Preparation and Optimization of a Diverse Workload for a Large-Scale Heterogeneous System

Ian Karlin[1], Yoonho Park[2], Bronis R. de Supinski[1], Peng Wang[3], Bert Still[1]
David Beckingsale[1], Robert Blake[1], Tong Chen[2], Guojing Cong[2], Carlos Costa[2], Johann Dahm[2], Giacomo Domeniconi[2], Thomas Epperly[1], Aaron Fisher[1], Sara Kokkila Schumacher[2], Steven Langer[1], Hai Le[1], Eun Kyung Lee[2], Naoya Maruyama[1], Xinyu Que[2], David Richards[1], Bjorn Sjogreen[1], Jonathan Wong[1], Carol Woodward[1], Ulrike Yang[1], Xiaohua Zhang[1]
Bob Anderson[1], David Appelhans[2], Levi Barnes[3], Peter Barnes[1], Sorin Bastea[1], David Boehme[1], Jamie A. Bramwell[1], Jim Brase[1], Jose Brunheroto[2], Barry Chen[1], Charway R. Cooper[1], Tony DeGroot[1], Rob Falgout[1], Todd Gamblin[1], David Gardner[1], James Glosli[1], John Gunnels[2], Max Katz[3], Tzanio Kolev[1], I-Feng W. Kuo[1], Matthew P. Legendre[1], Ruipeng Li[1], Pei-Hung Lin[1], Shelby Lockhart[4], Kathleen McCandless[1], Claudia Misale[2], Jaime Moreno[2], Rob Neely[1], Jarom Nelson[1], Rao Nimmakayala[1], Kathryn O'Brien[2], Kevin O'Brien[2], Ramesh Pankajakshan[1], Roger Pearce[1], Slaven Peles[6], Phil Regier[1], Steve Rennich[3], Martin Schulz[5], Howard Scott[1], James Sexton[2], Kathleen Shoga[1], Shiv Sundram[1], Guillaume Thomas-Collignon[3], Brian Van Essen[1], Alexey Voronin[1], Bob Walkup[2], Lu Wang[1], Chris Ward[2], Hui-Fang Wen[2], Dan White[1], Christopher Young[1], Cyril Zeller[3], Ed Zywicz[1]
[1]Lawrence Livermore National Laboratory [2]IBM T. J. Watson Research Center [3]NVIDIA Corporation
[4]University of Illinois at Urbana-Champaign [5]Technical University of Munich
[6]Pacific Northwest National Laboratory [7]Sandia National Laboratories
{karlin1, desupinski1, beckingsale1, blake14, epperly2, fisher47, langer1, le36, maruyama3, richards12, sjogreen2, wong125, woodward6, yang11, zhang30, anderson110, pdbarnes, sbastea, boehme3, bramwell1, brase1, chen52, cooper67, degroot1, falgout2, gamblin2, gardner48, glosli1, kolev1, kuo2, legendre1, li50, lin32, mccandless2, neely4, nelson99, nimmakayala1, pankajakshan1, pearce7, peles2, regier1, scott6, shoga1, vanessen1, voronin1, wang84, white37, young110, zywicz1}@llnl.gov
{yoonho, chentong, gcong, chcost, eunkyung.lee, xque, dappelh, brunhe, gunnels, jhmoreno, kmob, caomhin, sextonjc, walkup, hfwen}@us.ibm.com, {johann.dahm, giacomo.domeniconi1, saraks, c.misale}@ibm.com, tjcw@uk.ibm.com
{penwang, lbarnes, mkatz, srennich, guillaumet, czeller}@nvidia.com
sll2@illinois.edu, schulzm@in.tum.de, shivsundram@gmail.com, slaven.peles@pnnl.gov, paregie@sandia.gov

ABSTRACT

Productivity from day one on supercomputers that leverage new technologies requires significant preparation. An institution that procures a novel system architecture often lacks sufficient institutional knowledge and skills to prepare for it. Thus, the "Center of Excellence" (CoE) concept has emerged to prepare for systems such as Summit and Sierra, currently the top two systems in the Top 500. This paper documents CoE experiences that prepared a workload of diverse applications and math libraries for a heterogeneous system. We describe our approach to this preparation, including our management and execution strategies, and detail our experiences with and reasons for using different programming approaches. Our early science and performance results show that the project enabled significant early seismic science with up to a 14X throughput increase over Cori. In addition to our successes, we discuss our challenges and failures so others may benefit from our experience.

KEYWORDS

performance, large-scale applications, heterogeneous systems, GPUs, programming models, project management

ACM Reference format:
Ian Karlin[1], Yoonho Park[2], Bronis R. de Supinski[1], Peng Wang[3], Bert Still[1], David Beckingsale[1], Robert Blake[1], Tong Chen[2], Guojing Cong[2], Carlos Costa[2], Johann Dahm[2], Giacomo Domeniconi[2], Thomas Epperly[1], Aaron Fisher[1], Sara Kokkila Schumacher[2], Steven Langer[1], Hai Le[1], Eun Kyung Lee[2], Naoya Maruyama[1], Xinyu Que[2], David Richards[1], Bjorn Sjogreen[1], Jonathan Wong[1], Carol Woodward[1], Ulrike Yang[1], Xiaohua Zhang[1], and Bob Anderson[1], David Appelhans[2], Levi Barnes[3], Peter Barnes[1], Sorin Bastea[1], David Boehme[1], Jamie A. Bramwell[1], Jim Brase[1], Jose Brunheroto[2], Barry

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356192

Chen[1], Charway R. Cooper[1], Tony DeGroot[1], Rob Falgout[1], Todd Gamblin[1], David Gardner[1], James Glosli[1], John Gunnels[2], Max Katz[3], Tzanio Kolev[1], I-Feng W. Kuo[1], Matthew P. Legendre[1], Ruipeng Li[1], Pei-Hung Lin[1], Shelby Lockhart[4], Kathleen McCandless[1], Claudia Misale[2], Jaime Moreno[2], Rob Neely[1], Jarom Nelson[1], Rao Nimmakayala[1], Kathryn O'Brien[2], Kevin O'Brien[2], Ramesh Pankajakshan[1], Roger Pearce[1], Slaven Peles[6], Phil Regier[1], Steve Rennich[3], Martin Schulz[5], Howard Scott[1], James Sexton[2], Kathleen Shoga[1], Shiv Sundram[1], Guillaume Thomas-Collignon[3], Brian Van Essen[1], Alexey Voronin[1], Bob Walkup[2], Lu Wang[1], Chris Ward[2], Hui-Fang Wen[2], Dan White[1], Christopher Young[1], Cyril Zeller[3], Ed Zywicz[1]. 2019. Preparation and Optimization of a Diverse Workload for a Large-Scale Heterogeneous System. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis, Denver, CO, USA, November 17–22, 2019 (SC '19),* 13 pages.
https://doi.org/10.1145/3295500.3356192

1 INTRODUCTION

Supercomputer procurements optimize for scientific output per dollar. Currently, abstract performance studies indicate that a heterogeneous, GPU-based design supports many scientific domains. However, many applications are not yet sufficiently prepared to exploit such systems. Since procurements often begin long before system delivery, many institutions now include explicit support for the significant time and effort to prepare for them in a Center of Excellence (CoE) [17, 19, 32]. We document the experiences, successes, and failures of a CoE that targeted a diverse workload for a large, unclassified, GPU-based system. We modeled this CoE after the Oak Ridge National Laboratory (ORNL) CoE that helped to prepare for Titan [11], ORNL's first heterogeneous system.

We identified applications that require significant high performance computing (HPC) resources to produce significant scientific results and that lacked GPU versions. Most of our applications and libraries had scalable MPI implementations that were already used on Blue Gene/Q systems at up to 98,304 nodes [3, 10, 16, 28]. Thus, we focused on the system's new hardware features for parallelism within a node. To ensure the full range of skills and knowledge, our center's teams included application scientists and institutional and vendor computer scientists. Overall, our Institutional Center of Excellence (iCoE) had these high-level goals:

- Maintaining application performance on existing systems;
- Achieving high-impact scientific results with multiple applications immediately upon system delivery;
- Running the complete application workload, including supporting tools and math libraries, well before system acceptance;
- Preparing institutional computational and computer scientists to be resources to other teams at project completion;
- Preparing our HPC ecosystem for future systems; and
- Integrating data science applications and methods.

We report on the successful preparation of our iCoE applications for GPU-based systems. These applications enabled scientists to perform high resolution seismic simulations, to use complex workflows and machine learning to improve our understanding of molecular mechanisms of cancer, and to design a new drone that has flown successfully. We also successfully ported key math libraries, which we tightly coupled with each other and in some cases to applications. Most importantly, many institutional staff now have experience in heterogeneous systems and can help other application teams. Crucial contributions of vendor experts included: compiler improvements; new GPU BLAS routines that an institutional solver uses; kernel optimizations; workload scheduling; performance analysis; and critical training of institutional personnel by working with them daily and through events such as OpenMP hackathons.

