DOE Centers of Excellence Performance Portability Meeting

August 22 – 24, 2017
Denver, CO

Post-meeting Report

LANL LA-UR-18-31215

Table of Contents

EXECUTIVE SUMMARY

The 1st Department of Energy (DOE) Centers of Excellence (COEs) Performance Portability Meeting in 2016 was successful in bringing together application teams, tools developers and vendor partners across the COEs, to highlight the growing importance of performance, portability and productivity with DOE's large advanced architecture supercomputer procurements. The community agreed on the complexity and importance of the problem, and shared solution strategies outlined in the post-meeting report. Many of these solutions were early in their development cycle due to the lack of production hardware and software, as is common in DOE procurements. Tight procurement timelines require proxy applications to predict real application behavior and existing hardware to inform performance of future hardware.

Since the first meeting, a large number of DOE systems have been sited, accepted and put into production use. This provides code teams with the opportunity to evaluate the state of application performance portability on real hardware and software. Systems include:

- **Theta**, ANL [11.69 PF, Cray XC40, 4392 nodes, 64-core Intel Xeon Phi processor w/ 16 GB MCDRAM + 192 GB DDR4 RAM + 128 GB SSD per node] – production July 2017
- **Trinity**, ACES @ LANL [20.16 PF, Cray XC40, 9436 Intel Haswell nodes (dual socket 16-core Intel Xeon processor w/ 128 GB DDR4 RAM) & 9984 Intel Knights Landing nodes (68-core Intel Xeon Phil processor w/ 16 GB MCDRAM + 96 GB DDR4 RAM), Cray DataWarp Burst Buffer 3.7 PB] – Haswell production Feb 2016, KNL Open Science Feb 2017, merging partitions & combined production use July 2017
- **Cori**, NERSC @ LBL [14.01 PF, Cray XC40, 2388 Intel Haswell nodes (dual socket 16-core Intel Xeon processor w/ 128 GB DDR4 RAM) & 9688 Intel Knights Landing nodes (68-core Intel Xeon Phil processor w/ 16 GB MCDRAM + 96 GB DDR4 RAM), Cray DataWarp Burst Buffer] – production May 2017
- **Summit (ORNL) & Sierra (LLNL) Early Access Systems** [IBM Power8 + NVIDIA P100 nodes] – early access January 2017

The 2nd DOE Centers of Excellence Performance Portability Meeting in August 2017, was an excellent opportunity to move from projections to actual measurements on production systems or early access hardware, which more closely mirror performance on final delivery systems. This year Workshop participants also explored the concept of productivity in association with performance and portability.

GOALS

The high-level goals for this meeting were largely the same as the first meeting:

- Share experience between the COEs,
- Make connections between application teams working similar ideas or algorithms,
- Address the numerous challenges of performance portability,
- Engage vendors in helping us succeed in our performance portability goals,
- Broaden discussion to others outside our immediate COEs.

The one new goal for 2017 was the last: to broaden the discussion to include other interested parties outside of the immediate DOE Centers of Excellence. This intent of this new goal was to capture diverse opinions and ensure exposure to a wide range of up-coming solutions outside of our current scope. The Call for Participation encouraged participation from vendors outside of current DOE system procurements, university partners and international partners. A list of attendees can be found on [pg. 56] of this document.

HIGH-LEVEL TAKEAWAYS

The key takeaways from the meeting are as follows:

- Performance, portability and productivity are unanimously agreed to be important to application development on advanced architectures, but the relative weights of importance for each one varies significantly from person-to-person depending on their daily challenges and work scope.
- There has been significant progress on development and hardening of the tools for performance portability, going beyond proof of concept in smaller proxy applications to full applications, which was evident through a healthy set of Application Experience talks
- Fortran codes still do not have a good solution toward performance portability
- There is a strong need to effectively communicate with the standards committees and with vendors on their implementation of standards to ensure application needs are being met
- As a community, we need to better define what we want and then get that into the appropriate metrics and language to use in procurements

There was clear consensus for the need to continue with follow-on meetings at an appropriate cadence consistent with availability of new system hardware to inform the path to performance, portability and productivity. Lawrence Berkeley National Laboratory (LBL) is leading the organization of the next meeting, scheduled for April 2 – 4, 2019 in Denver, CO.

RECOMMENDATIONS FOR FOLLOW-ON ACTIVITIES

The community identified additional follow-on activities to ensure continued momentum and collaboration until the next meeting. Activities were focused on influencing the HPC/computing communities, communicating best practices, and developing additional resources:

Influence

- **HPC procurement through RFP and technical specifications:** Metrics that consider performance, portability and productivity equally have been lacking in HPC procurements, rather leaning more heavily on single-system performance. A working group will help to define additional metrics and technical specifications to ensure consistent measurement of HPC system characteristics purchased by the DOE. Definition of new metrics and consistency of measurement will shape future procurement language to better embody the needs of application developers in the new computing landscape.

- **Standards committees:** The committees that shape programming languages and APIs that impact PPP, such as the ISO C++ committee and OpenMP ARB, need to hear the concerns of this community, with a unified voice from DOE labs and our vendor partners to carry the community weight. Stronger communication is needed between application developers with local standards committee representatives to understand impact of specifications, and amongst all DOE standards committee representatives and our vendor partners.

Communicate best practices

- **Leverage existing conferences/meetings:** The community can continue momentum and communicate interim results through smaller meetings such as a workshop or Birds-of-a-Feather session at SC18 (November 2018) and/or the Exascale Computing Project All-Hands Meeting (January 2019).
- **Create new working groups:** As issues are identified, new working groups will be established to define the requirements and devise next steps.

Develop

- **Definitions and metrics for Performance, Portability and Productivity:** As of the 2nd meeting, definitions and metrics for success of PPP are not uniformly agreed upon. A hackathon would be a good venue to define and measure metrics on a variety of platforms and across multiple codes with friendly users to inform the community on measuring performance portability. Initial measurements would use the definitions provided by S. J. Pennycook et al. [A Metric for Performance Portability, Proceedings of the 7th International Workshop in Performance Modeling, Benchmarking and Simulation of High-Performance Computer Systems]
- **More online resources:** More generally available content is needed to keep everyone up to date on the current state of PPP tools and approaches, as well as inform newcomers to the issues.
    - A resource provided through LBL, has captured significant initial content - http://performanceportability.org. They are soliciting additional contributions (e.g. case studies) from the community.
    - Additional curated online resources for best practices in the community are needed.

BACKGROUND

The Department of Energy (DOE) Centers of Excellence (COEs) Performance Portability Meetings provide 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 Exascale plans for risk mitigation. Two primary goals of this meeting are to:

- 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.
- Identify major challenges toward the goal of performance portability and associated productivity, and work with vendors and tool providers to determine implementations and solutions to meet their own performance criteria without inadvertently impairing performance results elsewhere.

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 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 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.

Attendees included application developers at five COEs preparing their codes, vendors chosen to provide the next-generation platforms, and solution-providers (DOE or third party) who are developing software tools aimed at helping application teams approach the challenges of performance portability.

ACKNOWLEDGEMENTS

The 2nd DOE Centers of Excellence Performance Portability Meeting was made possible through a community effort. We'd especially like to acknowledge the work of:

- Dee Cadena, Association for High Speed Computing, and Gloria Montoya-Rivera, LANL HPC-DO, for their exceptional organizational support
- Tina Macaluso and Emily Simpson, NNSA/Leidos, for their incredible scribe capabilities to ensure workshop information and discussions were accurately and thoroughly captured
- LANL ASC Program, Integrated Codes for financial contributions for the meeting space
- Steering Committee

AGENDA / TALKS / PANELS

The final version of the meeting agenda is found at: http://ww.lanl.gov/asc/doe-coe-2017.php along with links to the presentations and posters. What follows are highlights of the talks, breakouts, and panels presented at the meeting. Through the exceptional work of Tina and Emily as scribes, many of the highlights are provided in the speakers' own words.

DAY 1 – Tuesday, August 22, 2017

Session 1 – Performance Portability, Best Practices, Words of Wisdom (Chair: Rob Neely, LLNL)

John Levesque (Cray) - Setting Expectations for Performance Portability between Companion Accelerator and Manycore Systems

- Regardless of the programming language and parallel programming model one selects**, the user must understand target architecture**, and that understanding can lead to factors of 3x-4x in performance.
- When looking at re-factoring legacy code, following the steps shown on slide 25 will help to set a good foundation; then work to reduce memory overhead, stack pressure and otherwise optimize memory bandwidth and vectorize the code. John gives specific examples of optimal code performance and how it was achieved.
- To get performance improvements of ~3x-4x, you have to recognize that you must have two versions of the code. You cannot get this improvement with a portable code.
- Performance should be more important than productivity.

Mark O'Connor (Arm) - When Performance Portability is Less than Perfect: Matching Applications to Architectures

- Even when performance is portable, it would not be the same on every architecture (or there would be no need for multiple architectures). And, portable performance does not equal equivalent performance.
- A case study comparing Haswell to KNL shows that different apps benefit from the KNL architecture to different degrees. We find there is a wide range of performance accessibility, even for well-optimized codes. Not every code needs to run well on every architecture.
- How do I match the right app to the right cluster? Machine learning (ML) is a popular approach these days for many things. Other options? Let users decide where to run their code. If we do nothing then we will get vendor-led performance measures developed that will be different on every system. It is not about saying which is best. **We have become such a diverse community that we need to assess which hardware is best for which apps.**
- Standardization helps – but what would a standard approach to characterizing app performance look like? Every vendor wants to promote what it does best (and hide what it does not do well). But, agreeing on standards does not enable the vendors to do this.
- You are noting that the performance counters are designed by CPU engineers. If we want to see standardized counters then we need to figure out what to ask the vendors to provide to you. Procurement is a blunt instrument but if/when it is heavy enough it can motivate behavior. I

talk with CPU engineers about possibilities and they respond by telling me that no one has asked for that yet. The community needs to identify a set of counters; doing that can be compelling.

Karl Rupp (ANL) - Exascale Computing without Templates

- This is a summary of what the PETSc team has experienced in making sure large-scale apps achieve performance portability. PETSc is the solver library and not a stand-alone application. As we have multi-physics codes coupling different things, the application can become part of a larger software stack (even if that is not the case today).
- In the long term, we find that **we are paying technical debt as apps become more complex and the template begins to confine what users are able to do.** What are the disadvantages of C++ templates? First is static dispatch when architecture-specific information is only available at run time. "Change code and recompile is not acceptable advice." Another disadvantage comes from dealing with compilation errors. There are problems associated with stuffing everything into type names (in fact, there is now a competition for "C++ Error Explosion", suggesting we have reached the point where things have been taken too far). Scope limitations are another disadvantage of using C++ templates. Template metaprogramming lacks state and optimizations across multiple code lines are difficult or impossible.
- Excessive use of C++ templates complicates debugging; stack traces get longer names and become deeper and setting good breakpoints can become harder as you reuse code with templates. Lack of a stable ABI can lead to crashes and tricky situations when things appear to run but ultimately run to an undefined state. And, based on our experience, a non-technical point associated with using C++ templates is the high entry bar.
- So, what is the path forward to performance portability? **We have to manage complexity through good interface design, re-factor code when needed, and hand-optimize small kernels only** (discussed BLIS methodology as example). Only at the very bottom level of the stack is architecture-specific information defined. We have to **adopt professional software development practices** (e.g. nightly tests, performance monitoring tools) and we should be prepared to **maintain different data structures, as needed**. Reinventing the wheel is not productive; we need to be able to reuse existing libraries and focus on domain-specific and application-specific aspects of code. **We need to provide incentives and reward those who work to build and maintain the app**. We claim that performance portability may be a byproduct of addressing software productivity aspects on a larger scale.

John Pennycook (Intel) - Implications of a Metric for Performance Portability (PP): Necessity of Specialization and Application-Specific Abstractions

- I disagree that performance portability means different things to different people. We need to come up with a definition that people can use to measure their apps or we will not be able to really discuss performance portability. And, even if/when we come up with a definition it needs to be the right one and have no hidden traps.
- Our definition is this: a measurement of an app's performance efficiency for a given problem that can be executed correctly on all platforms in a given set. Architectural efficiency is defined as the ratio of observed to achievable and application efficiency means how well you are doing

relative to the best known performance seen on that platform (this is used to indicate whether you are paying a performance penalty by using an abstraction layer). The metric we propose is the "harmonic mean of an app's performance efficiency on a set of platforms for a given problem". With this you can make specific claims and evaluate performance portability on platforms of interest to you.

- If you accept our proposed metric and definition then there are implications: users can compare PP apps/libraries/framework support for their platforms and pressure developers to focus on platforms with poor support. It also enables developers to ask about best practices.
- If you have a PP program - and some degree of specialization for target architecture is happening -then the question becomes how to specialize code for a target platform. As you go lower in the software diagram you have to live with decisions that have been made at lower levels. One can exploit compiler knowledge, exploit library knowledge, exploit app knowledge or exploit domain knowledge, and we think users should be the ones to decide where such exploitation should happen.
- We need to have shared definitions and metrics. With the metric and definition I presented we are proposing a realistic approach to achieving high performance portability (a,p,H). Methods for user-driven specialization need to be adopted via standards, so we need not all have to reinvent. We would like to begin to look at this with respect to real codes.
- How would it apply to a large code, recognizing that it is difficult to evaluate the best implementation of that code on a given platform? It is a concern. If you can isolate some sections of the code then you can assess the PP of those sections. If there is a mini-app then you can use it to guide your selection of the best option for specialization.

Performance Portability Panel Discussion - Rob Neely (LLNL), Discussion Moderator

- What did we hear from these four speakers? John told us that performance trumps portability and you need multiple versions of code. Mark says a standard way is needed to measure performance and the roofline model is a good place to start. Karl says C++ templates are not worth the pain at production scale and one must have multiple versions. John has a good start at a definition for performance portability, and the scale for assessing it is not binary. That is the summary of what we have just heard.
- ***Question:*** Some of us are working with .5M LOCs – how does that change what you say?
  Levesque – It makes my argument doubly important. The world is not the Stream benchmark or HPL or a mini-app (unfortunately).
- ***Question:*** I have to maintain multiple versions of my code?
  Levesque – Yes and no. You run a profile and recognize the most important, time-consuming chunks of code (and they may be large) and then you have to determine for each chunk what the best strategy is. This study was on a 6K line of a large climate model and they determined they do need two sources, one for the GPU and one for the cache-place machine.
  Rupp - Your .5M LOC is not performance critical in every line. There are likely hotspots that are more manageable. If your data structures can be well matched to the architecture then you can get far with reasonable effort. If your data structures are bad and have to be changed then that impacts a larger share of code (but is still localized, although it could mean refactoring 50K-100K

LOC). It is worth the effort to abstract away the data-structure specific accesses and then the app-specific aspects remain unchanged. You factor out things that are relevant for performance and that tends to be a smaller faction of the code you will need to break down to what is manageable.

O'Connor – There are codes that have hundreds of years of effort in them where correctness and performance are important. You can find hotspots. Some codes will be the reason why we are pulling MPI+OpenMP as a paradigm into Exascale. I believe multiple architectures benefit HPC because some codes are so important that you will buy a machine because it runs that code well. Task-based parallelism offers much for people writing code from scratch - but current code is not going to be converted; I don't believe that will ever happen.

Pennycook – Specialization is a graduated notion and is tied to how much performance you want. The first step is to get the code running on systems you care about and then it is about augmenting with app-specific abstractions.\

- ***Comment:*** Why go with MPI+X at Exascale? Because the task-based models are not proven to be portable and are still research-grade.

  O'Connor – I agree. We need to go with MPI to Exascale anyway because there are apps that will never port to task-based.
- ***Comment:*** There is a need for PP for hardware. How does that change the formula? If we looked at all apps running on a single hardware then the clustering on the axis changes. Does that change things?

  Pennycook – An average of all apps on one type of hardware? I had not considered that. You might be able to apply our metric to it in some way; looking at things like performance portability across different sets would tell you something like that. CPUs, Xeon and Xeon Phi are more likely to have similar PP scores if the platforms are similar. I need to think about it further. That would be an important metric for people doing procurements.
- ***Question:*** When you optimize for one architecture and then we are told by higher-level lab management that we have to be fast on Sierra and Trinity – and that means we have to have multiple versions of the code – how do we sell that to senior management?

  Pennycook – You cannot get high performance without doing specialization we think you have to do. You can set a lower bar but if you want "high" performance then you need some form of specializing and that means writing some portion of the code multiple times (e.g. CPU, GPU, FPGA).

  Levesque – For my study the code optimized for KNL and Xeon ran reasonably well on the GPU but you could get another 4x improvement if you destroyed performance portability. If you give up a factor of 4x on one machine then you might be able to get what you want.
- ***Question:*** Part of our app is a highly optimized gather/scatter library written in C 15years ago with pre-processor statements. It needs to be touched (and needed to be touched to do MPI multiple GPUs). Years back I thought about porting the whole thing to templates because the barrier to entry might be better. When you backed PETSc away from C++ templates, did you have to do some ugly work in C?

