# DOE Centers of Excellence Performance Portability Meeting

April 19-21, 2016

Glendale, AZ

Post-meeting Report

**LLNL-TR-700962**

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

## Table of Contents

## Executive Summary

Performance portability is a phrase often used, but not well understood. The DOE is deploying systems at all of the major facilities across ASCR and ASC that are forcing application developers to confront head-on the challenges of running applications across these diverse systems. With GPU-based systems at the OLCF and LLNL, and Phi-based systems landing at NERSC, ACES (LANL/SNL), and the ALCF – the issue of performance portability is confronting the DOE mission like never before.

A new best practice in the DOE is to include "Centers of Excellence" with each major procurement, with a goal of focusing efforts on preparing key applications to be ready for the systems coming to each site, and engaging the vendors directly in a "shared fate" approach to ensuring success. While each COE is necessarily focused on a particular deployment, applications almost invariably must be able to run effectively across the entire DOE HPC ecosystem. This tension between optimizing performance for a particular platform, while still being able to run with acceptable performance wherever the resources are available, is the crux of the challenge we call "performance portability". This meeting was an opportunity to bring application developers, software providers, and vendors together to discuss this challenge and begin to chart a path forward.

As the first large meeting of its size focused on multiple COE efforts and vendors, the main goals were to present the breadth of work happening across the DOE and vendor space, and begin building some lasting efforts to make progress toward these goals. Applications developers must understand the challenges and options they have, and vendors should understand the role they can play in helping the DOE succeed with this strategy. Early on in the planning, we agreed that despite the fact that much of the work going on within each COE still contains proprietary data held under Non-Disclosure Agreements, that we would make this meeting as open as possible. A set of guidelines were provided to participants to ensure that the tone of the meeting was not "us vs. them", but collaborative in nature. NDA sessions were held to a minimum, or as side meetings off the main agenda. However, participants were limited (with a few exceptions) to DOE staff covered under the proper NDAs from all vendors.

The meeting consisted of three days of prepared and invited talks, intermixed with breakout sessions aimed at encouraging group discussion on specific topics of interest. The agenda, links to the talks, and breakout session notes are available at https://asc.llnl.gov/DOE-COE-Mtg-2016. Along with some detailed descriptions of the breakout sessions and summaries and recommendations in this report, they indicate both an encouraging message that performance portability is within our grasp, but also a clear recognition that much work remains to be done in a relatively short amount of time.

A detailed account of the main takeaways is described later in this document starting on page 6. One of the first things we will do is arrange for monthly telecons between the COE leads to provide updates, and work toward a model for sharing more efforts between the labs and planning for a follow-on meeting to be held in the April 2017 timeframe.

Other takeaways are quickly summarized here as follows:

- There was great value in having multiple vendors available for discussion in one forum, and there is a desire to establish more of these outlets related to this topic

- MPI+X dominated the discussion. These is little evidence yet of alternative programming models addressing the performance portability issue in the context of large, complex applications
- Productivity should be included as an additional metric beyond performance and portability.
- Descriptive versus prescriptive programming concludes that overly-prescriptive styles in pursuit of performance on one platform can defeat performance portability. Compilers must ultimately take on more of the work
- Much of the audience is focused on future architectures under study in the COEs (KNL and GPUs). The timing of this inaugural meeting was unfortunately a bit early to have much in the way of concrete experience with these platforms
- Continued opportunities for cross-COE collaborations should be pursued where possible. However…
- Dissemination of information (e.g. best practices for performance portability) continues to be a difficult issue. There is no obvious central forum where multiple COEs can collect info while ensuring NDA information is protected
- C++ generic programming (templates) and other modern features offer nice abstraction options for generating platform-specific code from a single source (e.g. Kokkos and RAJA). The path for Fortran codes is far less clear
- The memory hierarchy (or NUMA domains) is seen as a major challenge with little in the way of portable solutions if the vendor-supplied hardware support for managing the memory spaces does not suffice for high performance codes
- COEs as an entity provide a good mechanism to bridge applications and software R&D
- COEs are adopting many practices that were successfully used in early co-design centers
- There is a natural tension in compiler implementations between supporting the latest features, and optimizing the ones they have.
- OpenMP4.x early experiences are promising, but directives are clearly not for everyone
- And finally, there is no universally accepted definition of performance portability.

There was general agreement from the steering committee and survey responses from attendees that a follow-on meeting (perhaps leading to an ongoing series) would be useful to pursue about a year following the original. A number of lessons learned and recommendations for a follow-on meeting are captured on page 42.

At the time this report was being finalized, Los Alamos National Lab had agreed to help pursue organizing a meeting in 2017. Ideas and suggestions can be sent to coepp-meeting@llnl.gov and they will be forwarded to the planning committee. We look forward to seeing you all there in 2017!

# Background

The Department of Energy (DOE) Centers of Excellence (COEs) Performance Portability Meeting provided an opportunity for personnel working with the five COEs to share ideas, progress, and challenges toward the goal of performance portability across DOE's large upcoming advanced architecture supercomputer procurements. The need for applications to run effectively on multiple vendor advanced architecture solutions (as well as on standard "cluster" technology) is pervasive across application teams within DOE and is a specified goal of the DOE's Exascale plans for risk mitigation.

Recognizing the challenges of porting and optimizing large applications to the advanced architecture systems planned for deployment within the National Nuclear Security Administration (NNSA) and Office of Science (SC) labs between 2016 and 2019, the DOE has established a COE at each laboratory siting one of these systems. These COEs provide direct vendor expertise to the application teams and, in turn, give the vendors deeper insight into how applications are run on those systems. Each of the five current COEs has a mission to optimize a set of applications for their specific platform—however the application teams are often motivated to maintain a code base that will run effectively across diverse vendor offerings. Making use of open standards, libraries, and software abstractions that allow for minimal code disruption without negatively impacting performance potential is the preferred path to programming, but it constitutes a large, as-yet-unsolved challenge.

This meeting built on past COE meetings or workshops, most of which were held at one of the labs hosting a particular COE. What this meeting attempted to provide was a forum for discussing best practices and ideas, focusing on achieving high performance on these emerging platforms without greatly sacrificing portability and maintainability of applications.

Participants at the Meeting included:

- Application developers and management at the existing five COEs (LLNL, LANL/SNL, ORNL, LBL/NERSC, and ANL) who are working on preparing their codes for next-gen architectures
- Vendors chosen to provide the next-generation platforms
- Solution-providers (DOE or third party) who are developing software tools aimed at helping application teams approach the challenges of performance portability.

Because it was open to all vendors participating in the COEs (IBM, Cray, Intel, and NVIDIA), a limited set of vendor talks were given under Non-Disclosure Agreements (NDAs), and accepted presentations were free of NDA material. From the outset participants were asked to join in the spirit of cooperation, and follow a base set of "ground rules" suggested to ensure a productive and non-competitive meeting. Attendance by parties outside of the six DOE labs, the lab collaborators, and the four vendors involved in current COEs was limited and by invitation only.

## Goals

Several primary goals were considered when planning for this meeting initially began. In addition, we list as sub-bullets a few secondary goals that emerged during the planning and execution of the meeting.

- Inform application teams and tool developers of activities and methodologies being used across the COEs, and foster informal relationships that can help DOE participants benefit from activities beyond their own COE.
    - Foster a better understanding of application strategies and research programs across the Office of Science and NNSA
- Identify major challenges toward the goal of performance portability, and work with the vendors and tool providers on determining implementations and solutions that will meet their own performance criteria without inadvertently impairing performance results elsewhere.
    - Take those challenges and begin thinking about a set of follow-on meetings that could build on this meeting
- Communicate to the vendor community the importance of performance portability to the DOE application community
    - Consider how performance portability can be emphasized in future procurements

As this report hopefully indicates, the goals of the meeting were met, but work must continue for them to be more fully realized.

## High-Level Takeaways and Follow-on Actions

What follows is a collection of high-level takeaways that were collected during and immediately following the meeting, presented in no particular order. Sub-bullets in italics represent some specific follow-on actions either to consider for subsequent meetings, or pursued and acted upon in the interim.

1) There was great value in having vendors involved. While meetings amongst DOE staff are common, as are interactions between individual labs and vendors, this forum provided a chance for vendor input to be heard in a broader forum. In particular, many participants noted that the breakout sessions offered great opportunities for candid vendor input on specific topics of interest that might not always come across in prepared talks

    - *Need to establish additional outlets for how to get vendors more engaged in discussions going forward. While vendor talks being shared between COEs are useful, it is the face-to-face back-and-forth on focused topics that held the most value for this meeting. The COEs are a natural place for this to happen, but are often limited to one lab <-> one vendor. Multi-lab and multi-vendor focused discussions (within the limits of NDA) are highly valuable as well, and we should consider how to foster those sorts of discussions going forward.*

2) MPI + X dominated the discussion, and on-node performance was almost always the focus of performance results

    - *MPI was assumed to be the de facto solution for scaling out, and "X" was largely accepted to be either OpenMP4, or abstraction layers that used OpenMP4 or CUDA beneath. It will be important going forward to both a) continue evaluating these approaches in the face of very large number of nodes and b) include alternative approaches (e.g. async task models) as they continue to mature in some applications.*

3) A third "P" needs to be considered – ***Productivity*** matters. Many of the abstractions and programming models presented as performance portability solutions have direct impact (either positive or negative) on application developer productivity.

    - *Perhaps the next meeting can be called COE-PP&P? Note that productivity is notoriously difficult to measure and it is probably worth trying to come up with some definitions beforehand to help avoid lengthy debate. At this meeting, the definition of performance portability was not agreed upon at the outset (see #14), and several speakers presented their own, sometimes conflicting, views.*
    - *One can hope that when confronted with these three metrics of performance, portability, and productivity – the answer is not "choose any two, and give up on the third". E.g. Matlab = Productivity + portability without performance. OpenMP = Portability + performance without productivity. Solutions that address all three metrics are desired.*

4) *Descriptive* beats *prescriptive* for portability. Jeff Larkin's talk captured this concept most succinctly, but it was an underlying theme for several other talks as well. Prescriptive programming may offer the best performance on a particular instance of a platform, but we must find the balance between over-prescribing (e.g. specific thread counts, scheduling mechanisms, etc.) and letting the compiler choose the best approaches.

- *Oscar Hernandez's talk on the rules that the SPEC HPC group uses for OpenMP4.x represent a possible starting point for some community accepted best practices around this concept of avoiding over-prescription.*
- *The OpenMP standard needs to explore more "descriptive" options in its design, which currently leans more toward prescriptive. This will put more onus on the underlying compilers and runtimes for performance, and users must understand the tradeoffs while compilers mature. However, current struggles with existing vendor-provided compilers being able to build, optimize and provide correct results make application developers leery of descriptive approaches that rely more on the compiler. Strengthening the requirements in procurement documents and providing adequate funding will help to ensure vendor provided compilers meet current DOE needs and pave the way for performance portability.*

5) There are many unknowns about the impact of specific approaches as of the time of this meeting. While we know that KNL and NVLINK machines are coming in the future, there was little in the way of early-delivery hardware available that could allow people to draw firm conclusions about performance impacts of various approaches. The same goes for OpenMP4.5 compilers, which are still largely in beta – or do not fully support GPUs yet.

   - *This will hopefully be alleviated by the time we have a next meeting – at least with KNL and Power8+GPU with NVLINK coming online in 2016. One anticipates that at a follow-on meeting, there will be much more opportunity to evaluate performance portability on hardware and compilers that will be representative of future pre-exascale procurements. However, even though reporting of performance on publicly available platforms may not be considered NDA in the future, we need to continue to abide by the original guidelines to avoid pitting platform vs. platform in a multi-vendor environment.*

6) Opportunities abound for more focused cross-COE collaborations. Many COEs are holding trainings, workshops, working groups, etc.… that would be of value to many of the participants of this meeting.

   - *LLNL will arrange for monthly telecons to take place between the COE leads, starting in the fall of 2016. Goals will be to increase opportunities for sharing of information across the COEs, and begin strategizing for a follow-on COE-PP meeting to be held in the April 2017 timeframe. Vendor participation will be considered on an as-needed basis, but for purposes of being able to discuss issues under Non-Disclosure, they will not be included by default.*
   - *Need to capture topic areas of broad interest and provide a forum for more collaboration. E.g. LANL-hosted Trinity's* knlchatter *email list and bi-weekly KNL working group meeting. LLNL is planning on a similar forum, and will invite others to participate in calls.*
   - *The labs widespread adoption of conferencing tools like WebEx is invaluable toward enabling collaboration and information sharing.*

7) We need to have best practices for performance portability that can broadly disseminated and vetted through multiple COE experiences – but there are questions about how to best capture, populate, and disseminate these.

   - *Everyone seems to agree that as best practices emerge from the COEs, there needs to be a way to communicate these broadly (e.g. dynamically updated websites and occasional publications). COE leads here should try to figure out a way we can share non-NDA* *information between each other. This could involve a centralized repository, but also needs* *<u>incentives</u>* *for people to share their experiences in a way that is generally applicable to a broader audience. This will be one topic of the above mentioned monthly COE meetings.*

- *There are many logistical obstacles (e.g. security, maintenance, access control) to sharing information between all labs. Future DOE data management infrastructure should include mechanisms for information sharing to minimize duplication of efforts and maximize progress on large scale platforms.*

8) C++ applications using Kokkos or RAJA (two of the more commonly discussed abstraction layers) seem to have a path forward if they choose to adopt either of these approaches. However, the path is less clear for Fortran – which doesn't have native support for generic meta-programming (templates).

    - *DSLs and code generation are perhaps a path forward (e.g. Richards, Van Straalen), as are best practices for uses of the language that can be optimized by the compiler. (e.g. elemental functions [Pennycook]).*
    - *Compiler support for these abstractions is critical to ensuring they can be a viable path to performance portability. Both RAJA and Kokkos have suffered through compiler issues that have impeded their development progress.*

9) There are many unknowns in how to deal with the memory hierarchy on node in light of a hardware managed single address space (e.g. KNL cache mode, or NVLINK coherence). Will the hardware-based solutions suffice, or do we need to be application managed? And if the latter, how can we do this portably?