In order to enable others to learn from and to improve on our experience, we also discuss our challenges and failures. Challenges arose primarily because some activities were too ambitious for the timeline or not well suited to GPU acceleration. Availability of code and other key resources also caused challenges. Lastly, some decisions at the start of the project, long before hardware delivery, proved suboptimal when the hardware was delivered.

Throughout the paper, we highlight these key lessons learned:

- No programming model can meet all needs: CUDA provides optimal performance while RAJA and directive-based languages provide portability and a mixture of these approaches is often necessary to balance the concerns;
- Vendor porting support before system delivery is essential;
- Mini-applications are crucial to explore porting strategies;
- Suboptimal early decisions can be acceptable to ensure that an application is ready; and
- Challenges that exceed the available time and existing knowledge can arise when moving domains to new hardware.

To allow others to learn from our experience, public repositories for all open source projects now include their modified source and test cases. Also, we have released several iCoE mini-apps, including, sw4lite, which is now part of the Exascale Computing Project's (ECP) proxy application suite [23]. Another ECP proxy-app, Laghos, depends on our *hypre* and MFEM work to run on GPUs. ECP vendor co-design activities use both of these mini-apps.

The following sections describe our project in detail. We first discuss our project management philosophy, which included an exploration/training phase and an execution phase. We then describe the approach of each application team, including the reasons for diverse choices. We then present science and performance results from the applications. Finally, we discuss the challenges and lessons learned with each team from the perspective of six months after the end of the iCoE, as well as their future challenges.

2 PROJECT MANAGEMENT

The iCoE project started in October 2015 and ended in September 2018. Each activity had institutional and vendor points of contact. Each activity had an exploration and prototyping phase of 12 to 18 months. The remaining time was in the execution phase. Contractual considerations delayed participation of vendor experts until February 2016. While vendor experts made many valuable contributions throughout the project, the delayed participation impacted the project's initial prototyping and training phase.

We initially identified ten application teams. We dropped one, Parallel Discrete Event Simulation (PDES), after the first year due to significant unsolved research challenges. This change supported an increase in the Data Science effort. Another original activity, Chemistry Equations of State (CEOS) required a larger effort than originally anticipated and was moved to alternative funding in March of 2018 to allow the effort more time to complete. Finally,

Activity	Science Area	Base Language	Programming Approach(es)
Cardioid	Heart Modeling	C++	**DSL,** OpenMP, **CUDA**
Cretin	Non-LTE Atomic Kinetics	Fortran	**OpenACC, CUDA**
ParaDyn	Dislocation Dynamics	Fortran	**OpenMP,** OpenACC
Molecular Dynamics (MD)	Molecular Dynamics	C	**CUDA**
Seismic (SW4)	Earthquakes	Fortran ported to C++	**RAJA, CUDA**
Virtual Beamline (VBL)	Laser Propagation	C++	**RAJA**
Tools and Libraries	Math Frameworks	C/C++	**DSL, RAJA, Kokkos, OCCA, OpenMP, CUDA**
Data Science	DL and Data Analytics	PyTorch, Spark, C++	**Accelerate PyTorch, Spark**
Optimization Framework (Opt)	Design Optimization	C++	**CUDA, Job scheduler simulator**

Table 1: Completed iCoE activities and programming model approaches (Bold font indicates final approaches)

the biology activity split into Molecular Dynamics and Cardioid to serve the needs of those application spaces better.

Table 1 lists the programming model approaches and application domain areas of the nine completed activities. Seven activities (indicated by italics in the table) and PDES and CEOS were already running on large-scale systems. The applications span many science domains (including laser science, biology, math libraries, and data science) and a diverse set of computational techniques (including low and high-order stencils, finite elements, many body, linear algebra, parallel graphs, and machine learning inference and training). Code sizes also vary widely, from thousands to hundreds of thousands of lines of code. Performance profiles are diverse: Opt, Cardioid, and MD have a few hot kernels; Paradyn has a nearly flat performance profile across hundreds of loops. The diversity of science domains, algorithms, code sizes and performance profiles make the challenges that we encountered broadly applicable.

Institutional leadership met with each team every two months. Before a team began its execution phase, it had to define milestones that measured the success of their activity. We planned vendor efforts in six-month increments that culminated in written reports on the interactions with each team. We allocated vendor effort in each period to ensure it could add value to each activity.

The exploration phase included significant training such as onsite classes on GPU fundamentals. Hackathons that focused on topics such as code porting and OpenMP 4.5 provided more advanced training. We evaluated a range of approaches for *traditional* HPC applications and libraries, including RAJA, Kokkos, OpenMP, OpenACC, and CUDA. In the Data Science activity, we considered various deep learning and data analytics workloads; the team chose video analytics and Latent Dirichlet Allocation (LDA) as workloads to optimize. Multiple activities had workflow challenges that required hundreds to thousands of jobs to be scheduled. Thus, the iCoE explored several scheduling policies and workflow tools.

The execution phase had multiple interaction strategies. The teams met regularly at varying frequency (weekly to monthly). The Data Science institutional and vendor teams had multiple exploration meetings but once the activity goals were set the meetings were infrequent. Other teams, such as Cretin, required weekly meetings due to the close interactions between all participants.

2.1 Hardware Resources

Access to early versions of the target hardware was crucial to prototyping work and programmer training. Application teams started by using two GPU clusters already available on site: a visualization cluster with Sandybridge CPUs and K40 GPUs; and a dedicated, but smaller, development machine with Haswell CPUs and K80 GPUs. These initial machines were useful for early exploration, but lacked some key features and performance capabilities of the final hardware, in particular, the NVLink™ host interconnect to GPUs that supports richer CUDA Unified Memory capabilities.

We deployed early access (EA) systems in the second year of the project and the final production system in early 2018. The EA systems, one generation earlier than the final system, have Minsky nodes with two POWER8™ (P8) processors and four Pascal (P100) GPUs connected with the first version of NVLink. Their beta software environments gradually approached the final one. The final system has Witherspoon nodes with two POWER9™ (P9) processors and four Volta (V100) GPUs. Because its software environment was still undergoing refinement, only teams that needed access to test new hardware features or code scalability had immediate access. Most teams completed porting efforts in May 2018 or later.

3 GENERAL APPROACHES

With ten activities and fifteen distinct software products, iCoE programming approaches varied widely. Nonetheless, the activities share some common themes. We now discuss shared themes and then detail the activities.

3.1 Software Improvement

Many institutional applications are funded for science, with little time or money devoted to software maintenance or improvement. iCoE funding allowed these teams to modernize and to improve their code base significantly without worrying about research results. These changes support sustainable software.

SW4 first converted code from Fortran to C with the help of an internal compiler expert. The team adopted programming patterns that allowed the C version to match the performance of the Fortran version. The change to C simplified the use of RAJA, as well as CUDA, which led to a more portable RAJA-based code at reduced overall effort compared to the original CUDA Fortran plan.

The Tools and Libraries activity used iCoE to integrate their math libraries more closely. This activity, with a closely integrated, on-site vendor expert, created a math library ecosystem that simplifies efficient application use of the suite of libraries.

iCoE activity improved Cardioid portability and maintainability. Its mechanics solver used an inadequate finite element framework, and its electrophysiology solver relied on intrinsics for performance. The team ported the mechanics solver to MFEM and *hypre* to leverage the integrated math libraries. They reimplemented the electrophysiology solver with Melodee, a Domain Specific Language (DSL) that automatically generates high performance C++ and CUDA.

3.2 Mini-Apps

Developing a mini-app was often the best approach to prototyping. SW4 created sw4lite to explore RAJA and CUDA. Cretin created minikin and minicoll to try various optimization and parallelization strategies. minikin was turned into production code and included in Cretin, by adding the missing physics. ParaDyn performed most of its work in Dyna3D, which is a less restricted subset of the application, and created smaller test examples for key patterns to give to the vendor compiler team. The Cheetah team worked on a mini-app, Cheap, to prototype GPU porting; unfortunately, Cheap was not completed in time. The VBL activity created a mini-app with a node-level parallelization strategy that could target both GPUs and multicore systems. The techniques demonstrated in the mini-app will be incorporated into the full VBL application through work outside the scope of iCoE.