  Rupp – Yes there is code in PETSc that is basically doing C++ with preprocessor statements. As

long as your template does not propagate across the interfaces you can have a localized set of templates. But you do have to balance the use of C++ compiler with the convenience the user sees. Developers have a better chance of understanding and maintaining the code versus asking the user to do magic. If you are designing within a small set and have control over how things go then you can use templates if/when that is the best thing to do.

- ***Comment:*** I don't believe we can achieve PP across architectures. Many apps were written decades ago, we are trying to adapt those apps because they are too big and that is different from the problem looking forward, i.e. a new generation of hardware requiring much parallelism which can be exposed in terms of tasks. We can revisit algorithms and data layouts relate to the underlying memory hierarchy (i.e. depth, size). There are ways in which algorithms can be written, and directives, to encapsulate some of the information needed. I prefer that we look at how to write a code that is capable of being PP and preserving language that scientists prefer. If we have a way to solve the science, can we address that more directly rather than be constrained by the legacy we have been handed? Think of users in terms of graduate students that will help us develop apps on the new systems that are coming.
- ***Comment:*** The GPU is the odd man out. If not for the GPU (and if you had only cache-based machines) then you could easily write PP code. GPUs have a definite play in HPC but they require intricate programming.
- *Pennycook* – As potential solutions I see a system for managing complexity so there can be different code paths. Or writing in an app-specific or domain-specific language and recognize that you will have to rewrite the back-end when a new architecture comes along.
- *O'Connor* – We are giving the vendors an easy pass. You could make the 4x better performance the vendors' problem. Google used TensorFlow as their standard runtime and as a result the amount of effort the vendors put into running it well is huge.

Session 2 – Memory Hierarchy (on-node) (Chair: Doug Doerfler, LBNL)

Stephen Olivier (Sandia) - Memory Management Extensions for OpenMP 5.0

- Last year OpenMP issued a technical report describing some things we wanted to try to do to manage memory in OpenMP 5.0, which we expect to release ~SC18. There are core features we are trying to put into a preview of OpenMP 5.0 now and then there are a number of less-mature features that might get into the full release.
- We are adding an OpenMP Allocator as an object that fulfills allocation requests. There will be an initial choice of pre-defined allocators (default, high capacity, constant which is read-only, high bandwidth, low latency, team local) to expose features.
- There will be an allocate directive and clause to specify that the OpenMP implementation will perform the memory allocation and indicate which allocator to use. We are trying to implement OpenMP, C++ and Fortran, and our doing that affects the features we consider. For C/C++ we want to provide new API routines that look like omp_alloc and omp_free to identify where you want the space to be allocated for the data. A default allocator will be used when an allocator is not specified on an allocation directive, clause or API routine call.

- We also want to define memory spaces, based on combinations of traits that span various dimensions. This will be a future feature. The implementation will map the group of traits onto physical resources. There will also be a fallback allocator to specify what to do when the memory request cannot be satisfied (with a number of options). We will look at better support for C++, explicit optimization for NUMA, being able to query available resources, special code generation required by some memories, and static (compile-time) allocator mappings.
- ***Question:*** Are there any prototypes we can use to judge performance?
  Several vendors have implemented their own directives (Intel for Fortran, Cray), and I have experimented with them. The resemblance to the proposal is not 1:1 but the notion of using directives has been experimented with. As we get into the technical report the vendors will focus on implementation.
- ***Question:*** How much flexibility do the directives give to the compiler to do things differently?
  For the extended set, the ability to set things specifically will be there. Anything else not said in the specification is generally up to the implementation, so you can do other things, so long as you are doing what the spec requires.

Tom Scogland (LLNL) - Deep Copy and Unified Memory in OpenMP

- What Stephen talked about is how to figure out where to place memory whereas these remarks are focused on how to tell the compiler, runtime what access you need. If everyone has unified memory, why use map? The answer is because not everything has unified memory and not all unified memory is the same (e.g. do atomics work across the full system?). Mapping provides more information to everything in the stack and the more information you can give to compiler, runtime, etc. the better the job can be done.
- How to specify unified memory in OpenMP will not change but we can do something to make non-portable features portable by specifying what they provide, give users a way to assert they need that and give implementers a way to react to that assertion. When the user makes an assertion then the compiler has an ability to react to it. This allows the user to write a code that will be portable to the machine it will work on (note that is not necessarily all of the hardware that OpenMP in general has to work on). One solution could be a way to assert what requirements you have of the system via extension declarations (extension name options, e.g. unified_address, unified_memory, system_atomics, system_coherence).
- Deep copy has been a topic for a long time. It is possible to use deep copy in OpenMP today. Manual deep copy works by pointer attachment while mapping deep structures. You map a structure, map the data inside with a link, and then it will map the data and attach it to the structure. But, this means using many pragmas if you have many pointers and is not ideal. And, if you have an array then you have to break up the code (something we would like to avoid). We want to offer a mechanism that allows the user to define what should happen for a given/type of group of mappings in terms of normal map clauses. The goal is to have separation of concerns – write once and apply where it is needed. No explicit map is required if you have an explicit structure and mapper for it.

- Any given mapping (assuming you are mapping to and from the device) can be broken into 4-6 stages and the user can map arbitrary data (mappers are declared by stages and all are replaceable). This means we are not writing a language with conditionals in pragmas.
- Stephen and I have both pointed out work that is going on to give OpenMP more refined control over memory visibility. Unified memory is becoming more common but not all unified memory is created equal and deep copy with composition can make mapping less painful.

CJ Newburn (Nvidia) - A Declarative Approach to Managing Memory Properties

- The HiHAT community project focuses on a declarative approach to memory. We face challenges due to the diversity in memory in platforms, program semantics, usage types and performance tuning.
- There are the declarative (i.e. "what" not "how" or "when" or "who" – this can be abstracted so the best available implementation can be plugged in) and imperative (i.e. more specification) approaches. For the former you need an abstraction so you can declare something (the runtime is asked to make it so) or, in some cases, the user has already made it so and needs to inform the runtime or other parts of the program. Implementations can be plugged in to enable best performance for each target. The developer and tuner need not know the "how" – at a higher layer (i.e. scheduler) the enumerator capabilities can be deployed.
- With HiHAT we want to enable the ability to declare the traits required so, if it is available for a given platform, one trait may be selected over another and then allocated. As a result, an abstraction is created with various properties. The greatest promise is at the boundary between what is target-agnostic and what is target-specific (and that is what our project seeks to do).
- Traits are associated with semantics (size, usage patterns, load/store versus block-oriented) and performance (e.g. device kinds, state, data layout). A data view is a logical expression to express what the user cares about and the tuner wants to manage. The implementation invokes the allocation registered for particular resources and enforces the traits.
- For resource enumeration, the goal is to have something that can work across platforms to assess what is there (including associated resources) and enumerate it once to avoid double coverage. A model would tell you properties of what things look like; you need a way to expose them to some form of abstractions to the cost model. We look at the APIs to assess the notion of complexity and find HiHAT has fewer unique APIs and few static API calls. This makes clear that we can accelerate optimization space exploration. We are not proposing HiHAT as a new user-facing interface (it is primarily something that runtimes would use).
- What's next? We are asking users to contribute because we have to listen to our customers: apps, runtimes, and programming models. We are not trying to replace OpenMP; we want to develop something that works well with OpenMP. We have user-facing memory abstractions – how do they layer on top? How do we expose the goodness available from the hardware? We encourage you to try out what we have to offer (it is all open source so everyone can contribute).
- ***Question:*** How is HiHAT different from what OpenCL is trying to achieve?
  Some OpenCL people have agreed to provide us with that analysis. There is a sense – among

the people who work on OpenCL - that OpenCL is both too high and too low. I think performance portability has to occur at a higher place on the stack (compared to OpenCL).

Sean Williams (New Mexico Consortium/LANL) - Simplified Interface to Complex Memory (SICM)

- SICM is an Exascale code project. We are addressing the problems we know are approaching with heterogeneous memory (whether Intel Xeon Phi/MIC, NVLink, Gen-Z, 3D XPoint). How to expose heterogeneous memory to users? NUMA? Block devices? Exotic buses? We care about coordination between processors and threads, and also about the general notion of portability (homogeneous exposure of at least a few operations, e.g. allocate, deallocate, migrate, introspect and arbitration/coordination).
- We have set up a way for NUMA nodes to specify a number of ordering (based on preference such as bandwidth, for example) and this allows a preferred normality as NUMA is configured on KNL. For now this is configured via sysfs and orderings become policy. This is an in-place solution that is unobtrusive (because memory policies are already part of Linux) and provides an initial answer to the problem of heterogeneous memory exposed via NUMA.
- What about arbitration problems? You can easily end up with imbalanced workloads. We have some ideas for a kernel solution or a user solution (although this would be tricky and require a custom allocator and could lead to bookkeeping concerns).
- What about block devices? At present data is mirrored in memory and there might be a future in swap. We have heard that Linux may be optimizing swap, anticipating the reality of solid-state drives and if you want an expanded memory pool via MIME then swap might be the answer.
- Files is a user-space problem and we are open to a general solution to problems that heterogeneous memory presents being – itself – heterogeneous; this turns into a problem of standards. The solution may simply be to enumerate things and may require a shared-memory allocator. We are leaning towards the conclusion that NUMA needs to be addressed at the kernel level.
- Right now we have kernel patches to add orderings for NUMA devices. Since there is nothing on the hardware side yet it relies on a sysfs file to specify the ordering for a node as writable files. Having a way to control the order of how pages are populated onto NUMA nodes is helpful.

David Beckingsale (LLNL) - Umpire: Resource Management for Heterogeneous Memory Hierarchies

- This project seeks to provide a high-level interface for the various types of memory we have to deal with on upcoming machines. Technology-specific APIs force application developers to commit to one implementation. For programming models this can be mitigated with approaches like RAJA and Kokkos and we want to provide a similar interface for memory and execution resources. The Umpire API design will be driven by app needs.
- Some of the questions that came out of the 2016 DOE COE Meeting helped to motivate us to develop Umpire. How can apps and libraries coordinate use of limited memory resources? How can we support flexible allocation strategies for different allocation types (e.g. temporary

arrays)? How can data be moved between places in the memory hierarchy so it will be accessible wherever you are executing?

- We are trying to provide a unified, high-level API that will decouple resource allocations from specific spaces in the machine and provide introspection capability for these allocations, allowing apps and libraries to make decisions based on allocation properties. We do not want to reinvent a wheel; we will design a unified, high-level and app-focused API for projects like tcmalloc, jemalloc, memkind and SICM.
- Coming up with the right concepts and names has been a challenge. A space is a way to abstract a memory location, providing an interface to inspect properties and allocate/free via a strategy. An allocator is a lightweight interface. AllocationStrategy decouples allocations from the area they are made in and operations allow allocations to be moved from one space to another.
- The user interface will be based around allocators (i.e., the user sees an object s/he can allocate or deallocate). All allocators are accessed by querying a central resource manager. Spaces are created based on accessibility of different memory resources. Once a space is constructed it will be tied to a "system" allocator. Operations will allow data movement between spaces; the resource manager will handle all data movement. Our goal is to provide a mechanism for doing these things without specifying policies.
- We will coordinate Umpire with other projects, e.g. RAJA, CHAI, and Sidre. Ultimately, Umpire will underpin Chai and Sidre. Chai gives you a managed-array object that works like a vendor; hidden behind the RAJA call it uses execution policy to move the data to a managed array. When you create managed arrays Umpire allocators will do allocation; actual caching and remembering details about the data will remain within the Chai layer.
- For our initial use cases we set up flexible pools for temporary allocation of GPU data and pass allocators from the app to the library so library data is allocated in the same place. Tomorrow Adam Kunen will discuss how Umpire is being used with RAJA. Our initial implementation will support allocators for CPU and GPU and simple arena allocation. A release is now in process.

Session 3 – Application Experience I (Chair: John Pennycook, Intel)

David Gunter (LANL) - Kokkos Port of CoMD Mini-app for Trinity-class Systems and NVIDIA Pascal Nodes

- We evaluate different programming models, assessing them using proxy apps. We began to look at Kokkos. We used Kokkos and CoMD [a Molecular Dynamics (MD) code]. CoMD is based on SPaSM, a Gordon Bell Prize-winning MD code that is important in materials science (and a simple, nested-loop structure we thought would be easy to port). CoMD is part of ExMatEx, written in C, is portable (in that there is a separately maintained branch for each type of architecture currently available), and can be found in the Github repository.
- The tuned version of CoMD contained 6K LOCs. Fifteen loops to calculate inter-atomic forces and time-stepping were instrumented in parallel for Kokkos. Some parts of the code have parallel reductions (i.e. reduction over four different variables); Kokkos does not do this natively but we created a simple structure to encapsulate the four variables and allowed them to be

summed. We found doing this was not too difficult and the time we spent porting was mostly devoted to our figuring out the need to do things like this.

- We look at single-node performance. We used 256K and 2M atoms across 100 time steps. We have had the most trouble with GPU compilation (and this is an area where we can use some help). Our metrics are wall-time, strong scaling and speed-up. Scaling numbers look reasonable until you get above 64 threads on KNL (which is not a Kokkos issue but we will explore this further). And, it is not a memory issue (because the same thing happens with the 2M atom problem, too).
- There is a false notion that Kokkos is running as well as the OpenMP version until we look at wall-clock time and then find Kokkos is running much slower. We suspect the reason has to do with vectorization optimizations. We see similar results for KNL_quad_flat and KNL_quad_cache. We find it runs faster on Haswell versus KNL but this does not have to do with Kokkos – same code, same vector optimizations, different flags for Haswell versus KNL and the difference between the two is striking. I think some optimization is being hidden from the compiler by Kokkos, and I am not sure why. Nothing has changed with the kernel itself. Estimated speed-up is 7x under Kokkos and 8.6x under OpenMP.
- How will our users deal with using programming models like Kokkos? It is difficult to find documentation for Kokkos' nitty-gritty details. CoMD is relatively easy to port to Kokkos due to the C++-like structure of the code but data structures in CoMD still need much work. We had to create special structures for handling parallel reductions of more than one variable. It is not clear why vectorization performance suffers. Code blocks inside Kokkos lambda functors are not optimized by the compiler to the level of the tuned, OpenMP version.
- ***Comment:*** When you go above 64 threads all you are trying to do is hide latency, but it is likely you do not have much latency.

Ramesh Pankajakshan (LLNL) - SW4Lite: Performance Portability using RAJA

- SW4Lite is a proxy for the SW4 (Seismic Waves 4^{th} order) code . The dominant compute kernel is 125-point stencil evaluation. 90% of the computation is on two kernels. It is written in C++ derived from Fortran. To port to RAJA first I had to change the memory allocator and then do memory marshalling for GPUs (pre-fetching and staging data). I use the RAJA::forallN construct for compute kernels.
- The initial performance was OK on Coral Early Access (EA) nodes but slow on CTS-1 and even slower on KNL. On the slide is seen the triple-nested loop kernel that requires the greatest part of the RAJA implementation. RAJA has a feature that allows permutation of part of the loop and when I did that the numbers for inline and permutation performance improved.
- You have to do pre-fetch if you want to get performance. When you work with triple-nested loops, make sure the stride-one access is on the biggest thread block. Thread-block configuration has many options and they must be chosen at compile time. On the CTS-1 and ATS-1 machines vectorization was the problem; so, be careful where you put the SIMD fragment. Cache and flat mode performed the same.
- When do you stop trying to improve things? Is 1.3x sufficient for using this code? In some respects the answer is how you are funded, meaning how much the code is going to run and

where it is going to run. On the slide (#15) are component runtimes for CTS-1, ATS-1 and Coral EA. In conclusion, loop abstractions are a viable path to performance portability, additional abstractions are required to meet GPU performance, optimizations are lagging behind for lambdas (but will likely catch up), and there is need for an abstract memory model.

- ***Question:*** How long did it take in general?
  If you know the code well then it takes a couple of weeks to work on the loops and the rest of the time is needed to make sure you are caching the data properly. I found bugs in the compiler and expect things will be faster when the machine is stable.
- ***Question:*** You had a reference OpenMP to compare against. When would you have stopped had you not had the reference implementations?
  The ratio of memory bandwidth provides one indication. You have to have an estimate to begin with and its level of sophistication comes from what you have to work with.