    - *Real code experience with KNL HBM and upcoming NVLINK will help answer the question of how much manual management is really necessary. This should probably be the focus of some talks at a follow-on meeting. Hard data is emerging on the effectiveness of KNL "cache mode" and GPU "unified memory".*

10) COEs provide a good mechanism to bridge applications and software R&D

    - *This meeting was a nice mixture of applications people, researchers, and production software and tool developers. That mix is important, and something we should continue to strive for in future meetings.*

11) Goals of the COEs are highly correlated with goals of co-design. Hack-a-thons, Dungeon Sessions, Boot Camps, … these are all highly effective co-design and training mechanisms.

    - *Best practices for how to run one of these sessions are emerging. We should capture these from vendors and attendees of successful sessions to ensure that as these opportunities grow and expand, we continue to build on successes.*

12) Are programming language standards perhaps becoming too big and/or complex? Where are DOE priorities in supporting new features vs. optimizing existing ones? And how effectively are we using our large procurements to drive our actual priorities regarding compiler support?

- *An interesting discussion that broke out during the conclusion of the meeting as well as in several breakouts, revealed the existing tensions the vendor compiler teams have between producing optimized implementations of existing language features versus addressing the rapidly emerging language standards (particularly in C++, which is now on the 3 year cadence). We all like to have additional, vendor-supported features (e.g. for parallelism or memory management) built into standards. However, we also like to have attention paid to vendors making existing features robust and optimized.*
- *Lesson: We need to be careful about asking for too much in the language standards, or at a minimum – make clear what our priorities are for features vs. optimizations. Large procurements are the best mechanism for making DOE priorities clear to the vendors, and more attention should be paid to clearly specifying compiler expectations in those procurement RFPs and contracts.*

13) OpenMP4.x early experiences are encouraging, but directives are not for everyone, and can severely clutter up otherwise simple code. Language features (i.e. part of the language standard, not add-on directives) seem to be generally preferred, but we don't yet have (m)any examples of where this has been successful.

   - *Need to continue to push OpenMP standards, and begin to understand practices that allow for it to work well across both multi-core and GPUs. Language features for parallelism and memory management are longer-term goals, but we must be careful not to ask for so many features in our HPC languages that we negatively impact vendors' abilities to support optimized compilers (see #12)*
   - *Standards adoption takes place on a lengthy timeline not always conducive to rapid adoption by code teams. Further discussion of this issue and the impact on code teams who might otherwise consider their use is needed.*

14) There is not yet a universally accepted definition of "performance portability"
   - *Several attempts were made by various speakers to take a crack at what it really means to be performance portable. While everyone is in basic agreement, there are nuances in the definitions. Examples:*
     - *"*For purposes of this meeting, it is the ability to run an application with acceptable performance across KNL and GPU-based systems with a single version of source code.*" (Neely)*
     - *"*An application is performance portable if it achieves a consistent level of performance (e.g. defined by execution time or other figure of merit, not percentage of peak flops across platforms relative to the best known implementation on each platform*" (Pennycook)*
     - *"*Hard portability = no code changes and no tuning. Software portability = simple code mods with no algorithmic changes. Non-portable = algorithmic changes*" (Pope, Morozov)*
     - *"*The same source code will run productively on a variety of different architectures*" (Larkin)*
     - *"*A code is performance portable when the application team says its performance portable!*" (Richards)*

*As one participant put it during a discussion on the topic during the Intel NDA session that Pennycook and Reinders led: "*Let us not get tied up in definitions. Performance portability means different things to different people and we need to accept that.  Both performance and portability are poorly defined and depend on the applications.  Every app has different constraints and there is no way to get around it.*" And that, folks, is probably where we have to leave it! (But it certainly didn't keep others from trying).*

# Agenda / Talks

What follows is an overview of the talks provided as part of the planned agenda. The descriptions provided within are meant to be short summaries aimed at getting readers interested in taking a look at the presentation, and making contact with the author for any follow-up discussions. Links to most of the talks are available at https://asc.llnl.gov/DOE-COE-Mtg-2016, and direct links are provided in the descriptions provided below as well.

## Day One

### Overview of the Five DOE COEs

After a brief kickoff talk to motivate the goals for the workshop, each of the COE leads from ORNL, LBL, LLNL, LANL/SNL (ACES), and ANL gave an overview of their Center – all of which are in various stages of being stood up. All have a common goal of application preparedness, with the main differences between the Centers being a) How the applications were selected, and b) How the vendor partnerships were being utilized.

### Recap of HPCOR

Nick Romero (ANL) gave an update on an HPCOR workshop held back in Sept 2015 where a number of best practices were discussed – some of which were directly related to the goals of performance portability. Many of the best practices are not surprising, and include practices such as use of good abstractions, libraries, good testing, training, and the development of the COEs themselves. Perhaps more interesting were the "failures", or anti-best-practices, which included: vendor proprietary code and libraries, OpenCL, portability via two common code branches, the number and variety of libraries growing beyond the ability to support them, the fact that libraries do not interoperate due to different programming/threading models, "hero" codes, translations from prototype-like codes to compiled C++ codes, and unspecified support models.

### Exascale Computing Project

Bert Still (LLNL) was one of several people representing the Exascale Computing Project in the audience, and offered an update on where the Applications Focus Area of the ECP stood. All signs point to the ECP and DOE continuing to procure diverse platforms at the different labs for risk mitigation, making performance portability a critical ongoing issue.

### Vendor NDA Sessions

The morning of the first day concluded with Intel and NVIDIA each taking an hour to discuss topics that could include developments currently under Non-Disclosure Agreements. Because the invitation list for the meeting was largely restricted to DOE employees and collaborators covered under NDA, most attendees (other than the competing vendors) were able to sit in on these discussions.

John Pennycook and James Reinders (Intel) took their hour to lead an open-ended discussion on issues of performance portability. Intel pointed out that despite their recognition in the HPC market being mostly for Xeon and Xeon Phi, that they are actually a very large GPU chipmaker and perhaps the largest users of GPUs, so performance portability is important to the company as a whole. A long debate on the definition performance portability ensued.

Cyril Zeller (NVIDIA) gave an overview of the NVIDIA programming model vision, and a recap of a number of announcements that were made at the recently concluded 2016 GPU Technology Conference (GTC). The Pascal (P100) GPU and corresponding CUDA 8 compiler greatly simplify the programming model with respect to memory – providing a unified memory space, and advice to the runtime on dynamic memory behavior. Further improvements are in the works, including partnering with Red Hat on OS support for heterogeneous memory with an open source driver that should be applicable to other devices as well.

Both vendors were asked a lot of questions about compilers and language support – an early indication of where much of the discussion would lead as the meeting went on.

## Session: Applications, Optimizations, and Algorithms

*The first session of talks focused on work that's ongoing from the applications side to optimize for advanced architectures using a variety of techniques.*

Jae-Seung Yeom (LLNL) discussed using learning/training techniques (specifically neural word embedding) to optimize sparse linear matrix solves given variations over the input, data representation, and algorithm. To optimize the training data set, they used a GBM (Gradient Boosted Machine) to select features based on relative importance. Different combinations of preconditioners and linear solvers were tested.

Charles Ferenbaugh (LANL) talked about performance research he's doing with a proxy app called *Pennant*, which includes arbitrary polygonal meshes and a subset of physics from the Rad-Hydro code *FLAG*. He concludes that at the time of his study (1-2 years prior), fine-grained concurrency was still far from being performant on Sandy Bridge processors, and that high-level (i.e. coarse-grained) threading consistently outperformed it at the expense of perhaps some additional code refactoring.

Scott Parker (ANL) discussed Nekbone, a proxy app for Nek5000 – a long-lived spectral element CFD solver for unsteady incompressible Navier-Stokes.  The smaller size of Nekbone allowed them to build a performance model. By comparing predicted to actual performance, they can focus in on areas where the machine is perhaps not reaching its potential. Their experience to date has been with BG/Q, but it is now up and running on KNL.

Kris Garrett (LANL) showed some results from early runs of a mini-app (*Tycho 2*) based on neutral particle transport sweeps on unstructured tetrahedral grids found in the Capsaicin code. His experience was that porting to KNL (B0 testbeds) was quite easy, and got comparable results running MPI everywhere or in a hybrid MPI-OpenMP mode.

Adrian Pope and Vitali Morozov (ANL) tag-teamed on a talk covering HACC, a Cosmology code that has won the Gordon Bell prize twice, and has been designed for performance as a "first-class requirement". They introduced a definition of three levels of portability: hard portability (no code changes required), soft portability (simple changes, none algorithmic), and non-portable (algorithmic changes required). Their design principle is for hard portability, and the use of microkernels tuned for a specific architecture if necessary. Their talk gave several examples of both, with some hard numbers for BG/Q, and indications of similar benefits on early KNL hardware.

Kristopher Keipert (Iowa State, partnered with ALCF) presented an overview how the GAMESS atomic structure code has evolved over the years to address new platforms. Early results on the Xeon Phi were presented. He then discussed development of a new Electron Integral library (libERD) and the process and results of integrating it into GAMESS.

Steve Rennich (NVIDIA) works with the Sierra COE team at LLNL, and under that COE has developed an optimized algorithm for neutral particle transport sweeps on a structured mesh, which he covered in this talk. The technique uses a combination of KBA and hyperplane methods to find a sweet spot for optimizing memory bandwidth and reducing synchronization points. The work thus far has been done in CUDA, but there is ongoing work with a colleague at IBM on an OpenMP4 version.

Balint Joo (Jefferson Lab, partnered with ALCF) represented work done by a large team on the USQCD Lattice QCD code – which is built on varying layers of abstraction (with the focus of this talk on the middle "level 2" abstractions). The code has historically been very portable, but due to an AOS (array of struct) data layout has been difficult to vectorize. He talked about the QUDA DSL, as well as experience with QDP-JIT for LLVM JIT compilation. He then turned to some template meta-programming examples to improve bottlenecks in QDP++ from the University of Edinburgh (Grid), QEX done at the ALCF based on the NM language, and Sandia (Kokkos) with some promising performance results of the latter compared to hand-tuned C++ code. He concluded with lessons learned from each of the approaches presented.

Alvaro Vazquez-Mayagoitia (ANL) concluded the session discussing the Electronic Structure Infrastructure Library (ELSI) designed to accelerate electronic structure simulations, which take a significant portion of compute time on ALCF. ELSI provides the user with an interface that uses optimized solvers for various matrix and eigenvalue solvers – specifically ELPA, libOMM, and PEXSI. Going forward, they plan to support GPU and many-core architectures.

## Session: Performance Portable Abstractions

*The first day concluded with a session on general software techniques being developed to support applications for performance portability. Many of these abstractions were then presented in a subsequent session that talked about application experience with these techniques, and were further discussed in a set of breakout sessions.*

Tan Nguyen (LBNL) presented work done with *TiDA* (Tiling as a Durable Abstraction) that handles locality through tiling abstractions – effectively partitioning work into smaller chunks, maintaining a small amount of meta-data associated with each tile, and managing the memory hierarchy to provide maximum data reuse by threads. Work to integrate TiDA into *BoxLib* (AMR library) was presented along with some promising performance results on the MG and SMC proxy applications. He then presented some work to extend TiDA to task-based parallelism in *Perilla*, which uses a runtime to manage data locality.

Jeff Vetter (ORNL) gave a broad overview of performance portability work his team has been doing, with some detailed results of 12 applications using the *OpenARC* compiler targeting the ARES HLIR (High Level Intermediate Representation) and LLVM across four architectures (NVIDIA GPU, AMD, ARM, and Xeon Phi). Looking forward, they are looking at FPGA portability, and thinking how to integration NVRAM into the mix of architectural features using NVL-C.

Christian Trott (SNL) talked about progress on Kokkos, a C++ template meta-programming technique that abstracts concepts of execution and memory spaces. Kokkos is in production and is tested and ported on a large number of different systems. Results on the widely used LAMMPS code were presented as motivation. He then went into some detailed discussion of the handling of atomic operations and how Kokkos can portably handle those using multiple techniques. He then discussed the ongoing development of *KokkosKernels*, which provides a portable interface to Level 1,2, and 3 BLAS, and other sparse matrix and tensor kernels. Finally, he shared some common performance issues when techniques like Kokkos are utilized, including issues with optimizations, elements of the language standards, and OpenMP4 deficiencies.

Rich Hornung (LLNL) then talked about RAJA, which has some similarities to Kokkos. RAJA provides an API (also based on C++ templates and lambda functions) to express different forms of parallelism, traversal policies, and reduction operations, and is designed to be approachable and simple to applications. In many cases, the loop body will remain unchanged from the original code. RAJA is meant to be somewhat customizable such that applications can tailor the look-and-feel to their application domain and common patterns. The concept of an *IndexSet* is used to abstract array traversal methods and potentially avoid race conditions that would be inherent in a standard loop. Updated results on the LULESH proxy app were presented, as well as some forward references to other talks at the meeting that are building on top of RAJA to support memory placement and motion. He concluded by talking about co-design with compiler and tool teams to support optimizations – a concern and approach shared with the Kokkos team.

Arpith Jacob (IBM) is working with the Sierra COE at LLNL, and talked about work ongoing at IBM to build an OpenMP4.5 compiler that can support RAJA on CPU/GPU architectures, and the challenges they ran into - including lambda capture of variables and defining a GPU kernel for the lambda loop body. Results using the LCALS proxy app were presented, showing promising early results.

## Day Two

### Session: Managing the Memory Hierarchy

*This session was intended to address current R&D aimed at managing multi-level coherent memories that will be common in future architectures. While coherent memory promises an easier programming model, it is unclear as of yet whether vendor-provided hardware solutions for managing movement of data into "fast" memory (e.g. MCDRAM on KNL, and GPU memory on Sierra/Summit architectures) will suffice, or if applications will still end up needing to explicitly manage motion somehow to achieve maximum performance. And if so – how can this be done in a way that is portable across these diverse architectures?*

David Poliakoff (LLNL) began the session with a talk on CHAI (Copy-Hiding Application Interface) which uses lambda capture of variables in programming models such as RAJA (or potentially Kokkos) in concert with a simple runtime to automatically determine which data needs to be copied into HBM or GPU memory. The programmer must define pointers using a special "managed" type declaration, and the rest is automated by a simplified runtime that keeps track of where data currently lives, and whether it needs to be explicitly moved at the time the kernel is launched and terminated. It is only managing memory motion on-node – there is no concept of cross-node memory motion. CHAI is in early development, but is showing promising results – and could potentially be more efficient than page-based or cache-line based hardware methods by copying only the data required for the kernel.