3.3 GPU Programming Approaches

With a diverse set of performance, portability, and productivity needs, application teams took differing approaches to port their code to GPUs. We discuss their final approaches individually. However, many teams first prototyped multiple options. SW4 looked at RAJA and CUDA. Cardioid explored their DSL, RAJA, OpenMP, and CUDA. MFEM looked at a tensor DSL, RAJA, OCCA, and CUDA. *hypre* and SUNDIALS tried RAJA, Kokkos, OpenMP, and CUDA.

Teams whose applications exhibit significant hot spots usually chose a low-level programming approach. With relatively little code to port, these teams can afford to rewrite their code for each new machine. Thus, the MD and Opt teams adopted CUDA to achieve a high percentage of peak – over 30% for the MD application. The Cardioid electrophysiology solver achieves a high percentage of peak through CUDA and use of a DSL to generate low-level code that can be customized for each system.

Applications with larger and more diverse code bases took higher level approaches. The C++ codes of SW4 and VBL used RAJA to support porting productivity including performance portability. The large Fortran codes of ParaDyn and Cretin used OpenACC throughout most of the project, but will switch to OpenMP once the compiler tool chain is mature since OpenMP is more widely supported and provides better performance on most platforms.

When possible, applications used math libraries to abstract their code while achieving high performance. VBL leveraged cuFFT, NVIDIA's CUDA Fast Fourier Transform library, to perform part of its split step algorithm [24]. *hypre* leveraged spmv calls in cuSPARSE, NVIDIA's CUDA Sparse Matrix library, to perform key operations in the AMG solve phase and Krylov solvers.

During the pathfinding stages of the project, multiple iCoE teams participated in OpenMP hackathons to speed the development and maturity of the OpenMP 4.5 compilers. During this time, the Cardioid and ParaDyn teams partnered with vendor application and compiler experts to port code to OpenMP 4.5. Since the work was on the early tool chain when things broke or did not perform as expected, fixes were often implemented on the spot. In addition, problems with the OpenMP standard were identified and reported back to the community, which led to future improvements [13].

4 ACTIVITIES

While some common themes emerged across activities, the iCOE project supported a diverse set of activities including production code optimization, deep learning exploration, large-scale data analytics, and the development of an experimental job scheduler. Below we discuss the unique challenges faced by each activity.

4.1 Cardioid

Cardioid solves the monodomain equations for electrical propagation in cardiac tissue. At every location and timestep, the application computes membrane ion transport (reaction kernels) and diffuses ions among nearby cells (diffusion kernels). The reaction kernels are embarrassingly parallel and computation-bound, usually involving the evaluation of 100-500 calls to math functions. The diffusion kernels are memory-bound stencil computations on a structured grid, with unique coefficients used at each point of the continuum.

In the exploration phase, the team implemented the reaction kernels in OpenMP and CUDA to find the fastest implementations. They manually performed several optimizations on the kernels such as changing memory layout and replacing expensive math functions with rational polynomials. The team found that replacing expensive functions with run-time rational polynomials was essential for top performance, and that changing run-time polynomial coefficients into compile-time constants could yield significant performance. Thus, they developed a DSL that automatically finds and replaces expensive math functions with rational polynomials, computes the coefficients at run-time, and uses an NVIDIA run-time-compilation library to produce high performance kernels for the GPU on-demand. After determining ideal transformations, the DSL automates their use across all available reaction kernels. The DSL also generates vectorized CPU code, yielding a portable code base that performs well on both CPU and GPU systems.

While exploring the diffusion kernels, the Cardioid team examined overlapping computation between the CPU (with diffusion kernels) and the GPU (the reaction kernels). OpenMP and CUDA versions of the diffusion code were generated and compared. However, the team found that the OpenMP and CUDA versions of their diffusion stencil performed comparably on the final hardware to CPU versions. Given the penalty for moving data to and from the CPU every iteration, the team decided to perform all computations on the GPU to minimize data migration.

In the execution phase, CUDA was used for diffusion and reaction kernels in order to exploit shared memory and run-time compilation

respectively on the GPUs. To achieve performance and portability, libraries were written to facilitate data transfer between the CPUs and GPUs and to parallelize simple loops on the GPU.

Lessons Learned: Explicitly instantiating constants at compile time can improve performance significantly. This lesson also applies to MFEM which uses JIT (Just In Time compilation) (see Section 4.10.3). OpenMP performance for some kernels is competitive with CUDA. This lesson also applies for Cretin with respect to OpenACC (see Section 4.3 below). Data transfer costs can be high enough that sometimes computation is better performed where the data is located rather than moving the data to the optimal processor.

4.2 Chemistry Equations of State (Cheetah)

The Chemistry Equations of State activity encountered multiple challenges. The work required to port the Cheetah code proved to be more than initially anticipated. The planned mini-app for the vendors to prototype GPU strategies was never delivered due to competing priorities for key developers and US Government export issues. Thus, the vendors never engaged with the team beyond answering a few general questions.

Despite these challenges, the team significantly improved the source code. The application was made thread-safe, variables were rescoped, and an initial GPU port was completed. However, GPU tuning was not completed. Instead, the activity transitioned to other funding in April 2018 to ensure continued GPU porting effort beyond the iCoE. Cheetah shows the challenge in scoping a three-year project as unanticipated issues can arise. Nonetheless, we still would have pursued this activity due to its strategic importance to the institution and the benefit that it brought to the code base.

Lessons Learned: When relying on outside expertise to assist in porting, non-technical issues that limit code access can cause significant delays. Understanding these risks sooner and being more aggressive about alternative paths is needed to ensure success.

4.3 Cretin

Cretin solves a system of rate equations to compute populations of various atomic configurations for situations in which a plasma is in non-local thermodynamic equilibrium [26]. Cretin uses atomic models that contain rate information for the different processes that connect the model's atomic configurations. The main computation calculates transition rates between pairs of states, forms a rate matrix from them, and inverts that matrix to update the populations. The populations are used to calculate frequency-dependent opacities required for a radiation transport calculation.

Many institutional multiphysics applications use the Cretin atomic kinetics package. Solution of the rate equations entails significant computation in calculating several types of transition rates and inverting the rate matrix. Thus, applications must limit the complexity of their atomic models. Cretin's GPU port will enable more complex models to be used in applications including inertial confinement fusion simulations. For example, Figure 1 shows a hohlraum model used in inertial confinement fusion experiments (ICF). Cretin GPU code performance and memory improvements allow the use of larger atomic models that more accurately capture the physics of the gold hohlraum walls.

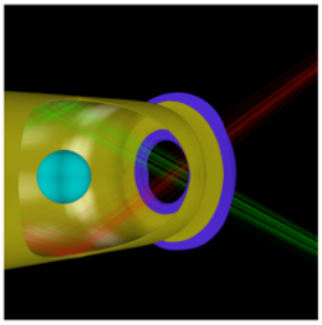

Figure 1: Gold hohlraum used in ICF experiments

Much of the exploration phase focused on using CUDA Fortran to port a few different mini-apps. Each mini-app focused on calculating one type of transition rate, as each type posed a different parallelization issue for GPUs. Although initial performance of the mini-apps was promising, the ports resulted in separate code paths for the CPU and GPU versions. Since Cretin contains many physics routines, this solution would be difficult to maintain. In the execution phase, the team focused on using OpenACC to port the minikin code, which included versions of each of the rate calculations. We found that OpenACC performance was identical to CUDA Fortran while only requiring one code path for both CPUs and GPUs.

Cretin uses both direct and indirect solvers to invert the rate matrices, depending upon matrix size. The direct solver uses NVIDIA's cuSOLVER library. For the iterative solver, we first wrote a Fortran interface to the NVIDIA AMGX library and its variety of solver options. Unfortunately, AMGX can only solve one (potentially large) system at a time, while Cretin must solve multiple systems simultaneously. Thus, we used the cuSPARSE library to construct our own iterative solver, which required some additional coding in CUDA Fortran and CUDA C. Cretin now combines OpenACC, CUDA Fortran, and CUDA C in different parts, demonstrating the value of interoperability across approaches. Also, the ability to link to AMGX supported exploration of different solvers and preconditioners, and identification of the best configurations for our problem. During this process, we encountered several CUDA and OpenACC (PGI) bugs that were fixed in later releases.

The CPU and GPU minikin versions have one important difference. The GPU version, which is threaded over atomic transitions, only needs enough GPU memory to process one zone. Each thread in the CPU version needs enough private memory to process one zone, which prevents the use of some CPU cores for large models.

The GPU version achieves good speedups over the CPU version. For our second largest atomic model, the GPU processing rate per node is 5.75X the rate for CPUs. GPU speedup is much higher for the largest atomic model because memory constraints require idling 60% of CPU cores. Future work will port the new threading approach to CPUs to enable large atomic models on CPUs. minikin has been installed in HYDRA, an important multiphysics application.

