Accelerated Application Development: The ORNL Titan Experience^{☆,☆☆}

Wayne Joubert^{a,*}, Rick Archibald^a, Mark Berrill^a, W. Michael Brown^a, Markus Eisenbach^a, Ray Grout^c,
Jeff Larkin^d, John Levesque^b, Bronson Messer^a, Matt Norman^a, Bobby Philip^a, Ramanan Sankaran^a,
Arnold Tharrington^a, John Turner^a

^a Oak Ridge National Laboratory, 1 Bethel Valley Road, Oak Ridge, TN 37831

^b Cray, Inc. Knoxville, TN

^c National Renewable Energy Laboratory 15013 Denver West Parkway Golden, CO 80401

^d NVIDIA Corp. P.O. Box 2008, MS-6008 Oak Ridge, TN 37831

Abstract

The use of computational accelerators such as NVIDIA GPUs and Intel Xeon Phi processors is now widespread in the high performance computing community, with many applications delivering impressive performance gains. However, programming these systems for high performance, performance portability and software maintainability has been a challenge. In this paper we discuss experiences porting applications to the Titan system. Titan, which began planning in 2009 and was deployed for general use in 2013, was the first multi-petaflop system based on accelerator hardware. To ready applications for accelerated computing, a preparedness effort was undertaken prior to delivery of Titan. In this paper we report experiences and lessons learned from this process and describe how users are currently making use of computational accelerators on Titan.

Keywords: High performance computing, accelerated computing, GPU graphics processing units, science applications, software development, code refactoring, software optimization

2010 MSC: 65Y05 [Computer aspects of numerical algorithms: Parallel computation], 68W10

[Algorithms: Parallel algorithms]

1. Introduction

1.1. Context

The use of manycore computational accelerators has become prominent across the high performance computing (HPC) landscape, with over 60 systems and roughly 35% of the computational power of the TOP500 list represented by systems utilizing accelerator hardware. While this seismic shift in HPC system design was foreseen as early as 2008[1] and the need for a fundamentally new approach to system architecture some time earlier[2], the exact nature of these systems and the means by which they would be programmed has only become clear over time.

[☆]This paper is dedicated to the memory of our colleague, Ricky A. Kendall (1961–2014). Ricky was instrumental in the formulation of the Titan early science project, and his insight and experience were valuable to all the code teams. His reputation in computational science was considerable, and the work described here bears his mark as clearly as any other piece of his legacy.

^{☆☆}This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy 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>).

*Corresponding author

Email address: joubert@ornl.gov (Wayne Joubert)

The Los Alamos National Laboratory (LANL) Roadrunner system, based on IBM's PowerXCell 8i processor, and the Oak Ridge National Laboratory (ORNL) Titan system, based on NVIDIA's Kepler K20X GPU (graphics processing unit), were pioneering in the use of accelerators, being respectively the first system to break the petaflop barrier and the first multi-petaflop system to use accelerated manycore processors. Titan, deployed for general use in mid-2013, has been the #1 system on the TOP500 list and is presently the largest open science computing system in the United States. Titan has been responsible for advances in diverse areas of science, technology and industry. It was used for petascale science runs by four of the six finalists in the SC13 Gordon Bell competition. The Titan computational resource has run many of the world's most scalable science application codes.

Titan was installed as a staged in-place upgrade of the ORNL Jaguar Cray XT5 system in 2012 and 2013. Titan is a Cray XK7 system composed of 200 cabinets containing 18,688 compute nodes, each equipped with a 16-core Advanced Micro Devices AMD Interlagos processor with 32 GB of memory and an NVIDIA Kepler K20X GPU accelerator with 6 GB memory, with Gemini interconnect. Titan's peak speed is in excess of 27 petaflops.

Each of the hybrid compute nodes in Titan consists of one AMD Series 6200 16-core Opteron processor and one NVIDIA Tesla GPU. Each node's GPU and CPU (central processing unit) are connected by a PCI Express Gen 2.0 bus with an 8 GB/second data transfer rate. The x86 portion of the Titan node contains an Opteron built of two Interlagos dies per socket. Each of these incorporates four processor groups called Bulldozer modules. Each Bulldozer module contains two independent integer unit cores which share a 256-bit floating point unit, a 2 MB L2 cache and instruction fetch. A single core can make use of the entire floating point unit with 256-bit AVX instructions. The four Bulldozer modules share a memory controller and an 8 MB L3 data cache. The processor die incorporating the four Bulldozer modules is configured with two DDR3 synchronous dynamic random access memory channels and multiple HT3 links. It is important to note here that each Titan node therefore contains two non-uniform memory access or NUMA domains, defined by the Interlagos dies on each socket. Memory operations across dies traverse the multiple HT3 links between the dies in a socket. The Tesla Kepler GK 110 GPU is composed of groups of streaming multiprocessors (SMX). Each SMX contains 192 single precision streaming processors called CUDA cores. Each CUDA core has pipelined floating point and integer arithmetic logic units. Kepler builds on the previous generation of NVIDIA Tesla Fermi GPUs with the same IEEE 754-2008-compliant single and double precision arithmetic, including the fused multiply add operation. A Kepler GK110 GPU has between 13 and 15 SMX units and six 64-bit memory controllers.

An easily overlooked difference between the Cray XK6/7 and prior generation XT systems is the Gemini interconnect. In contrast to the single SeaStar application-specific integrated circuit (ASIC) per node used in the XT series, each Gemini custom ASIC connects *two* nodes to the 3D torus interconnect. Cabling and backplane interconnectors between node boards are identical to the previous SeaStar system. The only physical difference is the mezzanine card on the node boards. The mezzanine card is a separate printed circuit board that attaches to the base XK7 node board and contains either the SeaStar or Gemini ASIC along with any support circuitry and the interconnections between the SeaStar or Gemini chips. This feature allowed ORNL to upgrade from an XT5/SeaStar system to an XK7/Gemini system while reusing the cabinets, cables and backplanes.

From an application point of view, Titan is, in many ways, an evolutionary step from what has become traditional supercomputer design (i.e., multicore CPU nodes joined via a high performance interconnect). This being said, it is the sheer number *and* nature of processing elements per node and the significant data locality requirements introduced by the PCI Express Gen 2.0 link within the node that make the architecture a challenging target for extant codes.

In previous times, a supercomputing system might be delivered with little to no application software ready to run on the system[3]. For the case of Titan, given the size and peak speed of the platform and its anticipated role as one of the world's fastest supercomputers at the time it was built, such a dearth of early application codes was deemed unacceptable by the Titan design team. As a result, the Center for Accelerated Application Readiness (CAAR) was formed over two years before the delivery of the system to prepare applications for Titan's accelerator hardware. This project stands in a long line of application readiness efforts for leadership systems deploying qualitatively new types of computer hardware,

such as vector supercomputers [4], reduced instruction set RISC-based distributed memory platforms [5], the LANL Roadrunner system[6], Titan, the upcoming National Energy Research Scientific Computing Center (NERSC) Cori system or the upcoming ORNL Summit system. Subsequent to the Titan CAAR project, many other applications have been ported and are using Titan’s GPUs to generate new science. Porting codes to a totally new architecture has been considerable work and has provided many insights regarding methodologies for achieving high performance for new and existing code bases on accelerator hardware. It is the purpose of this paper to document application porting experiences in preparation for Titan and in the early life of the system.