Nicholai Sakharnykh (NVIDIA) presented some novel accelerator techniques using HPCMG (an emerging multi-grid solver benchmark) to demonstrate how a program can dynamically select whether to run on the CPU or GPU based on the coarseness of the problem (which in turn depends on where in the AMG "V-cycle" it is). If the kernel is executing > 10k grid points, it is optimal to run on the GPU (TOC) – otherwise on the CPU (LOC). The use of unified memory simplifies the memory management, and minimizes memory paging.

Fabian Delalondre (ANL) discussed the Blue Brain project in the EU. The application is complex, but approximately 85% of the compute time is spent in a relatively small number of kernels. They use a DSL (NMODL) and some mini-apps to develop their techniques, which have been tested on BGAS (BlueGene Active Storage). They are working to support the Dynamic Exascale Entry Platform (DEEP) – a core of the EU strategy for exascale. Usability of the application and a focus on user workflow is paramount to their efforts.

Luiz DeRose (Cray) presented some early prototype work Cray is doing to support managing data motion on KNL through the use of directives. A pragma is used to determine memory placement, and they are working with the OpenMP committee to get the concepts adopted in OpenMP 5.0. The talk generated a number of good questions and suggestions, and Cray will take those into consideration as this work evolves.

Ian Karlin (LLNL) ended the session with a paper study performed by the three NNSA labs and AWE regarding the issue of memory management. A number of use cases (six in total) were presented that outline likely places where programmers will want to be able to query the system. For example, when applications are developed in a component-wise fashion (or in the general case, use libraries), there is no standard way to determine the state of memory across the "handoff". The other use cases are presented in the talk. The intent is to use these as a starting point for working with vendors on a set of APIs or a programming model that is portable across diverse systems.

## Session: Application Experience with Performance-Portable Solutions

*This session was a follow-on to the day-one discussions on general solutions being developed for performance-portability, with this session focusing on early application experience with these techniques.*

Changhoan Kim (IBM) kicked off the session with a talk on an architecture-independent abstraction for unstructured mesh objects. Concepts are decomposed into Acode (application code), which borrows the OO concepts of encapsulation, attributed variables, and inheritance. Acode concepts use class descriptions (fibers), parallelism (arrays/vectors of fibers) and a dependency graph. Scode (scheduling code) focuses on programmability of the automatic compiler and runtime scheduling. A real-world example was given using the IBM neuro simulator.

Adam Kunen (LLNL) talked about developing new concepts in RAJA to support nested loop structures common in structured deterministic transport. Nested loops over zones, angles, and energy groups are dependent upon a number of factors – including the target architecture, compiler, and even problem size. Thus a dynamic way to choose the nesting loop structure and data layout is required that doesn't require hand-coding all permutations of the loop nesting. Early work on the mini-app *Kripke* has been promising, and the work is moving into the production code that it represents. The nested-loop constructs he developed are part of the official RAJA release as well.

Stan Moore (SNL) discussed the use of Kokkos in a new Direct Simulation Monte Carlo code called *SPARTA*. He discussed the methodology and workflow used to introduce Kokkos incrementally through bottleneck identification and incremental addition of parallel for, reductions, and scan operations. Currently Kokkos is an optional package in SPARTA, so multiple versions of the code are maintained, but he found Kokkos to be relatively non-invasive, and allowed him to run on GPUs with relatively little effort.

David Beckingsale (LLNL) then discussed Apollo – an auto-tuning extension to RAJA that uses pre-trained models (machine learning) to select the best execution policies in RAJA on a kernel-by-kernel basis. Initial studies using the AMR mini-app *Cleverleaf* demonstrated up to a 4.8x speedup. Training is done off-line to generate the models, but is relatively inexpensive. Current work has been done on homogenous nodes, but work is being extended to GPUs to help predict which kernels will execute best there.

Geoff Womeldorff (LANL) discussed work being done in a structured deterministic transport mini-app called SNAP to use Kokkos and Legion. Kokkos is used for on-node parallelism, and Legion is used at the system level to manage coarse-grained parallelism. Together, Legion and Kokkos have the potential to greatly reduce programmer effort. A number of results showing different approaches and levels of tuning with Kokkos were presented, with promising performance results.

Ryan Bleile (LLNL) talked about early research in a small monte-carlo mini app to look at event-based techniques, which effectively batch particles into groups that will perform common operations so that low-level SIMD/SIMT style parallelism can be exploited. Work was initially done in CUDA, followed by implementations in Thrust and RAJA – which were relatively easy once the data structures were modified for CUDA. A factor 2x speedup was demonstrated in a 10M-particle simulation.

Matt Martineau (Bristol University) joined from overseas to discuss results that his team (led by Simon McIntosh-Smith) is doing to evaluate performance-portable solutions. They studied several applications (TeaLeaf, CloverLeaf and BUDE [molecular docking]) on CPUs, GPUs, and Intel KNC. They used a combination of OpenMP4, OpenACC, OpenCL, Kokkos and RAJA in various experimental configurations. Their results showed that in general the abstractions were able to achieve close to the results of native optimized results (CUDA or OpenCL), and simplified the programming effort – concluding that portability and reducing complexity for the application programmer can be balanced with good performance.

Leopold Grinberg (IBM) discussed his experiences in developing a performance-portable implementation of sparse linear algebra options (Sparse matrix-vector multiple and Symmetric Guass-Siedel) using a Power8 CPU and K80 GPU. His goals were to find a data layout that worked well on both the CPU and the GPU, could switch between devices dynamically at runtime, had no hardcoded kernel parameters, and was a single source code base – including no preprocessor `#ifdef`'s. This is one of the first known pure OpenMP4 implementations of SpMV and SYMGS, and the results showed good performance on the GPU relative to the CPU.

Slaven Peles (LLNL) talked about strategies being deployed in several solver libraries at LLNL (MFEM, SUNDIAL, and Hypre) with this talk focusing on SUNDIALS (numerical integrators and nonlinear solvers). Because these libraries often interoperate with each other, they are being treated as a suite that will be ported and optimized together. Results were promising for GPUs so long as the vectors were sufficiently large to overcome transfer costs (> 10k elements), and results on the CPU used the optimize MLK libraries. Next steps are to prototype and implement basic data structures (vector, matrix)

## Session: Experience with OpenMP and Recommendations on Guiding Future Standards

*OpenMP is currently the one open standard that is aimed at on-node performance portability. A significant update to OpenMP version 4.5 was announced at SC15, which largely addressed many of the criticisms of the original OpenMP4.0 standard – the first to address heterogeneous computing. Compilers are starting to emerge that support OMP4.x for GPU offload. This session was aimed at collecting some of the early experience with this important standard.*

John Pennycook (Intel) gave a pair of back-to-back talks to open this session. In his first talk, he reiterated his definition of performance portability (from the earlier Intel NDA session), which is that "*an application is performance portable if it achieves a consistent level of performance across platforms, relative to the best known implementation on each platform*" – with an emphasis on the performance aspects. A main thesis of his talk was that choosing the proper abstractions is the proper way to design your application, not to simply choose a portable language or programming model. He gave several examples where it was difficult to portably uncover all available parallelism in a kernel through threading and SIMD without making it difficult for the compiler. He used the concept of "parallel kernels as an abstraction" to drive home the point, and discussed the concept of elemental functions as a way to bridge the gap between different types of parallelism – namely threads and SIMD.

In his follow-on talk, Pennycook discussed ideas for a Domain Specific Language (DSL) concept that uses a YAML-based input specification to build a DAG and code generation tool for "rolling updates" to identify dependencies that are normally difficult and error-prone for the compiler to handle. Codes that had multiple loops/kernels over a single domain with known dependencies between domain elements were used in most of the examples. CEA's *Hydro2D* miniapp was used to present some promising results. The work on this DSL is informing some suggested additions to the OpenMP standard to extend the task syntax to specify dependencies between iterations of different loops. Described by example in his talk, they include new keywords *pipeline*, *intermediate*, and *depend*.

John Levesque (Cray) talked about techniques in OpenMP to help it achieve the good performance of MPI while taking advantage of the dynamic scheduling of OpenMP. Multi-core nodes (KNL in particular) have a "sweet spot" for the number of MPI tasks. One MPI task per node with all threads is not ideal, because of NUMA effects. All-MPI (one task per core) on a node is often not practical because of strong scaling (memory) limits. MPI works well on KNL because MPI forces locality. He then presented a number of techniques to use OpenMP effectively on KNL by having large parallel regions, forcing locality in the threads like MPI does, and avoiding synchronization. Regarding performance portability, he suggested that mapping those threads (which each contain a number of iterations) to CUDA streams was an effective approach used in S3D.

Carlo Bertolli (IBM) talked about some experiences porting the LULESH mini-app to an early version of the IBM OpenMP4 compiler and comparing the results to an optimized CUDA port. The initial naïve port showed very disappointing results, but several rounds of optimizations (mostly through simple changes to the OpenMP4 pragmas) helped greatly. The first important piece was to fix uncoalesced accesses by changing the scheduling to `schedule(static,1)` in order to assign successive iterations to successive threads within the same block, which improved performance by about 15x over the naïve version. Subsequent optimizations looked at compiler improvements to perform code synthesis and reduce overheads such as register pressure. Ultimately, they were able to show performance of OpenMP4 pragmas on par with the tuned CUDA port. An audience question asked if they had tried the resulting code on a CPU, which they had not yet done – but one of the LULESH maintainers piped in that the code looked very much like the OpenMP code generated for CPUs and should work well there.

Jeff Larkin (NVIDIA) presented a good description of prescriptive vs. descriptive parallelism. Prescriptive parallelism implies that programmers specify details to help the compiler optimize for a particular architecture, which in turn can hinder the performance portability of an application. Descriptive parallelism relies more on the compiler taking basic information about the program and deducing the correct parallelism model based on the target hardware. In OpenMP, the biggest difficulty in successfully implementing a descriptive approach was related to the default scheduling and whether the compiler could properly assume loop collapsing. The use of the SIMD pragma was suggested as a possible hint that could inform the compiler of the correct choice for GPUs. He concluded by encouraging the community to adopt a descriptive style (with prescription only as necessary) and to begin developing best practices that compilers can detect and optimize for. Some follow-up Q&A from the audience discussed issues of reproducibility, and knowing exactly what the compiler chose to do. Larkin noted that first, you can tell the compiler what NOT to do and still remain descriptive, and that compilers and tools must be enhanced to provide sufficient feedback to the developer, something the Cray and PGI compilers do a pretty good job at already.

David Appelhans (IBM) discussed some work being performed to port LLNL's Deterministic Transport mini-app *Kripke* to OpenMP4. Kripke is defined by a discretization space over zones, angles (or directions), and energy groups. He first noted that a batched DGEMM was used to achieve 60% of GPU peak on solution of the RHS – demonstrating drop-in optimized library technique. Kripke uses a diamond difference sweep in many of its kernels, and an optimized CUDA version was used as a baseline. The results of the OpenMP4 port demonstrated *better* performance, which he attributes to use of teams to launch a large number of thread blocks and the ability for OpenMP to easily merge new data structures and updates via collapse. (He noted that with sufficient coding, the CUDA version could match or beat the OpenMP4 version, but the resulting code would be difficult to maintain). In the spirit of the meeting, he then tested this GPU version on the CPU, and indeed some of the optimizations for the GPU hurt the threaded CPU performance relative to a baseline implementation. Some conditional coding (e.g. to manually hoist loop invariants) helped, but further work is needed both in the future OpenMP standard and the Kripke implementation to find a truly performance-portable solution.

Carlo Bertolli (IBM) returned to build on the theme of developing performance portable OpenMP code with some simple examples, and some suggestions for the OpenMP standard to allow directives that optimize for the GPU to perform well on the CPU. Specifically, the amount of parallelism exposed through directives for the GPU via devices, teams, and the innermost parallel for can hinder the ability for the CPU to do efficient coarser level threading. He suggested a number of enhancements that could help with portability, including an iterator construct over devices, a well-defined target construct when running on the host CPU, an "if-and-only-if (iff)" clause that would trigger GPU-specific pragmas only if an accelerator existed, and improved performance of collapsed loops (which are important for exposing sufficient parallelism for the GPU) on CPUs. He concluded that while current issues exist in the OpenMP4.5 standard to enable performance portability, early results are highly promising, and issues can likely be addressed in the standard with "minor tweaks".

Oscar Hernandez (ORNL) gave an overview of work being performed in the SPEC HPG on best practices they have learned for writing a series of 16 SPEC benchmarks using a descriptive (i.e. non-prescriptive) approach developed at an OpenMP4.0 meeting in Berlin. Their guidelines make for a useful start at community best practices for OpenMP4 performance portability, and are clearly outlined in his talk. In general, the recommendations were to use the OpenMP 4 accelerator model, but leave much of the decisions up to the compiler to achieve performance portability.

Tom Scogland (LLNL) wrapped up the session with a discussion of progress the OpenMP standards are making and a view toward the OpenMP 5.0 specifications. He started by sharing some aspects of the OpenMP 4.x standard that "surprised" developers, including the fact that all code to be run on the GPU (regardless of where it sits in a potentially deep call stack) must be decorated with the `declare target` pragma, and future work to alleviate this burden by automatically detecting functions in the same translation unit. C++ codes also suffer from the unavailability of virtual functions on the device, and the fact that STL is not currently supported on GPUs. The lack of a deep copy construct in the current standard was also noted, but is being addressed in the future standard. He pointed out that many challenges to performance portability with OpenMP4 exist (largely because – as noted by earlier speakers – optimizations that work well on one platform can hinder performance on others), and that work on issues such as platform/device hints are being addressed, as are issues specific to C++, multi-level memory, a more flexible collapse, and a series of others.