Lessons Learned: Cetin development has historically focused on improving the accuracy of the opacities. iCoE provided the opportunity to investigate code restructuring to improve performance. The fine-grained threading approach developed for the GPU will

lead to 2.5X speedups or more for large atomic models when implemented on the CPU. The code restructuring efforts during a focused GPU porting project are likely to improve the CPU performance of other codes that have spent many years focused on physics improvements.

4.4 Data Science: Data Analytics

The exploration phase of the Data Science Data Analytics activity focused on SparkPlug, a home-grown density estimation toolbox based on Apache Spark, and its Variational Expectation-Maximization (EM) implementation of Latent Dirichlet Allocation (LDA). LDA is a generative statistical model that can explain observation sets through unobserved groups that describe why some parts of the data are similar. This widely used clustering/latent factors model is often part of analytics pipelines. Our goal, improved scaling, would enable LDA on large dictionaries and topic counts.

The vendor team studied LDA performance and scalability in detail, which exposed performance dependencies, inefficiencies and limitations, and guided targeted optimizations. The team applied LDA to the entire Wikipedia corpus, including 390 different languages with a total dictionary size of more than 54 million unique words, which is a significant scaling challenge. In the profiling phase, the team identified the main scalability bottlenecks as overheads in the Java Virtual Machine (JVM) that Spark uses, Spark's implementation of shuffle (all-to-all communication), and Spark's aggregate (all-to-one communication) data management primitives.

In the execution phase, the team achieved significant performance impact and improved scaling by leveraging Spark-targeted optimizations in the IBM Java SDK (including the OpenJ9 JVM) and changes to the shuffle algorithm implemented in Spark. At the JVM level, more efficient garbage collection and lock contention schemes, as well as reduced serialization/deserialization overheads improved performance. At the Spark level, we reduced shuffle overhead with an adaptive shuffle mechanism [20, 21]. We also improved scaling by changing SparkPlug to use more scalable all-to-one operations. With a combination of these optimizations, the team mitigated scalability limitations and demonstrated LDA on large datasets at large scales. Figure 2 shows the breakdown of the aggregate time for the main compute and communication steps, showing a significant performance improvement of more than 2X over the default, non-optimized stack. The data shown in Figure 2 was collected on 32 nodes of the final system. These optimizations allowed the team to run LDA successfully on the entire Wikipedia corpus using 256 nodes of the final system.

The team also found an additional possible optimization with a Spark adapter for Data Broker. The Data Broker provides common shared, in-memory storage [25]. The work created new optimization opportunities that can scale topic modeling with LDA even further.

Data science work on the graph code HavoqGT [22] demonstrated the value of NVMe to applications. Table 2 shows historical graph data. Porting involved ensuring scalability and determining whether the use of CPUs or GPUs was more efficient. Using the 1.6 TB of NVMe on each node and CPUs for compute we can run larger graph problems faster.

Lessons Learned: Because new supercomputers usually contain the latest technologies, popular open-source middleware such as Spark cannot fully exploit the scale and technologies that the systems provide on day one. Therefore, significant investment is needed to use the systems successfully on day one in, for example, accelerator enablement, network enablement, and new scalable algorithms in open-source middleware. With the increasing importance of data analytics in scientific workflows, open science efforts could also benefit from a better engagement with the open-source data analytics community by facilitating early access to new systems.

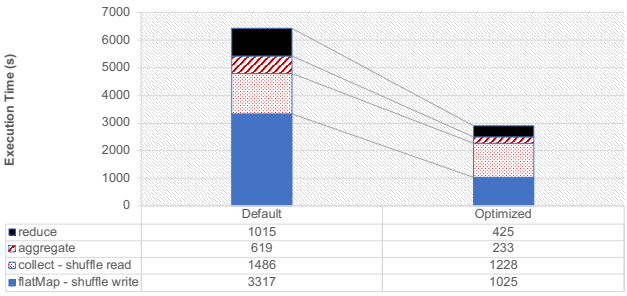

Figure 2: Default vs. optimized SparkPlug LDA performance

Machine	Year	Nodes	Scale	GTEPs
Kraken	2011	1	34	0.053
Leviathan	2011	1	36	0.053
Hyperion	2011	64	36	0.601
Bertha	2014	1	37	0.054
Catalyst	2014	300	40	4.175
Final System	2018	2048	42	67.258

Table 2: Historically best graph scale and performance

4.5 Data Science: Deep Learning

In the exploration phase, the Data Science Deep Learning activity investigated the performance of video analytics. Training on large video datasets can take days or weeks on small systems. The institutional team provided the initial video analytics application, and the vendor team experimented with performance optimizations.

The original application used Theano. The first task of the execution phase was to use single GPU computing resources fully. Since the original application used in-house versions of convolutions, the vendor team ported it from Theano to PyTorch, which is more mature and flexible. Also, PyTorch uses the highly optimized cuDNN (NVIDIA's CUDA Deep Neural Network library). This port reduced training time from weeks to days.

Efficient distributed training algorithms are required for further speedup. At the time, Asynchronous Stochastic Gradient Descent (ASGD) such as Downpour [6] and EASGD [33] were popular. As long as the gradient staleness is bounded by the number of learners, ASGD converges for nonconvex problems [14]. However, we found that, in theory and in practice, ASGD implementations have significant scaling issues [34]. Although ASGD has the same asymptotic convergence rate as SGD when the staleness of gradient update is bounded, the learning rate assumed for ASGD convergence is

	Combination Approach	UCF101	HMDB51
Single	Spatial Stream	85.06%	61.44%
	Temporal Stream	84.70%	56.34%
	SPyNet Stream	88.32%	58.69%
3-Stream	Simple Average	92.78%	75.16%
	Weighted Average	**93.47%**	77.45%
	Logistic Regression	92.60%	**81.24%**
	Shallow NN	93.18%	80.33%
	I3D [4]	93.40%	66.40%

Table 3: Validation accuracies for three stream approaches

usually too small for practical purposes. Also, an ASGD implementation cannot easily control the staleness in gradient updates as it is influenced by the relative processing speed of learners and their positions in the communication network. Further, the parameter server presents performance challenges with large GPU counts.

The team proposed a K-Step Averaging Algorithm (KAVG) that has better scaling properties than ASGD [34]. In KAVG, the learners start with the same initial weights. Each epoch partitions the training data among learners, and each learner runs SGD and uses a reduction to average their models every K steps. With KAVG, distributed training becomes bulk synchronous. KAVG scales better than ASGD. In addition, the optimal K for convergence is usually greater than one, so frequent global reductions are unnecessary for the best training results. KAVG, reduced training time for UCF101 and HMDB50 from weeks to a few hours on four EA nodes.

The team evaluated various neural network architectures and combination approaches for video analytics. Table 3 shows the validation accuracies of these approaches (higher is better). We developed a novel end-to-end trained temporal stream that extends SPyNet, a neural network for computing optical flow. This approach enhances the standard optical flow estimation (like TV-L1) by adding details from the original frames that can help to understand the movement context and, thus, action classification. Our ensemble method achieved state of the art accuracy for two stream video analytics training that does not rely on models pretrained from much larger datasets or costly 3D convolutions. For example, the validation accuracy on HMDB50 is 81.24% [5].

Another activity focused on extending LBANN [7] for large-scale training using a new distributed-training algorithm that enables training a model against a dataset that would otherwise be too large to fit in the device memory of a single GPU. Unlike the conventional parallel training algorithm, where only a mini-batch of samples is partitioned among multiple GPUs, the algorithm allows for further partitioning of each sample. It can also improve training performance by using a larger number of GPUs than the conventional data-parallel training method. The team developed an efficient implementation of the algorithm that exploits the system's unique capabilities such as NVLink. Figure 3 shows strong and weak scaling of neural network training for semantic segmentation using up to 2048 Volta GPUs. The four colored solid lines correspond to four configurations that use two to sixteen GPUs per sample to train the model and show good weak scaling trends when the total number of GPUs is increased. The model requires a large memory capacity to store the model in memory, which was larger than the available

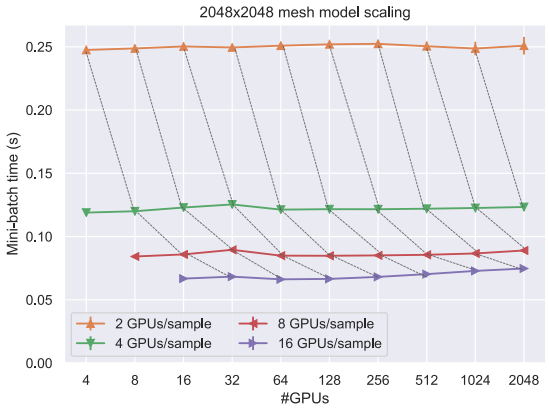

Figure 3: Performance of LBANN on up to 2048 GPUs [7]

capacity of the Volta GPUs, thus we had to use at least two GPUs per sample. The dotted lines that connect the points in the different colors indicate strong scaling trends where the model was trained with the problem size, i.e., the number of samples, being fixed but with different numbers of GPUs. Our results show near-perfect scaling when scaling from two GPUs to four GPUs per sample, and 2.8X and 3.4X speedups with eight and sixteen GPUs.