Table 1: Technology adoption phases: (a) early 1990s distributed memory HPC; (b) current accelerated HPC

Phase	Examples
Precipitating crisis: exhaustion of current technology	(a) Shared memory bandwidth scaling (b) Dennard scaling
Proliferation of hardware designs	(a) Connection Machine, nCUBE, KSR, Intel Paragon ... (b) NVIDIA GPUs, AMD APUs, Intel MIC, Adapteva ...
Multiple programming models	(a) CM Fortran, PVM, NX, Express, p4 ... (b) CUDA, AMP, Cilk, OpenCL, directives ...
Consolidation of hardware designs	(a) Distributed memory parallel systems (b) Vectors, threads, cores and nodes
Standardization of programming models	(a) MPI (1994) (b) OpenMP 5.X (?); OpenACC (?); OpenCL (?) (late 2010s)
Sustained advantage of the new technology	(a) To 2015 and beyond (b) To post-exascale

1.2. The accelerator technology adoption cycle

To better understand the current state of accelerated high performance computing and what can be expected in the near future, it is informative to understand the current position of accelerated computing within the technology adoption lifecycle model. This model interprets the pattern of adoption of a new technology through the stages of innovators, early adopters, early maturity, late maturity and laggards. It is also helpful to observe that the current rise of accelerated computing has many parallels with the rise of distributed memory parallel computing in the early 1990s.

The adoption cycle for accelerated computing was precipitated by the end of Dennard scaling in the mid-2000s and a growing belief by many that Moore’s law technology advancement is in some sense slowing. This has given rise to a multiplicity of new types of processor hardware—what has been described as the “compute jungle”—with an accompanying challenge to software for performance and performance portability across diverse architectures. Similarly, in the early 1990s, the “supercomputer market crash” occasioned the transition from traditional single processor and shared memory supercomputers to networked distributed memory systems and a concomitant challenge for porting software to the new systems[7]. See Table 1.

According to this view, we are already well into the early adopter phase of the technology adoption cycle, with many accelerated systems deployed and being used to deliver new computational science results. Until now, the emphasis for applications has been to obtain high performance on accelerated systems, to advance science in the respective domains, and to develop community experience regarding best practices for programming accelerators. In the near future we may enter the early maturity phase of technology adoption. This will be characterized by increasing convergence of hardware designs and standardization of programming models. For applications, this will mean a growing emphasis on performance portability across different accelerated architectures and software maintainability of applications into the foreseeable future. Performance portability of applications is in fact a growing concern for many centers. In this paper

we report findings on how applications have addressed the concerns of high performance and performance portability in the context of Titan.

1.3. Organization of this paper

The remainder of this paper is organized as follows. Section 2 describes the applications chosen for Titan early readiness and how they were selected. Section 3 explains the methodology used to port each of the CAAR applications to Titan, and Section 4 describes how other applications have been ported to Titan. Section 5 reviews the lessons learned from the readiness process. Following this, Section 6 reports how Titan is being used for accelerated science. Finally, conclusions are given in Section 7.

2. Early science applications

Given the marked difference in node architecture inherent in the new hybrid CPU-GPU architecture, it was recognized early in the project to build Titan that substantial effort would need to be expended to bring scientific applications to the point of effective use of the new platform. The Center for Accelerated Application Readiness (CAAR) is a collection of application teams, vendor partners, and tool developers that was first brought together by the Oak Ridge Leadership Computing Facility (OLCF) to produce successful ports of a set of representative applications to Titan. The intent was to ensure this suite of codes was capable of “day-one” scientific output when Titan was put into production, as well as to serve as a laboratory to establish a set of best practices for other application ports.

However, given the level of effort anticipated to accomplish each application port to Titan, only a limited number of applications could be attacked. A process to select a diverse set of codes to give broad coverage to represent use cases for our users was designed, consisting of several steps and evaluation points. First, the OLCF used two separate surveys of computational science projects to gauge interest, need and current thinking in scientific subfields regarding application redesign and maintenance going forward.

An OLCF application requirements document was developed and produced in 2009 [8] which relied heavily on the responses to a new comprehensive requirements questionnaire. The questionnaire included requests for information involving science motivation and impact, application models, algorithms, parallelization strategies, software development processes, usage workflows and application performance. In addition to this information-gathering effort, a Science Driver Survey was deployed. This survey was developed in consultation with more than 50 leading scientists in many domains. Key questions in the survey included:

- What are the science goals, and will a system with the proposed capabilities enable them to be met?
- What might the impact be if the improved science result occurs?
- What does it matter if this result is delivered in the 2012 timeframe?
- What science will be pursued on this system and how is it different (in fidelity, quality and predictability, and/or in productivity and throughput) from the current system?
- What might the impact be if this improved science result occurs? Who cares, and why?
- If this result is delivered in the 2012 timeframe, what does it matter as opposed to coming five years later (or never at all)?
- What other programs, agencies, stakeholders and/or facilities are dependent upon the timely delivery of this result, and why?

Both of these surveys produced a potential list of roughly 50 applications that might have been chosen for the initial CAAR suite of codes. This list was cut down by an order of magnitude via a set of cuts based on several criteria. First, the impact and timeliness of the expected science results were evaluated, as was the alignment of these results with the U.S. Department of Energy’s science mission. The final list was intended to provide broad coverage of science domains and implementations (including models, algorithms

and software). In addition, broad coverage of relevant programming models, environments, and languages was desired. A variety of data structures, algorithmic motifs and scientific library requirements was yet another source of desirable diversity. The size and makeup of the code’s user community—both current and anticipated—was also considered. The degree to which institutional and developer/user involvement could be anticipated in the CAAR effort was gauged, based on available information. Finally, a good representation of the current and anticipated OLCF workload was required. This consideration included an evaluation of the availability of OLCF liaisons with adequate skills and experience in the respective application area and the availability of key code development personnel to engage in and guide readiness activities.

The final result of this analysis was a manageable list of initial applications. The six primary CAAR applications targeted for early science results on Titan were:

- CAM-SE – A highly scalable atmospheric model that is part of the Community Climate Systems Model.
- Denovo – A deterministic neutron transport code for shielding and nuclear reactor analysis.
- WL-LSMS – A first-principles density functional theory code (local density approximation) used to study magnetic materials.
- LAMMPS – A molecular dynamics general statistical mechanics based code applicable to bioenergy problems.
- S3D – A direct numerical simulation code for compressible reactive flow for simulating turbulent combustion.
- NRDF – A 3D non-equilibrium radiation diffusion code.

Although CAAR efforts were driven by specific simulation targets and particular scientific questions, since the inception of the project an additional aim has been the collection of “lessons learned” and the dissemination of the techniques developed to a much broader community of application developers. Care has been taken to ensure that acceleration enhancements to all CAAR codes are amenable to inclusion back into the production trunks of the individual applications. By engaging the primary developers of all the CAAR codes early in the development process, choosing scientific targets in collaboration with them, and taking care to make enhancements that can survive near-term changes in code structure that are planned irrespective of acceleration work, we have confidence that CAAR development efforts will not be “dead ends.” We expect that structural changes in all the CAAR codes will provide benefit on non-accelerator-based architectures, and, in any event, will not impact portability in a negative way. Portability and the prevention of version bifurcation is critical. As compilers, libraries, and tools mature and change on accelerated systems, there will be fewer instances where GPU-specific code needs to be written. When separate GPU-specific code is required for performance, it must be enabled through conditional compilation and must be done in collaboration with the application development team. This is important to ensure that future changes in the base code, which may require changes in the accelerator code, will be visible and understood.

3. Application porting strategy

The following steps were followed to bring each application from its initial state to a point of entry into the code porting process.

- A multidisciplinary code team was set up, including an OLCF application lead, a Cray engineer, an NVIDIA developer, and others such as application developers and tool or library developers. The teams worked independently but also met regularly with the other teams and technical management to discuss progress and issues. Cross-cutting support from external tool and library developers was also provided.

