M., Chhetri, H., Streich, J., Climer, S., Thornton, P., Joubert, W., Jacobson, D.: Climatic clustering and longitudinal analysis with impacts on food, bioenergy, and pandemics. *Phytobiomes Journal* **0**(ja), null (2022). <https://doi.org/10.1094/PBIOMES-02-22-0007-R>
- [14] LSMS: Lsms: scalable first principles calculations of materials using multiple scattering theory. <https://github.com/mstsuite/lms> (2022), accessed: 12-30-2022
- [15] Luo, L., et al.: Pre-exascale accelerated application development: The ornl summit experience. *IBM Journal of Research and Development* **64**(3/4), 11:1–11:21 (2020). <https://doi.org/10.1147/JRD.2020.2965881>
- [16] Matthes, A., Widera, R., Zenker, E., Worpitz, B., Huebl, A., Bussmann, M.: Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the alpaka library (Jun 2017), <https://arxiv.org/abs/1706.10086>
- [17] McClure, J.: Lbpm software package. <https://github.com/opm/lbpm> (2022), accessed: 12-30-2022
- [18] McClure, J.E., Berrill, M.A., Prins, J.F., Miller, C.T.: Asynchronous in situ connected-components analysis for complex fluid flows. In: 2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV). pp. 12–17 (2016). <https://doi.org/10.1109/ISAV.2016.008>
- [19] Oak Ridge Leadership Computing Facility: OLCF user documentation. <https://docs.olcf.ornl.gov/> (2022), accessed: 12-30-2022
- [20] orringa, J.: On the calculation of the energy of a bloch wave in a metal. *Physica* **13**, 392–400 (1947)
- [21] Perilla, J., Schulten, K.: Physical properties of the hiv-1 capsid from all-atom molecular dynamics simulations. *Nat. Commun* **8**(15959) (2017)
- [22] Phillips, J.C., et al.: Scalable molecular dynamics on cpu and gpu architectures with namd. *The Journal of Chemical Physics* **153**(4), 044130 (2020)
- [23] Ravikumar, K., Appelhans, D., Yeung, P.: Gpu acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism. In: SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–22 (2019). <https://doi.org/10.1145/3295500.3356209>
- [24] Schneider, E.E., Ostriker, E.C., Robertson, B.E., Thompson, T.A.: The physical nature of starburst-driven galactic outflows. *The Astrophysical Journal* **895**(1), 43 (may 2020). <https://doi.org/10.3847/1538-4357/ab8ae8>
- [25] Schneider, E.E., Robertson, B.E.: Cholla: A new massively parallel hydrodynamics code for astrophysical simulation. *The Astrophysical Journal Supplement Series* **217**(2), 24 (apr 2015). <https://doi.org/10.1088/0067-0049/217/2/24>