Lessons Learned: ML training frameworks sacrifice either accuracy or time to solution at scale. Efficient scaling requires additional research in distributed training algorithms and model parallelism.

4.6 Molecular Dynamics (MD)

Another project leveraged our molecular dynamics project to develop a novel, massively parallel, Multiscale Machine-Learned Modeling Infrastructure (MuMMI) to couple macro and micro simulations [18]. The micro simulations, the most computationally intensive component in MuMMI, use ddcMD [27] which is offloaded to GPUs. The iCoE Molecular Dynamics activity focused on ddcMD. Figure 4 illustrates how MuMMI uses the iCoE efforts.

Our ddcMD port uses CUDA because other approaches do not offer the same optimization possibilities. Currently, the biggest bottleneck for many MD codes is CPU-GPU bandwidth. To reduce the frequency of data transfers, we moved the entire MD loop to the GPU, including bonded and nonbonded energy terms, neighbor list construction, Langevin thermostat, Berendsen barostat, velocity Verlet integrator, constraint solver, and restraint.

Many force fields used in classical MD simulation build up the interaction between particles in terms of various short-range functions of the distance separating pairs of particles. Simple pair potentials examples include Lennard-Jones and exp6. However, various many-body potentials, such as embedded atom potentials, are also common. Given the ubiquitous need to process pairs of particles in MD potentials, we developed a templatized generic pair processing infrastructure that can be used to efficiently implement a diverse set of potential forms on GPUs. Our approach assigns multiple threads to each particle neighbor list while ensuring that each thread accesses contiguous memory regions. We tune the number of threads to achieve better latency hiding. To improve locality, we converted

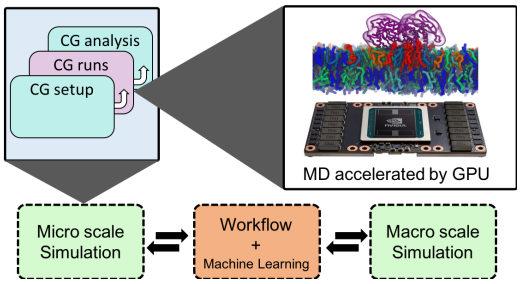

Figure 4: GPU acceleration of the MuMMI workflow

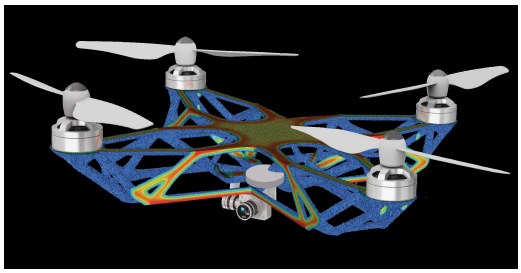

Figure 5: Drone designed with the Opt Framework

the array of structs to a struct of arrays. In lieu of shared memory reductions, we use shuffle-sync based reduction for the summation of forces, energies, and virials. The constraint solver kernel is an iterative kernel and relatively expensive to calculate. To ensure fast access to memory, launch time code generation is employed for rapid constant memory access and loop unrolling. The main challenge for the bonded kernels is serialization and marshaling of the nested, pointer-rich CPU data structures for a memcpy to the GPU. Other kernels are simple and straight-forward to implement.

Due to the iCoE efforts, ddcMD with the Martini force field [15] outperforms GROMACS [1], the community standard for Martini MD simulations. GROMACS uses an automated load balancing scheme to distribute calculations between GPUs and CPUs. For the Martini simulation, only 8 CUDA kernels are used in GROMACS as compared to 46 CUDA kernels in ddcMD. GROMACS uses single precision while ddcMD uses double precision. The average elapsed time for each MD step of ddcMD is 2.31 ms while it is 2.88 ms for GROMACS when using a combination of 1 GPU and 1 CPU. When using 4 GPUs, ddcMD is faster by a factor of 1.3 even though GROMACS also uses multiple CPUs. In the MuMMI framework, ddcMD is faster than GROMACS by a factor of 2.3 because MuMMI uses CPUs for the macro model and *in situ* analysis.

Lessons Learned: Performance dominated by a single kernel presents an opportunity to apply focused, low-level optimizations in order to achieve code improvement. Optimizations may take multiple paths (e.g., Cardioid was improved using a code generator, MD and Optimization Framework were improved using low-level CUDA implementations).

4.7 Optimization Framework (Opt)

The Opt activity had two goals: 1) an efficient GPU code; and 2) the ability to schedule thousands of small jobs that constitute a topology optimization design workload under uncertain conditions. Since the Opt team included developers with significant GPU knowledge, the vendor team assisted with the workflow aspect.

The collaboration began with the Opt team educating the vendor team about topology optimization workflows. The primary challenge is that a variable number of expensive GPU jobs are often necessary for topology optimization under different loading conditions. The team decided to develop a job scheduler simulator to study job scheduling policies with job requests that represent the behavior of the topological optimization application.

In the execution phase, the vendor team developed a job scheduler simulator and studied job requests that follow an arrival rate distribution and compared that to job requests that arrive in a batch. When job requests arrive according to a distribution, the team concluded that job arrival rate should be throttled to less than the aggregated processing capacity of the GPUs. When job requests arrive in a batch, the team concluded that Shortest Job First with Quota should be used to increase GPU utilization (assuming availability of job duration information). The team recommended that the resource manager implement these policies.

The Opt code is relatively small with a few hot kernels. By using a matrix-free solver implemented in CUDA and texture cache memory, the team achieved good performance on the EA system compared to an MPI-based CPU implementation. However, Opt did not benefit from texture caching on the final system due to improvements in Volta GPU caching. If this improvement was known in advance, the team may have used RAJA rather than CUDA.

Figure 5 shows the image of a drone that was designed using Opt. The fully functional drone has flown a test flight.

Lessons Learned: Starting early comes with risks and rewards. In hindsight, the decision to use CUDA for the Optimization Framework was suboptimal from a performance portability and code maintainability perspective. The decision to use CUDA was suboptimal because an abstraction layer such as RAJA would have been sufficient. However, CUDA use was necessary early in the project to make forward progress and to ensure the application performance goals were met on time.

4.8 ParaDyn

Optimization of ParaDyn for GPUs began with an OpenACC version. The initial port was slow due to kernel launch overheads because ParaDyn contains many small loops. To improve performance, we merged many loops into a single subroutine. This change reduced the number of kernel launches and CPU-to-GPU memory traffic since it stores fewer intermediate results. We also changed some key arrays to use a GPU-friendly data layout that redefines the array dimensions to support memory coalescing. We also explicitly specify OpenACC loop parallelization with the *gang* and and *vector* clauses. Otherwise, the PGI compiler makes suboptimal choices. Further, using large derived types for global parameters in OpenACC loops led to excessive GPU stack memory. Our optimized code replaces the derived types with scalar variables inside the loop. Finally, we changed the algorithm to avoid the use of GPU atomics.

Unfortunately, these optimizations, in particular, the merged loops, significantly decreased CPU performance. The existing small loops operate on a subset of the data that remains cache resident

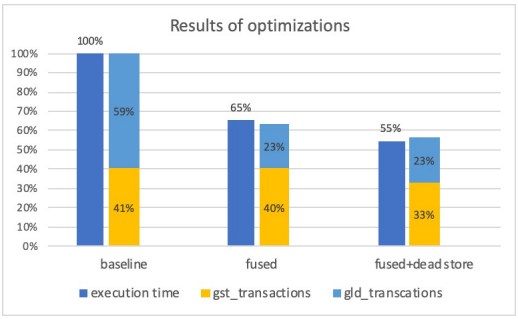

Figure 6: ParaDyn execution results: time and load/store

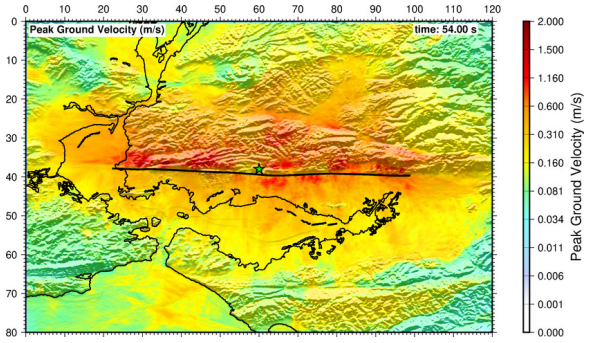

Figure 7: Simulated earthquake on the Hayward Fault

across loops, resulting in good CPU performance. Thus, the team investigated compiler-based methods to achieve good performance on both GPUs and CPUs.