- A testbed GPU cluster was acquired to provide a resource for early code development and testing.
- A code inventory was performed, to assess application code structure, suitability for refactoring, algorithm structure, data structures, and data movement patterns. Also assessed were typical code use cases and problem sizes. Importantly, the execution profile and the scaling behavior were also evaluated, to determine whether the code had high-usage hot spots (code locations where most of the runtime was spent) that might be suitable candidates for porting to the GPU.
- A parallelization approach was determined for each application. This required determining which algorithm components of the code to port to the GPU, what problem dimensions to map to the GPU thread hierarchy, and how data movement to and from the GPU and between the GPU processors and memory would be scheduled.
- A GPU-based programming model for the code port was decided upon, whether CUDA, OpenCL, directives, use of libraries or a combination.
- A code development strategy was determined. Issues addressed included the decision to rewrite versus refactor, portability to other platforms, incorporation of GPU code support into the build system, and relationship to the code repository main trunk.
- Representative test problems were chosen to guide the porting and code optimization process, and performance metrics were formulated for measuring success.

The port of each application posed a unique set of challenges. The following is an overview of the porting effort for each code.

3.1. *Denovo*

Denovo is a 3D deterministic radiation transport code with applications to nuclear energy, shielding, medical physics, homeland defense and nuclear criticality/safeguards. The Denovo code system solves the forward or adjoint linear Boltzmann transport equation. The primary algorithms of Denovo include the multigroup discrete ordinates (S_n) method with finite element collocation in angle, multiple spatial discretization options, eigensolvers such as Arnoldi's method, Krylov solvers including GMRES, and the KBA sweep algorithm. Denovo uses MPI for message passing and the Trilinos library for solvers. Denovo is written in C++ under an agile development process with rigorous software quality engineering including extensive unit testing.

The KBA sweep algorithm of Denovo consumes 80-99% of the compute time of a typical run and thus was a key target for acceleration in the CAAR work. As part of the OLCF CAAR effort, the sweep operation has been ported to NVIDIA GPUs [9].

The solution of the Boltzmann equation in the context of Denovo is inherently recursive based on the directions of particle flow being modeled. However, considerable work in algorithm design has been applied to this problem to give good parallel scalability. GPU thread parallelism has been applied to multiple problem dimensions including space, angle, moment, energy and octant. In the porting process, the code was also restructured to reduce memory traffic and improve data reuse.

The results in Table 2 are for a weak scaling study in which Denovo is used to solve a reactor eigenvalue problem with 16 energy groups, 16 moments, 256 angles, linear discontinuous elements, and $16 \times 32 \times 64$ spatial cells per compute node. It can be seen that the code gives good weak scaling behavior and has significant performance improvement with use of GPUs. We anticipate similar or better performance gains to be possible by use of Intel Xeon Phi processors or other accelerators.

3.2. *WL-LSMS*

The magnetic materials code WL-LSMS has two major components: (1) a finite temperature, statistical physics calculation (Wang-Landau sampling) and (2) an underlying order-N scaling, first-principles density functional code (LSMS). This multiple scattering code allows the fully relativistic investigation of finite

Table 2: Weak scaling results for Denovo (times in seconds).

Nodes	Sweep CPU	Sweep GPU	Denovo CPU	Denovo GPU
1	585	302	835	553
2	645	305	898	558
4	617	319	871	572
8	579	291	803	514
16	750	365	1025	640
32	833	407	1122	700
64	1010	441	1318	750
128	1071	439	1361	732
256	1153	498	1447	794
512	1489	572	1801	886
768	1505	618	1816	931
896	1590	645	1903	961

Table 3: WL-LSMS multicore scaling on Titan.

# Walkers	# Cores	Time [s]	Perform. [TFLOP/s]	Par.Eff.
2	2064	6575	13.0	1
10	10256	6576	65.0	0.9998
40	40976	6579	260.0	0.9994
80	81936	6582	519.8	0.9989
160	163856	6585	1040.0	0.9985
263	269328	6584	1708.5	0.9986

temperature magnetism and alloy stability in a wide range of materials (e.g., materials for electric motors, magneto-caloric materials and nuclear fuels).

In far too many instances, DFT explanations of phenomena include the caveat “at $T=0$,” which we aim to replace by “performed at experimental temperature.” We use the Wang-Landau (WL) statistical method to minimize the number of DFT energy evaluations needed to survey the energy landscape. WL utilizes a trial density of states (DOS) to force the walkers toward undersampled regions of phase space. At convergence, the walkers are performing a random walk in energy, and the trial density of states has converged to the true density of states. Each of the many walkers enlists another level of parallelization through the Locally Self-Consistent Multiple Scattering (LSMS) code that uses multiple nodes to evaluate the energy of each trial configuration. On each node the LSMS work is distributed to standard compute cores and accelerators, where the compute-intensive linear algebra (over 95% of all floating point operations) is allocated to the accelerators via library calls (e.g., using MAGMA or CULA).

Recent scaling measurements for WL-LSMS on Titan are given in Table 3. The test problem for these runs is for a 1024 iron atom system with a local interaction zone radius of $12.5\text{\AA}$. The same system was tested also using the Kepler GPUs on Titan, yielding the results in Table 4.

3.3. LAMMPS

LAMMPS, the Large-scale Atomic/Molecular Massively Parallel Simulator, is a classical molecular dynamics code. LAMMPS is parallelized via MPI, using spatial-decomposition techniques that partition the simulation domain into smaller subdomains, one per processor. It is also designed in a modular fashion with the goal of allowing additional functionality to be easily added. This is achieved via a variety of different

Table 4: WL-LSMS scaling results with GPUs on Titan.

# Walkers	# Nodes	Time [s]	Perform.	
			[TFLOP/s]	Par.Eff.
2	65	938.2	45.6	1
32	513	939.7	728.3	0.9984
128	4097	960.0	2851.8	0.9773
526	16833	1185.8	9489.0	0.7911

style choices that are specified by the user in an input script that control the choice of force field, constraints, time integration options, diagnostic computations and so forth. At a high level, each style is implemented in the code as a C++ virtual base class with an appropriate interface to the rest of the code. For example, the choice of *pair* style (e.g., lj/cut for Lennard-Jones with a cutoff) selects a pairwise interaction model that is used for force, energy and virial calculations. Individual pair styles are child classes that inherit the base class interface. Thus, adding a new pair style to the code (e.g., lj/cut/hybrid) is as conceptually simple as writing a new class with the appropriate handful of required methods or functions, some of which may be inherited from a related pair style (e.g., lj/cut). This design has allowed us to incorporate support for acceleration hardware into LAMMPS without significant modifications to the rest of the code while maintaining full support for all of LAMMPS features.

We have been researching methods to attain efficient acceleration for molecular dynamics in LAMMPS for several years. In our current approach, implemented in the main LAMMPS distribution available for download, LAMMPS supports both CUDA and OpenCL builds allowing for portability to compute devices from many vendors. Currently LAMMPS supports acceleration for neighbor-list builds, long-range electrostatics via PPPM and short-range force calculations for a variety of simulation models important for biological, materials and mesoscale simulations. In addition to porting existing models for use on accelerators, we have investigated alternative models for molecular dynamics that are better-suited for future hybrid and heterogeneous machines.

The flexibility of LAMMPS leads to a rather disparate set of results when speedup achieved with hybrid code is considered. Depending on the particular physical problem, our realized speedups range from 1.92 (for an atomic fluid made up of roughly 32K particles with a 2.5σ cutoff) to 5.82 or more (for a liquid crystal simulation of 32K particles)[10].

3.4. CAM-SE

The Community Atmosphere Model or CAM code itself is composed of two major sub-components: (1) the dynamical core (dycore) and (2) the physics (CAM5) and global 3D chemical reaction model (MOZART) packages. The spectral element (SE) version of the CAM dycore—CAM-SE—is based on a finite element, variational discretization of the partial differential equations (PDEs) on a cubed-sphere grid (though adaptable to arbitrary grids). The system is semi-discretized (via the method of lines) and evolved in time with a Runge-Kutta (RK) time integrator. During each RK stage, elements only communicate their boundary values to nearest neighbors in a boundary-averaging process required in the SE method, giving rise to the excellent scaling properties of CAM-SE.