Alejandro Vaquero (University of Utah) - Performance Portability Experiments with the Grid C++ Lattice QCD Library

- USQCD and its partners are working on a software stack that will be Exascale-ready under ECP. We want the code to be efficient, flexible and easy to use, and also to be portable without sacrificing performance. A candidate for us is The Grid Library (a data-parallel C++ mathematical object library), which has been adapted for quantum field theories and abstracts away the complexities of the target machine.
- We also want to be able to run on GPUs; for this we need to understand the data layout in QCD (and computer simulated QFTs). QCD is our current theory to describe the strong force (discussed SIMD parallelism in Grid). Grid has an expression templates engine; using C++11 templates we can abstract complex datatypes and treat them as numbers in an expression. This structure is useful for us (and presents challenges with data copy, so unified memory was very useful for us).
- We found porting Grid to GPUs to be quite challenging because C++11 code is strongly based on STL, which is not well supported in GPUs. We tried several approaches including OpenACC/OpenMP, Jitify (just-in-time compilation) and CUDA. On slide #8 is a table showing some kernel code using these approaches. We find that Grid's native layout can be used by the GPU coalesced access, which requires some modifications to make the Expression Templates work properly. Slide #10 shows the code at the lowest level; the following slide (#11) shows the results we got.
- We tried our code across different architectures to test performance portability (results for Xeon, KNL, Quadro Pascal and Tesla Volta are seen on slide #13). We find results of native datatypes almost independent of vector length and this can be improved if we assign several vectors per block.
- Our main difficulties are associated with C++ compilers. OpenACC was easy to port but did not give us the performance we wanted (so far we have stayed with CUDA). Jitify must place kernels and datatypes in header files, which can be cumbersome.
- Another team is working with Kokkos on a Dslash test. They tried a naive Kokkos implementation to see if they could get good performance (although they were not expecting

it). Kokkos on the GPU resulted in good performance. They found that specializing vector types and using some intrinsics gave a major performance boost. They found GPU performance was very good, especially on Volta.

- ***Question*:** Performance was double-precision?
  No, single-precision but what is most important is memory bandwidth. Double-precision would yield the same results (the left axis on slide #11 would be halved).

Brian Ryujin (LLNL) - Experiences Porting a Multiphysics Code to GPUs

- Ares is a massively-parallel, multi-dimensional, multi-physics code with capabilities such as ALE-AMR hydrodynamics, high-order Eulerian hydrodynamics, 3T plasma physics, HE modeling, diffusion, SN radiation and other types of physics. The user can select any number of these capabilities for applications such as ICF modeling, pulsed power, NIF debris and HE experiments.
- Ares is 21 years old with ~800K LOCs and written in C/C++ with 60+ libraries in C, C++ and Fortran. It is used daily by an active user base on our current supercomputers, so we cannot stop production because it is being used every day (while we are continuously asked to develop new capabilities).
- We have ~5K mesh loops with limited hotspots. The Lagrange hydro problem runs 80+ kernels and grey radiation diffusion runs 250+ kernels. The code is mostly bandwidth-bound. We have to maintain a single code base and cannot afford to have multiple versions. We also have to be able to run on all architectures available to us.
- What is the Ares strategy for Sierra? We want to keep strategies simple and leverage existing capabilities and infrastructure, along with keeping concepts that are familiar to developers. Our overarching approach is to use RAJA to get the code to run on GPU (with a CUDA back-end) and use unified memory to get mesh data onto the GPU; we use 1 MPI task per GPU rank and keep all data resident on the GPU.
- We use RAJA for our execution model. RAJA is an abstraction layer for on-node parallelism. We implemented a code-specific layer above RAJA for Ares to improve readability by giving loops additional context and provide an easy place to put hooks in for all our loops. You will hear details more about this on Thursday morning.
- We find the code looks similar after the port to RAJA. Lambda semantics helped the porting process to some degree. There is still a learning curve for developers to change how they are coding but challenges are largely overcome after a week or two of use. Over 98% of our loops could be ported in a straightforward manner because we use the same code on GPU or CPU; a few patterns had to be changed but it was clean for the most part. The remaining loops are serial that need to be reworked for performance (and we did this before when we went to clusters and had to redefine our algorithms). We find RAJA's ability to use multiple back-ends to be very helpful. It is easy to switch between Serial, OpenMP3 and CUDA and we have not seen degradation in performance. We are using CPU thread-analysis tools to track down race conditions we see on the GPU by using the OpenMP3 execution policy and we are waiting to compare OpenMP4.5 with CUDA. We are finding the code benefits directly from performance improvements in RAJA.

- Getting the code to run on the GPUs has been relatively easy and now we need memory management strategies. Ares has ~5K malloc calls that are each wrapped by a macro. For performance we have a three-tiered system. Originally, we used unified memory for everything and it worked - but was slow, so we separate out now what we use the code for (e.g., malloc, cudaMallocManaged (UM) and cudaMalloc (cnmem memory pools). Switching from the naïve single-tier system to a three-tier system gave a 14x speed-up on current hardware. We will switch to Umpire in the near future (we have baseline work now).
- We are seeing good speed-ups on our EA systems (an EA node has 20 Power+ CPU cores, 4 NV P100 GPUs, 16 GB memory). Some packages are more amenable to GPUs. Increasing zone count provides a better speed-up (table with speed-up details seen on slide #9). Slide #10 compares GPU performance with CPU performance for ALE hydro problems.
- With respect to strong-scaling properties of GPUs, we find packing and unpacking the communication layer is taking substantial time and that inhibits strong-scaling performance. But, even with nominal zone counts, we find improvement of 15x.
- RAJA and UM is a viable strategy for some degree of performance portability; over 98% of the loops port clearly. GPU performance is what we'd expect compared to CPU. Kernel launch overhead can become a significant portion of the runtime (>25%) in moderately-sized problems. We are investigating simple techniques for packing and unpacking calls and using threads. We know dealing with large physics libraries will be challenging and GPU strategies for libraries must be compatible. We are changing the way we run code, going from 30K zones/rank to 1M zones/rank).

DAY 2 – Wednesday, August 22, 2017

Session 4: Abstractions, DSL (Chair: David Richards, LLNL)

Ben Bergen (LANL) - The Flexible Computational Science Infrastructure (FleCSI)

- FleCSI is a C++ programming system for developing multi-physics simulation codes. We want to separate concerns with a high-level interface for app development, a mid-level specialization (the intent is that a computational scientist will write this and the app will be implemented on top), low-level building blocks (these comprise the core part of FleCSI), tasking (using notions from Legion – tasks are pure functions and work well to distribute memory communication) and fine-grained threading back-ends. The programming model has control, execution and data models and we offer useful data structure support (e.g. mesh, tree). Our target for next year is to add kernel abstraction in the form of FleCSI runtime abstraction.
- On the screen (slide #10) is a FleCSI program I pulled from a development test. You begin by registering a data client (something that can access the data manager, e.g. mesh or tree specialization) and assign an arbitrary name space, then you add fields (which are registered against data client types), next you define tasks (a task is the basic unit of execution for distributed-memory parallelism) with associated permissions. After I define my tasks I register them with the runtime and define the user driver. I have registered my data client and now I need to get a data client handle. I execute tasks; all distributed-memory parallelism is

embedded in permissions and not explicitly exposed to the user. We find this to be a clean way to handle distributed-memory.

- FleCSALE includes 2D/3D cell-centered Eulerian and Lagrangian solvers and can handle multi-materials. On the slide is an example of a specialization to define interfaces that make sense for an unstructured hydrodynamics code. Mixed-mesh, arbitrary polyhedra can be handled by FleCSALE (showed example of moving mesh calculation on pure Lagrange).
- FleCSPH is a smoothed-particle-hydrodynamics (SPH_ solver built on the FleCSI programming system) and is being used to run simulations of neutron star collisions.

CJ Newburn (Nvidia) - HiHAT: A Way Forward to Performance Portability with Targetable Infrastructure

- We will continue to see heterogeneity within a platform and users will need software architecture and infrastructure to make various components work. The trend towards specialization will continue with respect to host, accelerators, kinds, layers and locations of memory and interconnect. Choices and tradeoffs will need to be made. Meanwhile, we want a single code to be able to run across difference architectures because "same code" + different architectures → efficient performance. We have much differentiation in the stack at the bottom close to the targets and also much differentiation with user interfaces. But, in the middle is a promising area that can be neither target-agnostic nor target-specific and that is the space where HiHAT resides (Heterogeneous Asynchronous Tasking memory abstractions).
- When it comes to language people should be able to use their preferred language inside a task. Let them use their preferred compiler so long as they can provide a function handle. For data layout the user may want tailored-task implementations and the scheduler should be able to choose to re-layout data off the critical path. As long as you can boil down to a common set of primitives then things should work out.
- Portability is in the eye of the beholder. We begin with pluggable implementations on which there is a sequence of target-agnostic primitives to invoke, manage data, move data, coordinate and enumerate. We are not yet performance portable though. Above the sequence of primitives is a scheduler to handle binding and ordering, based on the cost model (which is not part of HiHAT but is built on top of HiHAT) and then the scheduler invokes the primitives where and when most appropriate. At this point we begin to have some performance portability. On slide #12 is seen a notional graphic for a common retargetable software infrastructure.
- Do we really need a scheduler? For many people the answer is no but the answer might be more yes this year than in the past because of increasing lack of predictability, growing complexity and the trend towards asynchrony. In the past we have used bulk-synchronous approaches but that is wasteful when scaling up substantially, particularly on a heterogeneous platform.
- HiHAT is about APIs for retargetability. We have some initial prototypes now and recognize that for HiHAT to be adopted it will have to be the easiest and best solution, and be open with respect to architectures. Nvidia, as an example, has half a dozen tasking frameworks and by using HiHAT we are looking at how we can share infrastructure. Yesterday I described an

experience using HiHAT to map between CUDA and HiHAT for molecular orbitals. We were able to accelerate exploration of the application space.

- Portability comes at the scheduling layer, on top of target-agnostic primitives. By using dynamic scheduling we have the most promising path to portability and scaling and the HiHAT prototype looks promising as a retargetable infrastructure.

Adam Kunen (LLNL) - Recent Work with RAJA, and a Nested-Loop Update

- I work on a LLNL transport code (ARDRA) and have contributed to RAJA for nested-loop abstractions and atomics. RAJA is a C++ abstraction layer that enables portability. We want to balance code performance and developer productivity. The goal is for the developer to only have to write a kernel once. RAJA is responsible for executing loops and dealing with performance issues on-node. Apps choose execution policies and select loop iteration patterns. RAJA development is driven by ATDM/ASC requirements.
- We are trying to abstract the for-loop such that the loop body becomes a lambda but is mostly unchanged. We want a have a light touch and require a low barrier to entry. LLNL maintains ~6M LOCs and we want to have the lowest possible impact on the codes but also enable them to find as much parallelism as possible. We have an application-design philosophy.
- Since the 2016 Performance Portability Workshop we have reorganized the RAJA source code to make it clearer, refined APIs to make them more user-friendly, added back-ends for OpenMP4.x, OpenACC and TBB, added parallel scans, re-factored how index sets are implemented, added execution policy to enable features such as dynamic dispatch, added hooks for integration with Chai (and we will add more hooks for other libraries), built the RAJA performance suite to help guide compiler NRE work, and expanded and refined nested loops.
- We identified performance issues in nested loop abstractions. On the slide is a triply-nested loop we are trying to extract. Right now RAJA provides a pseudo-code that peels a policy to execute a loop and binds the loop index to the lambda, wrapping the lambda with another layer. We had to capture the original loop body by value so in the inner most loop there are as many loop body interactions (equal to the number of loop iterations) – this is known as "peel and bind" and was a straightforward way to design RAJA. Early on we saw no performance issue with Kripke but it was because the compiler was hiding our performance inefficiency. Our (new) reengineered nested loop execution invokes the loop body with a tuple of indices, leading to (O)1 copy constructions of the loop body and improving performance.
- We have much work going on in RAJA right now, and RAJA is available on Github: https://github.com/LLNL/RAJA. We ask you to tell about your experience using it.
- ***Question:*** Experience with LLVM?
  It has gone well; we use it a lot, especially on the Sierra EA machines and do not seem to have issues.
- ***Question:*** Is there a user manual, programming guide for RAJA?
  No and that is a gap we need to fill.
- ***Question:*** Is the tuple work hidden behind the user interface?
  Yes, you do not have to use it directly. It happens directly. But if you need a tuple it is available to use.

- ***Question:*** Can you say anything about compile times with and without the model?
  We do not see any difference in compile times. Ardra is 250K LOCs and takes ~1.5 minutes to compile (longer if using nvcc).

Carter Edwards (Sandia) - Kokkos Task-DAG Parallel Capabilities & Evolution of Kokkos Back-ends

- Seven years ago we began to work on performance portability to the GPU and CPU back-end. We want to expand our back-ends and maintain them. Key to doing this is mapping to the best programming mechanism the back-end provides (e.g. CUDA, Intel) as abstractions and programming mechanisms evolve.
- Previously we worked only with data parallel patterns/policies whereas our present work is focused on new directed-acyclic graph of tasks parallel patterns/policies (this is being done via a Sandia LDRD research effort). Tasks can be heterogeneous (need not be single, same-function body) and the DAG is dynamic.
- For Kokkos back-ends we are now using OpenMP for CPU and KNL+ and will soon require at least OpenMP4+ so we can use those features. We want to be able to map to the best back-end. We are also working on a strategy to interoperate between AMT (Uintah style) and then have heavyweight tasks calling Kokkos underneath.
- We will move to CUDA 9+ for the NV GPU and will plan to jettison CUDA 7 and CUDA 8 as our new machines come up to speed. We want to maintain a thread back-end so we can compare to OpenMP to ensure OpenMP back-ends remain performant. We will work on a backend for Arm and for AMD (ROCm compiler) and an OpenMP4.5 offload to GPU.
- The new pattern is Directed Acyclic Graph (DAG) of tasks, i.e. parallel execution of computations (tasks) that have "execute after" dependencies. With dynamic and heterogeneous Task-DAG the DAG changes constantly as the program executes and completely different functions execute - but there is much associated overhead. We are presently working on a "Work-DAG" use case based on the Tycho2 application (neutral particle transport via sweeps): https://github.com/lanl/tycho2 and just got it running on the KNL. We will next work on data structures. This is an exciting mini-app that we have running now. (Discussed details of approach).
- This work was a good target for LDRD funding several years ago. We have a dynamic Task DAG with heterogeneous functions for which we need scalable dynamic allocation of tasks with low-latency scheduling and dynamic creation/completion of execute-after dependencies. There are GPU constraints we had to overcome, e.g. non-blocking tasks had to be forced into a reconceptualization to "re-spawn", changes to the memory pool and dealing with non-coherent L1 caches. Details about the scalable memory pool and task scheduler are found on the slide (#10).
- Our initial Task-DAG capability is working and is portable to CPU and GPU architectures. Our initial Work-DAG capability is also portable to CPU and GPU architectures.

Abstractions, DSL Panel Discussion - David Richards (LLNL), Discussion Moderator

- ***Question:*** FleCSI *et al* were focused on writing code once. Are you committed to this or do you believe you can have multiple implementations of a single kernel for different devices?
  Carter – When we map kernels onto hardware the mappings are different, so you could call that a different implementation with the same source base. Sometimes that is not sufficient - but for the most part the hope is most kernel device-mapping is sufficient with a handful of kernels that need to be specialized, usually due to algorithmic differences (which is the user's domain). We have closures and classes we template on the back-end and then you (the user) can partially-specialize.
  Kunen – You want to write the kernel once; that is a trade-off you are looking for when developing large codes. You will likely end up rewriting algorithms to make them more efficient in general but in many cases you can reach a happy medium with acceptable performance everywhere with one rewrite.
  Bergen – Ideally, I would like a structure to support the idea of needing to have different kernels written in the same language with a mechanism for choosing them. We are looking at developing our own fine-grained parallel model and I would like to adhere to the principle that the user can go around that when needed. If you have a machine on the floor that has to run fast then you do what you have to do.
- ***Question:*** How does each of your system fail?
  Kunen – You are at the whim of whatever back-end you choose. If you fail at a GPU then you only have what the GPU offers. At the front-end, when writing CPU code, you leave it up to the user.
  Carter – In Kokkos, if you try to allocate memory where there is none then it will tell you memory is not available. If you ask for a shared memory resource that is not available then the kernel will tell you that you asked for too much and will have to do something different in your algorithm. For cases of algorithmic failure we advise you to make changes to the algorithm.
  Newburn – If you run out of a resource like memory there are opportunities to ask the system what is available so you can work within that budget. We want to develop a notion of firing something asynchronously and then work in the completion framework to find out whether it succeeded, and if it did not, there would be something that would trigger a retry. We need to find methods to work those things into our systems when we cannot pre-manage to stay within the budget. I think there is promise for asynchronous frameworks to be able to deal with that.
- ***Question:*** What has been your experience pushing task-parallel paradigms to domain scientists?
  Kunen – We do not really have a task-parallel approach in RAJA. We have task-based-looking efforts but LLNL has had little adoption of task-based models.
  Carter – We have conceptually partitioned the task-programing model into micro-tasks and macro-tasks. Uintah has macro-tasks - but the tasks do not migrate, so you have to determine whether you are running a DAG that moves data. So, the answer is that it depends … with micro-tasks the barrier is low, as it is with non-migratable tasks, whereas at higher levels Legion and HiHAT worry about data migration (as Charm++ has for years).
  Bergen – So far, so good. I think we could run into some problems in the future. With FleCSI we

have MPI interoperability, so data can be written into a task to access MPI calls but if you want FleCSI to handle distributed memory communication then you have to address that in the runtime and people have not had problems adopting that.