Existing compilers such as GCC, LLVM, and IBM XL cannot exploit data reuse even in sequential code due to the lack of sufficiently powerful program analysis and performance prediction. Usually, parallelization complicates optimization further. However, we applied an optimization-aware parallelization approach to ParaDyn. When parallelizing ParaDyn with OpenMP, we explored how to extend existing OpenMP directives to convey programmer knowledge to the compiler. We also designed and implemented mechanisms in the compiler to extract useful information in OpenMP directives and an interface to help traditional optimizations. Specifically we added two new components to the IBM XL Fortran compiler.

First, we added a Single Level No Synchronization Parallelism (SLNSP) pattern to the compiler. In this pattern, each thread executes exactly one iteration of each loop without any added synchronization. Therefore, traditional data flow based optimization can work across different loops without explicit loop fusion.

Second, we modified the compiler to propagate the variables specified by private clauses to data flow analysis. This extra information helps compiler analysis identify and eliminate dead stores through the existing dead code elimination mechanism.

We tested the performance of these optimizations on a ParaDyn kernel. In addition to execution time, we measured the number of global load/store operations with NVProf. As Figure 6 shows, SLNSP improves performance by almost 2X, which roughly matches the reduction in the number of load operations. Dead store elimination improves performance by an additional 20%.

Lessons Learned: Existing performance portability tools are not always sufficient for all use cases. Some applications will need modifications or additional tool support.

4.9 Seismic (SW4)

We prepared the SW4 proxy app, sw4lite, in the exploration phase of the activity. We debugged RAJA, OpenMP, and CUDA versions of sw4lite. We also identified optimization opportunities.

In the execution phase, we focused on the CUDA version of sw4lite. The team improved CUDA kernels that perform stencil computation by almost 2X using fast on-chip shared memory, resulting in almost 40% of theoretical peak performance for some CUDA kernels. Performance was also improved by offloading all work in the main time-stepping routine to the GPU, merging small GPU kernels into larger ones, and overlapping GPU communication with computation. For example, offloading the forcing computation code yielded a 2X speedup with similar speedups achieved for boundary forcing. sw4lite is now an ECP proxy application.

We replaced the Fortran parts of the original SW4 code with C++ code in order to use RAJA. Thus, we adapted SW4 to use GPUs based on experience with sw4lite. sw4lite experiments indicated that this choice only moderately reduces performance (approximately 30%) compared to pure CUDA code for substantially less porting effort. Further, we can mix RAJA and CUDA to improve performance by recoding the most computationally intensive routines in CUDA.

As initial verification, SW4 simulated a magnitude 7.0 earthquake on the Hayward fault in the San Francisco Bay Area. The computation resolved frequencies up to 5 Hz and used 26 billion grid points. We compared results on the GPU system with those for the same computation run on Cori-II at NERSC. The computation on 256 GPU-equipped nodes executed in 10 hours on Sierra, which was almost the same time as required on Cori-II, and results from the runs agreed to within machine precision.

Other projects are now using SW4 to perform parametric studies of earthquake scenarios on the Hayward fault. These computations use regional size domains that resolve frequencies on the order of 5 Hz. The largest runs require up to 200 billion grid points. Figure 7 shows an example shake map with peak ground velocity in which red indicates strong shaking. The maximum resolved frequency is 5 Hz and north is directed to the left.

Lessons Learned: Switching to C/C++ provides more performance portability paths. At this time, the C/C++ ecosystem is much richer, and provides more open source solutions to leverage. Further, Fortran lacks features such as functors and lambdas, which eliminates some performance portability paths.

4.10 Tools and Libraries

The iCoE effort included key tools and libraries. Tools work enabled reading of key hardware performance counters. The libraries ported included *hypre* [2, 9], SUNDIALS [8, 31], MFEM [29], and SAMRAI [30]. As with many other activities, their preparation combined restructuring of the existing code with new code that specifically targets GPUs. However, these activities faced unique challenges because the libraries not only had to be ported to GPUs but also must integrate with each other and into a broad range of applications.

The exploration phase investigated methods to integrate the libraries while meeting the needs of the applications. Nonetheless, different libraries have unique application requirements and therefore used different porting strategies.

4.10.1 hypre. *hypre* historically used an MPI/OpenMP model but was not GPU-enabled. *hypre* provides both structured and unstructured algebraic multigrid (AMG) solvers, which use different data structures. The team focused on porting the solve phase of BoomerAMG, the unstructured AMG solver, and the structured solvers to the GPU. The structured solvers exploit problem structure and are abstracted with macros called BoxLoops. These macros were completely restructured to allow ports of CUDA, OpenMP 4.5, RAJA and Kokkos into the isolated BoxLoops. BoomerAMG consists of a setup and a solve phase. The solve phase, which can completely be performed in terms of matrix-vector multiplications, was ported to CUDA and OpenMP 4.5 with the inclusion of NVIDIA's cuSPARSE matvec routine for better performance and currently requires the use of Unified Memory. The setup phase, which consists of complicated components, has been kept on the CPU.

4.10.2 SUNDIALS. The team addressed support for GPU execution and integration with other libraries in GPU memory. SUNDIALS already expresses its vector and algebraic solver operations generically by abstracting the specific operations behind methods in backends. The team's approach leaves high-level control to the time integrator and nonlinear solver calls on the CPU, and supplies vector implementations that operate on data in GPU memory. The user calls a vector constructor on the CPU that allocates, then moves, a vector's data to the GPU so the time integrator and solvers can call its vector methods. The only time vector data needs to transfer back to the CPU is when a user needs it for I/O purposes.

4.10.3 MFEM. The MFEM team determined early on that the library's existing algorithms were the wrong choice for GPUs. Thus, they sought not only to port the code to GPUs and to integrate with other libraries, but rewrote the core algorithms to use sum factorization and to employ partially or completely matrix-free operator representations. The team explored the performance of several new algorithms on the EA hardware. A significant portion of the later effort focused on integrating MFEM with other libraries, since MFEM integrates with *hypre* and SUNDIALS internally in its linear solvers and time integrators.

In order to achieve the highest performance with these matrix-free algorithms, the loop bounds must be known at compile time. Thus, just-in-time compilation was identified as an area where software tools and compilers must improve. Several libraries have been developed to address this need, including Acrotensor, which calls the NVIDIA run-time-compilation library, and OCCA, which calls the NVIDIA NVCC compiler to perform code translation.

4.10.4 Integration. As the *hypre*, MFEM, and SUNDIALS teams prepared their libraries, they also coordinated and worked toward library interoperability by adding new interfaces. For example, CUDA Unified Memory support was added to the MFEM matrix classes to support integration of *hypre*'s unstructured solvers. MFEM owns the vector data wrapped by the SUNDIALS GPU vector class, so that the data stays on the GPU. The reduced CPU-to-GPU memory copies have proven critical for performance.

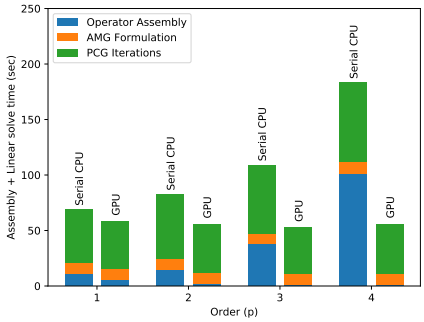

Figure 8: Timing breakdown of nonlinear diffusion problem

Unknowns	$p = 2$	$p = 4$	$p = 8$
20.8×10^3	2.88	2.78	4.97
82.6×10^3	6.67	8.00	12.47
329.0×10^3	10.59	13.71	19.00
1.313×10^6	12.32	14.36	20.80

Table 4: GPU speedup using MFEM, HYPRE, and SUNDIALS

The GPU integration of the libraries has provided an enormous scaling and performance boost. In order to quantify this benefit, we solved a nonlinear time-dependent diffusion problem using high-order continuous finite element discretization, the SUNDIALS CVODE implicit time integrator, and *hypre*'s BoomerAMG preconditioner on a low-order refined version of the finite element operator as a preconditioner for the linear solver.

Figure 8 shows the timing breakdown of the linear system formulation using the SUNDIALS time discretization, and preconditioner and solve phases using MFEM and *hypre* on a nonlinear diffusion problem with 1M degrees of freedom. The timings with the new partial assembly algorithms compare the time to solution with a single CPU thread on a P8 with that on one P100. Table 4 shows the GPU speedup using the P9+V100 system on 20 timesteps of the transient problem over the serial GPU code performance with the same algorithms. These new algorithms reduce not only the time to solution but also the memory footprint, so many applications can now tackle larger and more realistic problems because they can scale to more accurate high-order methods.