The new CAM5 physics improves many biases inherent to CAM4 (prior generation physics) and is also less sensitive to resolution and mesh refinement. Interactive chemistry is supported via the Model for Ozone And Related chemical Tracers (MOZART), a global 3D chemical reaction model. For the simple moist processes in CAM4, only the three phases of water are used. For advanced cloud microphysics, CAM5 requires 22 more tracers to represent intermediate and more complex forms of precipitation. For treatment of aerosols and other chemical species, another 81 tracers are required by MOZART. Using CAM5+MOZART, tracer transport becomes the most expensive component in high-resolution configurations of CAM-SE.

CAM-SE scalability has been demonstrated on a variety of platforms. We previously verified that 14km resolution CAM-SE with CAM4 physics and MOZART chemistry exhibited strong scaling with 60% parallel

Figure 1: Parallel efficiency under strong scaling for CAM-SE 14km global simulation with CAM4 physics and MOZART chemistry.

efficiency to 172,800 cores on the Cray XT5 compared to 6,144 cores (see Figure 1). The code is implemented in a hybrid OpenMP+MPI fashion. The OpenMP can be run in two ways: (1) the elements can be chunked so that each chunk is run on a given thread in a data-independent manner, or (2) loop-level OpenMP can be invoked to place multiple threads on a single element.

As the first step in the CAAR port of CAM-SE, the tracer transport routines, which dominate the runtime profile for the science target configurations, were ported to GPUs [11]. This port was accomplished with PGI's CUDA Fortran language extensions. Efforts are currently underway to migrate to OpenACC directives for a comparison in efficiency and to facilitate better software engineering practices by reducing source code duplication.

Since these initial efforts, additional work has been carried out to overlap MPI transfers, PCI-e transfers, kernel execution on GPUs, and CPU execution, leading to an overall CAM-SE speed-up of 2.7x using a single NVIDIA Fermi GPU per node compared to a single Interlagos CPU per node. This speedup is computed using best case CPU code and end-to-end wall timers including all PCI-e communication for all of CAM-SE except initialization, which is the same for both versions of the code.

3.5. S3D

S3D is a massively parallel direct numerical simulation (DNS) solver for turbulent compressible reacting flows. S3D solves the fully-compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry. The governing equations are supplemented with additional constitutive relationships, such as an ideal gas equation of state, and models for reaction rates, molecular transport and thermodynamic properties. S3D performs a fully explicit time integration of the governing equations with high-order accurate, non-dissipative approximations for the spatial derivatives.

S3D is written in modern Fortran and employs a pure MPI approach for parallelization between on-node cores as well as between nodes. The computational domain is decomposed in all three dimensions, and each MPI process advances the solution in one piece of the 3D domain. The Cartesian mesh and the domain decomposition ensure that all MPI processes have the same number of grid points and the same computational load. The explicit flow solver requires interprocessor communication only between the nearest neighbors in the 3D MPI topology. All-to-all communications are required for occasional monitoring and synchronization ahead of I/O events.

As the first step of the CAAR effort, S3D was converted from a pure MPI code to a hybrid MPI+OpenMP application. Almost all the computations in S3D are in loops traversing the three-dimensional grid that are

Figure 2: Grind time (in seconds) as a function of compute nodes for S3D. The squares denote timings for the initial OpenMP, CPU-only port. The triangles show the performance and scaling for the final hybrid OpenMP/OpenACC version of the code.

relatively simple without complex dependencies or critical regions. These could be parallelized automatically by many of the current compilers. However, for the hybrid parallelism to be scalable to $O(10)$ threads in the near future, and to $O(100)$ threads in the long term, it was necessary to restructure S3D to express much finer granularity of parallelism than was available at the loop level. Also, the higher granularity OpenMP regions developed in this step would also become OpenACC regions for the accelerator later on. The resultant code was able to cover close to 100% of the computation in only a few OpenMP regions, compared to the large number of individual loops in the original code. This also led to a low memory footprint because the need for temporary arrays to carry forward the intermediate results between successive loops and OpenMP regions was minimized.

Once the hybrid version of S3D was finished and checked both for correctness and performance, the next step of the CAAR effort was to port the hybrid code to the accelerator [12]. When the CAAR effort was started, the only defined directives for acceleration were the proposed OpenMP extensions for accelerators. Before the work was completed, a new, more complete definition of directives (OpenACC) was proposed by a collaboration of Cray, NVIDIA, PGI and CAPS. Given the approach that we had taken with the directives, the adoption of the new OpenACC directives required only a trivial change in syntax. Even early versions of the proposed OpenACC directives included all of the functionality required for S3D. S3D target problems are typically not limited by available system memory, so a strategic decision was made early to target an accelerator that had enough memory to store an up-to-date version of the major computational arrays. With this approach, the only data that would need to be shipped back and forth from the host to the accelerator would be those data from halo regions, as they must be communicated to other nodes in the system. Given this strategy, we broke up the code into code that runs only on the host—mostly composed of the routines dealing with communication, boundary treatment, and I/O—and the code that runs on the accelerator, i.e., the computational OpenMP regions. We augmented the reworked communication infrastructure to encapsulate accelerator/host communication and overlapped communication with computation on the accelerator [12]. Figure 2 illustrates the performance of the hybridized S3D code on the Titan system, the performance improvement due to the use of the GPU accelerator, and the near-ideal weak scaling.

3.6. NRDF

NRDF is a 3D non-equilibrium radiation diffusion code[13] that solves the time dependent non-equilibrium radiation diffusion equations that are important for solving the transport of energy through radiation in optically thick regimes with applications in several fields including astrophysics and inertial confinement

fusion. The associated initial boundary value problems that are encountered often exhibit a wide range of scales in space and time and are extremely challenging to solve.

$$\frac{\partial E}{\partial t} - \nabla \cdot (D_E \nabla E) = \sigma_a(T^4 - E) \quad \text{in } \Omega, \quad (1)$$

$$\frac{\partial T}{\partial t} - \nabla \cdot (D_T \nabla T) = -\sigma_a(T^4 - E) \quad \text{in } \Omega. \quad (2)$$

NRDF depends on a number of external packages for its capabilities, including SAMRAI, PETSc, Hypre, HDF5 and MPI, many of which are not optimized for the target architecture. The non-equilibrium radiation diffusion problem is discretized on structured adaptive mesh refinement (SAMR) hierarchies which consist of nested refinement levels with each level a union of non-overlapping patches at the same resolution. A method of lines (MOL) approach is used with a cell-centered finite volume spatial discretization followed by discretization in time. At coarse-fine boundaries where fine and coarse levels interact, linear interpolation into ghost cells aligned with fine grid cells enable the use of regular finite volume stencils for interior and boundary cells. To solve the nonlinear systems arising at each timestep, an inexact Newton method is used with GMRES for the linear solver. The linear system is preconditioned on SAMR grids with components that involve either a multilevel Fast Adaptive Composite Grid (FAC) solver or an asynchronous version of FAC (AFACx). Table 5 shows a comparison of the performance of the two solvers. The asynchronous version of FAC allowed for the use of multiple threads to process the preconditioners in parallel and overlap the communication in the call to the boundary conditions (ApplyBC), resulting in significant performance improvements. The use of multiple threads within AFACx further improved performance by utilizing idle processor cores.

Table 5: NRDF calculation with 250 timesteps, final time $t=0.0002879$, 4096 processes on 1024 nodes with different preconditioners with and without threading

	FAC	AFACx (serial)	AFACx (2 threads)
Total Time (s)	4639	1750	1656
Preconditioner (s)	4205	847	726
ApplyBC (s)	3675	241	130
# Nonlinear Apply	519	621	621
# Linear Apply	637	967	967

An additional solver capability utilizes block iterative methods for smoothers within the FAC and AFACx solvers. This presented an opportunity for utilizing the GPUs available by porting the block solvers to the GPU. Figure 3 shows the kernel performance on different CPU and GPU architectures implemented on GPUs using CUDA. Moving the kernel to the GPU resulted in a net speedup for all blocksizes, with a maximum speedup of 2x. The majority of the NRDF code and all communication remained on the CPUs.