- ***Question:*** For the code Ben showed, the task seems to be a large and chunky loop, and that is easy to accept from a developmental perspective. The HiHAT task is agnostic but still big and chunky. How do you get wide graphs when the parallelism is in the task? (And you need wide graphs for parallelism).
  Newburn – If you want to be relevant to graph analytics, you cannot have large and chunky tasks.
  Bergen – This is a problem with the definition of task. I would call a task as being defined as to whether it is a pure function or not and call what Carter is designating a macro-task as a kernel. You can do fine-grained tasks with Legion. The goal would be to get low overhead so you can have as fine-grained task as you want.
- ***Comment:*** Legion, Kokkos and RAJA have all implemented their own scheduling algorithms.
  Carter – Kokkos is all on-node (whereas Legion and HiHAT are between nodes).
  Newburn – Legion is built on top of Realm, which is built on top of HiHAT (and Kokkos is potentially built on top of HiHAT). The scheduler I described is actually above the HiHAT layers.
- ***Question:*** Is Intel interested in HiHAT?
  Newburn – Intel needs to decide what they will do. Intel and other companies have not given formal endorsements. We have talked with Arm, IBM, Intel and others and they are watching with interest. Ultimately, apps and runtime need to decide what they are going to do. For a CPU we ported from x86 to Power to Arm and it worked in a number of minutes.
  Bergen – If the companies try to kill it (as NV tried to kill OpenCL) then it is done. If it is at a low-enough level - and everyone agrees to use it – then it could solve some problems, but that only happens if all the vendors agree.
- ***Question:*** When operating in async mode what is a state snapshot you can take for either analysis or checkpoint/restart?
  Bergen – We are talking about formats to use for checkpoint/restart with FleCSI.
  Dubey – As some point you have to interrupt the DAG to have a consistent state.
  Bergen – A task with requirements on data you want effectively does that, getting access to data in a consistent way.
  Dubey – If you have a high-cadence output requirement for analysis purposes then you have another challenge.
  Bergen – I see that as opportunity for task-parallel runtimes because it is easier to do things like *in-situ* analysis to generate output data.
  Dubey – I want to use your framework and am not doing *in-situ* analysis.
  Bergen – The data and arguments are immutable within the task and you are free to write that. There is no issue with doing that.
  Ferenbaugh – You can also make your output to the task run as soon as the data is ready, in parallel with other tasks.

Session 5: Application Experience 2 (Chair: Charles Ferenbaugh, LANL)

Micah Howard (Sandia) - Performance Portability in SPARC – Sandia's Hypersonic CFD Code for Next-Generation Platforms

- SPARC is a compressible CFD code using hybrid structured-unstructured finite volume (FV) methods. We use it for research on high-order unstructured discontinuous collocation element methods. It has perfect and thermos-chemic non-equilibrium gas models, along with RANS and hybrid RANS-LES turbulence models. Enabling technologies include scalable solvers, embedded geometry, meshing, UQ and model calibration. In FY18 we are performing a V&V study on the code in partnership with DOD and a university partner. SPARC is built on Kokkos. Two years ago we began with a stripped down MPI-only FV code, brought in Kokkos and built from there as our avenue towards achieving performance portability.
- With heterogeneous architectures the question is what to do with C++ virtual functions. We have a dispatcher to convert runtime polymorphism to compile-time polymorphism in a continuous way so we can dispatch functions statistically as well as dynamically. We use a runtime-to-compile-time chain (rt2ct) so we can chain different instantiations of physics models for efficient execution on a GPU. In FY16 and FY17 we converted structured grid discretization to a threaded assembly by introducing a MeshTraverserKernel, allowing a physics code to operate on a structured block. Details showing what we do for a CPU or KNL system compared to what we do on a GPU are seen on slide #7. As a result of our FY16 work, the structured grid implementation of SPARC is running, end-to-end (equation assembly+ solver) on the GPU. In FY17 our work focused on the unstructured grid implementation of SPARC.
- SPARC is running on all of the current Sandia testbeds. The table on slide #9 shows the end-to-end speed-up we saw for SPARC versus Sierra/Aero; SPARC runs ~2x faster with better parallel efficiency compared to Sierra Aero. And, as of the end of FY17, our unstructured grid implementation of SPARC is performing slightly better than the structured grid implementation.
- Strong-scaling analysis of SPARC is seen on the following three slides (#10-#12). Threaded KNL is ~1.5 faster than MPI KNL, so threading is important and provides pay-off. Performance on Haswell and Broadwell is ~1.5x faster than threaded KNL (this is a common thing we have found). We hope to get some of the improvement back with SIMD vectorization. With Pascal we get good performance with KNL assembly. The halo exchange on the GPU is not good but we understand it is going to be fixed. Solves on the threaded KNL is ~2x faster than Haswell or Broadwell. GPU-based solves are not seen (yet) on the graphs because that work is still underway. SPARC weak scaling shows results similar to SPARC strong scaling.
- Kokkos has helped us develop a performant, portable code and faster code has resulted from our focus on PP. There are some negatives: first is DevOps challenges (associated with building the codes and dependent libraries on different architectures, running it, testing it), performance analysis challenge (collecting meaningful performance data on each architecture), and we have found that developing for the GPU in general has been difficult and takes substantially more time than completing clean and incremental builds on CPU.

Scott Parker (ANL) - Portability and Performance of the Nekbone Mini-App

- We are looking at performance we can achieve with the Nekbone mini-app on different architectures. NEK5000 is a spectral element CDF solver and provides the basis for Nekbone. It is written mostly in F77, with some routines written in C and MPI for parallelization. It is available and is open source. Nekbone was derived from NEK5000 (which is impractical to use for exploratory investigation of performance of programming models). Nekbone represents significant kernels in NEK5000 and is used in DOE machine acquisitions (e.g. CORAL) and Exascale co-design activities as well as in DOE FastForward and DesignForward programs.
- We have looked at Nekbone performance in some detail to identify on a function-by-function basis the operations in each function (e.g. memory-bandwidth intensive, floating-point intensive, network-intensive, memory-operation intensive and wrapper routines).
- We configured the code to run in a single-node-only configuration, so we turned off network-intensive operations. We defined a Nekbone-compute performance model (details seen on slide #6) and looked at performance on Xeon Phi KNL and NV Tesla P100. Key for us is peak floating-point rate and Peak DGEMM rate. We use the STREAM benchmark to characterize memory bandwidth.
- Slide #8 shows solver time for different problem sizes on the KNL (both single-core and running on all 64 cores), along with different code optimization levels. With single-core results, the time to solution increases linearly with problem size (this is not the case for the 64-cores where time to solution does not increase with problem size until the cores are saturated). Overall, on the KNL, the performance analysis is seen in the table found on slide #9. We find memory-bandwidth intensive kernels obtain 64-95% of STREAM and compute-intensive kernels obtain 40%-51% of peak DGEMM rate. I think we understand performance on KNL and are close to the upper bound.
- Next we look at what we can do on the GPU, using OpenACC. We had to make changes to the code, some simple, some what we consider to be "moderately complex" and some complex (details of required changes found on slide (#10). Nekbone performance on the K100 and KNL is seen on slide #11. There is room for improvement in GPU optimization; we expect to ultimately exceed performance seen using optimized code on the CPU. We find memory-bandwidth-intensive kernels obtain 89-95% of STREAM (better than KNL) while compute-optimize kernels do worse (which is not a surprise). Kernel efficiencies for Nekbone performance on KNL and P100 are comparable for un-optimized kernels. Higher small-matrix-multiply efficiencies require architecture-specific routines. The lowest time to solution was seen on KNL with optimized matrix multiply. With respect to portability – we can have single source portability using OpenACC recognizing we will get poor performance on both architectures because the matrix multiply operations will not perform well.
- ***Question:*** For the more complex changes, did you test the effect of those changes on other architectures?
  Yes and the results are effectively the same. The change was primarily complex because of the implementation of the NEK code (where the devil is in the details based on historic decisions made by its developers years ago).

Brian Friesen (LBNL) - Performance Portability Experiences at NERSC

- We attempted to implement OpenMP 4.x and Kokkos in three codes at NERSC: (1) BoxLib (a large code, now retired), (2) BerkeleyGW and (3) Dslash. So far the results have been underwhelming. We struggle with the portable part (recognizing that performance will not come for some time).
- For OpenMP 4.x target directives, we have five compilers on three systems and have one result. We found the results vary widely with the compiler. With OpenMP we would like to see a direction when there is a target construct but there is no device.
- Kokkos requires a memory model for a performant-portable framework and that can cause a problem if the existing code already has one.
- For BoxLib we only tried to port four kernels. Most floating-point work is done in Fortran. Kokkos and BoxLib is not working especially well because BoxLib has a data infrastructure that we had to rewrite in order to run Kokkos.
- For the BerkeleyGW kernel (this is a material science code written in pure Fortran), we rewrote the GPP kernels in C++ because we wanted to experiment with Kokkos. There are no complicated data structures in BerkeleyGW (BGW) and it does actually run.
- Some of what we found when running OpenMP in BoxLib and BGW is described in slides #14-#15. So far, Cray does not compile anything, so we have no Cray numbers to report to you. We also have no IBM numbers to share.
- We are reporting all of the bugs as we find them and they are being fixed. We do have some numbers for BGW. The compilers are getting to where we need them to be but there is still a long way to go. Performanceportability.org is a website we have established to document our experiences.

Arnold Tharrington (ORNL) - Portability Initiatives for Scientific Computing and Simulation: Molecular Dynamics as a Case Study

- We sought to develop a performance-portable MD library. Targeted platforms include CUDA HPC architectures. To me, a performance-portable algorithm achieves acceptable performance across a variety of HPC architectures and would require only minor modifications when porting to novel architectures. Design characteristics include a consistent, unified front-end interface. Long-range electrostatic interactions forces and short-range 2-body (pair-wise) forces are the components of the non-bonded-forces calculations (which comprise the bottleneck).
- What can I do with given resources (and one post doc)? First I profile the code as close to production status as possible because using a microkernel risks missing things. You have to identify computational bottlenecks, dominant data structures, and determine whether the application is suitable for acceleration. We had non-existing code and chose to write our libraries in C and C++. On the back-end I use simple, flat arrays (nothing exotic).
- For each domain I need charges and position. We are trying to replicate the ease of use in a manner like the FFTW library. My solver has one main step which is a large-grid calculation. My direct kernel has a virtual class that specifies what I want to do with my calculation, along with a wrapper between my algorithm and my device. I keep the device separate from the implementation. The accelerator device wrapper is actually an API between the CPU and

accelerator device. The Accelerator device is a C++ class allocated on the device which performs the computation on the HPC compute device. The accelerator device abstracts the programming model and is made up of hand-written CUDA kernels. I learned that it is important to separate the algorithm interface from its implementation. If coding goals and resources permit me to do so I will try using an accelerated library or OpenACC (other alternatives include Kokkos, RAJA).

Session 6: Off-Node Hierarchical Memory and I/O (Chair: Charles Ferenbaugh, LANL)

Elsa Gonsiorowski (LLNL) - SCR and Preparing for Burst Buffers

- SCR is a scalable checkpoint/restart library. I will discuss Burst Buffer (BB) technologies and an overview of SCR. The machine-global BB is now on Trinity and provides storage close to all of the compute nodes in contrast to the node-local model of BB we will see with Sierra (which is only visible from compute tasks on that node).
- Using the BB relies on integration with the resource scheduler and is different for machine-global versus node-local storage. For checkpoint/restart you might want to pre-stage the last checkpoint, create checkpoints throughout your job, and then flush checkpoints when the job completes. You want to periodically create check-points; checkpoint/restart is also known as "defensive I/O" and is related to the size of system memory. And, checkpoint/restart depends on the resiliency of the machine, which can change over time.
- SCR is a scalable checkpoint/restart library designed to enable apps to do checkpointing and take advantage of system-storage hierarchies. SCR manages efficient file movement between storage layers (writing first to a local RAM disk and then SCR moves it to local file system when most efficient). SCR also manages data redundancy that is helpful in the event of a node failure.
- SCR has a C library (and also C++ and Fortran interfaces) that the app redirects its I/O through, front-end scripts to do prefetching and post-stage scavenging and configurations for SCR that can be changed with machine or user change.
- The SCR back-end library redirects app files and supports both synchronous and asynchronous flush operations (with hardware-specific capabilities). It supports both checkpoint and output (e.g. vis) data. (Described details associated with SCR back-end and front-end scripts – see slides #9-#12 for details). SCR configuration files define levels of available hierarchy, modes/groups of failure (e.g. two nodes on the same power switch) and checkpointing and data residency needs. If you know details about your location and power groups and want to be resilient then you can provide details to SCR.
- Pre-staging for machine-global (Trinity-style) is fairly simple. Node-local requires new software, close integration with the scheduler, and is most useful for DATs or half+ system jobs. The post-stage scenario is similar for machine-global and node-local BBs and can take advantage of vendor API asynchronous approaches.
- Unaddressed concerns at present include applications without checkpointing, shared files and arbitrary data movement (e.g. machine learning).

- Our VELOC effort combines FTI and SCR codes (FRI is a variable-based checkpointing scheme) and will support existing FTI and SCR apps. If you want to write shared files across the node-local BB the UnifyCR project presents a shared-namespace across the node-local BB via an I/O interception layer.
- Moving a large number of files around is done via MPI File Utils. With machine learning when you want to put files on every node-local BB there are useful tools. There is a team of four of us at LLNL on the SCR team and we are looking for applications to work with. We are working with the Mercury team to prepare for Trinity and I believe we have successfully run on Trinitite.
- ***Question:*** Portability in terms of the underlying batch scheduler – is SLURM required?
  SCR presently supports APRun, SLURM, and we are working on LSF as well. As part of the ECP project we will look at taking out the MPI layer and look at the ANL Mercury framework.

Francois Tessier (ANL) - Toward portable I/O performance by leveraging system abstractions of deep memory and interconnect hierarchies

- TAPIOCA is an optimized data-movement library based on the two-phase I/O scheme for topology-aware data aggregation at scale on BG/Q (with GPFS) and Cray XC40 (with Lustre); it features topology-aware aggregator placement, taking into account architecture topology and data-access patterns. We capture the data model and data layout to optimize I/O scheduling, facilitate pipelining, and provide abstractions for interconnect topology.
- Topology-aware aggregator placement is based on quantitative information gathered from compute nodes, e.g. latency, bandwidth, and distance between node (equations for placement analysis are found on slide). The next slide shows results obtained with TAPIOCA on the two target architectures (BG/Q and Cray XC40); we find TAPIOCA outperformed MPI I/O on the I/O kernel of HACC.
- We want to extend this I/O library towards a generic data movement library for data-intensive applications exploiting deep memory. We are working on abstractions for memory and storage and have some open questions associated with the blurring boundary between memory and storage, how to express the need for one (or more) process(es) at destinations for data movement, scope of memory/storage tiers (PFS versus node-local SSD) and data partitioning to take advantage of fast memories with smaller capabilities.
- We are working towards performance portability on Mira and Theta with the same app code running on both platforms and the same optimization algorithms, using an interconnect abstraction. We have seen promising preliminary results with memory/storage abstractions and are working to come up with a model to help make smart decisions about data movement.
- ***Question:*** Have you thought how this might work on a GPU machine? Would you need to move all data to the CPU to store it?
  That is a good question. We have not yet considered GPUs. I am not sure it would fit into our abstraction because we cannot consider GPUs as another kind of memory. It is something for us to think about but we do not have much GPU hardware to work with, so it is not currently a pressing concern.
- ***Question:*** What about working with HDF5?
  We would like to work on an HDF5 driver.

Session 7 - Languages, Compilers, Frameworks, Tools (Chair: CJ Newburn, Nvidia)

Carlo Bertolli (IBM) - Performance Analysis and Optimizations for Lambda-based Applications in OpenMP 4.5

- We receive reports about how our OpenPower compiler is working. We take the reports and look at how a generic code runs to understand why we are getting this performance. We went into LULESH 2.0 that runs on top of RAJA and ran it through our compiler, looked at the compiled code and compared it to our generic code performance. The notional graphic on slide #3 shows a generic lambda-based framework with OpenMP and lambdas on the host. The main function does not contain OpenMP. You are capturing variables used within the loop body. This is capture by copy. You extract the three pointers and the loop executes on the pointers.
- If you want to go on a device there is mapping of the three pointers from the host to the device and the compiler creates a lambda struct to map the three pointers onto the GPU. The compiler issues a set of statements to force the OpenMP to translate the three pointers. This will become part of the OpenMP standard for OpenMP 5.
- Looking at some simple examples for using a lambda structure for the simplest case (vector add) we compare #parallel for with and without lambda. We see a difference with the small vector-add problem size (we do not see the difference with large problem sizes). Looking at the same example on the GPU we again see difference only at smaller sizes and that disappears with large iteration space size. The code generated is identical, except the lambda version has to retrieve pointers from the struct.
- Next we looked at LULESH 2.0 using RAJA. We used four versions of the code: host versions with plain OpenMP parallel for, RAJA with domain, RAJA with direct-array access and device versions with plain OpenMP target region, RAJA with array capturing. For the OpenMP target implementation we modified LULESH to access domain arrays from within the capture expression. At small iteration sizes, missing vectorization shows significant slowdowns whereas at large iteration sizes the difference is within 10% for most kernels. The only kernel that is not performing is CalcLangrangeElements.