4.10.5 SAMRAI. The SAMRAI team developed a portable and high-performance GPU implementation that uses the RAJA and Umpire libraries. This implementation can also be used on CPUs, replacing older FORTRAN code used for numerical kernels in SAMRAI. The team also used the iCoE project to modernize the code and build system, and to implement modern software development best practices, including moving the software repository to GitHub, setting up continuous integration tests, and replacing library dependencies with new C++11 features. The GPU implementation achieves high performance by ensuring that data remains in device memory as long as possible, reducing unnecessary CUDA Unified

	Full Node	P9 vs. V100
CPU time (s)	127.5	74.0
GPU time (s)	17.86	5.0
Speedup	7X	15X

Table 5: CleverLeaf mini-app performance using SAMRAI

Memory traffic. Further, all data is allocated from memory pools that Umpire provides, which amortizes the cost of these allocations.

We assessed SAMRAI performance with the CleverLeaf mini-app, which solves the Euler equations. Although the performance improvement compared to the original code is difficult to assess since it does not run on GPUs, the results in Table 5 demonstrate an improvement of approximately 7X using four V100 GPUs compared to two P9 CPUs. The CPU results employ coarse-grained parallelism with 11 MPI processes per socket while the GPU results use the RAJA CUDA backend with 256 threads per block.

4.10.6 Tools. Many HPC applications are memory-bandwidth bound. Understanding the bandwidth that an application uses is crucial to performance tuning. Historically, for security reasons, no production systems at our computer center have allowed access to uncore (Intel) and nest (IBM) counters that are not bound to a core.

Our tools experts worked with vendor partners to define requirements for the necessary hardware and kernel software modifications. They also improved the interfaces to access memory counters. The facilities tools team then used Performance Co-Pilot [12] to access the nest counters on the P9. The improvements now allow regular users to access the nest counters on the P9.

Lessons Learned: Performance gains from tight coupling of libraries can be significant. Success requires coordination of data ownership, efficient passing of data structures between libraries, and clear, well-designed interfaces to reduce data motion and copies. Close coordination of teams enables accepting data from other libraries through wrappers and added support for other memory models. Examples include coordination of the library that is responsible for the allocation and deallocation of data structures with the ability of other libraries to accept pointers to them.

4.11 Virtual Beamline (VBL)

During the exploration phase, the VBL team chose RAJA as the parallelization approach since it supports multiple types of node-level parallelism without dictating data structure implementations. Computationally, VBL's split-step algorithm has two main parts [24]: discrete fast Fourier transforms and triply-nested loops that update the electric field. cuFFT was used to perform the FFTs, and RAJA's forallN was used to parallelize the multiply nested loops.

The execution phase had two parts. The first part parallelized the VBL mini-app with RAJA. Parallelizing the full amplifier step, which is a major computational component of the VBL mini-app, significantly improved overall GPU performance. The team identified the transpose as an algorithmic bottleneck. They implemented a tiling transpose in RAJA and directly in CUDA. Ultimately, the native CUDA transpose significantly outperformed the RAJA one.

The second part explored the use of GPUDirect™ and updated the RAJA loops to the new RAJA Kernel API, the successor to forallN. Initial measurements showed that using cudaMemcpy for transfers from CPU to GPU will overtake GPUDirect for transfers of a few kilobytes or more; and for transfers from GPU to CPU for a few hundred bytes or more. VBL uses CUDA Unified Memory, which is equivalent to transferring blocks of 64 kilobytes.

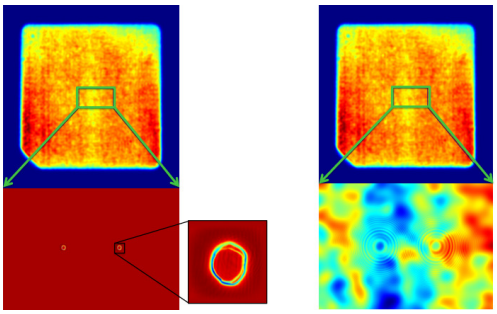

Figure 9: VBL simulation results

With the upgraded computational capabilities effects that were not resolvable before can be seen. Figure 9, in which two 150 micron phase defects (lower left) cause ripples to appear in the fluence of the beam after propagating 10 meters (right), shows one such simulation advantage.

Lessons Learned: Using an abstraction layer that is embedded in the base language increases flexibility significantly. Being able to mix RAJA and CUDA enables productivity when needed and performance when required.

5 CONCLUSIONS AND LESSONS LEARNED

The iCoE project shows the value of dedicated effort and funding to prepare for a technologically advanced system. Being able to have applications run from day one was crucial to extracting scientific value from the system and to stress testing the system during installation, stabilization, and acceptance. Our iCoE demonstrates the importance of pulling together computational and computer scientists from the host institution with vendor experts to leverage unique expertise and skills. In particular, without the vendor partnership, the project could not achieve the compiler and math library improvements that were critical to the success of some applications. Throughout the project, significant training opportunities with vendor experts were key to the project's success.

Starting this project three years before system delivery brought risks, rewards, and challenges. The hardware was not fully defined at project start. The efficacy of some features was unknown. The target software stack existed but was often incomplete and alpha quality. However, opportunities to influence software design to support early success accompanied these challenges.

The project would have required less time and effort if we waited until we had a stable software stack and production hardware. However, early science runs on the final system would have been delayed, and the lost system productivity would have exceeded the cost of the extra effort. Significant time would have been saved by not having to prototype multiple solutions. However, this early exploration allowed us to address multiple software issues.

As with any large project, flexibility was key. We shifted funding emphasis and vendor support to maximize effectiveness as needs

and priorities changed. Our Data Science activities grew throughout the project, while we moved applications with significant research or longer-term needs to other funding. The Data Science activities led to important contributions in scaling ML algorithms, ML frameworks, and data analytic frameworks that enable new ways to use our system. These activities will continue through another CoE project. Ongoing research will port the AMG setup phase in *hypre* to GPUs. The system's computing capabilities coupled with high performing code exposed a gap in the SW4 algorithms. Further SW4 development work under ECP will address these gaps to model slower wave speeds. Other funding continues other iCoE activities including SAMRAI, MFEM, Cretin, and Cheetah.

While EA systems were crucial to start iCoE activities early to port the applications, these systems did not always provide accurate guidance. Opt ran into one risk of starting early. Its preferred portability solution, RAJA, was not performant enough on the EA system because texture memory was needed for performance. Therefore, a CUDA implementation was used. However, the improved caching of Volta made texture memory unnecessary so RAJA would have been sufficient. However without starting early we would not have already used the application on our system to design drones. SW4 day one readiness resulted in significant science. It was also used to track system performance as the software stack improved. The iCoE had many smaller auxiliary benefits. Its activities led to improvements in the OpenMP 5.0 specification. The initial system software stack became significantly more robust. The lessons learned from the iCoE are helping others and ourselves to reduce porting and tuning efforts.

iCoE applications were early adopters of new software engineering technologies. Multiple applications used the RAJA programming model and the Umpire data movement framework. To continue the iCoE's momentum towards an integrated software strategy, the Radius project was started in October 2018. This project will help institutional applications to improve their software quality by leveraging common packages and best practices in software development, such as continuous integration. Similar to iCoE, this project will fund computer scientists to work with application teams to integrate and to improve common institutional software.

The iCoE project significantly improved developer skill sets and application software quality. Thus, the state of the institutional computing ecosystem is much stronger. However, most importantly, the project ensured that the institutional computing environment supports its diverse workload.

6 ACKNOWLEDGEMENTS

We dedicate this paper to Bert Still who ran the iCoE project until his untimely passing in September of 2018. His impact on computational science and HPC was large and his legacy will live on both technically and through the people he worked with and mentored. Prepared by LLNL under Contract DE-AC52-07NA27344. LLNL-CONF-772139. IBM and NVIDIA participation was supported under CORAL NRE Contract B604142.

REFERENCES

[1] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (2015).

[2] A.H. Baker, R.D. Falgout, T.V. Kolev, and U.M. Yang. 2012. Scaling *hypre*'s multigrid solvers to 100,000 cores. In *High-Performance Scientific Computing*. Springer, New York, NY.

[3] A.H. Baker, A. Klawonn, T. Kolev, M. Lanser, O. Rheinbach, and U.M. Yang. 2016. Scalability of classical algebraic multigrid for elasticity to half a million parallel tasks. In *Software for Exascale Computing - SPPEXA 2013-2015*, H.-J. Bungartz, P. Neumann, and W.E. Nagel (Eds.). Springer, New York, NY.