## Day Three

### Session: Tools for Performance Portability and Analysis

*After hearing about the number of challenges with performance portability seen by developers thus far, this session focused on the importance of good tools that can help developers reason about achieving good performance portability, and the enduring need for some co-design in this space.*

Jeanine Cook (SNL) kicked off the session with a discussion on the "Good, Bad, and Ugly" of Performance Monitoring Units, or PMUs. The good being: that they exist. The Bad and the Ugly being: that they are totally non-standardized across platforms, and documentation is spotty and often inconsistent with reality. PAPI presets are one attempt at providing a standard interface, but are also fraught with potential error and inconsistent results across platforms. She described a new tool being released soon at Sandia called *perfminer* that attempts to solve this problem by presenting a standardized way to collect PMU data in a scalable underlying no-SQL database, and present the data using visual feedback in a Java-based GUI. It uses a continuous monitoring approach (i.e. you can opt in/out). She concluded with a plea to vendors to begin movement toward a more standard approach to collecting PMU data.

Juan Gonzalez Garcia (IBM) discussed a prototype tool in development at IBM meant to capture a system-wide view of application performance by integrating profiling techniques for CPU, GPU and MPI – and understanding the interactions amongst those hardware subsystems. The tool uses modules and interactions to collect data from those three sources in a MySQL DB. He showed some early results using the LULESH mini-app with 8 MPI ranks, 1 CPU and 1 GPU. The combined results are displayed in a standard profiling report, with more complex and visually oriented output available in a post-processing tool that queries the SQL DB. Future work is to integrate more profiling sources and test on a broader array of applications.

Ignacio Laguna (LLNL) introduced *STATuner*, a tool that uses machine-learning techniques to predict the optimal number of CUDA blocks and block size for a kernel. With too small a number of thread blocks there is not sufficient parallelism to keep the GPU busy, and if the block size is too large there is too much contention for resources. After a brief discussion of some existing tools (NVIDIA's occupancy calculator and autotuning) and some of their strengths and weaknesses, he motivated the reason for building STATuner using a machine-learning based approach based upon a set of statically determinable metrics and a set of training data. Results showed better prediction of performance than simple occupancy, and while the results may not always be the best prediction, the error is bounded by a minimum.

Si Hammond (SNL) talked about the motivation and design for *KokkosP* – a performance profiling interface that overcomes many issues that standard tools have with understanding the complex abstractions that Kokkos employs, and provides profiling feedback for each Kokkos kernel instantiation. KokkosP is currently connected to several tools, including VTune and Nsight, and more are on the way. The design includes being dynamically loaded, and the tools are stackable. Use of KokkosP comes "for free" for codes that adopt Kokkos, and additional research is in the pipeline, including dynamic feedback to the running application, debugging connectors, and vectorization and instruction analysis. Some follow-up Q&A discussed the need for getting teams doing some similar things to work together on standardizing hooks.

Protonu Basu (LBL) opened up his talk with an example using *miniGMG* where a baseline of 13 lines of code exploded into 170 lines for a hand optimized CPU version, and over 300 lines for a GPU optimized version. This motivated their work on using CHiLL to perform compiler-based transformations and code generation. The results he presented were based on experiences with Geometric Multi-grid (in the form of miniGMG), using a variety of different stencil types. Several examples were provided showing how complier transformations could reduce memory traffic relative to computation at various stencil sizes. CUDA-CHiLL was briefly discussed as a thin wrapper on top of CHiLL to directly generate CUDA code. In summary, he stressed that compiler technology should not be overlooked as a performance portability tool – performance results can rival manually tuned code for well-understood patterns.

Heidi Poxon (Cray) talked about tool developed by Cray called *Reveal* to help developers through the process of identifying how to target loops for parallelization. She started by motivating why you would want to consider OpenMP – namely when bottlenecks from MPI-everywhere begin contending for shared resources or network injection rates. Once the decision to move to OpenMP is made – Reveal guides the user through a series of steps: 1) Identifying key high-level loops through runtime profiling 2) Performing parallel analysis and scoping (with dependence analysis included in the tool) 3) Adding OpenMP parallelism to those loops, and 4) Analyzing performance and guiding the user toward additional optimizations such as vectorization of inner loops.

## Session: The Input/Output Bottleneck and Use of Burst Buffers

*While much of this meeting focused on the challenges of various node architectures, burst buffers represent an important emerging feature of new architectures to use multiple levels in the storage hierarchy to help alleviate the I/O bottleneck. This session addressed some early work going on to think about how to portably address burst buffers.*

Mark Miller (LLNL) discussed a new proxy app *MACSio* developed to help explore parallel I/O models and performance in a flexible plug-in architecture. MACSio was motivated by the fact that many existing IO benchmarks tend to work at lower levels of abstraction than some of our sophisticated applications that use abstractions such as HDF5, Silo, Exodus, or ITAPS. MACSio allows the user to set up a representative data set using a simple JSON format that best mimics the output (i.e. restart or plot) characteristics of the application. Field data can be initialized with realistic data fields so that tests such as compression techniques are not stymied by data initialized as constant or random data. The back-end plugins allow different I/O models (e.g. file-per-processor, collective writes, or multiple independent files) to be tested. Early results motivated some work with the HDF5group to work on some peak memory issues uncovered in recent HDF5 releases. He concluded with a pointer to the github site and a request for interest in contributing additional plug-ins.

Andrey Ovsyannikov (LBL) began his talk with a motivation for why burst buffers are an important emerging innovation, using a workflow consisting of *ChomboCrunch* and *VisIt* to motivate the data analytics needs in the context of Carbon Capture and Sequestration simulations. After showing how the burst buffers were enabled in a SLURM script, he presented results of an I/O bandwidth study of the burst buffer versus straight lustre, showing a consistent 3-5x improvement in bandwidth using the burst buffer. He then showed results of a straight time history analysis, with clear spikes in the runtime when the lustre version performed I/O compared to the relatively non-existent overhead using the burst buffer. These early results were promising, and he concluded with some planned future work including dynamic component load balancing, managing burst buffer capacity, component signaling, and including additional components into workflow (e.g. pore graph extractor).

Kathryn Mohror and Mike Pozulp (LLNL) concluded the session with a description of the Scalable Checkpoint Restart (SCR) library and some application results. Kathryn began the talk with an overview of SCR, and their plans to expand from its initial approach of providing resilient checkpointing to abstracting away the complexities of using the burst buffer. Mike picked up the talk from there with a description of how SCR was used in some early science runs on Trinity Phase I. His first result showed how SCR using RAMDISK held time spent checkpoint almost flat as the number of nodes increased from 1 to 4096, with the time spent doing traditional Lustre I/O increasing exponentially. He then presented time history results for an 1152 processor run showing the Lustre vs. SCR RAMDISK consistently showing about a 20x performance improvement in checkpoints. He concluded with a brief description of the ease of which integrating SCR into Mercury was, and future plans to replicate the results using the NVRAM burst buffers on Trinity with SCR when the capability comes available.

## Session: Use of Domain-Specific Languages for Performance Portability

*The final session presented a few talks on successful examples of the use of Domain Specific Languages to support performance portability.*

David Richards (LLNL) talked about work that came out of the X-stack D-TEC project to use the ROSE compiler on high order stencils to generate optimized code that is then passed on to the vendor compiler. He started with some examples from the *Cardiod* heart modeling application, where he got good results on the GPU (66% of the roofline peak). But he spent most of his time talking about the *SW4* seismic code. Some of the lessons learned had to do with converting some of the Fortran to C, in which case a fair bit of performance was lost due to missed vectorization opportunities. While the C could be made to match the Fortran performance, the native Fortran was much easier for the compiler with the naïve implementation. The GPU work on SW4 was more complicated, and performance was limited by register pressure. While tiling is the typical approach, they instead used loop fission – splitting the complex kernel into a number of smaller kernels. Performance on the GPU was good, but still only about 15% of the roofline bound. Overall, the DSL for stencils demonstrated very promising results – allowing the user to write simple "MatLab" type code, and the D-TEC ROSE transformations taking care of handing the backend compiler more optimized code.

Brian Van Straalen (LBL) wrapped up the session with a discussion of AMRStencil – a DSL embedded in C++11. It is being used in a rewrite of Chombo – replacing the original Fortran kernels with the eDSL. The default version works with C++11, and the use of ROSE allows for some additional optional cross-platform optimizations. AMRStencil tries to be very clear about what is compile-time vs. runtime bindable – Stencils are compile-time, while Boxes are runtime. He showed some example results with Geometric multigrid, demonstrating the expressiveness of the compact source code. Performance results on an AMR shift calculation demonstrated very good results on a Haswell processor with SIMD. He touched on how the ROSE tool can be used to manage the memory hierarchy – turning a few tens of lines of code into almost 1000 lines of highly optimized tiled code. He concluded with a list of current challenges – including some complexities with the C++ language.

## Breakout Discussions

Perhaps the most useful aspect of this meeting was the chance to get people talking during the breakout sessions. These sessions were designed to cover topics for which an earlier session of talks had completed so as to give participants a basis on which to form their discussions. Each topic area had two independent groups meeting in parallel – both to keep the number of people involved in each discussion manageable, and to ensure that a diversity of ideas were brought out between the two groups. Breakout leads were asked to moderate the discussion, provide an outbrief, and summarize the content in this report. In general, the breakout leads coordinated beforehand to formulate a common set of questions for the overlapping breakouts. Their original outbriefs in their raw form are available on the meeting web site. Below is a summary of their discussions.

Note: We've included in each section below notes that were collected by Tina Macaluso and Emily Simpson during the short outbriefs of the breakout sessions (out-brief presentations are available on the meeting web site), in addition to the write-up summaries provided by the session leads. **Thanks to Tina and Emily for their incredible efforts** – without which so much information would be lost forever!

### Breakout Session A: Managing the Memory Hierarchy

#### Lead: Doug Doerfler (LBNL)

The session had a number of participants, with active participation from LLNL, LBNL, Sandia, LANL, Nvidia and Cray. The breakout charge questions developed for the session were used to lead the discussion:

- What are the practical limitations of using current programming models for managing the memory hierarchy
  - Do you plan to integrate multi-level memory support into your code?
  - What are your memory capacity requirements in the 2020 timeframe?

  - Can you live with 16, 32, 64 GB per node? Per NUMA domain?
  - How much effort are you willing to do to support multi-level memory?
- Languages, directives, attributes, other?
  - Are you willing to use a "non-standard" memory management programing model?
  - Do you need memory management interoperability of C, C++ and Fortran in a common code?
  - Would you like to see a type attribute for variables to declare fast memory storage?
- What is the proper balance between user control and runtime control for memory placement and management?

The discussion started off by setting some boundary conditions on what constitutes the memory hierarchy: a) on package memory (MCDRAM, HBM), b) off package, bulk capacity memory (DDR) and c) byte addressable non-volatile memory (upcoming NV technologies). There was some discussion if in the future high-bandwidth, on-package memory alone would satisfy application needs for capacity, but it was the consensus of the group that developers will have to deal with a multi-level memory hierarchy, as it will be necessary to satisfy adequate Byte/FLOP balance ratios.

The group discussed the practical limitations of current programming models and identified the desired features of future models. The group identified that developers want control at a low level and current library-based solutions provide that, but these solutions assume that data allocations are static where as in reality codes go through multiple phases and you want data storage and attributes to be dynamic. Developers want to be able to describe the attributes and have introspection of the node to help manage data placement, some combination of compiler and runtime support. The group also discussed the merits of higher level, higher productivity solutions but did not reach a consensus of what that would be, CHAI, UVM, OpenMP, etc.?

We identified a need for variable type attribute extensions to specify the characteristics of memory. Attributes, as opposed to declarations, allow type characteristics to propagate through the system. However, there was some disagreement that a declarative statement is sufficient and there was some argument that the extra semantics would help in using a data structure with this information. This is a language issue and is applicable to Fortran and C/C++. The group also expressed concern on the amount of time it would take to get through a language committee.

Enhanced tools were identified as a desirable means to identify data structures that would benefit most from "fast" memory. For example identify hotspots for memory accesses and identify if it's latency or bandwidth bound.

Action Items

- Cray agreed to explore language attribute features in compilers (point of contact Luis De Rose).
- It was recommended that the DOE COE's conduct tutorials for tools available today. Cray and Nvidia identified to support a future tutorial, but all the COE vendors should participate.

***Outbrief* notes (Doerfler – Managing the Memory Hierarchy):**

- Both discussion groups answered the same breakout charge questions. What are practical limitations of package memory for managing the memory hierarchy? What are ramifications for language directives, attributes, etc.? What is the proper balance for user control? I began the discussion by setting some boundary conditions: managing on package memory (MCDRAM, HBM), off-package, bulk capacity (DDR) and byte addressable non-volatile memory (and we did not really get to this last discussion). A quick survey of participants revealed a quarter of the group members were app developers who are actively incorporating multi-level memory (MLM) into their code, another quarter said they would do so in the near future; half of the attendees were not app developers.

- Are we sure we need MLM concepts in next generation machines? There was no indication that we can avoid MLM in future machines. There was skepticism that on-package memory only can satisfy adequate Byte/FLOP balance ratios.
- What's wrong with memkind? It assumes that where data resides is static but real codes go through multiple phases, so you want to dynamically change data attributes. The memkind solution is completely developer-managed. The group was not sure why one would want a library-based solution – but they still want a way for developers to work at this low-level. What developers really want is to be able to describe the attributes of data at the language level and have introspection of the node to help manage data placement, i.e. some combination of the compiler and a "runtime" to help manage the data. The sense was there is also need for a higher-level, higher productivity solution: CHAI? UVM? OpenMP?
- There is desire and a need for variable type attribute extensions to specify "memory characteristics". Instead of an attribute that fast memory is needed for a given data structure, consider saying when fast memory is not needed. The attribute needs to be broader than fast or slow and capture other characteristics such as latency. Cray has agreed to explore this notion and solicited input from participants during the discussion. This is not a specific language issue – it needs to be addressed across languages. Attendees said they would appreciate not just a programming model but also a tool to tell us what data structures would benefit most from various types of memory. There was agreement that a proper balance is needed between user control and memory management.

*Question* – Would attributes for memory be static?

- No they would likely be dynamic via the runtime.