Jeff Larkin (Nvidia) - Early Results of OpenMP 4.5 Portability on NVIDIA GPUs

- Since the 2016 Workshop several OpenMP implementations for Nvidia GPUs have emerged or matured. Can I take the same code between compilers, run them on GPUs and expect reasonable performance? I will show results for CLANG, Cray, GCC and XL – all running on the same GPUs (for Cray on XC40 and for the others on P100).
- OpenMP in Clang has improved significantly since the last time I made a similar presentation. Cray's OpenMP 4.x compiler was the first to market for NV GPUs (but does not adhere to OpenMP as strictly as others). GCC is the *de facto* open-source compiler with support for OpenMP offloading to NV GPUs. It is a mature compiler for the CPU - but the GPU still needs work. XL is IBM's compiler suite on Power and now has support for offloading to GPUs.
- I took a simple app kernel (Jacobi iteration) we use for teaching. On the slide is the code I tested with. We move data once. GCC performance is bad compared to the others. When I put SIMD on the inner-most loop I find improvements with GCC and Cray.

- Currently our distributed and work-shared parallelism comes from the same loop. We could collapse them together or we could move the parallel to the inner loop. The next slide shows the code for the collapse clause – I find adding SIMD does not help either the GCC or the XL case. I would say Clang and Cray show similar performance (meaning there is fairly good portability with the collapsed code). XL lags somewhat (but not too far) behind. I can split directives and only use SIMD parallelism on the inner loop. GCC was far behind without SIMD but catches up somewhat when SIMD is added. Clang and Cray again look quite portable; XL does not like this parallelization scheme.
- Many users want to fall back to the CPUs – how well can I run one set of directives for both the host and device - is this possible? Using the "if" clause forces the compiler to build both CPU and GPU versions of the code and then select which type of processor to run on at runtime. When we look at team version versus collapse version versus split version for each of the four compilers, we see results as shown on the slide. Cray gets ~33% performance with all three approaches (because there is a limit of one thread). XL did a good job on the split (by adding one "if" clause I can run effectively on CPU or GPU) and with the teams versions - but the collapse version did not run.
- In summary, there are compilers you can use in production. The SIMD support/requirement is inconsistent. Whether the OpenMP target code will be portable with the host is compiler-dependent (the answer is yes for XL, with a fairly good possibility for Clang whereas GCC and Cray did poorly).

Max Katz (Nvidia) - Adaptive Mesh Refinement for Exascale

- I support Sierra deployment and also work with the AMReX team at LBNL. AMR-enable science includes astrophysics and cosmology, combustion, accelerators and multiphase flow. AMReX does block-structured AMR refinement so you can zero in on areas of interest. Grids can be dynamically adjusted at runtime and data can be cell-centered mesh data as well as zone or corner centered.
- For on-node CPU performance we use vectorization, code transformation and auto-tuning but here, too, what works well on CPUs does not necessarily work well on GPUs.
- Logical tiling is used to improve serial, parallel performance. You want to expose enough fine-grained parallelism to keep the threads busy - but you do not want to have to deal with overhead associated with keeping regions open. You can make the tiles as small as you want and have quite a number to work the threads you need. An MFIterator is configured to loop over grids or tiles (and could do particles instead).
- For our compressible hydrodynamics code we developed StarLord as a min-app version and measured performance with a standard figure of merit. We ran on KNL with OpenMP threading over the node and the results (with varying numbers of threads per core) are seen on slide #8.
- For the WarpX electromagnetic PIC code being ported to AMReX the results are seen on the next slide. For a GPU strategy we use unified memory and C++ iterators to hide complexities of data motion and keep data resident as long as possible to avoid transfer costs. We will also explore OpenMP and OpenACC for offloading.

- My results for running the app on P8 and on P100 on SUMMITDEV are seen on the slide (#12). I hope this type of approach can be used in future procurements (because these issues are not only associated with astrophysics codes). AMReX codes are available on Github.

Xiao-Yong Jim (ANL) - Performance portability via Nim metaprogramming

- You can push complexity to the compiler, low-level libraries or the user, or use a different version of the source code for each device. On slide #4 is the code for the micro benchmark we use. First we ran on CPU threads to set the diagonal elements of the matrix and then we run the statement block on the GPU and again on CPU. The code handles memory management. A CPU thread takes a block of code statement and wraps it in a function with lexically-scoped thread-local objects. References to variables outside the code block are managed by Nim; a custom iterator over the array indices takes care of actual data parallel operations.
- Why are we doing this? Our domain is lattice-gauge theory via a large 4D (5D) grid. Nim is a modern language (developed in 2008) that is "efficient, expressive and elegant". It generates C or C++ code and compiles with any compiler. It has an integrated-build system; you copy the main program, modify and compile. It is available on nim.org
- Why use Nim? It can be used at both the low-level and high-level; it can be used to manually manage memory and also can be used as a high-level wrapper. Nim has templates, generics, macros and AST-based overloading.
- We have begun to develop a data-parallel library for small tensor objects on the lattice (QEX – Quantum Expressions), and it is available on Github. It supports arrays on both CPUs and GPUs now.
- Slide #15 shows performance results on Tesla P100 and 2x Xeon; when the memory footprint is small, there is not enough work. Slide #16 shows the same code compiled only for the CPU (and everything runs). When the memory footprint is small not enough threads are working. One can see the impact of saturating the caches and the DRAM bandwidth.
- Nim metaprogramming helps to hide architectural differences under a unified data-parallel API. A toy benchmark saturates GPU bandwidth. We are considering an API to use both the CPU and GPU in a heterogeneous setting. There are possibilities for using AST transformations.

Dmitry Lyakh (ORNL) - Portable Heterogeneous High Performance Computing via Domain-Specific Virtualization

- Currently I am working on the ExaTENSOR framework. Basic Tensor algebra equations are seen on the next few slides; this is my use case. Slide #11 shows a notional graphic of the domain-specific virtual processor architecture. We have multiple virtual processors (DSVPs) to hide hardware specifics – one for node level and one for network level. We also want to encapsulate the HPC scale, so we define V0 as a virtual system processor, followed by additional DSVPs at levels below v0 (and the number of rows of virtual processors grows based on the size of the system). To build DSVP we use domain-specific microcode, resource allocation primitives, data transfer primitives, data decomposition/aggregation methods and virtual architecture specification.

Carter Edwards (SNL) - High Performance Atomics are Critical for Thread Scalability

- What do we do about increasing thread counts? We have issues now with GPU architectures. We want to enable on-node parallel algorithms and this can be challenging. We need to keep changes to the code to a relative minimum and keep initial algorithms similar to what we have in the serial world, so there is an incremental path forward. Atomic operations are an application enabler to keep (roughly) serial algorithms. A disadvantage is floating-point-rounding differences because floating point operations are not associative and variations in runtimes if/when contention rates change between runs and atomic operations can be expensive.
- Alternatives to using atomic operations include development of new algorithms (e.g. coloring on well-behaved structured grid) - but there is not enough parallelism to support coloring schemes and this can lead to significant code churn.
- C++11 introduced atomic memory updates into the standard but std::atomic is clunky and requires a special data type to be declared and – with a large matrix - can become intractable.
- In Kokkos we wrapped atomic overloads so atomic operations can be applied to ordinary data types, making atomics simpler to use. We identified three categories for atomic-issue rate and contention levels (defined as the likelihood of multiple threads hitting the memory location and doing atomics at the same time): histogram, MD and matrix assembly. On our current testbeds we look at GigaUpdates per second, the ratio of using atomics to standard memory operations, running in the "best configuration" and the ratio of using non-atomics to atomics. Looking at results we are excited about what the GPUs are doing for us using atomic units (although we are not excited about what the CPUs are doing).
- We are concerned about high-thread counts on non-GPU architectures going forward. Nvidia has done something right on the Pascal (P100) GPUs with hardware enablement of the cache. CPUs have historically struggled with fast atomic updates because they add a significant number of additional operations.
- Most algorithms have relatively low-content rates. Atomics are used to enable correctness. Atomic memory operations are potentially a lightweight programming choice but we need better hardware support for atomics.

Gheorghe-Teodor Bercea (IBM) - Towards a portable OpenMP data sharing implementation for NVIDIA accelerators in the CLANG/LLVM toolchain

- We want to make a more "upstream-able" data sharing scheme for CLANG/LLVM trunk. We look at mapping OpenMP to GPUs in a simple problem and find that on GPUs threads cannot share a variable located on the local stack. We have the master region set up the function point for the workers. To share across functions we have to maintain a list of references to our shared variables. The master region initializes the list and the worker region retrieves the list. Most allocations are allocated to device-agnostic lower-level structures.
- There are four alternative ways to lower a shared variable and today I will talk about a scheme for lowering alloca to shared memory. We store the variable in a shared-memory stack and there is a limit to how much shared memory we have. If we use no sharing then the c variable will be allocated to threads and cannot be shared. But if we allocate c in shared memory, and

have the runtime have a list of shared arguments then all threads can have access. To make this possible we have to change the address space, using some tricks by introducing a new LLVM-IR pass. We add a new pass to the NVPTX that will lower the frame index of shared values.

- In addition to the shared memory scheme, we have to ensure the runtime is able to construct the list for the master and the worker and make sure that we change the back-end to lower the variables to the appropriate address space. There are limitations that need to be addressed in future work. We need to come up with a smart way to identify shared variables and figure out how to communicate the information to LLVM back-ends. We need a back-up scheme that supports all OpenMP cases (even those that use a number of shared variables as well as sharing between parallel and SIMD regions and other use cases). We are working with the CLANG and LLVM communities to get this to trunk as fast as possible.

Languages, Compilers, Tools Panel Discussion - CJ Newburn (NVIDIA), Discussion Moderator

- ***Question:*** We had several talks on OpenMP in this section and throughout the workshop. What things remain to be investigated?
  Bertolli – We know performance portability – as far as OpenMP is concerned – will be about selecting the proper pragma for the specific target.
  Katz – You can tune so much for magic black-box compilers that you can un-tune for other architectures. We are looking at ways to provide per-target customizations that can be selected at runtimes.
  Bercea – I think portability is presently focused on data-parallel examples but there are also cases that have hidden problems such as data sharing that must be considered because there is impact on performance portability.
  Katz – We have not touched on tasking or asynchronous behavior in general, which is challenging to get correct and for the compiler to map the task graph appropriately depending on the architecture.
- ***Question:*** On a KNL I was concerned that if I want to divide ~60 cores among 4 different tasks and want the tasks to execute concurrently and if it takes time to submit something then ¼ of the threads are waiting while the serial action of managing the tasks happens. I would favor being able to have a way to specify what resources are used and that will help with heterogeneous architectures. We had an example of a primitive used (atomics). Other than atomics, what do we need help with?
  Edwards – Synchronization, as needed for internal data parallelism.
  Newburn – Should that stop at primitives within a single target or span targets?
  Edwards – In CUDA 9 it is codified with group collectives. We need atomics and collectives.
  Katz – Being able to allocate memory per thread efficiently – right now we have to re-factor everything. (I know it is a hard problem.)
- ***Question:*** Jeff noted last year that descriptive beats prescriptive. Are we getting there?
  Larkin – At the OpenMP face-to-face meeting a month from now we will discuss that. Today all directives have specific meanings and performance depends on which ones you use. We would like to emulate OpenACC and have the compiler make the first round of decisions. We will continue to work on getting that into OpenMP. I think the collapse data I showed was so good partly because the compiler has complete freedom (but one is not always able to write code that way).

Bertolli – Right now compiler code generation is not something people like. I support this but we need to keep an eye on those users.

- ***Question:*** In the CLANG compiler, do the error reports indicate what the compiler does not know?
  Bertolli – For the LLVM compiler you will be told what happened and why. We do not have a diagnostic that tells you what happened, and I was not aware the CLANG compiler did that (this is good feedback).
  Larkin – Potential for aliasing in C/C++ kills many optimizations. Sometimes having negative information can be more valuable than positive information.
  Comment – We try to translate messages for the user as well. Sometimes several options are needed, depending on the audience.
- ***Question:*** About responsibility … are there areas where the responsibilities of the vendor and those building/evaluating and the user are not clear (e.g. atomics)? Are there other areas where there is tension with respect to targeting specific platforms effectively?
  Bercea – With OpenMP we were forced to do most of the development in the front-end of CLANG for code generation and as a result we are not able to touch it to deliver solutions to other problems.
  Comment – My understanding is that Google owns the back-end to LLVM (and they are the 800-lb gorilla).
  Bertolli – Overall, you are right. Google is a big player in the open source community.

DAY 3 – Thursday, August 24, 2017

Vendor Performance and Portability Panel Discussion – Dave Bernholdt (ORNL), Discussion Moderator

Kathryn O'Brien, IBM Research

- We do not have an IBM Performance Portability roadmap but we have an approach that we want to provide tools, programming models and onsite help to help app developers develop performant-portable codes. We do not try to define performance portability or provide metrics for it because we believe that to be the purview of the app developers.
- In the CORAL procurement there was a strong request for COEs but – at that time – we did not anticipate performance portability would be a big part of the COE activity. At IBM we are trying to promote performance portability in the OMP standards group, driving proposals around affinity with a current proposal focused on the "Super if" (i.e. define a set of pragmas depending on target architecture), prescriptive versus descriptive. Earlier this week we heard how OMP is easier to use but as we move towards 4.5, I would argue it is harder to user. We want the ability to get good performance across a range of platforms with less onerous programming practices and OMP is an important part of that.
- The move to LLVM has been effective in the drive to performance portability. It allowed us to provide users with compilers early in the development process. We held hack-a-thons and some of the things that were identified were deficiencies in the standards, so we have had good dialogue across all of the parties involved.

- We prototype explicit-declare-target in LLVM. Intel is also implementing OMP in LLVM and AMD has worked with us to use our implementation for their architecture.
- We have many people working with the COEs on OMP and RAJA, lambda support and we have good involvement there.

Geraint North, Arm

- Arm made its name in the mobile market and many people assume that Arm is solely a low-power CPU play - but that is not the value proposition. Arm has lowered the barrier to entry for people who want to build their own SoCs; we developed a common platform that can move between chips from different vendors easily. Arm allows others to bind to a flexible ecosystem.
- This model only works if I can take code built for someone's Arm core and move it to another entity's Arm core. We want cores to look similar enough to have portability while leaving enough room for our implementers to differentiate. We implement some performance counters. SVE allows our implementers to change vector width (but without requiring code to be compiled).
- It is important to have a portable performance analysis process so people can use tools they know, apply them to Arm and have them work. To us, that is continuity.
- Community is another focus because we work collaboratively – some chips are our own design whereas others are not.
- We have many people developing and optimizing chips for markets. It is in Arm's interest for you to have performance portability (although it is not necessarily in the interest for any one of our partners). Arm needs to make certain it is easy for you to move between cores, so we have commercial tools, math libraries to enable you to work on all of the technologies we produce.

Jonathan Gallmeier, AMD

- We provide flexibility in terms of how our system integrators integrate our hardware and abilities to modify the software stack. We participate in a number of open consortiums and standards effort to provide flexible hardware - but the software performance issue is more challenging.
- From our LLVM perspective, our collaborative efforts allow us to extend compilers to support Kokkos, RAJA and other frameworks as we move into the future. We support various Linux distributions using open-source drivers. Anyone who has an LLVM-based project can plug into our systems easily.
- We have components and runtime with the flexibility to support underlying architecture and we support the broad HPC and machine-learning markets. There is much opportunity and extensibility from the Open Source perspective that is key to our strategy.

CJ Newburn, Nvidia

- We will continue to see specialization to get the best absolute performance, to tailor the hardware, and to bring together lots of computation. We are seeing trends towards specialization in areas such as deep learning, graphics and video.