[4] J. Carreira and A. Zisserman. 2017. Quo vadis, action recognition? A new model and the Kinetics dataset. *CoRR* (2017). arXiv:1705.07750 http://arxiv.org/abs/1705.07750

[5] G. Cong, G. Domeniconi, J. Shapiro, C.C. Yang, and B. Chen. 2019. Video action recognition with an additional end-to-end trained temporal stream. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV '19)*. IEEE.

[6] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A.Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 1 (NIPS '12)*. Curran Associates, Red Hook, NY.

[7] N. Dryden, N. Maruyama, T. Benson, T. Moon, M. Snir, and B. van Essen. 2019. Improving strong-scaling of CNN training by exploiting finer-grained parallelism. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '19)*. IEEE Press, Piscataway, NJ. *To appear.*

[8] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. 2005. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)* 31, 3 (2005).

[9] *hypre* Team. 2019. *hypre*: High Performance Preconditioners. https://www.llnl.gov/CASC/hypre/. (2019).

[10] H. Johansen, A. Rodgers, N.A. Petersson, D. McCallen, B. Sjogreen, and M. Miah. 2017. Toward exascale earthquake ground motion simulations for near-fault engineering analysis. *Computing in Science Engineering* 19, 5 (2017).

[11] W. Joubert, R. Archibald, M. Berrill, W.M. Brown, M. Eisenbach, R. Grout, J. Larkin, J. Levesque, B. Messer, M. Norman, B. Philip, R. Sankaran, A. Tharrington, and J. Turner. 2015. Accelerated application development: The ORNL Titan experience. *Computers and Electrical Engineering* 46 (2015).

[12] N.L. Petroni Jr., T. Fraser, J. Molina, and W.A. Arbaugh. 2004. Copilot - A coprocessor-based kernel runtime integrity monitor. In *Proceedings of the Conference on USENIX Security Symposium - Volume 13 (SSYM '04)*. USENIX Association, Berkeley, CA.

[13] I. Karlin, T. Scogland, A.C. Jacob, S.F. Antao, G.-T. Bercea, C. Bertolli, B.R. de Supinski, E.W. Draeger, A.E. Eichenberger, J. Glosli, H. Jones, A. Kunen, D. Poliakoff, and D.F. Richards. 2016. Early experiences porting three applications to OpenMP 4.5. In *OpenMP: Memory, Devices, and Tasks*, N. Maruyama, B.R. de Supinski, and M. Wahib (Eds.). Springer, New York, NY.

[14] X. Lian, Y. Huang, Y. Li, and J. Liu. 2015. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 2 (NIPS '15)*. MIT Press, Cambridge, MA.

[15] S.J. Marrink, H.J. Risselada, S. Yefimov, D.P. Tieleman, and A.H. de Vries. 2007. The MARTINI force field: Coarse grained model for niomolecular simulations. *The Journal of Physical Chemistry B* 111, 27 (July 2007).

[16] A.A. Mirin, D.F. Richards, J.N. Glosli, E.W. Draeger, B. Chan, J.-L. Fattebert, W.D. Krauss, T. Oppelstrup, J.J. Rice, J.A. Gunnels, V. Gurev, C. Kim, J. Magerlein, M. Reumann, and H.-F. Wen. 2012. Toward real-time modeling of human heart ventricles at cellular resolution: Simulation of drug-induced arrhythmias. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '12)*. IEEE Press, Piscataway, NJ.

[17] H. Nam, G. Rockefeller, M. Glass, S. Dawson, J. Levesque, and V. Lee. 2017. The Trinity Center of Excellence co-design best practices. *Computing in Science Engineering* 19, 05 (2017).

[18] F. Di Natale, H.I. Ingólfsson, H. Bhatia, T. Carpenter, T. Oppelstrup, S. Kokkila Schumacher, X. Zhang, S. Sundram, T. Scogland, G. Dharuman, T. Bremer, L. Stanton, M. Surh, C. Neale, C. Lopez, S. Gnanakaran, C. Misale, L. Schneidenbach, C. Kim, B. D'Amora, D. Nissley, F. Streitz, F. Lightstone, and J.N. Glosli. 2019. A massively parallel infrastructure for adaptive multiscale simulation: Modeling RAS initiation pathway for cancer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19)*. IEEE Press, Piscataway, NJ. *Submitted.*

[19] R.J. Neely and B.R. de Supinski. 2017. Application modernization at LLNL and the Sierra Center of Excellence. *Computing in Science Engineering* 9, 5 (2017).

[20] B. Nicolae, C.H.A. Costa, C. Misale, K. Katrinis, and Y. Park. 2016. Towards memory-optimized data shuffling patterns for big data analytics. In *Proceedings of the International Symposium on Cluster, Cloud and Grid Computing (CCGRID '16)*. IEEE Press, Piscataway, NJ.

[21] B. Nicolae, C.H.A. Costa, C. Misale, K. Katrinis, and Y. Park. 2017. Leveraging adaptive I/O to optimize collective data shuffling patterns for big data analytics. *IEEE Transactions on Parallel and Distributed Systems* 28, 6 (June 2017).

[22] S. Poudel, R. Pearce, and M. Gokhale. 2015. *Towards scalable graph analytics on time dependent graphs.* Technical Report. Lawrence Livermore National Lab

(LLNL), Livermore, CA.
[23] D.F. Richards, O. Aaziz, J. Cook, H. Finkel, B. Homerding, P. McCorquodale, T. Mintz, S. Moore, A. Bhatele, and R. Pavel. 2018. *FY18 proxy app suite release. Milestone report for the ECP proxy app project.* Technical Report LLNL-TR-760903. Lawrence Livermore National Lab, Livermore, CA.
[24] R. Sacks, K. Mccandless, E. Feigenbaum, J.M.G. Di Nicola, K.J. Luke, W. Riedel, R.J. Learn, and B.J. Kraines. 2015. The virtual beamline (VBL) laser simulation code. *Proceedings of SPIE - The International Society for Optical Engineering* 9345 (Feb. 2015).
[25] L. Schneidenbach, C. Misale, B. D'Amora, and C.H.A Costa. 2019. IBM Data Broker. https://github.com/IBM/data-broker. (2019).
[26] H.A. Scott. 2001. Cretin–A radiative transfer capability for laboratory plasmas. *Journal of Quantitative Spectroscopy and Radiative Transfer* 71, 2 (2001).
[27] F.H. Streitz, J.N. Glosli, and M.V. Patel. 2006. Beyond finite-size scaling in solidification simulations. *Physical Review Letters* 96 (June 2006). Issue 22.
[28] F.H. Streitz, J.N. Glosli, M.V. Patel, B. Chan, R.K. Yates, B.R. de Supinski, J. Sexton, and J.A. Gunnels. 2005. 100+ TFlop solidification simulations on Blue Gene/L. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '05)*. IEEE Press, Piscataway, NJ.
[29] MFEM Team. 2019. MFEM: Modular Finite Element Methods Library. https://mfem.org. (2019).
[30] SAMRAI Team. 2019. SAMRAI: Structured Adaptive Mesh Refinement Application Infrastructure. https://computation.llnl.gov/projects/samrai. (2019).
[31] SUNDIALS Team. 2019. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. https://www.llnl.gov/CASC/sundials/. (2019).
[32] S.S. Vazhkudai, B.R. de Supinski, A.S. Bland, A. Geist, J. Sexton, J. Kahle, C.J. Zimmer, S. Atchley, S. Oral, D.E. Maxwell, V.G.V. Larrea, A. Bertsch, R. Goldstone, W. Joubert, C. Chambreau, D. Appelhans, R. Blackmore, B. Casses, G. Chochia, G. Davison, M.A. Ezell, T. Gooding, E. Gonsiorowski, L. Grinberg, B. Hanson, B. Hartner, I. Karlin, M. L. Leininger, D. Leverman, C. Marroquin, A. Moody, M. Ohmacht, R. Pankajakshan, F. Pizzano, J. H. Rogers, B. Rosenburg, D. Schmidt, M. Shankar, F. Wang, P. Watson, B. Walkup, L. D. Weems, and J. Yin. 2018. The design, deployment, and evaluation of the CORAL pre-exascale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ.
[33] S. Zhang, A. Choromanska, and Y. LeCun. 2015. Deep learning with elastic averaging SGD. In *Proceedings of the International Conference on Neural Information Processing Systems - Volume 1 (NIPS '15)*. MIT Press, Cambridge, MA.
[34] F. Zhou and G. Cong. 2018. On the convergence properties of a K-step averaging stochastic gradient descent algorithm for nonconvex optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '18)*. International Joint Conferences on Artificial Intelligence Organization.