## Breakout Session B: Managing the Memory Hierarchy

### Lead: Bronson Messer (ORNL)

This breakout session included participants from ORNL, LBNL, LLNL, LANL, IBM, and Intel. The discussion began with a series of somewhat specific questions regarding the methods the participants were currently using (or considering) to manage hierarchical memories and how they saw those techniques changing as hierarchies become deeper on the CORAL platforms and beyond. The overarching question was: What are the practical limitations of using current programming models for managing the memory hierarchy? In particular, the breakout discussed:

- Do you plan to integrate multilevel memory support into your code?

- What are your memory capacity requirements in the 2020 timeframe? (Can you live with 16, 32, 64 GB per node? Per NUMA domain?)
- How much effort are you willing to expend to support multilevel memory?
- Are you willing to use a "non-standard" memory-management programing model?
- Do you need memory-management interoperability of C, C++, and Fortran in a common code?
- Would you like to see a type attribute for variables to declare fast memory storage?

**Current practices and expectations**

There was a strong consensus that managing the memory hierarchy was a common requirement. Many application developers in the session reported a need to manage memory hierarchies on current machines. Electronic structure, lattice QCD, deterministic transport, other multi-physics codes were given as examples of codes that will not fit into small, fast memories, necessitating memory management. Indeed, several participants reported that they often manage memory themselves, in some cases writing their own memory management software to accomplish this (see comments later regarding the question of whether this is a desirable state of affairs). For codes that need some sort of explicit control, one possibility is to use high-bandwidth memory as a cache, but managing that cache explicitly as a user seems daunting. MPI-3 shared memory programming was offered as an example of a development that was envisioned as a method to reduce complexity (by allowing changes to existing MPI codes incrementally in order to accelerate communication between processes on the shared-memory nodes) that can actually result in increased effort on the part of the programmer (i.e. using MPI-SHM can lead directly to writing a handmade memory manager).

Questions regarding memory footprint need a normalization to be truly meaningful, e.g. bytes/peak TFLOP or memory-size/memory-bandwidth. Nevertheless, for many codes, there is a minimum amount of memory required per MPI rank to make the MPI+X model feasible.

In discussion of memory management, it is important to separate the distinct, but related, issues of data placement and data layout. Data placement refers to the physical memory location where a piece of data is housed. Data layout refers to how a program traverses data structures (e.g. for example, is the data laid out as a structure of arrays or a array of structures?). For each of these related issues, the implementation of a truly shared address space—where the system "finds" the data for you—opens up several possibilities for tools and finer user control.

In discussing possible remedies for many of these issues, libraries and consistent API's were overwhelmingly preferred to directives by the breakout participants. Portability was cited as the primary driver for this choice, as libraries are not directly dependent on individual compiler support. Though not desirable, this strategy allows an extreme solution: If required, individual libraries can be packaged with the code. Nevertheless, it is important to note that directives can be used and "hidden" from noncompliant compilers with macros. Waiting for language standards to take hold was not considered part of a realistic strategy for the future, as time is short. Indeed, though memory management is a current concern already, it may be premature to designate features for standards. Guidance from vendors to determine the scale of the gap between what is possible and what is desired would be helpful here. Understanding that the vendors have to optimize over finite development resources to effectively answer the question, a concise list of guaranteed features would be of significant benefit.

**User control vs. runtime control**

Finally, the breakout participants discussed the proper balance between user control and runtime control for memory placement and management. There was a strong consensus for a combination of reasonable defaults and the possibility of fine control. Total runtime control connotes a level of opaqueness, which can lead to performance problems. Participants noted this sort of opacity is already a problem in, e.g., PGAS languages. In particular, page faults or allocations exceeding device memory need to be discoverable by developers. The most robust and safest mechanism for discovery would be stopping program execution.

Nevertheless, a balance is desired, especially when development is beginning. If developers have to confront the full complexity to get started, that is a problem.
The experience of developers with the Cell processor was suggested as a lesson learned in this respect. With the Cell, developers had to learn essentially all the details of the hardware before productive software work could begin. Tool developers will need to determine a set of abstractions to provide information between the robust program failure mode and hardware-level details of paging. Ideally, these abstractions will be the same as those under user control.

Two closing thoughts from the session relate to what might be considered legacy code issues. First, we often talk about “memory management” when we, in fact, mean “dynamic memory management.” This is a limiting assumption. We should also consider static memory and how the runtime will manage it, particularly thread-static memory.

Finally, all participants agreed that intransigent adherence to outdated implementations in legacy codes renders most of this discussion moot. If a developer insists on maintaining major parts of a code frozen in time, they have to accept that new capabilities and hardware features may well be beyond the reach of the code to exploit (e.g. the use of Fortran COMMON blocks and the standards constraint this introduces is an especially stark example).

***Outbrief* notes (Messer – Managing the Memory Hierarchy):**

- What we discussed in our breakout group validates what Doug just presented. One thing that was pointed out in our discussion is we don’t really know what we need now. There was consensus that many users have identified needs for memory management (electronic structure, QCD, deterministic transport, other multiphysics codes). Memory footprint needs some form of normalization to be meaningful, e.g. bytes/peak TFLOP or size/bandwidth. The group discussed the reality that for many codes there is a minimum amount of memory required per MPI rank, and this is not going to change.
- The group agreed that “COE Platforms” (SC ASCR 2018-era platforms at NERSC, OLCF and ALCF and ASC platforms at LLNL and LANL around same time) will have hardware features that smaller platforms will not share (e.g. Linux clusters with older GPUs). It remains to be seen whether we will have to design to the lowest common denominator. One possibility is to use HBM as a cache but managing that cache explicitly - as a user - seems daunting. MPI SMH provides a picture of how bad things can be and can lead to the need to hand-write a memory manager.
- The notion of true shared address space – where the system “finds” the data for you – opens up possibilities for tools and finer user control. The group agreed it is important to separate the distinct - but related - issues of data placement and data layout. Data placement is physical memory

location whereas data layout is about how the user lays out arrays, manipulates the data structure, and in which order the process is traversed.

- Libraries and consistent APIs were preferred to directives by members of our group. Portability is the prime driver here. You can package a library with the code and importantly, you can hide directives with macros. This is more of a compiler support issue.
- Wrapping platform-dependent allocation and placement methods is already common (and was not seen as a big deal but also not seen as the ideal solution). We need vendor guidance to determine the scale of the gap we need to bridge between what is possible and what is desired. The vendors have to optimize over finite development resources.
- Some group members argued that "balance" is the wrong word – what is needed is a set of sensible defaults and a way to exert control when you want to. There was agreement that there are always tradeoffs between absolute performance, maintainability, resources and portability.
- Opaqueness for performance is a huge potential problem and already a problem today for PGAS. Getting detailed information in a timely and effective manner is important.
- Page faults or allocations exceeding device memory need to be able to be seen, even to the point of stopping program execution. If people have to confront full complexity in order to get started then that is a problem.
- Finally, we often talk about memory management and mean dynamic memory management, and that is a big assumption. We should also discuss static memory and how the runtime will manage it, particularly thread-static.

## Session C: Performance Portable Abstractions

### Lead: David Beckingsale (LLNL)

The goal of this breakout session was to understand and answer questions about the programming systems required for writing performance portable applications. The session was attended by around 15 participants, including vendor representatives, and national laboratory members. The session focused around 5 questions:

1. What is performance portability?
2. At what level(s) do we want abstractions?
3. What are the tradeoffs of different approaches?
4. What do we need from: Vendors and the community?
5. What do we want to see supported by our programs?

Our discussion surrounding each of these items is summarized below.

***What is performance portability?***

In one of the workshop presentations, John Pennycook (Intel) volunteered the following definition for performance portability: "An application is performance portable if it achieves a consistent level of performance across platforms, relative to the best known implementation on each platform." This definition caused considerable discussion during the talk, and a similar level of discussion during the breakout session. Whilst we agreed to use the definition as a point of reference for the remainder of the breakout, some additional qualifiers were added to ensure it was more applicable to the needs of national laboratory applications. A key point for a performance portable application was that a certain fraction of the codebase be shared between implementations, although the fraction of shared code could limit the performance that can be achieved. An additional qualifier was that the implementation should aim to exploit available architectural features. The original definition leaves some parameters unspecified, such as the level of performance the application achieves. It was noted that this level of performance need not be high for an application to be considered performance portable.

***At what level do we want performance portable abstractions?***
There are many levels at which performance portable abstractions can be provided. This question captured two different interpretations that were discussed at the breakout. For an application developer, the ideal level for an abstraction could be the library level. If the application requires a matrix to be solved, they could call the appropriate library that would then contain high-performance code targeted specifically to the platform the application is running on. For an application developer, using a different library depending on the architecture they are using is not a problem. However, this would require high-performance libraries to be implemented. It was noted that the abstraction of functionality in a library is already abstracted out frequently in the case of MPI.

For the computer scientist, rather than the application scientist, having abstractions at the lowest possible level was preferred. These low-level abstractions should still exhibit some degree of portability, as this increases developer productivity and minimizes the amount of functionality that has to be rewritten on a per-architecture basis. This comment means that DSLs might not be appropriate here, but could work for other use cases. Specifically, a DSL used to develop a new application could allow the computational scientist to focus on the science, and the computer science to focus on the optimized implementation of the operations exposed by the DSL.

When adding abstractions to legacy applications it is important that they can be adopted incrementally, since replacing everything in the code all at once is impossible. Whilst most of the discussion focused on C++, it was noted that there seem to be a lack of these performance portable approaches for Fortran.

***What are the tradeoffs of different approaches?***
Each approach for performance portable abstractions has different pros and cons. DSLs provide the opportunity for lots of optimizations (e.g. kernel fusion) and can be highly tuned for a specific application. However, they cannot be incrementally adopted. One success story was using a DSL to generated optimized Fortran code from Python. C++ abstractions like RAJA and Kokkos are good at allowing users to rapidly prototype new versions and optimizations, but by using cutting edge language features (e.g. C++ lambda functions), these abstractions are at the mercy of compiler support. One solution to this would be to push language standards to include parallelism support. However, this would bloat an already large standard and create more work for complier vendors. It was clear that each abstraction would be better suited to a different user community and that successful applications tended to have three stakeholders: domain scientist, computer scientist, and implementers.

***What do we need from vendors and the community?***
Improved compiler support was a key issue that the breakout attendees felt needed to be addressed by the vendors. Both in terms of supporting modern programming language standards, but also in ensuring that code could be optimized effectively when using performance portable abstractions. In addition to compiler support, it was noted that tool support for our chosen abstractions was critical. For example,

debuggers that could understand the abstraction and provide meaningful information when finding and fixing application bugs.

One important point raised was that ensuring that different abstractions can co-exist is critical. For example, if one part of an application uses one abstraction, but calls a library that uses another, the two abstractions must be able to work together to ensure correctness and performance.

A final area that would benefit from improved support is build systems. Programming models like CUDA require separate compilers and can be difficult to combine with others libraries and existing application build systems. Applications will be run on desktops as well as supercomputers, so need to be built in both places.

***What do we to see supported by the programs?***
Program support should focus on system software such as the compilers, and breakout attendees stressed the value of programs supporting software that wasn't applications. Finally, to help ensure applications are ready for the next-generation systems, it would be good to have more expertise on the available performance portable abstractions, perhaps in the form of evangelization experts who can visit application teams and share knowledge and teach application developers.

***Outbrief notes (performance portable abstractions – Beckingsale):***

- We started discussion with John Pennycook's definition of performance portability (i.e. an app is performance portable if it achieves a consistent level of performance) and added some qualifiers including X% of the code must be shared between implementations (e.g. 95%), and the implementation must exploit available architecture features.
- App developers can use library-level abstractions. There was a recommendation to leave the problem of performance to the library developer. One library per architecture is no big deal.
- Developers want abstractions as low-level as possible while remaining portable. Legacy apps require incremental adoption. Fortran cannot be ignored – but what can we do? Pre-processing, RAJA/Kokkos interoperability with Fortran, f2c.
- Tradeoffs? DSLs provide opportunity for kernel fusion but inhibit incremental adoption. IRP DSL is a success story of generating optimized Fortran from Python. RAJA/Kokkos is great for rapidly prototyping optimizations. Abstractions using cutting edge features are at the mercy of compilers. The push to add parallelism to language standards makes them more bloated (and this adds more work for compiler developers).
- What do we need from the vendors? Focus on compilers, meaning support standards and optimize code even through our chosen abstractions. Provide tools that support our chosen abstractions (e.g. debug information) and better build systems because it is difficult to combine wrappers, preprocessors and compilers.
- What do we want to see the programs support? Explicit support targeting system software (e.g. compiler research). Have experts who know application abstractions well spread their expertise via visits to app teams to share knowledge.

*Comment* – Having DOE fund some of the things that everyone uses directly (versus having the labs fund this work) would be helpful.
*Comment* – There are efforts by several compilers via a "Link-time" technique.

- Link-time optimization does work in some cases.

*Comment* – It would be helpful if we could put our own work in there.
*Comment* – The problem is that we do not know what the best possible solution is, so it is impossible to gauge whether we have the most performance portable solution.

## Breakout Session D – Performance Portable Abstractions

### Lead: Rob Hoekstra (SNL)

This discussion session was 1 of 2 to focus on the future of parallel programming models and the value of abstractions that allow performance portability across the diversity of hardware architectures the community is grappling with already today. In common with the other session, focused on 5 questions.

1. What is meant by performance portability?
2. At what level of the programming models do we most need these abstractions?
3. What are the tradeoffs of different approaches to these programming models and abstractions?
4. What do we need from the vendors and community?
5. What are the priorities we believe the programs should support?

***What is meant by performance portability?***
Isn't the real question, what level of performance are we targeting for our codes or what is 'good enough' when it comes to performance portability? What is 'good enough' clearly varies depending upon the community. In some cases, within an order of magnitude of peak is good enough (climate?) while others want as close to peak as possible (astrophysics?). In many cases, code teams have a limited amount of time to invest in performance improvements and they rely on slow incremental improvement especially when the code base is large (NNSA). This implies that a crucial value proposition of performance portable abstractions (PPAs) is productivity. Can an existing code incrementally adopt this technology to gain performance and portability? Can a new code utilize this technology to incrementally improve their productivity and performance portability? Perhaps the most appropriate metric is "How rapidly can a code be brought up and running productively on a new platform with 'good enough' performance"? Performance portable abstractions are valuable when they contribute positively to that metric.