- To get performance portability in the face of specialization, you have to work at a higher level. We want to support diversity within a common software architecture, which is something that many of us want to go after. With a common scheduler and common set of primitives, we can enable data layouts when needed.
- We see a number of things are converging: cloud, HPC and AI. These three communities can all learn from one another, e.g. making good use of dense nodes and heterogeneous resources with retargetable frameworks (HPC), apps as a service (cloud), highly-tuned libraries and frameworks and fast collectives (AI). In the face of diversity and specialization we will see emergence of new apps – e.g., churn through big data – pick out a subset – do HPC simulation and bring in AI.
- On the slide (#5) is a taxonomy showing where we see HPC and AI are converging. You need diverse resources in the platform and will pick and choose where to implement.
- We see two trends – the first is too many LOCs and for this limited effort is needed – this is the area where we see a need for standards interfaces and libraries. Other codes where you perform aggressive tuning for the target platform – this is going to take work to expose parallelism and scale as needed, along with use of tailored abstractions.

John Pennycook, Intel

- I want to make three main points. First, portability comes from open standards and we are committed to open standards because we know the community is involved and the features in the standards are important to the community (so we will give you what you want when we support those features).
- Intel has no plans to enable proprietary programming models/runtimes. We will help to build open source runtimes and models based on open standards.
- Performance portability is a shared responsibility. It is in Intel's interest for you to be able to program on our architecture in a way that is productive. We contribute to OMP in a number of ways to make performance more productive.
- We do believe the apps have to take some responsibility for performance and performance has to be portable across different platforms.

John Levesque, Cray

- I will present my personal view (which is close to Cray's). The most important thing is adhering to the standards. We spend great effort on this. We have been weak in C++ and we are working on improving that. As far as parallel-programming paradigms are concerned, we are working hard on OMP and have a question about OpenACC. There was discussion about OpenACC at this meeting - but there seems to be a trend with significant movement towards OMP and away from OpenACC (I believe the latter is easier to program to).
- I am not sure what to do about the abstractions. We have excellent performance tools. Will those tools work with RAJA and Kokkos? My last trip to LANL was to gather statistics on 65,000 MPI tasks on the Trinity KNL system and from this data we discovered enlightening information. As you produce your codes it is important for you to be able to look at performance

measurements and determine where the bottlenecks are, so you can fix them. At this meeting I did not hear talk about debugging or looking at performance from using these abstractions.

- Why should we be interested in performance portability? We do want performance portability across our systems and we do sell accelerators - but the reality is we do not want your programs to run as well on systems provided by other vendors. Our interconnects introduce the problem of performance portability because we can do remote memory accesses well so OpenSHEM, etc. run better on our interconnect.

Panel Discussion

- ***Question:*** We heard much about standards and working with the community. What are gaps in the current portfolio of standards?
  Intel – The idea of having a way to specialize functions for different targets/devices (and Super If may solve some of that for OMP 5) but there are probably gaps around that topic.
  Arm – We ask ourselves whether we should do a sparse library and what it should look like. We see sparse math in your codes with no standardization on what it would look like
  AMD – Asynchrony – greater freedom to tolerate unpredictability.
  Cray – Better support for locality, especially threading on manycore/multicore system. It is hard to get good performant OpenMP on a NUMA node.
  IBM – Memory is still an open issue. A problem is the increasing complexity in standards, making them harder for users to use.
  AMD – I echo the need to support multi-level memory and help the developer reason about it. I also echo the need for additional libraries to help compiler and library writers.
- ***Question:*** I have not worried about the performance tools working because people like John Mellor-Crummey have shown they can get through RAJA and Kokkos if the compiler gets the information it needs. How do we bridge source code to compiler to useful information so you can deliver features we need without having to support massive complexity?
  IBM – CORAL Working Groups (e.g. tools) are essential. Everyone needs to work together to get functionality; avoiding complexity is something we need to continue to harp on.
  Arm – There are opportunities with LLVM. We can preserve things in the binary. Building a standard framework within LLVM so higher layers can attach notes that survive to deployment time so everyone's analysis tools builds on that could be helpful. We are all looking at LLVM and we would like to standardize on it as much as we can.
  AMD – Work needs to be done to standardize how the tool interacts and gathers information from the application.
  Intel – If/when you come across something that you do not understand using our tools then please tell us.
  Cray – We need to deal with this because it is so important going forward.
- ***Question:*** There is clear centralization on LLVM as compiler technology but I hear that Google and Apple control aspects of the front-end. Is that a hindrance in terms of developing new features?
  AMD – We have used the LLVM environment in CLANG successfully as an innovation platform. Getting things upstream has not been as big of a focus for us. We expect the development

community to pick up on some of what we have done and help to push it into the environment. In terms of co-generation in the back-end, I think there is enough room for a vendor to provide optimizations in the space. At the front-end, the LLVM community is doing a fairly good job keeping up with standards as they evolve.

Arm – Within the LLVM community personal credibility is important. You have to be active participants in order to get your thing upstream. When vendors like Arm and others want to push new features into LLVM the end users who care have to step up and say they need that feature. People in the LLVM community are not comfortable with the timescales that HPC works with.

- ***Question:*** How do we demonstrate our need?

  Arm – Participate on the mailing lists, which is where things are debated. Seeing a public debate and Q&A is visible and helpful.

  Nvidia – Slowness in upstreaming can be both good and bad. We are driving something for heterogeneous memory management upstream into a Linux kernel but the community tends to be impatient and you have to find the right "bite-size" chunks.

  IBM – It is not a simple process and it is time-consuming - but we have been successful getting most of our patches upstream (and getting these accepted by the community has been important to IBM).

  Arm – LLVM has realistic expectations. We know there is work to do to get it to look like Clang or like Flang. There is work to do there.

- ***Question:*** The roofline model and method was highlighted as a best practice and has been useful for bringing up Trinity and Cori. It has good acceptance (Intel adopted it in their tool chain). Is that part of the Allinea tool chain? Are the vendors thinking about adding roofline? The methods we are using need to be supported in your tool chains.

  Intel – We are open to it. For the roofline plot it is not clear you can always get the data you need from the CPUs to produce one.

  Arm – From the Arm hardware perspective, we are working to get better at a top-down view of what performance counters should be put in the hardware. How do we put those counters in place so they are useable? Up to now we have not looked at it from the perspective of the user.

  Doerfler – I have trouble getting value out of the cache-aware model. Consider performance counters for DRAM traffic (or whatever the final memory hierarchy is).

  AMD – I would like to suggest we turn roofline into an actual tool.

- ***Question:*** In the old days vendor lock-in was a corporate strategy and I am encouraged that everyone here supports our goals and need for portability - but is it still an argument to convince your senior corporate decision-makers that open standards is the right path?

  Cray – We understand and over the last 10-15 years have concluded that the company cannot lock in customers. That is the wrong thing to do. What you want to try to do is differentiate within the standards and enable the hardware so the hardware can enable the standards to run better.

  AMD – AMD Executives have bought into open standards and open source development and see that we can add our own value by understanding what hardware we need to add.

IBM – IBM has adopted the open-source approach and will make code available in an open-source fashion.

Arm – From a hardware perspective, having our math libraries tuned for other vendors by Si was a surprisingly easy discussion. Our Executives understand that to have value in HPC Arm - as a whole - needs to be a value proposition. From a software perspective all commercial tools developers have a primary mission to enable the ecosystem (not to make money from the tools).

Nvidia – We seek to figure out what is in the best interest of the customers and try to create opportunity for people to play on the field and have the best offering inside the field.

- ***Question:*** If I want to use an Intel compiler and a GPU - to my knowledge - the Intel compiler does not support targeting directives to the GPU. Can we add a standard for separation of concerns? When a target directive occurs could it be separated so you could specify two compilers – one for the CPU and one for the target?

  Arm – I think that is how GCC works with target offload.

  Garrett – And if we standardized it then the vendors could write for their own architectures.

  Pennycook – It needs to be explored. It would need to be part of OMP … i.e., this is what the host needs to do to obey target directives and this is what the target needs to do.

  Arm – You would need a compatible runtime on both sides as a good first step,

  AMD – We provide a lightning compiler which is just the GPU-generation portion. We would support anyone who wants to extract and use that.
- ***Question:*** BLAS was a unique success for dense algebra but the sparse BLAS standard never caught on. Sparse linear algebra computational kernels that take more specialized knowledge to write are harder to encapsulate and it takes more domain-specific knowledge to design an interface for a kernel. Is it time for the HPC community to revive a standardization effort?

  Arm – These things are always better when driven by users. I would value a specification. There is much to be said for providing people with tools they need to do things themselves, and that is something we could be better at. I don't know what the appetite is within the community to determine the right balance for building high-performance stacks.

  AMD – I think this is the right time. Other communities also need sparse matrix. If you want to dangle a carrot in front of us and can demonstrate what you are proposing is of interest to other markets as well, we will pay attention.
- ***Question:*** On most of the roofline plots I have seen the app is nowhere near the roofline. Figuring out why this is the case is challenging and requires detective work. Is there anything we can do to make that easier?

  Pennycook – We have tried using something other than Gigaflops as the metric. Looking at instructions/byte can help in some situations.

  Nvidia – That is consistent with having specific goals to work towards and figuring out which are the most important. Decompose the problem into specific goals to explore further.

  AMD – Roofline analysis would need co-design at the level you are talking about. Having access to someone who knows code well sitting alongside one of our engineers would be valuable.

  Richards – And we are willing to make that happen.

Karlin – The tools need to show things better.
Pennycook – Roofline's biggest strength is giving a snapshot of performance; I think it was originally envisioned as a teaching tool.
Richards – It is a good way to start the discussion.
Cray – A problem is having the right hardware counters to find out what the issue is.
Langer – An issue with physics algorithms is latency (not memory bandwidth) – if you can give me something that tells me I need to hide memory latency that I can give to a developer then that would be very helpful. I can give you a Monte Carlo where I can guarantee latency is the problem.

Session 8 – Applications Experience 3 (Chair: Ian Karlin, LLNL)

Olga Pearce (LLNL) - Experiences Utilizing CPUs and GPUs for Computation Simultaneously on a Heterogeneous Node

- 95% of the FLOPS on Sierra will be on the GPUs (4x Volta) and 5% on 2 P9 CPUs. Right now most people are using a single-MPI task/GPU. I will propose a way to use different cores to do different things. You still have a single-MPI task/core and reserve some cores to compute and others to drive the GPU. So, if driving the GPU is done sufficiently well by a single-MPI task then you have other cores to work with.
- I am working with Sierra Early Access (EA) machines at the present time. With heterogeneous MPI tasks I add control code, so some MPI processes drive the GPUs and other MPI processes compute on the CPU cores (being careful about the CPU core/GPU binding). Memory is allocated according to the task on the particular CPU core (and doing this was fairly straight-forward for a spatially-decomposed MPI app where each MPI process owns its data). I ran a proof-of-concept implementation in ARES (structured mesh, multi-physics code) that allocates memory use by context. For the heterogeneous approach, I inject additional context into the communication scheme using the control code. There are some "gotchas", e.g., dependencies may assume that "USE CUDA = allocate on GPUs".
- For loop execution ARES (which uses RAJA) has its own policies to which I added architectural information. To split work between CPU cores and GPUs I use a "sliver". Decomposition affects memory access and therefore performance; my baseline for comparison is one MPI/GPU (our standard approach).
- Slide #9 shows a performance comparison of 1 MPI/GPU with the heterogeneous approach. The former performed better when the inner-most loop usage was large (meaning memory use was optimal). With the heterogeneous approach I could not always give the CPUs a small enough amount of work to speed up the computation but when I was able to do so then the heterogeneous approach performance did improve. I expect this will change once some host device issues are fixed, resulting in the CPU being able to run faster. If we are clever about decomposition I believe we can see some performance improvement. With each software update the MPS performance improves. The direction of the split impacts memory performance.

- In summary, I have a proof-of-concept implementation and performance portability, courtesy of RAJA, with same source code for CPU-executing loops and GPU. I divide work via domain decomposition. Load balancing between GPUs and CPUs is not trivial. The memory access pattern dominates performance. The "ground truth" that we need to make our domain as square as possible may need to be evaluated.
- ***Question:*** If the best strategy depends on the problem size, would it be feasible to put heuristics in the code, depending on the situation?
  We can probably differentiate large versus small problems but pinning it down in a fine manner will not be feasible. There are technical challenges. Once you decide to run with MPS you incur the overhead of the server running. When there is problem-size dependency such as this, the bigger problems may want a different strategy from the smaller problems.
- ***Question:*** What was the performance gain by using CPU and GPU?
  10%-20% improvement (for all of the cores). I left memory on the CPU.

Li-Ta Lo (LANL) - Performance Portable Halo and Halo Center Finding in HACC

- HACC is a cosmology simulation code developed at LANL. Within the "Friends of Friends Halo", the Most Connected Particle (MCP) has the most friends and the Most Bound Particle (MBP) is the particle within the Halo with the lowest number of particles. In most cases the MBP is not the same as the MCP. In HACC there is an algorithm to find the Halo and particles are distributed to compute nodes according to their domain decomposition. Overload zones are defined so the largest halo will be fully contained within a single node. Each node executes a performance portable halo finding algorithm to find halos.
- Performance portability is achieved through parallel primitives (parallel algorithms are provided by the Thrust library). How did we parallelize the halo-finder algorithm? We initialize each particle with a pointer, and in parallel we try to find one of its neighbors. If we define an edge between pairs of particles within linking length then connected-component algorithms give the Friend of Friends halo.
- We ran results on Moonlight and also on Stampede with Xeon Phi. We saw faster runs on the GPU than on the CPU.

Roshan Quadros and Brian Carnes (SNL) - Scaling Post-meshing Operations on Next Generation Platforms

- *Quadros* - We use two steps to generate large meshes. CUBIT provides capabilities for preparing geometry and generating an initial coarse mesh; on top of the CUBIT mesh we run refinement (via Precept) → projection and smoothing operations. The refined meshes have new points that are projected onto the geometry. Smoothing is done to improve mesh quality. This is supported on block-structured, unstructured and hybrid meshes. Brian will discuss refinement and smoothing. I will focus on post-meshing operation #1 – projection. We use OpenNURBS for our geometry kernel because it is open source and lightweight. Our programming model is MPI + Kokkos (OpenMP); the slide shows some sample LOCs. We identify hot spots, re-factor the code around the spots and replace the LOCs with the projected code. Looking at point-

projection-scaling results on KNL, I find as I increase the MPI ranks the code performs well, until I hit 64 ranks.

- *Carnes* – I will talk about refinement and smoothing. We have done much work on unstructured meshes but we switched to doing block-structured meshes due to the SPARC product that Micah discussed yesterday. We process the structured blocks serially and then can do a parallel refinement. The slide shows our scalability results for a sequence of refinement meshes. As we make the problem size larger, we can get more scalability (we were getting the scalability we wanted to see). The complexity involved in the process for smoothing was much higher than it was for refinement. The performance plot is similar to what I showed for refinement (but it took more work). We have learned much doing this. Initially, we supported unstructured meshes and the developer wanted to put a mesh abstraction between the mesh and the smoother so the smoother could operate on any type of mesh - but this proved to be very bad for performance on structured meshes, so we ended up making the smoother more aware that the mesh was hybrid. The unstructured part of the smoother is not likely to be ported to GPUs, so we are working on a different unstructured-mesh refinement built from the beginning to support threading and to run on GPUs. We found memory layout caused very poor performance, initially. We will investigate other smoothing algorithms.

Mark Govett (NOAA) - Parallelization and Performance of the NIM Weather Model for CPU, GPU and MIC Processors

- For a decade we worked on the NIM weather prediction model (developed specifically for GPUs) and the FV3 model, which has been selected by the National Weather Service (NWS) for their global prediction model. It is more difficult to achieve good performance with FV3 (compared to NIM), and we are still trying to figure out why. They are different in many respects. NIM was developed in 2008 when fine-grained computing was seen as the future for highly scalable weather apps. The work on NIM has now ended because the NWS selected the FV3 model, which was developed in the late 1980s as a local (latitude/longitude) model that was then patched together.
- NIM had a simple time-stepping routine and FV3 has a more complex one along with more branching in the code (so finding parallelism was more of a challenge). FV3 was a broader scope for the model – both weather and climate run in fine or coarse scales. NIM had 4K LOCs whereas FV3 has ~23K LOCs. The model grids are seen on the slide. NIM had a uniform, cube-sphere grid and used an indirect addressing scheme to access neighboring points with <1% penalty resulting. The indirect addressing scheme saved time and communications.
- Both codes are written in Fortran; NIM has ~21 routines whereas FV3 has ~165. OpenMP regions are relatively small for NIM and larger for FV3 (with many subroutines). Both codes achieved 10% of peak on Haswell. Both are efficient on the CPU; why are they not as efficient on the fine-grained?
  For NIM we generally see 2x performance improvement with successive generations of hardware. We ran scalability studies on Titan.
- In the fine-grained study of FV3 we found there was not enough work for the GPU to do, so we pushed the vertical loop into the routines – and doing that has taken massive effort. We are

adapting FV3 dynamics for GPUs (we found we could have speed-up) but the team has spent a year doing what was needed. We struggled with promotion of variables to 3D (which blew our caches). Preliminary FV3 performance results (we are still working on 3D results and GPU results) comparing original 2D code with Intel compiler and 3D code with the Intel compiler show the latter is 2x slower. We continue to try to figure out why the FV3 code is still not able to use the fine-grained capabilities on the GPU. KNL gave ~15% performance improvement over the 2D code.