4. Other applications

In addition to the CAAR applications, many other codes have been ported to Titan’s GPU accelerators, with more applications currently in progress. In the following, a subset of these codes is briefly reviewed, with discussion of the relevant algorithms and the porting methodology for Titan. Further details regarding the approach by which each code was accelerated can be found in the respective references.

1. AWP-ODC is used for earthquake simulation and hazard analysis. It solves the 3D velocity-stress wave equations with an explicit staggered-grid finite difference scheme. AWP-ODC is written in C and uses MPI and CUDA. A 2D decomposition is used to parallelize the 3D structured grid computation, and the finite difference calculations are performed on the GPU with aggressive overlapping of communications and CPU-GPU transfers and with significant use of GPU shared memory and constant memory[14].

Figure 3: NRDF kernel performance for chaotic block Gauss-Seidel using different block sizes with patch size 64^3 .

2. **DCA++** is used to solve high-Tc superconductivity problems via quantum cluster algorithms. It uses the dynamical cluster approximation (DCA) to study the superconducting transition in Hubbard-type models. The code is written in C++ with template metaprogramming to support multicore and GPU-based systems in a single code base and uses MPI and CUDA. The primary computational expense is a submatrix update requiring DGEMM and DTRSM operations. To use the GPUs, the Monte Carlo walkers are deployed to GPU threads which are used to perform the submatrix updates[15].

3. **HACC** is a cosmology code for modeling cosmic structure formation by solving the gravitational Vlasov-Poisson equation. To compute short-range forces for the P³M method on the GPU, an asynchronous buffering scheme is used. The 3D grid is divided into 2D slabs which are processed in sequence; each slab requires input from its two neighbor slabs. The force calculation is computed on the GPU for the current slab while the previous slab result is copied out and a future input slab is copied in. The code uses MPI with OpenCL for accelerated parallelism [16].

4. **MUPHY** performs the numerical solution of fluid motion with suspended macromolecules using the lattice Boltzmann method on a Cartesian grid and a molecular dynamics-like method for particles. A major computational expense is an LU factorization which is performed on the GPU using the MAGMA library. MUPHY is written Fortran 90 with CUDA C and with MPI for communication [17].

5. **NAMD** performs molecular dynamics calculations and is based on the Charm++ parallel programming system and runtime library. It uses CUDA to perform short-range force calculations on the GPU with use of asynchronous CPU/GPU computation overlap and uses GPU shared, constant and texture memories[18].

6. **PiConGPU** performs 3D fully-relativistic particle-in-cell simulations of plasmas. It is written in CUDA and C++ using metaprogramming techniques and compile-time polymorphism. PiConGPU solves electric and magnetic fields on a regular rectangular Cartesian mesh. It performs particle and mesh operations on the GPU using asynchronous communication and transfers and with data and task parallelism. Performance is accelerated by caching cell updates from particles using the GPU shared memory[19].

7. **PSC** solves the Vlasov-Maxwell equations for particle-in-cell simulation of plasmas. The finite difference time domain method is used to advance the problem in time. A space-filling curve method is used to decompose the 3D Cartesian grid to facilitate load balancing. The GPU is used for the particle advance

and current deposition processes, making use of atomic add operations in GPU shared memory. The code is written in C with an object-oriented design style with MPI and CUDA[20].

8. QCD API is an interface specification through which quantum chromodynamics (QCD) codes such as Chroma and MILC are able to perform calculations. The QUDA library is a package conforming to this API which enables QCD calculations for these codes on GPU-based systems[21, 22].

9. QMCPACK uses the quantum Monte Carlo method to solve Schrödinger’s equation. The code parallelizes across multiple Monte Carlo walkers with loose synchronization. Multiple GPU kernels are implemented which distribute threads to electrons, atomic cores or orbitals. QMCPACK is written in C++ with MPI/OpenMP parallelization and CUDA[23].

10. RMG performs quantum mechanical electronic structure calculations using multigrid methods. GPU acceleration has been applied to the diagonalization routines via CUBLAS calls and some finite difference routines with CUDA. The code is parallelized using MPI, OpenMP and Pthreads[24].

11. SPECFEM3D.GLOBE models global seismic wave propagation with linear elastodynamics using spectral finite elements. The code launches kernels on the GPU at each timestep to perform finite element calculations such as element assembly and numerical integration with the discrete Jacobian. The GPU kernels make use of constant, shared and texture memories, and the algorithm performs asynchronous communication and GPU operations. A grid coloring scheme is used to avoid race conditions for the element assembly. SPECFEM3D.GLOBE is written in Fortran 90 with MPI and OpenMP support and uses CUDA and OpenCL for accelerated kernels [25].

12. 2HOT performs cosmological simulations using a parallel hashed octree N-body algorithm and multipole methods. It uses a space filling curve domain decomposition and tree traversal computations. To obtain high computational intensity, particles with the same interaction list are bundled together and sent to the GPU for force calculations. The code aggressively overlaps execution of multiple concurrent kernels. 2HOT is written in C and uses MPI, CUDA and OpenCL [26].

13. XGC1 is a full-function gyrokinetic particle-in-cell code designed for simulating edge plasmas in tokamaks. It solves Maxwell’s equation on an unstructured triangular spatial grid and uses a finite difference discretization of the Fokker-Planck collision operator on a structured grid. The particle push calculations which require the majority of runtime are offloaded to the GPU using streams for kernel launch and data copy. XGC1 is written in Fortran 90 and uses CUDA Fortran [27]

5. Lessons learned

We here discuss lessons learned from Titan application readiness efforts, primarily from the CAAR effort but also from engagement with other application teams.

5.1. Application assessment

Multiple factors affect the level of difficulty and likelihood of success for an acceleration readiness effort for an application:

- A large code base with many lines requiring change can be difficult to port due to a flat profile requiring many code parts be accelerated, a suboptimal looping structure spanning many levels of the call tree, or a cross-cutting concern such as data structures requiring global changes. For example, the CAM-SE code base has length in the hundreds of thousands of lines and required acceleration and data structure changes in multiple parts of the code.
- Models and codes with high opportunity for extracting more parallelism and with high computational intensity are more favorable candidates for acceleration. The LSMS code, for example, spends a large portion of its runtime in computationally intense BLAS-3 operations that can be offloaded effectively to the GPU.
- Algorithms with complex branching at the lowest level present challenges to vectorization or mapping efficiently to GPU warps. Monte Carlo radiation transport codes, for example, can have complex branching logic in innermost loops resulting in warp divergence and difficulty with vectorizing.

- Many present-day science codes were not written to minimize memory traffic for modern cache-based architectures. These codes are likely to require major loop restructuring and/or updated data structures. An example from the CAAR effort is S3D, which required loops to be permuted to improve data locality and expose parallelism.
- 455 • Codes may require substantial unrelated prior optimizations in order to realize gains on an accelerator, e.g., optimizing MPI communications. For many codes, the performance characteristics may never have been analyzed in depth or may not be known accurately, potentially necessitating significant preparatory work. Optimization of MPI communications was required of CAM-SE in the CAAR effort. Needless to say, this optimization work is also beneficial to the CPU version of the code.
- 460 • Codes written in an older programming style, e.g., with poor code structure or poorly organized data structures, may be difficult to port.
- Likewise, codes for which the science and algorithms are expressed with software engineering styles that are not cognizant of performance issues may need redesign.