***At what level of the programming models do we most need these abstractions?***
How many 'levels' are there? We can define 2 and perhaps 3. High-level programming models are in the category of domain specific languages (DSLs) or library APIs that support domain specific algorithms. At the lowest level, the focus is on direct manipulation of hardware features such as vector units, etc. A middle level could be defined which captures 'patterns' which are common across many communities such as gather-scatter or cache oblivious kernels supporting hierarchical block algorithms, which can be tuned to particular architectures. This middle layer is where our PPAs are most often found. We see high-level abstractions as being most valuable as a productivity enhancement for a particular domain. They can have the weakness of being overly rigid and difficult to customize to support multiple code bases even if they are in the same domain. Lower level abstractions can bring a greater degree of customization but at the cost of complexity for the developer. It is unclear where the sweet spot lies. The ability to work primarily to a high level abstraction while having the access to the lower levels when necessary seems to be the consensus view. A crucial issue is community support for these abstraction tools. Adoption of these technologies leaves a code team vulnerable to an outside team/community,

which may have differing priorities and timetables. We see many successes and failures in this space from which we can perhaps learn some lessons. High Performance Fortran has largely failed due to lack of adoption by the community while OpenMP has seen significant adoption. However, OpenMP and the vendor-developed runtimes have struggled to meet the performance expectations of the community. The newer players are template meta-programming C++ based layers. They are showing initial success and the hope is that they will strongly influence future language standards adoption of parallel constructs. Will Fortran be a part of that movement? There is hope that the size of the community that cares about performance on parallel architectures has grown and this will drive more aggressive pursuit of mature parallel programming models and environments.

***What are the tradeoffs of different approaches to these programming models and abstractions?***
There are a wide variety of approaches including directives, language extensions, libraries/APIs, DSLS to name a few. A focus of the discussion was on language and language extension-based approaches verses directive-based approaches. Directives clearly have the advantage of being language agnostic. There appears to be a significant cost however due to the lack of ability to manage data attributes such as type and layout. This can significantly hamper the compiler's ability to infer intent and optimize effectively. It was noted that there are good reasons why the most successful parallel programming model/environment (MPI) is library/API-based rather than directive-based. Language specific approaches are showing significant progress especially in the C++ arena. If these technologies are successfully integrated into the language standard they could prove to be a compelling approach. Currently, however, they are driving issues with complexity, compile times and executable sizes. OpenMP appears to be the most broadly adopted shared memory parallel environment. There are significant concerns about its current performance and complexity. A potential opportunity for a language agnostic solution may lie with the LLVM-IR. With many of the compilers adopting LLVM, this presents an opportunity to leverage the LLVM-IR layer to implement parallel performance enhancements. There appears to be little interest from the community in DSLs currently. Lower level constructs seem to be the direction the community is headed with the expectation that they will be more broadly applicable. However, it was noted that other communities such as the neuromorphic community are investing in DSLs and we should track this.

***What do we need from vendors and the community?***
With regard to the vendors a key aspect is development of better communication paths. Multi-lab communities (such as the co-design community) appear to be effective in allowing a 'common voice' to speak to the vendors. The vendors can be highly responsive to this approach especially when we can articulate the value of a capability to a broader segment of their market. Delivery timelines can vary dramatically from vendor to vendor and we clearly need to move the vendors towards response times that meet our mission needs. Early and regular engagement with the vendors has proven highly valuable. A key example from the Centers of Excellence is early access to the development and execution environments. Many issues that would only have been identified after a platform was delivered can now be addressed significantly earlier. Compilers are seen as a key part of the toolchain that can benefit from strong engagement between the vendors and the labs. The vendors' movement to adopt LLVM as a key part of their compiler toolchain is seen as a very positive direction. A forum for engagement has informally developed and we should strive to formalize it. In addition, we believe there should be more emphasis placed on open or shared tools especially with regards to performance analysis. In general, moving away from vendor proprietary tools to open source tools is a preference in the community. There are open source efforts in some cases driven by large companies such as Google that we should better leverage. A model for success appears to be development of an open source baseline implementation that the vendors then optimize for their platforms.

***What priorities do we want to see supported by the programs?***
Several points of emphasis where made here. With regards to large procurements stronger emphasis should be placed on the software environments. The Centers of Excellence are driving earlier evaluation of the software and we need this to continue throughout the machine lifetime. The vendor must be committed to continuous improvement of the development environment as a whole and the compilers in particular. Performance improvements for the codes need to continue to be a joint effort with the vendors beyond the initial deployment of the platform and acceptance testing. Another point of emphasis was the need for strong engagement by the labs with the standards communities (C++, Fortran, OpenMP, MPI, etc.). This engagement has increased in the past year or two and it is already paying dividends. In particular we need to explore what the communities needs are within the Fortran arena. Overall, the group would like to see better collective focus from the programs with a heavy emphasis on meeting the needs of the mission codes.

***Key Points***
The key points that were exposed in this discussion were:
- Performance portability is hard to quantify but you know it when you see it.
- Maintainability and productivity are a crucial part of the equation.
- Dependencies on more tools and libraries are clearly in our future but this can be vulnerability.
- OpenMP appears to be the de facto winner in the on-node parallelism wars but language-based approaches are showing a lot of promise.
- Procurements should emphasis the software environment more.
- We need to continue to increase our engagements with the standards communities.

***Outbrief notes (Performance portable abstractions – Hoekstra):***
- What is "good enough", i.e. a reasonable target for our code performance? The responses were that the answer varies by user community. Some communities need peak performance while others need less performance. Performance portability should be about creating more maintainable code and getting higher productivity from the code. How agile is the code base with respect to getting up and running on a new platform? There is an enormous time sink required to make this happen now.
- At what level do we want abstractions? People like high-level abstractions but many caveats are required to make full use of them. High-level abstractions may constrain too much. Where is the demand for such high-level abstractions? Community support must be developed, and DSLs provide a good example of this. We are seeing some success in the low-level abstraction layers. Template metaprogramming, Fortran, and Thrust are examples. Fortran has been effective as a performance portable approach for years across CPU-based architectures but has so far not truly responded to changes in hardware. Why? Largely because the parallel constructs in Fortran are not being supported. Are we going to come up with more parallel constructs that are not fully supported? That is a concern.
- Looking now at tradeoffs. In general, productivity drives everything else. Meta-template programming drives up complexity and compile times. Most group attendees do not want to work with OpenMP but prefer to work with something that involves more language-based constructs. There is a reason why MPI is not directive-based and that is because it is library-based - and we should learn from that.

- What do we need from vendors? We are getting better at working with vendors (this meeting is a good example). Vendors tell us what we ask for should have broad market appeal. And the vendors are listening but timelines can vary widely. Compilers are a crucial area. One of the best interactions I see is when code developers and compiler developers work together. We need to create a forum for code developers and compiler developers to talk.
- In the open community the greatest success right now is the move to LLVM. Do we want to work at this level to add language extensions? That is an open question and should be explored.
- In the (ASCR, ASC) programs we want to see stronger emphasis on the software environment in the procurements, including compiler updates and support through the life of each platform. We need stronger engagement by the labs with standards communities. We are not presently involved with Fortran standards right now, for example, and that needs to change. Can we get ASCR, ASCR and ECP to have a more common focus?

## Breakout Session E: OpenMP Futures

### Lead: David Richards (LLNL)

During this session we discussed the current state of the art as well as future research and development directions for OpenMP. We addressed questions in four specific areas:

1. **Performance Portability**: Does the OpenMP standard offer an adequate set of constructs to write code that will perform well on multiple architectures? I.e., is it possible to write performance portable code in OpenMP
2. **Usability & Interoperability:** What are the most important usability and interoperability concerns with OpenMP
3. **Memory Management:** Does OpenMP provide adequate facilities to manage memory hierarchies, especially the lowest levels?
4. **Low-level Optimizations:** Should OpenMP add features to give greater control over low-level optimizations such as loop tiling, loop unrolling, pipelining, dependency lists, etc.?

Regarding performance portability, the consensus of the participants is that the only practical approach to portability at this time is lots of ifs and ifdefs. This isn't very satisfactory since such code is difficult to understand and maintain and can end up as two (or more) separate codes that just happen to sit in the same files. Key challenges also include large performance differences between compilers and little or no support for cases where different hardware characteristics require different algorithms to obtain good performance. The best path forward appears to involve finding methods to separate the description of the work from the mapping of the work to the hardware. Relationships between work units will also likely need to be defined. Some participants expressed a hope that directives that were more descriptive (as opposed to prescriptive) might help compilers do the right thing. This lead to a discussion of whether we believe in magic compilers.

Regarding usability, we find that there are few good examples or best practices to follow, especially for OpenMP4.5. A set of such examples for non-trivial codes would be of significant value, especially since it appears that writing good code in OpenMP4.5 will require new patterns and practices. Interoperability also needs significant work. Many scientific libraries have been developed without any notion of a thread or memory context. In the past, there was rarely any meaningful context to worry about. Moving forward it will probably be essential to pass some sort of resource allocation handle to a library—perhaps something akin to an MPI communicator—to specify how a library can acquire and use resources. Tasking constructs offer a form of help, but aren't fully up to the job. One can ask for example, what would support for "unbound" threads look like and how would different modules acquire resources from a pool of unbound threads.

Regarding memory management we again agree that OpenMP is not currently adequate. Existing map constructs are far too tedious for complex data structures and need improvement. For lower levels of the memory system such as caches and registers, the situation is also not good. OpenMP writes a low-level code for the developer but offers very little control over how that code is written. Our best thoughts are that some kind of mark up on data attributes might help, especially if such mark up could be "sticky" and carry data through multiple functions, modules, or compilation units.

Finally, our discussion of low-level optimization found that optimization control directives could be beneficial, but that feature creep is a significant concern. The best driver for progress would be specific use cases. Doacross, collapse of non-rectangular loops, and loop tiling were identified as possible starting points with at least some potential for wide adoption. Future work on auto-tuning or runtime adaptive tuning would be welcome.

**Outbrief *notes (OpenMP Futures – Richards)***

- Sriram and I put together a unified set of questions in four areas. First performance portability - are there machines abstractions? What is the basic state for OpenMP? Solutions tend to involve lots of "Ifs" and moving them to other places – that is not portability.
- Descriptive options would help the compiler, but we have to agree on some magic. That will be subject of work over time.
- We have CPUs and GPUs and put in "ifs" - they're portable if the app developers say they are. Otherwise, specify units of work and mappings that then lead to more abstractions. The programming style of OpenMP needs to evolve too. We need a better STL – one that is not inherently non-threadsafe.
- Do we need best practices? We agreed "yes," it would be a good idea but no one knew where they could be found and archived. If anyone could volunteer. COE should be doing a better job of it.
- There was some discussion of classifying codes based on characteristics; conversation wandered and did not gel. We talked about nesting and parallelization, no real consensus/message.
- On interoperability, there are significant problems dealing with third part libraries. It's the spiritual cousin of a MPI communicator to inform a library what it is working with – that would be nice. The tasking constructs of OpenMP provide some support, but don't do exactly what we need. Our libraries have been developed in a context fee environment. Many things you want to do in C++ don't map to OpenMP.
- Is OpenMP adequate to manage the lower levels? We think the answer is "yes" but don't have tools to aggressively mange those levels. CUDA manages memory space and kernel launches and is good at it. How do we get there without losing performance portability? We don't know traits or where problems are, directives are sticky so that's something the runtime could trace/track (but how do you instantiate based on runtime tracking, maybe late binding - a lot of pieces to work through).
- Do we need more control? There are tensions between portability and performance. These kinds of directives are useful to a section of the community. We could use the loop collapse if it could be applied more often (i.e. non rectangular loops). Standardizing directives would serve a real purpose.

In communicating with vendors, we could do a better job of collecting examples when we can't do what we want – we need to keep pushing.

Explorations of auto tuning and runtime adaptive tuning would help.

## Breakout Session F – OpenMP Futures

### Lead: Sriram Swaminarayan (LANL)

*Write-up pending*

***Outbrief* notes (OpenMP Futures – Swaminarayan):**

- There were three main things on our wish list:
  1. We do not have enough users on the standards group in the apps area to direct where we want it to go to help the hardware side.
  2. We need a high-level way of doing memory management across platforms, so we are not reworking it for each platform.
  3. A subgroup of OpenMP needs to talk to the OpenMP subcommittee.
- We agree on a need for best practices but there are not many resources, plus they are changing rapidly in terms on implementation and performance. If you don't have good documentation features, users would not know how to use it. The standard comes out before it has been "tried in anger"; we saw as OpenACC went through growing pains.
- Examples on OpenMP tutorials – The one provided at the Supercomputing conference was a good resource, versus those on the website.
- The "gotchas" are crossing NUMA domains, i.e. simple things everyone learns the hard way. The NERSC tutorial from GCC is available. Best practices for performance portability are not documented since we don't know them yet.
- Interoperability with other threading systems is something we want OpenMP to deal with in the future.
- We would like OpenMP to be more descriptive on the execution and memory model. There are some things we can't do in OpenACC since they are not descriptive enough. You may be able to write a descriptive model on top of a prescriptive model, so you don't need both.
- We need to be able to handle shared data properly. There is no easy way path from OpenMP3 to OpenMP4.5. It would be nice to have defaults that behave in similar ways in different architectures. OpenMP needs to be more prescriptive than it is.
- There's a proposal that exists (although not accepted yet) that the configuration space has grown large and we might be able to reduce it by using something like Raja so all OpenMP directives are in one space.
- It is tiresome to use a new API on a new machine, we need consistency.
- It would be nice to assign a priority to memory.
- We discussed sequential optimization – there is much responsibility placed in the compiler. Doing an analysis for dependencies may be no trivial matter for the compiler. We also discussed fusion and proactive placing of a large number of parallel regions next to each other to avoid joining/swarming threads again.
- The configuration space keeps expanding.