- Takeaways? Code simplicity is important; with NIM we stayed away from things that would be hard for the compilers to analyze. We were aware of the limited memory available to us so smaller NIM kernels were good and bigger kernels in FV3 were not. FV3 uses no shared memory whereas NIM used it extensively (and there are multiple reasons for this).
- Stride-1 is essential for vectorization, SIMT and memory access. We should minimize branching to the extent possible. Use parallel algorithms and avoid complex algorithms where possible.
- We did achieve performance portability with a single code with NIM. The design did target GPUs and we saw every performance benefit that benefited GPU also benefited the CPU. Adapting FV3 has been difficult and we are still seeking to resolve FV3 performance issues. The NIM was collaboratively developed; we find writing in a code that the scientists prefer and working with them as part of the team is a good template for how to proceed.

Keita Teranishi (SNL) - Portability and Scalability of Sparse Tensor Decompositions on CPU/MIC/GPU Architectures

- This work is part of the high-performance data analysis tensor project. We are developing production-quality library software to perform CP factorization for Poisson Regression Problems for HPC platforms. We will use node parallelism (manycore, multicore and GPUs). There are questions associated with software design and perormance tuning that we have yet to answer.
- Slide #3 shows a graphic for CP tensor decomposition – the goal is to express the important feature of data using a small number of vector-outer products (discussed equations found in the next several slides for Poisson and Poisson regression). The algorithm will converge to a constrained stationary point if the sub-problems are strictly convex and solved exactly at each iteration. We looked at two methods: multiplicative update (MU) and Projected Damped Newton for Row-sub-problems (PDNR).
- To parallelize CP-APR we use Kokkos. We use Kokkos::View for data structure and we try to avoid atomics because atomics degrades performance on CPUs and manycore, so we use extra indexing data structure.
- Our implementation shows good scalability with the random tensor - but we see scalability issues with realistic tensor problems. We have developed portable on-node parallel CP-APR solvers and we need to study further the solver performance on realistic problems.

Rob Blake (LLNL) - Melodee: Solving ODEs with platform-specific code generation

- The Melodee is a DSL in the domain of cardiac electrophysiology. One reaction takes 80% of the flops and the code is embarrassingly parallel and computation-bound. Cardioid was a code

developed specifically for Sierra and was a Gordon Bell finalist. Exactly one reaction model was optimized for BG/Q. I am tasked to port this code to GPUs.

- The reaction model changes constantly. I developed a DSL for describing ODEs (and it is available on Github). There are languages that do similar things; my main goal was to separate the implementation of the equations from the description of the equations. I wanted the system to be agnostic to my domain so I built in extensions to separate domain knowledge from semantics.
- Melodee is a language you write models in. The developer writes code generators in python; the Cardioid generator is only 600 LOC. It is backwards-compatible with the many cell machine-learning models that already exist and has been used in three simulators.
- Slide #11 shows the optimizations needed for BG/Q, P100 and KNL. There is no way one code would be able to run on all three of these types of hardware. For the GPU, I find embedding the coefficients in polynomial code is much faster. We can do better on the CPUs by unrolling loops (which is as fast as embedding coefficients) - but on GPUs every access of the coefficient has to be a constant-access expression. Explicit vectorization for these polynomials must be used for performance on CPUs.
- On BGQ and Haswell I have to put in manual vectorization. On the GPU, coefficients must be known at compile time. On KNLs the rational polynomials are slower than they should be (and I do not understand why). No intrinsics are necessary for GPU but vector intrinsics are needed for BGQ, Haswell and KNL. Even the way I lay out data has to be done differently, depending on the platform I run on. (Detailed changes that had to be made to get the code to run on various types of hardware – details found on slides #15-#20). In summary, DSLs make us more productive and code generation is essential for portable performance.

BREAKOUT DISCUSSIONS

Multi-Level Memory Current Experiences: Abstractions for Managing Memory for Systems with Diverse Memory Resources

Session Lead: CJ Newburn, Nvidia

- We worked to find places where we could agree, taking the talks from Session 2 and some from what we will hear tomorrow to look at the different frameworks and address what a layered Venn diagram might look like (seen on Slide #1).
- At the top we have apps/framework/runtimes. Getting everyone to use the same one of these is elusive - but we can have a common notion of infrastructure and look at available options to identify what they are trying to do and then use our collective knowledge to identify how complete solutions are and what we agree on, along with what remains to be resolved. On the slide you see a rough diagram. At the top are user interfaces we know well (and we plan to add MPI). Does it make life easier for things at the top to have a set of primitives the community agrees on that captures what we want to do? People seemed open to discussing that further and we are gathering data for that through HiHAT.
- Then there are things that could be plugged into frameworks: Tapioca, Libpmem, SICM and collections of different allocators and low-level primitives. We also looked at what differentiates things that are higher and lower and asked each presenter where work fits in the diagram. We also asked where we think there is agreement: we know there are many details and they often differ on a per-platform basis. The sense was that abstraction could be helpful here. Making room for customized interfaces is a good idea. Having things at mid-level to build on is helpful via C ABI. The notion of traits interested participants, i.e. taking more of a declarative approach. There is interest in collaborating in these areas and leveraging vendor architectural features. As we move forward, we want to share requirements via user stories, so if we express what we need in concrete ways then we can accelerate progress by trying proof-of-concept.
- It is important to get vendor support for what we are doing but we do not want to put vendors on the critical path. There are a number of areas that are still open, e.g. how to do layering, common abstractions. An emerging idea is how to do this in an asynchronous context. As it becomes harder to tell what is memory and what is storage we need to think about common abstractions on top.

Session Lead: Ian Karlin, LLNL

- Most of the first hour was spent on discussion about how to handle things manually. We talked for the rest of the time about different vendor-specific solutions for managing memory. People want sensible portable defaults first, and then add knobs.
- Many people gain benefits from vendor-based hardware solutions but that is not the case for all. What is the future of multi-level memory? Some app teams say it is hard to use and we should ask for a flatter-space because people are strong-scaling to fit in the fast memory. There will be things that cannot be solved only with fast memory in the current budgets for systems.

- Getting abstractions right is very important. John Levesque used the HPF example – the team had the right abstraction and then were given an easier abstraction, which they ended up using in a hard way. It was pointed out to us that we missed discussion of getting the space simpler before we can begin to work on this, and this is interoperability, which is a recognized challenge for working with multi-physics codes. Building a common infrastructure might simplify this.
- Do people want a clear distillation of requirements and options? Everyone says they want this but it is not clear there is willingness to do the hard work necessary to make it happen. Whether through the OpenMP community, HiHAT or other groups there is going to have to be a solution to this widely-recognized problem. And, we noted that too many implementations are a huge waste of productivity.
- ***Question:*** Is there a path forward?
  Newburn – It is useful to have a Working Group and good for us to be held accountable. It starts with requirements and acceptance criteria as driven by DOE apps. If we can get that and understand better what you want to use under what circumstance and learn enough about frameworks to speak intelligently then we have an opportunity to come up with something that is usable, robust and product-worthy.
  Karlin – There are Working Groups now but they are talking across each other and need app people feeding the Groups their requirements.

Performance, Portability and Productivity: Definitions & Metrics

Session Lead: John Pennycook, Intel

- We tried to answer five questions, most of which I raised in my talk earlier today. The answer to Q1 was essentially yes, definitions and metrics are useful but they can be misused and it would be helpful to push some of the definitions back to the vendors via standardization ("using standards as a weapon") to make sure the hardware delivers.
- We spent most of our time on Q2 and defining portability and productivity. There were two camps: those people concerned with application portability and those who focus on code portability, given the introduction of a new platform. We argued about what portability meant and ultimately agreed to use both definitions. Based on this, it is possible that we need to consider two different metrics for portability. What I presented this morning might be used for cross-platform performance but we should also look at how many LOCs need to be added to a code in order to be able to port it to another platform.
- On tracking applications and their performance portability today, we concluded this is what people should be doing but whether people are actually tracking performance via regressions varied. Portability is tracked via the app portability definition or by users telling developers that the code no longer runs when given a new architecture. Some people on the other hand use issue trackers (on Github, for example) or scrum trackers and story points.
- We did not spend much time on Q4 and Q5 - but there was a sense that we need to accept that code specialization is necessary and we need to figure out how to do that in a productive way. It was noted that software is the hard part and hardware is the soft part. What about third party libraries and programming models? How do we factor in the performance portability of those

components? If they do not support a new architecture, can you consider the model that uses it portable?

- Productivity was seen as being very squishy and no definition was acceptable. We could have spent another hour discussing productivity. We talked about whether things we cannot define or measure can still be useful in guiding thinking about performance portability, and the consensus was that is the case. we can use approximation to guide our design decisions.

Special Concerns of Large Multi-physics HPC Codes

Session Lead: Anshu Dubey, ANL

- I am speaking from the perspective of the codes that have the most to be concerned about performance portability, and have the hardest time achieving it. I use PP in the loosest sense of the words. For these codes, the lifecycle covers several platform lifecycles, so they do not have the luxury of optimizing for one specific type of architecture. Different components can have different performance requirements. They program to a common machine model to achieve some fraction of peak performance, so there are tradeoffs that must be considered.
- What can vendors do? One answer is to continue supporting features of the language and not pull the rug out from under the apps. If there is a good reason for pulling the rug out from the developers then document the reason so the developers can redesign the code such that it does not happen over and over again.
- Abstractions should allow the codes to express calculations in a way such that they need not express every operation in an imperative manner. Perhaps resources can be used for buffering next-stage data. Automating this does not work but having constructs that programmers can use are helpful.
- Typical refactoring challenges include selection of abstractions – are they enduring? Are they the best answers for your productivity issues? At what level should you import them? Cost estimation and resource allocation for future platforms should be done carefully.
- Some codes have had good results with RAJA. In general it is a good abstraction model, so long as you have a C++ code. Another code example comes where incremental changes were not a challenge but figuring out how to have a good ramp-on process for the changes was a challenge (and then it became easy once they figured it out). It is helpful to generate a mini-app to understand the code data structure requirements and be able to perform and test alterations on the mini-app before altering the full code base.
- Your code is as good as the tests you run on it. If you don't test a feature then you don't know whether the feature works, or not. We need to recognize the important of testing and verification (it should have the same value as experimental design in other fields of science). It becomes difficult to distinguish perturbations due to machine precision from a bug. Some useful testing ideas include a pyramid with unit tests at the bottom and higher granularity tests above, followed by integration tests and system-level tests. The community needs to figure out how to put tests into codes that are critical but where the original developer knowledge about the code has been lost. Bugs need to be injected into the code to verify the code is working properly. To test coverage, use a matrix with classes of features across each dimension and select tests carefully.

- For performance testing it is easy to check for catastrophic issues but it is much harder to catch slow degradation. Productivity awareness is more mature in the UK with RSE and there is no equivalent yet in the US but the culture is changing and projects such as ECP are helpful. Equipment design is recognized as a critical scientific contribution and software plays the same role.

Feedback on OpenMP (OMP) 5.0

Session Leads: Tom Scogland, LLNL, and Kevin O'Brien, IBM

- We began with a list of what will likely make it into 5.0 and another list of what we are working on (but likely will not make it into 5.0). There were questions about clarifying the existing standard and asking about interoperability (especially with device constructs) and from that comes the first finding which is interoperability with streams – there was general agreement that would be useful.
- There was general agreement that interoperability would be useful on systems with NV processors so you could leave some kernels in CUDA and use streams to communicate with the OMP sections. There are implementation issues that have to be considered. The discussion led to support for events and tasks on the GPU and the host and communication. We ended up saying the OMP Accelerators and Tasking subcommittee should look into this further.
- Next we discussed specialization of functions (or generic) code for different devices. Kevin points out the fact that this request has been discussed for some time. As of now you can tell the code is being compiled for the host or the device and in most cases the code is processed twice or in a separate file. There is a question as to how we will support this. "Super if" is an option – if on a target region that falls back to the host it is not the same as if the target were not there – and Super if will allow the user to turn something off. We would like to at least get host and no-host as options. How do you tell what the type of a device is when you cannot say what types are?
- The next item is support for shared memory or memory that is only usable if it is attached to the execution in some fashion. There is a sense this could be possible with support for different types of memory in OMP 5 but if you have more than one compiler you may not have interoperability, so the onus on the vendors is for us to talk with one another (and we do, to some extent). Most vendors are engaged with LLVM so there is potential to use that as a reference implementation to iron out some of these issues. Stephen Olivier presented some things we hope will get into OMP 5, e.g. you would be able to tell the compiler to use shared memory.
- We have direct support for reductions because they are interesting to implement and often every platform needs a slight different implementation. A suggestion was made to consider other algorithms, e.g. scans, and we have seen demand for this from wrapper libraries like RAJA.
- We talked about making teams usable without target (presently you can only use teams tightly nested within a construct). Hopefully, we will fix that.
- With respect to updating base languages, we are making progress and there was discussion about what that actually means. There are corner cases that have to be thought through, e.g. interactions between similar features. It is not straight-forward and we have to go through

clause-by-clause and figure out what can be done. It is not likely the whole of the base language will be supported fully for some time.

- Perhaps we should have some form of next touch available or a way to do first touch without having to be as explicit about it as we do now could be a good idea.
- There was recognition that OMP has become a large implementation with much complexity. There are app subsets that care about some sets of features more than others. It would be useful to have a way to tell the compilers which features you might use, e.g. tasks are required on GPUs but make everything else harder.
- Having to assume there may need to be a full runtime requires more work. Because the issue of whether we were going to remove anything from the spec came up we asked what people want to see go away - but heard no answers. We may deprecate some things or change some things but removing anything from the spec when there is a large backwards-compatibility version is not likely.
- There was discussion about error model (and this is something that has come up for many years). At the moment the error model is to die (preferable loudly) and that is likely to be where we will be for some time yet.
- There were suggestions for ways to build the spec, for example, have more transparency on proposed extensions (maybe examples posted on the public repository) and more time or experimentation with prototype implementations (hard for a number of reasons). There is general consensus that more participation from users would be good. This falls partly on site representatives meeting their responsibilities to work closely with users, along with having reps and sites participate in in the discussions and meetings. All of the DOE sites are members and you have the opportunity to shape the spec, if you choose to do so.

Performance Portability Best Practices Website

Session Lead: Jack Deslippe, LBNL,

- http://performanceportabilty.org
- A year ago SC charged NERSC/ALCF/OLCF with a task to document best practices and make recommendations for performance portability. We built the website and will advertise it to our users. The notion is users will find recommendations for using both major types of platforms located at the SC facilities.
- The workload is very diverse – no single code dominates cycles; users care about getting science done and do not want to have to try each performance-portability option possible.
- On the website we tried to define what performance portability means and find there is apprehension about coming up with a single definition. “All definitions of performance portability are wrong, but some are useful.” Our definition is found on the performance portability site.
- We talked about how to give users a way to measure performance portability – not to come up with a single metric but to frame the conversation. Our approach is based on roofline and participants said that was a good start and encouraged us to use a set of well-defined benchmarks that is associated with the application.

- Does our having performance portability on today's systems say anything about likely performance portability on future architectures? That is an important point to consider.
- There were comments made specifically about the website and there is interest in turning the paper we are writing now into a living document. The site lives on Github; I will distribute the link and ask for contributions over the next few weeks.
- There are a number of specific recommendations, in particular the notation that GPUs are "cache-less" along with the need for more case studies. We think facilities are a good way for users to funnel budget and feature requests to vendors and standards bodies, and we want to encourage users to reach out.
- A number of action items evolved from the discussion including developing a process to contribute to the performance portability website and link the performance portability website to a new website from the IDEAS project.

DOE Participation in ISO/C++ Standards Committee

Session Lead: Carter Edwards, SNL

- We discussed DOE lab participation in the ISO/C++ Standards Committee. DOE is not a member of the Committee but each lab is a corporate member of the committee with a single vote (meaning 6 votes out of a total of 60-70). On the slide is a list of DOE lab representatives; we would like to have at least two people from each lab to give us a stronger voice. Sandia is hosting the meeting in November so it will be easy to get to. On the slide now is a notional graphic showing how ISO/C++ Committee collects input and (slowly) develops the standard. Things move from subcommittee to the full committee official voting (which is where there is one company/one vote to get things into the standard). Participation is very important. Those of us who participate regularly could use alternates.
- ***Comment:*** If people want to go to the meeting in November they can see proposed papers via their lab rep.
  3-4 weeks before each meeting all of the papers to be discussed are available to the reps. I send it around my lab to get input and feedback where there is particular interest in vectorization data types, as an example. Your lab rep can provide details about what will be discussed at the next meeting.