5.2. Algorithm / code transformation

465 The bulk of the work of application acceleration is code restructuring for improved parallelism and data locality. Some of the recurring themes of this work are:

- finding more useful threadable work for the GPU;
- modifying or changing numerical algorithms to expose more parallelism
- improving memory access patterns, e.g., obtaining stride-1 access and more data reuse;
- 470 • bundling work together to make GPU work in a kernel call as coarse-grained as possible;
- making data on the GPU more persistent to reduce the cost of transfers;
- using asynchronicity to overlap use of hardware components as much as possible (CPU, GPU, processor interconnect, CPU-GPU interconnect).
- flattening data structures and/or the call tree.

5.3. Implementation

475 A key choice to be made when porting to an accelerator is the programming model. Current choices of API for programming NVIDIA GPUs include CUDA, OpenCL and OpenACC. Each developer team must decide the API to use based on the needs of the project. CUDA allows a more close-to-metal approach for performance programming but is proprietary. OpenCL is a widely-portable open standard but is low-level, may require platform-specific customizations, and may be less performant than CUDA. OpenACC directives provide portable high level programming that is potentially less invasive to application code but does not provide access to all features of the underlying hardware and currently has less tool support. CAAR applications use OpenACC directives (S3D), CUDA (Denovo), CUDA with libraries (LSMS), CUDA Fortran (CAM-SE) or CUDA / OpenCL (LAMMPS).

485 Applications can access the accelerator in different execution configurations. For example, each CPU core can be assigned a single MPI rank, with each rank executing GPU operations in a coordinated or asynchronous fashion. Alternatively, an MPI rank can spawn OpenMP threads on multiple CPU cores that each launch operations on the GPU.

490 Separation of concerns is useful for designing portable, future-proofed accelerated codes. Design of applications such that platform-specific code is separated from the business logic of the application is desirable but difficult since the issues are increasingly intertwined as computer hardware becomes more complex. It is helpful to isolate use of the accelerator to a lightweight adapter layer which is easily modifiable to support CUDA, OpenCL, OpenACC or other APIs. Access to data can be abstracted through accessor functions

to opaque objects that are inlined for performance and allow data structures to be changed in a localized place in the code for different architectures. Generic programming techniques can be useful for managing the complexity of mapping algorithms to different kinds of hardware while obtaining high performance.

Recurring code transformations across development efforts for application accelerator readiness include: the need to permute loops, sometimes across multiple levels of the call tree, to improve locality of memory reference; fusing loops to obtain coarser granularity of GPU work; and flattening or other modification of data structures to improve accelerator performance.

Many developers have found it useful to write code with a dual CPU/GPU programming style so that most of the code can execute on either CPU or GPU. This reduces code redundancy, improves portability and makes debugging easier.

A tension exists between writing performant code and writing maintainable code based on good software engineering practices as specified by the project. Each project must negotiate this relationship.

5.4. Development workflow

Tools such as compilers, debuggers and profilers were lacking early in the CAAR project but have matured and continue to do so. Debugging and profiling tools were helpful at various points in the porting process.

A multidisciplinary team with diverse expertise is increasingly beneficial for such porting efforts, as science models, algorithms and computer hardware and software grow in complexity.

A performance model for application behavior can be useful for guiding coding decisions during development. Writing small test codes can help to evaluate performance of small representative operations.

5.5. Project management

CAAR applications each required roughly two person-years of work to port to accelerators. This figure is expected to be less now, as tools as well as community understanding of how to port code to accelerators have matured.

An estimated 70-80% of developer time was spent in code restructuring for accelerators; thus it is anticipated that porting to alternate programming APIs or other accelerator hardware will be less difficult than the initial port to accelerators.

The relationship between the porting effort, the code owners and the science team must be managed carefully. The science models used in the application can change mid-project, making the porting objectives a “moving target.” Other variances such as personnel changes must be dealt with.

5.6. Performance portability

It is common for HPC users to run applications on multiple architectures at different centers. As there is currently no single best way to program accelerators, applications are addressing performance portability in different ways, such as:

- interfacing to a library that can swapped with minimal code changes for different platforms (LSMS);
- creating a thin code layer to interface easily to alternate accelerator APIs (LAMMPS, Denovo);
- using directives that can be modified for different platforms (S3D); or
- using C++ generic programming techniques to generate code at compile time that is optimal for the architecture (Trilinos).

Fortunately, since most of the effort of porting to accelerators has to do with code restructuring irrespective of the API, it will likely be possible to repurpose the ported codes to other accelerators or APIs with comparatively little effort. This being said, the lack of a ubiquitous programming standard for accelerators remains an outstanding problem to the community which vendors must address. It is possible that a directives-based approach such as OpenMP or OpenACC will serve the need for performance portable programming. However, some codes may still require small amounts of alternative code that uses a more close-to-metal programming approach, e.g., CUDA or vector intrinsics. Furthermore, different processor

offerings will have different cache sizes, vector lengths, hyperthreading capabilities, relative bandwidths and latencies or other more subtle differences, even within the same product family, which may require use of tuning parameters or custom code to capture good performance.

6. Reception

6.1. Performance

Table 6 shows early performance results for science applications on Titan [28, 29]. Here the Titan Cray XK7 system which has one CPU and one GPU per node is compared against a corresponding alternative Cray XE6 system containing two CPUs per node, showing the value of replacing a CPU socket with a GPU. Note that on a per-node basis, the XK7 flop rate is roughly 5X that of a corresponding XE6, and main memory bandwidth is roughly 3X; these ratios give performance targets for porting applications to Titan, though these figures are generally optimistic since some code constructs perform with limited efficiency on the GPU. As the table shows, due to readiness efforts, applications have been able to take significant advantage of the performance benefits of GPUs on Titan.

Table 6: Application performance, Titan Cray XK7 vs. Cray XE6.

Application	Performance Ratio
S3D	1.8
Denovo sweep	3.8
LAMMPS, mixed precision	7.4
WL-LSMS	3.8
CAM-SE, estimated	1.8
AWP-ODC	2.1
DCA++	4.4
QMCPACK	2.0
RMG	2.0
XGC1	1.8

6.2. Usage

Figure 4 shows application usage of Titan’s GPUs in the early months of general availability beginning June 2013, taken from [30]. This shows the percentage of Titan core-hours that used GPUs, for applications run under 2013 and 2014 projects from the INCITE (Innovative and Novel Computational Impact on Theory and Experiment) program, which account for 60% of the core-hours awarded on Titan. These results are tentative, as software to measure GPU usage on these systems is improving. The results show that application readiness efforts have enabled users to exploit the capabilities of Titan’s GPUs to a significant degree at the outset of system availability.

6.3. Survey results

In 2013 the Oak Ridge Leadership Computing Facility went through a requirements gathering exercise to collect input for determining the center’s next leadership computing system, similar to the 2009 requirements process [8] leading up to Titan. As part of this effort, a user survey was conducted, covering projects using the Titan system, to better understand requirements for the next generation of accelerated system for advancing science [31]. The survey results represented 21 application code teams for 18 projects using the center’s facilities across 15 science domains, accounting for the majority of the 31 active INCITE projects at the center. In this survey, users described how they were using or planned to use computational accelerators to reach their science goals. The primary findings of this survey include the following:

Figure 4: Monthly percentage of Titan INCITE workload using GPUs

- 81% of respondents believed their applications still have moderate to large amounts of additional parallelism available to exploit, as systems support increasing numbers of compute threads.
- About three-quarters of users indicated their codes already use computational accelerators in some capacity.
- Though users said they were very willing to take advantage of accelerated compute hardware, 85% of respondents felt that the level of programming effort required to extract performance from accelerated systems was moderate to high. Clearly, vendors and tool developers must present a better user-facing experience for the code development process on these systems. As programming models become standardized and as compilers and tools mature, this situation is expected to improve.
- Most users stated they were currently using CUDA to program GPUs, but many showed interest in directives-based approaches such as OpenACC as these programming models become more mature.