## Break out Session G: Tools/Compiler/System Software Requirements

### Lead: Edgar Leon (LLNL)

This session of about 20 participants included representatives from Cray, Intel, and IBM as well as two centers of excellence: LLNL and SNL/LANL. Six questions were posed to the audience:

(1) What tools exist today that help achieve performance portability?
(2) Is memory management the most important area to achieve performance portability? Should vendors provide an interface for tools to extract information of interest?
(3) How can an application and libraries share resources amiably? Can an application specify what percentage of resources they are willing to give up to a library?
(4) What role should compiler technology play in performance portability? What features do we already have? What features are needed?
(5) Will machine learning play a significant role for performance portability? and
(6) What are some guidelines for performance portability with respect to I/O patterns (e.g., burst buffers)? How about memory layouts (for HBM, large shared caches, etc.)?

From these questions four of them were discussed and are summarized below.

**1. Existing tools that help with performance portability.**
Several areas were identified including compiler-based tools (e.g., Cray's Reveal, OpenARC), libraries (e.g., SCR, MACSio), profiling interfaces (e.g., KokkosP, OMPT), and DSLs (CHiLL, AMRStencil). The group also identified a need for tools to work across architectures and provide an "integrated" view on heterogeneous systems. There is a clear desire to have common interfaces between tools across different platforms. For application developers it is time consuming to have to learn a new tool when moving to a new vendor's platform and thus even though the tool may help address a certain need, it is not used. Totalview and ScoreP are good examples of tools that work well across multiple platforms.

The DWARF debugging standard (http://dwarfstd.org/) was brought up since vendors spend a significant amount of time modifying/extending it to address its shortcomings. However, none of this work is contributed back to the community. It would be helpful to have the vendors work together to update the DWARF standard to include the extra information that tools developers are needing to create executables and extract relevant information from them. For example, the Intel compiler adds extra information that Intel tools consume; it would be useful if that information was standardized in DWARF version 5 (future) so that other tools could use that information.

Tools that provide a more unified view of a heterogeneous system seem to be lacking. One promising approach is IBM's HPM. Another promising tool to add/identify parallelism in codes is Cray's Reveal but it does not work with GPUs (OpenMP 4.5). Having Reveal work with GPUs would be extremely useful.

**2. Tools for memory management on multi-level memories.**
There is a strong desire for tools that give application and tool developer's visibility into where data is moving as the application executes. Based on Karlin's presentation "Fundamental Cross Architecture Multi-Level Memory Support," the following features were identified to help application and tool developers understand data movement and improve the placement of data on a multi-level memory system: (1) Ability to mark user (subset) data structures and track them in the memory hierarchy throughout a code region. Associate location, effective access latency and bandwidth. Latency, in

particular, was emphasized. For example, having a histogram of what data structures experience the highest latencies would be helpful; (2) Ability to query memory properties/attributes: available space (dynamically), latency, bandwidth (numactl-like); (3) Control placement using an open interface that would work across architectures, i.e., KNL and GPUs. KNL uses libNUMA, which is supported in Linux but libNUMA does not allow access to the features described here; (4) Track data transferred from one memory to another and correlate to application's objects/data structures. And, importantly, what entity moved the data (e.g., runtime system, user initiated, OS).

PAPI was brought up as a useful tool in the past but does not provide the memory information desired and PAPI is CPU centric. It would be useful to expand PAPI to be node-based but it may include work throughout the software/hardware/OS stack. Another issue with PAPI is that the maintainers, apparently, have lost most of their funding and thus the tool, in many cases, is not accurate as brought up by Jeanine Cook in the presentation "The Importability of Performance Tools." There may be an opportunity for DOE to support PAPI or perhaps investigate other tools like Perfminer. There are also tools like Memspy, which seem to provide standard interfaces to get information about the memory subsystem but it is not clear whether this would be enough to support multi-level memories.

A key observation regarding these types of tools is that we need them to be accessible at the user-level, otherwise cannot be used by regular application or tool developers on DOE clusters. Finally, it is also desirable to identify different application uses cases of intended use of a multi-level memory system.

**3. Applications and libraries playing nicely**
There is a significant concern due to contention for resources between applications and libraries: How can application and libraries share resources effectively? What happens when the application needs all the memory, but then a library or other tool needs to execute and use resources? For example, an OpenMP application may need to call an MPI-only library or a library with Pthreads. We need a mechanism to orchestrate friendly coordination between these. This may result in primitives or a "contract" to coordinate the needs of the application and libraries. Libraries could be parameterized based on this contract but the interface would need to be defined. Cray does this implicitly with their optimized libraries, e.g., BLAS. There is an implicit handshake between the app and the library. The library may move data between memories or decide, based on problem size, which device to execute on (e.g., CPU or GPU). It would be helpful to further understand what decisions are made in these libraries and also what "application state" was changed after a library was executed.

**4. Compiler technology and performance portability.**
The group focused on two areas: code-to-code generation and descriptive vs. prescriptive programming abstractions (e.g., OpenACC and OpenMP). First, code-to-code generation (source-to-source compiler transformations) can be of significant value to achieve performance portability. The transformed code can retain important semantic information that can then be harvested by architecture-specific compilers and achieve good performance. During the workshop several examples were demonstrated including OpenARC and the importance of high-level intermediate representations and Domain Specific Languages regarding stencils through frameworks such as CHiLL and ROSE. There are some disadvantages though mainly the ability to map back to the original source code. This is a major issue when debugging for example. In addition, it takes many months for a code team to "trust" a compiler and compiler options.

Second, descriptive approaches such as OpenACC give the compiler more flexibility to optimize code and reduces application developer's burden to tell the compiler exactly what to do to optimize their codes. In many cases, however, the compiler fails to optimize code because of many constraints that ultimately the application developer can easily resolve. Thus, the group proposed a combined approach that would allow the use of a descriptive interface first and depending on the results a prescriptive approach might be necessary.

OpenACC was developed to fill the gap between OpenMP and device computing capabilities. Now that OpenMP 4.5 has similar device support, OpenACC is in maintenance mode but feature frozen. It seems that some vendors will continue to support OpenACC with bug fixes but are targeting OpenMP for future work and improvements. Considering that OpenMP is a prescriptive approach, there is a need for the OpenMP forum to have a discussion about the role of descriptive approaches within OpenMP.

***Outbrief* notes (tools/compiler/system software requirements – Leon):**
This breakout session was about tools, compilers and system requirements. We had a small but productive group with IBM, Intel and Cray present. Participants from LLNL and SNL and LANL were present on the DOE NNSA side. I will focus on the three questions:

1. ***Existing tools that may be portable across multiple architectures?***
   There was a general consensus on the desire for a more common interface between tools across different platforms. It is painful for application developers to learn a tool for a specific architecture. Another example is a tool like Totalview, particularly how they deal with profiling information - memory introspection is a big challenge for performance portability. We got into the WARF library and spent a few minutes on it. Vendors have to do a lot of work to deal with shortcomings of this library (and third party tool developers' shortcomings). If a lot of work has to be done on these existing standards, why don't vendors put more effort into changing the standard or providing feedback for the standard? Vendors would really like to see a use cases and a joint meeting (including vendors and DOE) providing use cases on what is important so it could be worked into a standard via discussions with a subset of the community.
2. ***Tools for managing memory management patterns? What is happening to data structures?***
   We need more feedback from the runtime system about the data structures. We discussed PAPI, and it was useful when it provides the right information. Architectures now make this data difficult to understand. PAPI seems to be centric so maybe moving to a more node-based-execution-like interface for PAPI. Another important area is user level tools. Many do not have access to the root level across the clusters. We need to focus on user level access. It would be useful to be able to associate latency bandwidth to certain data structures, especially latency with a certain data structure. Maybe a histogram to optimize code.
3. ***Interoperability between application and the library, and in how they share resources?*** When you come to a library, you are in a working set of your application, and that is problematic. What about setting a contract between the application and library defining how they are going to share resources? If we could come up with an API, that would be useful. Cray provided feedback and mentioned they do a lot on this with their optimized libraries. Determining whether the data is GPU, CPU, and size of data will inform the data movement accordingly. Perhaps having a better understanding of how Cray does it for their libraries could help us.
4. ***We discussed code generation, and what would be useful...how can we go back to the original source code?*** Prescriptive and descriptive approaches for things like OpenMP were discussed. It is useful to have both. It seems best to start with a descriptive approach, and then have the prescriptive approach available as an additional option.

## Breakout Session H: Tools/Compilers/System Software Requirements

### Lead: Brian Freisen (LBNL)

***Q: What tools exist today that help achieve performance portability?***
Our group had trouble answering this question, which indicated strongly that such tools probably don't exist. Or, if they do exist, they are not readily available. For example, ALCF has found that vendor-supplied tools (e.g., from IBM), are rarely sufficient or useful, and as a result they rely almost entirely on 3rd-party tools, e.g., TAU. Unfortunately there is a trend that some of these tools are supported directly or indirectly by DOE projects, which have limited lifetimes (funding). If the developers of such tools can't find a way to support themselves outside of DOE, then they'll be stuck when the project ends. TAU is a success story that has avoided this fate; ParaTools sustains itself independently of DOE. Additionally, the group agreed that the most valuable tools available to everyone in the HPC community are **compilers**. In fact, there were some interesting comments regarding compilers as tools in the "afterword" from some of the vendor representatives at the conclusion of the meeting. John Levesque, for example, cautioned that developers often want to use bleeding-edge features of compilers without fully appreciating the performance implications involved in doing so. Similarly, James Reinders said that vendor compiler teams' goals are often driven by the specifications in the procurement RFPs, which often place more emphasis on support for bleeding-edge features than on performance and stability. (One code team has a battery of 18,000 regression tests for their >1M LOC code, and Intel v13 (!) is the most modern version of the compiler suite which can pass all of the tests.)

Very quickly the discussion then focused on a tool that already is available and is probably the most valuable to everyone: **dissemination of knowledge**. Many developers in the HPC community have followed many different paths in the pursuit of performance portability, but because the knowledge gained from these exploits seems to be shared so infrequently, we often re-invent the wheel, or are never made aware of solutions that have worked well for others who are working on similar problems.

The entire breakout group agreed strongly with the notion that disseminating knowledge and experiences with regard to performance portability strategies needs to be a top priority. We brainstormed several different avenues for addressing this, including

- More frequent (and perhaps less dense) meetings such as the one just concluded in Glendale
- Collaborative forums of some form
- A webinar series
- Discussion forums
- Wiki pages

Things of this sort are beginning to precipitate already: the Trinity COE already has begun monthly KNL meetings, and IXPUG has similar regular meetings. However, a major complication in attempting to implement any of these solutions is the NDAs that each lab has with various vendors. We would therefore need a way to institutionalize these kinds of venues in a way that protects confidential information, e.g., by allowing access only to people with e-mail addresses from corresponding institutions.

***What tools/interfaces should vendors provide to extract information interest about memory access patterns, data layout in memory subsystems, etc.?***
This is a very difficult task to accomplish, in part because doing so requires access to things like hardware counters. Doing things like tracking loads/stores is possible (e.g., VTune's "memory access" mode), but the complex cache hierarchy in KNL systems makes it difficult to follow individual pieces of data in any more detail than that. As a result, we can obtain statistical samples of somewhat generic memory access patterns (loads and stores), but gleaning insight from those statistics can be challenging. The group also discussed the barrage of available flavors of malloc() and prospects for leveraging them to do memory access analysis.

***How can applications and libraries share resources amiably? Can an application specify what portion of available resources it is willing to give up for library calls?***
Intel's Math Kernel Library (MKL) is a mixed bag in this regard. On one hand, it allows the user to force MKL routines to use the same set of existing threads from the application that called it, rather than spawning new threads inside each MKL call (leading to unnecessary nested thread parallelism and perhaps degradation in performance). On the other hand, users have found that MKL often creates unnecessary copies of small matrices during calls to certain linear algebra routines.

The group also discussed the prospect of an application allowing (or restricting) a library call's access to various subsystems, e.g., is the library allowed to use high-bandwidth memory, or should it be restricted to DRAM? It's not clear whose responsibility that should be. On one hand, the purpose of libraries is to be of use to a wide range of applications and therefore should support generic, flexible interfaces; on the other hand, library developers should not be expected to re-architect their entire interfaces each time a new architecture emerges.

***What role should compiler technology play in performance portability? What features do we already have? What features are still needed?***
As discussed in an earlier section, it may not be wise to demand cutting-edge feature support from compilers when they already suffer from stability issues stemming from features which have long since been implemented. (One code team is forced to use v13 of the Intel compiler suite because no newer version can pass their entire regression test suite.)

Developers have often experienced lackluster response from compiler vendors when reporting bugs. To motivate vendors to focus more heavily on compiler stability, perhaps we should take John Levesque's and James Reinders's comments seriously, e.g., by focusing more heavily on compiler support in procurement RFPs. As it stands currently, a common experience is for a site to lose leverage with regard to compiler support once the machine has been accepted. The group gravitated significantly toward this idea in particular, and suggested assembling a suite of benchmarks (composed perhaps of mini-apps contributed from the community) that must compile and run on new systems (with new compilers) *before* they are accepted.

Another common experience with regard to compiler technology has been the direct interaction of compiler engineers with code teams. This has manifested in various forms already, including the "dungeon sessions" which NERSC and others have participated in with Intel over the last year as part of the NERSC-8 procurement. Compiler bugs that can be shown directly to engineers are often elevated to high significance within the vendor software teams and are fixed more readily. Someone also suggested inviting compiler engineers to COE meetings such as this one, both to provide/receive feedback among

HPC consumers/developers, and also to provide perspective on what goes into writing performant/stable/cutting-edge compilers.

***Outbrief* notes (Tools/Compilers/System Software Requirements – Friesen):**

- We discussed the same questions as Edgar's group. What exists already? There are a few platform-agnostic things available, like Cray "Reveal," Intel "Advisor XC." OpenMP is not a tool, but it is a mixed bag. As we've learned, there seem to be many tools and libraries, and the numbers speak for themselves that this problem is not solved. It is the sheer volume of tools that makes this learning curve so steep.
- Lack of interaction between appropriate groups is an issue; more frequent meetings, perhaps smaller groups, maybe a wiki page, and holding conference calls might help. A COE may be in a position to help with this though NDAs make it challenging. With such interactions we could quantify problems with stencils, tree traversals, and categorize them by their issues in terms of performance portability. More frequent interaction may lead to agreement in the terminology for "performant" and "portable" and "performance portable" codes. Many were excited about the idea of more interaction.
- With tools, it's interesting that at ALCF they have less reliance on vendor-supplied tools and more reliance on third-party tools. Vampir was a great success story, as an example. There are issues with third-party tools if support runs out for them. The most important tools from vendors are compilers.
- On topic #2, tools for analyzing memory patterns, this is hard to implement especially following pieces of memory through the memory subsystem (pre-fetched versus cache line). It is tricky with counters and the outer branch. It would be nice to have finer-grain information on what the memory is doing. We talked about the main "Mallocs" out there. Memkind can be useful for placing memory in a coarse way. Some would like more control.
- With respect to applications and libraries playing together nicely - MKL is an event variable. If I implement threads down to spawn new ones – that's a nice success. But it also results in unnecessary copying of data.
- Being able to inform a library about what you want to do with data instead of refactoring your library would be helpful. Those libraries don't know who the applications are. It is hard to have a generic interface.
- When libraries make certain development choices (e.g. PETSc), there is probably a balance to be struck in libraries and the applications to use them - we want both, to have freedom to choose, especially when architectures don't exist yet. Could we use emulator technology to trace data? That is an interesting talk.
- For compilers, intermediate representations of interest were discussed. LLVM was used the most, and GCC was too complicated for many.
- There are many things we want compilers to do, but we have trouble getting them to work as-is. There are even cases where codes have to use three-generations-old compilers to pass their entire suite of tests. That is a long time in terms of compiler lifetimes. The more we want from the compilers, the more complicated they will get. Solving all these problems with our code is undesirable. We should be mindful of the burden we place on compiler technology.
- A fruitful environment is when compiler teams and apps developers are in the same room. Isolated bugs can be time consuming. Showing a bug to a compiler team member results in a faster turnaround time for the solution. Intel Dungeon Sessions have observed this many times.

## Recommendations for Follow-on Meetings

What follows is a summary of recommendations that came out of survey responses (thanks to everyone who participated in those), steering committee discussions, and some side conversations with the organizers. These are meant to be advisory for the planning of a follow-on meeting.

- Fewer talks
  - Recognized that part of the point of this meeting was to lay some groundwork, but went overboard on talks. Thus - need a better process for selecting talks that keeps a balance between the labs and vendors, while focusing on the topic at hand
- Do some panel discussions.
  - Perhaps JOWOG 34-ACS style (an ASC/AWE collaboration meeting) – e.g. 3-4 short related talks followed by a panel of the speakers answering questions and debating for 15 minutes
- Post talks earlier so people can see them during the meeting
  - This can be hard to do because people are updating things in real time -but if we had a common site (a la Google site) that authors could upload directly to that would help. LBL could help with this.
- Going forward, having lessons learned from hack-a-thons should be brought out at this meeting to broadly share
  - Probably will need hardware on the floor before we get there.
- Consider lunches every day to keep things moving along – we lost a fair bit of time on Tues and Wed when people had to go offsite. Don't do dinner on the last day – some people want to leave early.
  - Note that we did the dinner on the last day to incentivize people to NOT leave early, and it appears to have worked somewhat.
- More I/O talks – e.g. sharing burst buffer experience.
  - This will come more naturally once hardware is on the floor to use (Trinity/Cori)
- Compiler-focused session – could it be multi-vendor?
  - We have the compiler working group with Sierra/Summit. But this doesn't include application people directly.
- Tools API – what hooks and performance APIs. Vendor participation? Tool community, too.
  - This is probably better done in the context of a different meeting, with a deep-dive summary at this higher level meeting
- The iPad timer worked wonders
  - Keeping the speakers to their time slots was critical for this meeting to stay on schedule and not run into the late evening hours. The use of an iPad in the first row counting down the time remaining for each speaker was hugely effective in keeping everyone on time.

If attendees reading this report have additional input that was not captured here, please let us know with a brief email to coepp-meeting@llnl.gov. Your input will help our next meeting be much improved and valuable.

## Conclusions

Overall, this meeting was deemed successful in raising awareness of the issues vis-à-vis performance portability, allowing a broad spectrum of speakers to weigh in on a huge variety of potential solutions to the challenge, and breakout discussions to drill down on the most important matters. But it in no way solved the problem. Instead, it will hopefully inspire continued ongoing discussions on this topic – one that will become critically important as the next generation of large-scale procurements is landing at the DOE sites.

The Centers of Excellence are a hugely valuable confluence of applications, facilities, vendors, and software providers aimed at working together on the challenge of using these machines effectively. Application teams have clearly indicated that performance portability is important to them, and this is manifesting itself within the strategies of these COEs and the feedback to the vendor community. While large meetings such as this are important for organizing our thoughts and sharing experiences, the biggest takeaway from this meeting is that we need to strive to establish meaningful and lasting cross-COE collaborations in the interim and for the benefit of the entire community. The motivation is there – what's needed is strong leadership, coordination, continued cooperation, and sense of a "shared fate" that this is something that only a community as strong as the DOE, the vendor partners, and our universities can solve together.

## Attendee List

| *Last Name* | *First Name* | *Organization* | *Email Address* |
|---|---|---|---|
| Benali | Anouar | ANL | benali at anl.gov |
| Keipert | Kristopher | ANL | KWK at IASTATE.EDU |
| Kumaran | Kalyan | ANL | kumaran at anl.gov |
| Morozov | Vitali | ANL | morozov at anl.gov |
| Parker | Scott | ANL | sparker at anl.gov |
| Pope | Adrian | ANL | apope at anl.gov |
| Romero | Nichols | ANL | naromero at alcf.anl.gov |
| Thakur | Rajeev | ANL | thakur at mcs.anl.gov |
| Vazquez Mayagoitia | Alvaro | ANL | vama at alcf.anl.gov |
| Berry | Michael | CRAY | mrberry at lanl.gov |
| DeRose | Luiz | CRAY | ldr at cray.com |
| Gould | Jay | CRAY | JayGould at Cray.com |
| Levesque | John | CRAY | levesque at cray.com |
| Poxon | Heidi | CRAY | heidi at cray.com |
| Delalondre | Fabien | EPFL | fabien.delalondre at epfl.ch |
| Appelhans | David | IBM | dappelh at us.ibm.com |
| Bertolli | Carlo | IBM | cbertol at us.ibm.com |
| Changhoan | Kim | IBM | kimchang at us.ibm.com |
| Cordery | Matthew | IBM | mcorder at us.ibm.com |
| Gonzalez Garcia | Juan | IBM | jgonzal at us.ibm.com |
| Grinberg | Leopold | IBM | leopoldgrinberg at us.ibm.com |
| Jacob | Arpith | IBM | acjacob at us.ibm.com |
| Kim | Changhoan | IBM | kimchang at us.ibm.com |

| | | | |
|---|---|---|---|
| Curley | Joseph | INTEL | joseph.c.curley at intel.com |
| Gabrielson | Liza | INTEL | liza.gabrielson at intel.com |
| Pennycook | Simon | INTEL | john.pennycook at intel.com |
| Reinders | James | INTEL | james.r.reinders at intel.com |
| Bergen | Benjamin | LANL | bergen at lanl.gov |
| Bird | Robert | LANL | bird at lanl.gov |
| Ferenbaugh | Charles | LANL | cferenba at lanl.gov |
| Garrett | Charles | LANL | ckgarrett at lanl.gov |
| Gunter | David | LANL | dog at lanl.gov |
| Kelley | Timothy | LANL | tkelley at lanl.gov |
| Lee | Stephen | LANL | srlee at lanl.gov |
| Loncaric | Josip | LANL | josip at lanl.gov |
| Long | Alex | LANL | along at lanl.gov |
| Nam | Hai Ah | LANL | hnam at lanl.gov |
| Nystrom | William | LANL | wdn at lanl.gov |
| Rockefeller | Gabriel | LANL | gaber at lanl.gov |
| Swaminarayan | Sriram | LANL | sriram at lanl.gov |
| Womeldorff | Geoff | LANL | womeld at lanl.gov |
| Basu | Protonu | LBL | pbasu at lbl.gov |
| Deslippe | Jack | LBL | jrdeslippe at lbl.gov |
| Doerfler | Douglas | LBL | dwdoerf at lbl.gov |
| Friesen | Brian | LBL | bfriesen at lbl.gov |
| Hartman-Baker | Rebecca | LBL | rjhartmanbaker at lbl.gov |
| Nguyen | Tan | LBL | tannguyen at lbl.gov |
| Ovsyannikov | Andrey | LBL | aovsyannikov at lbl.gov |
| Van Straalen | Brian | LBL | bvstraalen at lbl.gov |
| Beckingsale | David | LLNL | beckingsale1 at llnl.gov |
| Biagas | Kathleen | LLNL | biagas2 at llnl.gov |
| Black | Aaron | LLNL | black27 at llnl.gov |
| Bleile | Ryan | LLNL | bleile1 at llnl.gov |
| Brugger | Eric | LLNL | brugger1 at llnl.gov |
| Collette | Michael | LLNL | collette1 at llnl.gov |
| Dawson | Shawn | LLNL | dawson6 at llnl.gov |
| de Supinski | Bronis | LLNL | bronis at llnl.gov |
| Draeger | Erik | LLNL | draeger1 at llnl.gov |
| Epperly | Thomas | LLNL | epperly2 at llnl.gov |
| Futral | Winfield | LLNL | futral2 at llnl.gov |
| Glosli | James | LLNL | glosli at llnl.gov |
| Haque | Riyaz | LLNL | haque1 at llnl.gov |

| | | | |
|---|---|---|---|
| Hornung | Richard | LLNL | hornung1 at llnl.gov |
| Jones | Holger | LLNL | holgerjones at llnl.gov |
| Karlin | Ian | LLNL | karlin1 at llnl.gov |
| Keasler | Jeff | LLNL | keasler1 at llnl.gov |
| Kunen | Adam | LLNL | kunen1 at llnl.gov |
| Laguna | Ignacio | LLNL | lagunaperalt1 at llnl.gov |
| Leon | Edgar | LLNL | leon at llnl.gov |
| Loffeld | John | LLNL | loffeld1 at llnl.gov |
| Miller | Mark | LLNL | miller86 at llnl.gov |
| Mohror | Kathryn | LLNL | kathryn at llnl.gov |
| Neely | Rob | LLNL | neely4 at llnl.gov |
| Nelson | Jarom | LLNL | nelson99 at llnl.gov |
| Pankajakshan | Ramesh | LLNL | pankajakshan1 at llnl.gov |
| Peles | Slaven | LLNL | peles2 at llnl.gov |
| Poliakoff | David | LLNL | poliakoff1 at llnl.gov |
| Pozulp | Michael | LLNL | pozulp1 at llnl.gov |
| Richards | David | LLNL | richards12 at llnl.gov |
| Ryujin | Brian | LLNL | ryujin1 at llnl.gov |
| Scogland | Thomas | LLNL | scogland1 at llnl.gov |
| Shoga | Kathleen | LLNL | shoga1 at llnl.gov |
| Sinha | Punita | LLNL | sinha2 at llnl.gov |
| Still | Charles | LLNL | still1 at llnl.gov |
| Tagani | Bujar | LLNL | bujar at llnl.gov |
| Voronin | Alexey | LLNL | voronin at llnl.gov |
| White | Daniel | LLNL | white37 at llnl.gov |
| Yeom | Jae-Seung | LLNL | yeom2 at llnl.gov |
| Kelly | Suzanne | NNSA | Suzanne.Kelly at NNSA.Doe.Gov |
| Macaluso | Antoinette | NNSA, ASC | antoinette.macaluso at leidos.com |
| Branch | Greg | NVIDIA | gbranch at nvidia.com |
| Feldstein | Bob | NVIDIA | bfeldstein at nvidia.com |
| Gibbs | Tom | NVIDIA | tgibbs at nvidia.com |
| Larkin | Jeff | NVIDIA | jlarkin at nvidia.com |
| Rennich | Steven | NVIDIA | srennich at nvidia.com |
| Sakharnykh | Nikolay | NVIDIA | nsakharnykh at nvidia.com |
| Zeller | Cyril | NVIDIA | czeller at nvidia.com |
| Bernholdt | David | ORNL | bernholdtde at ornl.gov |
| Hernandez | Oscar | ORNL | oscar at ornl.gov |
| Lopez | Matthew | ORNL | lopezmg at ornl.gov |
| Messer | Bronson | ORNL | bronson at ornl.gov |

| | | | |
|---|---|---|---|
| Straatsma | Tjerk | ORNL | str at ornl.gov |
| Vetter | Jeffrey | ORNL | vetter at computer.org |
| Agelastos | Anthony | SNL | amagela at sandia.gov |
| Brunini | Victor | SNL | vebruni at sandia.gov |
| Clausen | Jonathan | SNL | jclause at sandia.gov |
| Cook | Jeanine | SNL | jeacook at sandia.gov |
| Dinge | Dennis | SNL | dcdinge at sandia.gov |
| Glass | Micheal | SNL | mwglass at sandia.gov |
| Hammond | Simon | SNL | sdhammo at sandia.gov |
| Harstad | Eric | SNL | enharst at sandia.gov |
| Hoekstra | Robert | SNL | rjhoeks at sandia.gov |
| Kenny | Joseph | SNL | jpkenny at sandia.gov |
| Laros | James | SNL | jhlaros at sandia.gov |
| Lin | Paul | SNL | ptlin at sandia.gov |
| Moore | Stan | SNL | stamoor at sandia.gov |
| Ruggirello | Kevin | SNL | kruggir at sandia.gov |
| Simpson | Emily | SNL | emily.simpson at nnsa.doe.gov |
| Stevenson | Joel | SNL | josteve at sandia.gov |
| Trott | Christian | SNL | crtrott at sandia.gov |
| Wilke | Jeremiah | SNL | jjwilke at sandia.gov |
| Paisley | Beau | Allinea | bpaisley at allinea.com |
| Perks | Oliver | AWE | oliver.perks at awe.co.uk |
| Joo | Balint | TJNAF | bjoo at jlab.org |
| Martineau | Matthew | Univ of Bristol | m.martineau at bristol.ac.uk |