7. Conclusions

This paper has demonstrated how a focused application readiness effort has enabled a set of applications to transition successfully to a new generation of HPC architecture. The lessons learned from this activity provide information relevant to future accelerator readiness efforts.

The primary findings of this application readiness effort are:

- A carefully managed readiness effort is essential to preparing for new types of HPC system hardware.
- Applications generally require substantial amounts of code restructuring to port to accelerated architectures. This task encompasses the bulk of the development work required.
- Exposing more parallelism and minimizing data traffic are common themes when porting to accelerated architectures.
- Performance gains of 2X-7X have been realized for accelerated application codes on Titan.

The prediction has been made that the world's top ten HPC systems will soon all be accelerator-based. It is also believed that future exascale systems will most likely be based on some form of manycore accelerator technology. In past experience, technologies originally deployed for the HPC space have over time migrated to the mainstream, and indeed features of accelerators such as large numbers of cores, vector units and hyperthreading are becoming common across the general processor landscape. Porting applications to

accelerated systems addresses two of the fundamental exascale challenges: exposing substantially higher levels of parallelism and restructuring codes to minimize data traffic. As Moore’s law is, in the view of many, facing headwind, additional hardware changes are likely to come for HPC systems, which could further impact applications, such as deepening memory hierarchies, nonvolatile memories and possibly application-level power management and application-managed resiliency beyond checkpoint/restart. The porting of applications to accelerated systems is a step forward in making these codes well-poised for migration to exascale and beyond.

Acknowledgments

The CAAR project was a large-scale effort requiring the participation of many individuals. The authors would like to thank Gregory Ruetsch, Adrian Tate, Massimiliano Fatica, Peng Wang, Yang Wang, Aurelian Rusanu, Ilene Carpenter, Paulius Micikevicius, Kate Evans, Jim Hack, Oscar Hernandez, Mark Taylor, J. F. Lamarque, John Dennis, Jim Rosinski, Sarah Anderson, Scott Le Grande, Steve Plimpton, Paul Crozier, Axel Kohlmeier, Kevin Thomas, Cyril Zeller, John Roberts, Thomas Evans, Christopher Baker, Gregory Davidson, Steven Hamilton, Joshua Jarrell, Nathan Wichmann, Peter Lichtner, Rebecca Hartmann-Baker, Manuel Rodriguez, Zhen Wang, Doug Kothe, Steve Poole, Ricky Kendall and Jim Schwarzeimer.

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.

References

- [1] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid GPU accelerated manycore systems. Technical Report UT-CS-08-632, University of Tennessee, LAPACK Working Note 210. <http://www.netlib.org/lapack/lawnspdf/lawn210.pdf>. [Online; accessed 7-October-2014].
- [2] S. Poole, Darkhorse: A proposed petascale architecture, in: High Performance Extreme Computing Conference 2006, https://www.ll.mit.edu/HPEC/agendas/proc06/Day2/27_Poole_Panel.pdf [Online; accessed 15-January-2015].
- [3] Cray-1 supercomputer 30th anniversary, <https://www.youtube.com/watch?v=J9kobkqAicU> [Online; accessed 7-October-2014].
- [4] B. L. Buzbee, J. F. Morrison, Proceedings of the 1978 LASL workshop on vector and parallel processors <http://www.gbv.de/dms/tib-ub-hannover/528549871.pdf>. [Online; accessed 7-October-2014].
- [5] A. E. Koniges, K. R. Lind, P. Dubois, Parallelizing code for real applications on the T3D, Computers in Physics 9 (4) (1995) 399–405. doi:<http://dx.doi.org/10.1063/1.4823422>. URL <http://scitation.aip.org/content/aip/journal/cip/9/4/10.1063/1.4823422>
- [6] P. Henning, Programming Roadrunner (and everything else) http://www.lanl.gov/orgs/hpc/roadrunner/rrinfo/RR%20webPDFs/Henning_Milagro_final1.pdf. [Online; accessed 7-October-2014].
- [7] A. E. Koniges (Ed.), Industrial Strength Parallel Computing, Morgan Kaufmann, 1999.
- [8] W. Joubert, D. Kothe, H. A. Nam, Preparing for exascale: ORNL leadership computing facility application requirements and strategy <https://www.olcf.ornl.gov/wp-content/uploads/2010/03/olcf-requirements.pdf>. [Online; accessed 7-October-2014].
- [9] C. G. Baker, G. G. Davidson, T. M. Evans, S. P. Hamilton, J. J. Jarrell, W. Joubert, High performance radiation transport simulations: Preparing for TITAN, in: Proceedings of Supercomputing Conference SC12, 2012.
- [10] W. M. Brown, T. D. Nguyen, M. Fuentes-Cabrera, J. D. Fowlkes, P. D. Rack, M. Berger, A. S. Bland, An evaluation of molecular dynamics performance on the hybrid Cray XK6 supercomputer, Procedia Comput. Sci 9 (2012) 186–195.
- [11] M. Norman, L. Larkin, R. Archibald, I. Carpenter, V. Anantharaj, P. Micikevicius, Porting the Community Atmosphere Model spectral element code to utilize GPU accelerators, in: Proc. of Int. Cray User Group Meeting, Hamburg Hamburg, 2012.
- [12] J. M. Levesque, R. Sankaran, R. Grout, Hybridizing s3d into an exascale application using openacc: An approach for moving to multi-petaflops and beyond, SC Conference 0 (2012) 1–11. doi:<http://doi.ieeecomputersociety.org/10.1109/SC.2012.69>.
- [13] B. Philip, Z. Wang, M. A. Berrill, M. Birke, M. Pernice, Dynamic implicit 3d adaptive mesh refinement for non-equilibrium radiation diffusion, Journal of Computational Physics 262 (2014) 17.
- [14] Y. Cui, E. Poyraz, K. B. Olsen, J. Zhou, K. Withers, S. Callaghan, J. Larkin, C. Guest, D. Choi, A. Chourasia, Z. Shi, S. M. Day, P. J. Maechling, T. H. Jordan, Physics-based seismic hazard analysis on petascale heterogeneous supercomputers, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC ’13, ACM, New York, NY, USA, 2013, pp. 70:1–70:12. doi:10.1145/2503210.2503300. URL <http://doi.acm.org/10.1145/2503210.2503300>

- [15] P. Staar, T. A. Maier, M. S. Summers, G. Fourestey, R. Solca, T. C. Schulthess, Taking a quantum leap in time to solution for simulations of high-Tc superconductors, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 1:1–1:11. doi:10.1145/2503210.2503282.
URL <http://doi.acm.org/10.1145/2503210.2503282>
- [16] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, HACC: Extreme scaling and performance across diverse architectures, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 6:1–6:10. doi:10.1145/2503210.2504566.
URL <http://doi.acm.org/10.1145/2503210.2504566>
- [17] M. Bernaschi, M. Bisson, M. Fatica, S. Melchionna, 20 petaflops simulation of proteins suspensions in crowding conditions, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 2:1–2:11. doi:10.1145/2503210.2504563.
URL <http://doi.acm.org/10.1145/2503210.2504563>
- [18] J. C. Phillips, J. E. Stone, K. Schulten, Adapting a message-driven parallel application to GPU-accelerated clusters, in: International Conference on High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008, 2008, pp. 1–9.
- [19] M. Bussmann, H. Burau, T. E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W. E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, R. Widera, Radiative signatures of the relativistic Kelvin-Helmholtz instability, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 5:1–5:12. doi:10.1145/2503210.2504564.
URL <http://doi.acm.org/10.1145/2503210.2504564>
- [20] K. Germaschewski, W. Fox, N. Ahmadi, L. Wang, S. Abbott, H. Ruhl, A. Bhattacharjee, The plasma simulation code: A modern particle-in-cell code with load-balancing and GPU support <http://arxiv.org/abs/1310.7866>. [Online; accessed 7-October-2014].
- [21] B. Joó, M. Clark, Lattice QCD experiences on TitanDEV, in: Accelerating Computational Sciences Symposium, Washington D.C., March 29, 2012, https://www.olcf.ornl.gov/wp-content/training/ascc_2012/thursday/ACSS12-Balint-Joo_s.pdf [Online; accessed 7-October-2014].
- [22] J. Foley, Lattice qcd - using milc and quda: Accelerating calculations at the high-energy frontier, in: GPU Technology Conference, March 25, 2014, <http://on-demand.gputechconf.com/gtc/2014/presentations/S4641-lattice-qcd-milc-quda.pdf> [Online; accessed 7-October-2014].
- [23] K. P. Esler, J. Kim, D. M. Ceperley, L. Shulenburger, Accelerating quantum Monte Carlo simulations of real materials on GPU clusters, Computing in Science and Engineering 14 (1) (2012) 40–51. doi:<http://doi.ieeecomputersociety.org/10.1109/MCSE.2010.122>.
- [24] S. Moore, E. Briggs, M. Hodak, W. Lu, J. Bernholc, C.-W. Lee, Scaling the RMG quantum mechanics code, in: Proceedings of the Extreme Scaling Workshop, BW-XSEDE '12, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2012, pp. 8:1–8:6.
URL <http://dl.acm.org/citation.cfm?id=2462077.2462085>
- [25] D. Komatitsch, D. Göldeke, G. Erlebacher, D. Michéa, Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs, Computer Science - Research and Development 25 (1-2) (2010) 75–82. doi:<http://link.springer.com/article/10.1007/s00450-010-0109-1>.
- [26] M. S. Warren, 2HOT: An improved parallel hashed oct-tree N-body algorithm for cosmological simulation, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 72:1–72:12. doi:10.1145/2503210.2503220.
URL <http://doi.acm.org/10.1145/2503210.2503220>
- [27] E. F. D'Azevedo, J. Lang, P. H. Worley, S. A. Ethier, S.-H. Ku, C.-S. Chang, Hybrid MPI/OpenMP/GPU parallelization of XGC1 fusion simulation code, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, http://sc13.supercomputing.org/sites/default/files/PostersArchive/tech_posters/post215s2-file3.pdf [Online; accessed 7-October-2014].
- [28] B. Bland, Titan: Early experience with the Titan system at Oak Ridge National Laboratory, in: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 2189–2211. doi:10.1109/SC.Companion.2012.356.
URL <http://dx.doi.org/10.1109/SC.Companion.2012.356>
- [29] J. Wells, What does Titan tell us about preparing for exascale supercomputers?, in: Programme du 32eme Forum ORAP, Maison de la Simulation, Saclay, France, October 10, 2013, <http://www.irisa.fr/orap/Forums/Forum32/Presentations/Wells.pdf> [Online; accessed 7-October-2014].
- [30] F. Foertter, Accelerating research and development using the Titan supercomputer, in: GPU Technology Conference, March 25, 2014, <http://on-demand.gputechconf.com/gtc/2014/presentations/S4760-accelerated-rd-titan-supercomputer.pdf> [Online; accessed 7-October-2014].
- [31] V. Anantharaj, F. Foertter, W. Joubert, J. Wells, Approaching exascale: Application requirements for OLCF leadership computing https://www.olcf.ornl.gov/wp-content/uploads/2013/01/OLCF_Requirements_TM_2013_Final1.pdf. [Online; accessed 7-October-2014].

Vitae

Rick Archibald obtained his Ph.D. in mathematics from the University of Alberta-Edmonton in 2002 and is currently a staff scientist in the Computational Mathematics group at ORNL. His research interests include developing uncertainty quantification algorithms for climate models and designing algorithms for the next generation of high-performance architecture.

Mark Berrill received his Ph.D. in Electrical Engineering from Colorado State University in 2010. He was a Wigner Fellow at Oak Ridge National Laboratory from 2010-2012 and was a DOE CSGF fellow from 2006-2010. His research interests are computational infrastructures, laser created plasmas, x-ray lasers and large scale parallel computing.

Michael Brown received degrees in Biochemistry, Computer Science, and Biomedical Sciences from the University of New Mexico. His interests generally involve enabling better research in life/material sciences through advances in high performance computing.

Markus Eisenbach is a computational scientist in the Scientific Computing Group at ORNL. His expertise lies in first principles calculations for magnetic materials and large scale scientific computing. He was a co-recipient of the SC'08 Gordon-Bell prize for his contribution to the DCA++ code development and the SC'09 Gordon-Bell prize for the Wang-Landau-LSMS code.

Ray Grout is an HPC applications researcher at the National Renewable Energy Laboratory. His current research interests include algorithmic advances to facilitate integrating partial differential equations numerically on future architectures and development of future computational fluid dynamics capabilities with particular emphasis on reacting flows.

Jeff Larkin received his B.S. and M.S. in Computer Science at Furman University and The University of Tennessee, respectively. He works in the developer technologies group at NVIDIA where he assists users develop applications on GPU platforms and represents the company to the OpenACC and OpenMP standards. Jeff previously worked in Cray's Supercomputing Center of Excellence at ORNL.

Wayne Joubert received his Ph.D. in mathematics from The University of Texas in 1990. He worked at Los Alamos National Laboratory in the 1990s developing solver algorithms and software for high performance systems. Since 2008 he has been a member of the Scientific Computing Group at ORNL, where he specializes in performance analysis of applications and porting of algorithms and software to advanced architectures.

John Levesque graduated from University of New Mexico in 1972. He is now in his 47th year in HPC working on state-of-the-art supercomputers. He is currently in the CTO Office of Cray Inc. and Director of Cray's Supercomputing Center of Excellence for the Trinity installation at Los Alamos, NM. Mr. Levesque specializes in the optimization of scientific applications for the latest architectures and has authored numerous books on optimization during his career.

Bronson Messer received his Ph.D. in physics from the University of Tennessee in 2000. He was a postdoctoral fellow at Oak Ridge National Laboratory and The University of Chicago before becoming a staff member in the Scientific Computing Group at ORNL in 2005. His research interests include supernovae, stellar evolution, radiative transfer, astrophysical combustion, high performance computing and numerical relativity.

Matt Norman received his Ph.D. in Atmospheric Sciences from North Carolina State University under the Computational Science Graduate Fellowship. He is currently a computational climate scientist with the Scientific Computing Group at Oak Ridge National Laboratory. His research includes scalable algorithms development, code optimization for massively parallel computers and porting large codebases to accelerators.

Bobby Philip has a Ph.D. in Applied Mathematics from the University of Colorado at Boulder. He is a Senior R&D Staff Member at Oak Ridge National Laboratory. His research interests include adaptive mesh refinement techniques, nonlinear and linear solvers for coupled systems of partial differential equations,

preconditioning, the design, development and optimization of scientific computing frameworks, and high performance computing.

Ramanan Sankaran received his Ph.D. in Mechanical Engineering at the University of Michigan, Ann Arbor in 2004. He has worked as a computational scientist at ORNL since 2006. His research interests are in high fidelity numerical simulations of problems in combustion and other engineering sciences using high performance computer systems.

Arnold Tharrington received his Ph.D. in Physics at the University of Pittsburgh in 2000. He now works as computational scientist in the Scientific Computing Group of the National Center of Computational Sciences at Oak Ridge National Laboratory.

John Turner received his Ph.D. in Nuclear Engineering from North Carolina State University in 1990. Until 1997 he worked at Los Alamos National Laboratory (LANL), when he joined Blue Sky Studios, earning credits on the feature film “Ice Age” and an Oscar-winning short animated film. In 2001 he returned to LANL and since 2008 has been a Group Leader at Oak Ridge National Laboratory.

Application performance improvement from use of GPUs on Titan

Titan Cray XK7 vs. Cray XE6