POSTER SESSION

Arlie Capps, Peter Robinson, Joseph Chavez (LLNL): Progress Porting ALE3D to the GPU (pdf)
We present an overview of progress porting core algorithms of ALE3D to the GPU. We use the RAJA performance portability layer for execution abstraction and the CHAI (Copy Hiding Application Interface) memory manager for heterogeneous memory abstraction. We will show performance results and challenges for test problems of varying complexity. We also will demo SPOT (Software Performance Optimization Tracker), used for tracking performance across multiple platforms.

Appelhans, David (IBM): Overlapping Data Movement and Compute with XLF and OpenMP 4: Experiences in UMT (pdf)
This poster describes the successes and challenges of overlapping data movement using OpenMP 4 for the CORAL benchmark application UMT. The memory footprint of UMT is very large and will not fit entirely in the GPU memory, so data and compute must be asynchronously pipelined into the GPU. Topics covered will be how to handle deep copy of data structures in OpenMP 4 and how to use parallel CPU threads to overlap data movement with kernel compute. Results will be compared to a highly optimized CUDA streams implementation.

Gorentla Venkata, Manjunath & Ferrol Aderholdt (ORNL): SharP: Towards Programming Hierarchical-Heterogeneous Memory based Extreme-Scale Systems (pdf)
The poster describes, SharP, a data-structure based and data-centric programming construct. We will demonstrate with empirical results the performance and portability advantages achieved for Big-Compute and Big-Data applications while using SharP. Particularly, we will show (i) the performance, portability, productivity, and data resiliency advantages achieved for QMCPack and 1D Stencil kernels on multiple architectures while using the SharP constructs, (ii) the performance advantages achieved for Memcached, Graph 500, and the implementation of a KV store for extreme-scale systems, and (iii) the simplicity of using SharP on different memories including, DRAM, High-bandwidth Memory (HBM), and non-volatile random access memory (NVRAM).

Slaven Peles, John Loffeld and Carol Woodward (LLNL): Performance portability of numerical time integrators in SUNDIALS library (pdf)
Numerical integrators are an essential tool for almost every dynamic simulation. Typically, dynamic simulations also involve linear algebra, nonlinear solvers, and spatial discretization tools. Thus, performance portability of numerical integrators needs to be investigated in a context that involves all codependent libraries required for dynamic simulations. In this talk, we present performance analysis results for integrators from the SUNDIALS library on different GPU---based hardware architectures. We show results for standalone SUNDIALS computations and for SUNDIALS running with MFEM, a finite element discretization package. The observed performance is consistent across all tested hardware architectures. Finally, we discuss software architecture solutions using the hardware abstraction layer RAJA, which streamline development, simplify the use, and reduce maintenance cost of performance portable libraries.

North, Geraint (ARM): Experiences with performance portability across ARM microarchitectures (pdf)
The ARM Performance Libraries aim to provide the fastest possible BLAS, LAPACK and FFT kernels on ARM systems, irrespective of whether the processor on which they are executed was designed by ARM or one of our many architecture licensees (such as Cavium, Fujitsu or Qualcomm). I will describe the tools and processes that our teams use to identify the characteristics of a given microprocessor implementation, and how those are in turn used to produce high-performance BLAS kernels. I will present some data highlighting interesting differences in microarchitectural detail between different processor types, as well as the performance differences that you might see if you run the "wrong" kernel for a given implementation.

Prashant Rawat, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan (Ohio State University): High-performance GPU code generation for high-order stencils: Alleviating register pressure (pdf)
High-order stencils are often used in accurate numerical solution of PDEs. They feature significant reuse of data and the high operational intensity often means that the limited bandwidth to GPU global-memory is not a performance bottleneck. However, the high-degree of data reuse poses a problem with excessive register pressure. While the current state-of-the-art register allocators in production compilers like Nvidia nvcc are very effective for most applications, they are unable to effectively manage register pressure for complex high-order stencils, resulting in a sub-optimal code with a large number of register spills.

The StencilGen code generator for stencils addresses this problem by implementing a new statement reordering framework that models stencil computations as a forest of trees with shared leaves. It adapts an optimal scheduling algorithm for minimizing register usage for tree computations. The effectiveness of the approach is demonstrated through experimental results on complex stencils from the SW4 seismic modeling code from LLNL.

Aravind Sukumaran-Rajam, Arjun Suresh, Jyothi Vedurada, Sriram Krishnamoorthy, and P. Sadayappan (Ohio State University): TTLG: Tensor Transpose Library for GPUs (pdf)
TTLG (Tensor Transposition Library for GPUs) is an efficient tensor transpose library implemented with the aim of high performance on GPUs. TTLG also includes a performance prediction model, which can be used by higher level optimizers using tensor transposition. For example, tensor contractions are often implemented by using the TTGT (Transpose-Transpose-GEMM-Transpose) approach -- transpose input tensors to a suitable layout and then use high-performance matrix multiplication followed by transposition of the result. The performance model is also used internally by TTLG for choosing among alternative kernels and/or slicing/blocking parameters for the transposition. The performance of TTLG is compared with existing code generator as well as a library implementation of tensor transpositions on GPUs. We demonstrate comparable or better transposition times for the "repeated-use'' scenario and considerably better "first-use'' performance than current state-of-the-art alternatives. Tensor contraction implementations using TTLG and cuBLAS achieve considerably higher performance

than the current GPU implementation of the triples calculation for the CCSD(T) method in the NWChem computational chemistry suite.

Elsa Gonsiorowski (LLNL): MACSio Development and Proxy Application Validation (pdf)
Proxy applications are a vital resource for evaluating and preparing for new systems and hardware. A proxy application for I/O workloads will be essential as new storage tiers and vendor solutions are being developed and deployed. MACSio is a multi-purpose, application-centric, scalable I/O proxy application designed to imitate a variety of multi-physics applications. Using a plug-in structure, it operates at the same level of abstraction as real applications. That is, MACSio can utilize a wide selection of the I/O software stack. It also implements a variety of multi-physics code features to closely mimic the way in which data flows in-to and out-of these applications. This talk will focus on the challenges faced in developing a proxy application. We will discuss validation efforts and show latest results for a burst buffer system.

Lixiang Luo (Research, ORNL/IBM CoE): OpenACC/OpenMP4.5 Interoperability (pdf)
The maturity of OpenMP4.5 compilers with support for offloading has improved significantly lately, so that many developers are giving serious consideration for migrating their GPU-enabled codes to OpenMP4.5 for better portability in the future. One may seek an "incremental porting" strategy to migrate the codes between OpenACC and OpenMP4.5, such that directives can be ported at smaller increments. After each increment, the code still works as a whole, so that its results can be verified before proceeding to the next increment. This approach has been adopted extensively when CPU codes are ported to OpenACC. Indeed, the option to do incremental porting is a major contributing factor to the success of OpenACC, as it can save enormous development efforts. Since OpenACC and OpenMP4.5 share a similar programming model, one would expect a straightforward transition one to the other. However, because no compiler can fully support both OpenACC and OpenMP4.5 interchangeably, the interoperability between OpenACC and OpenMP4.5 compilers becomes a prerequisite for any code which uses both paradigms.

Contrary to the relatively ease mixing CPU codes and OpenACC offloading, getting OpenACC and OpenMP4.5 offloading to work together is much more challenging. For simplicity, we will restrict our discussion to IBM POWER-based Linux systems and the two compiler suites with most comprehensive support for OpenACC and OpenMP4.5, that is, PGI and IBM XL, respectively. As a proof of concept, a simple OpenACC Fortran program is partially ported to OpenMP4.5, where only one of the two OpenACC kernels is ported to OpenMP4.5, leaving the other kernel and data management directives unchanged in OpenACC. Careful coordination is necessary to allow the runtime libraries by the two compiler suites to work properly together. The general interoperability of Fortran is another hurdle, due to the lack of standard ABI for Fortran. C and C++ ABI standards provide better support for compiler interchangeability, so it is generally easier for C and C++ programs to migrate between OpenACC and OpenMP4.5.

Chunhua "Leo" Liao (LLNL): AutoPar: Semantics-Aware Automatic Insertion of OpenMP directives (pdf)

AutoPar is an open-source tool that automatically adds loop-level OpenMP directives into sequential C/C++ source code. Besides using classic compiler analyses to discover loop-carried dependencies and recognize data-sharing attributes, AutoPar is able to optionally incorporate semantics of standard and user-defined abstractions in order to discover more parallelization opportunities. Example supported semantics include these related to function side effects, pointer aliasing, and indirect array accesses. These semantics can be either manually provided by users or automatically generated by third-party tools. In recent experiments using two proxy apps, AutoPar discovered 63.6% to 124.8% more parallellizable loops, compared to a baseline classic approach without semantics-awareness. AutoPar also has other user-requested features, such as generating patch files and checking correctness of existing OpenMP directives in input codes.

Attendee List

Total Attendees: 127

ANL	11	Cray	5	NNSA	6
LANL	21	IBM	7	Other	6
LBL	5	Intel	2		
LLNL	31	NVIDIA	7		
ORNL	4	AMD	2		
SNL	17	Arm	3		

Last Name	***First name***	***Organization***	***Email Address***
Chunduri	Sudheer	ANL	sudheer at anl.gov
Dubey	Anshu	ANL	adubey at anl.gov
Jin	Xiao-Yong	ANL	xjin at anl.gov
Luo	Ye	ANL	yeluo at anl.gov
Messina	Paul	ANL	messina at anl.gov
Parker	Scott	ANL	sparker at anl.gov
Rahaman	Ron	ANL	rahaman at mcs.anl.gov
Romero	Nick	ANL	naromero at anl.gov
Rupp	Karl	ANL	me at karlrupp.net
Tessier	Francois	ANL	ftessier at anl.gov
Thakur	Rajeev	ANL	thakur at anl.gov
Davis	Mike	Cray, Inc.	u3186 at cray.com
DeRose	Luiz	Cray, Inc.	ldr at cray.com
Levesque	John	Cray, Inc.	levesque at cray.com
Poxon	Heidi	Cray, Inc.	heidi at cray.com
Wagenbreth	Gene	Cray, Inc.	gwagenbret at cray.com
Appelhans	David	IBM	dappelh at us.ibm.com
Bercea	Doru	IBM	gheorghe-teod.bercea at ibm.com
Bertolli	Carlo	IBM	carlo.bertolli at gmail.com
Cordery	Matthew	IBM	mcorder at us.ibm.com
Luo	Lixiang "Eric"	IBM	lixiang.luo at ibm.com
O'Brien	Kathryn	IBM	kmob at us.ibm.com
O'Brien	Kevin	IBM	caomhin at us.ibm.com
Jacobsen	Doug	Intel	douglas.w.jacobsen at intel.com
Pennycook	John	Intel	john.pennycook at intel.com
Adedoyin	Toks	LANL	aadedoyin at lanl.gov
Baker	Randal	LANL	rsb at lanl.gov
Bergen	Ben	LANL	bergen at lanl.gov
Bird	Bob	LANL	bird at lanl.gov

Brock	Jerry	LANL	jsbrock at lanl.gov
Ferenbaugh	Charles	LANL	cferenba at lanl.gov
Garrett	Kris	LANL	ckgarrett at lanl.gov
Green	Ron	LANL	green at lanl.gov
Gunter	David	LANL	dog at lanl.gov
Hjelm	Nathan	LANL	hjelm at lanl.gov
Ionkov	Latchesar	LANL	lionkov at lanl.gov
Lang	Michael	LANL	mlang at lanl.gov
Lo	Li-Ta	LANL	ollie at lanl.gov
Loncaric	Josip	LANL	Josip at lanl.gov
Long	Alex	LANL	along at lanl.gov
McCormick	Pat	LANL	pat at lanl.gov
Nam	Hai Ah	LANL	hnam at lanl.gov
Nystrom	Dave	LANL	wdn at lanl.gov
Robey	Bob	LANL	brobey at lanl.gov
Rockefeller	Gabriel	LANL	gaber at lanl.gov
Williams	Sean	LANL	swilliams at newmexicoconsortium.org
Doerfler	Doug	LBL	dwdoerf at lbl.gov
Friesen	Brian	LBL	bfriesen at lbl.gov
Gayatri	Rahulkumar	LBL	rgayatri at lbl.gov
Leak	Steven	LBL	sleak at lbl.gov
Deslippe	Jack	LBL	jrdeslippe at lbl.gov
Beckingsale	David	LLNL	beckingsale1 at llnl.gov
Blake	Rob	LLNL	blake14 at llnl.gov
Burmark	Jason	LLNL	burmark1 at llnl.gov
Capps	Arlie	LLNL	capps2 at llnl.gov
Collette	Mike	LLNL	collette1 at llnl.gov
Draeger	Erik	LLNL	draeger1 at llnl.gov
Epperly	Tom	LLNL	epperly2 at llnl.gov
Gonsiorowski	Elsa	LLNL	gonsiorowski at llnl.gov
Hornung	Rich	LLNL	hornung1 at llnl.gov
Jones	Holger	LLNL	jones19 at llnl.gov
Karlin	Ian	LLNL	karlin1 at llnl.gov
Kunen	Adam	LLNL	Kunen1 at llnl.gov
Langer	Steven	LLNL	Langer1 at llnl.gov
Le	Hai	LLNL	hle at llnl.gov
Leon	Edgar	LLNL	leon at llnl.gov
Liao	Chunhua "Leo"	LLNL	liao6 at llnl.gov
Maruyama	Naoya	LLNL	maruyama3 at llnl.gov

Neely	Rob	LLNL	neely4 at llnl.gov
Pankajakshan	Ramesh	LLNL	pankajakshan1 at llnl.gov
Pearce	Olga	LLNL	olga at llnl.gov
Peles	Slaven	LLNL	Peles2 at llnl.gov
Pozulp	Mike	LLNL	pozulp1 at llnl.gov
Richards	David	LLNL	richards12 at llnl.gov
Ryujin	Brian	LLNL	ryujin1 at llnl.gov
Schulz	Martin	LLNL	schulzm at llnl.gov
Scogland	Tom	LLNL	scogland1 at llnl.gov
Sundram	Shiv	LLNL	sundram1 at llnl.gov
Tagani	Bujar	LLNL	bujar at llnl.gov
Vargas	Arturo	LLNL	vargas45 at llnl.gov
Voronin	Alexey	LLNL	voronin1 at llnl.gov
Wong	Jonathan	LLNL	wong125 at llnl.gov
Norris	Edward	NNSA	edward.norris at nnsa.doe.gov
Macaluso	Antoinette	NNSA ASC/Leidos	antoinette.macaluso at leidos.com
Hoang	Thuc	NNSA/DOE	thuc.hoang at nnsa.doe.gov
Simpson	Emily	NNSA/DOE	emily.simpson at nnsa.doe.gov
Sinha	Punita	NNSA/DOE	Punita.Sinha at nnsa.doe.gov
Voigt	Bob	NNSA ASC/Leidos	rvoigt at krellinst.org
Branch	Greg	NVIDIA	gbranch at nvidia.com
Feldstein	Bob	NVIDIA	bfeldstein at nvidia.com
Katz	Max	NVIDIA	mkatz at nvidia.com
Larkin	Jeff	NVIDIA	jlarkin at nvidia.com
Newburn	CJ	NVIDIA	cnewburn at nvidia.com
Norton	Dave	NVIDIA	dnorton at nvidia.com
Rennich	Steven	NVIDIA	srennich at nvidia.com
Aderholdt	Ferrol	ORNL	aderholdtwf1 at ornl.gov
Bernholdt	David	ORNL	bernholdtde at ornl.gov
Liakh	Dmitry	ORNL	liakhdi at ornl.gov
Tharrington	Arnold	ORNL	arnoldt at ornl.gov
Ang	Jim	SNL	jaang at sandia.gov
Brunini	Victor	SNL	vebruni at sandia.gov
Carnes	Brian	SNL	bcarnes at sandia.gov
Cook	Jeanine	SNL	jeacook at sandia.gov
Edwards	H. Carter	SNL	hcedwar at sandia.gov
Glass	Mike	SNL	mwglass at sandia.gov
Hammond	Simon	SNL	sdhammo at sandia.gov
Hoemmen	Mark	SNL	mhoemme at sandia.gov

Howard	Micah	SNL	mhoward at sandia.gov
Langston	William	SNL	wllangs at sandia.gov
Lin	Paul	SNL	ptlin at sandia.gov
Merewether	Mark	SNL	mtmerew at sandia.gov
Olivier	Stephen	SNL	slolivi at sandia.gov
Quadros	Roshan	SNL	wrquadr at sandia.gov
Ruggirello	Kevin	SNL	kruggir at sandia.gov
Teranishi	Keita	SNL	knteran at sandia.gov
Zinser	Brian	SNL	bzinser at sandia.gov
Benton	Brad	AMD	brad.benton at amd.com
Gallmeier	Jonathan	AMD	jonathan.gallmeier at amd.com
North	Geraint	ARM	geraint.north at arm.com
O'Connor	Mark	ARM	mark.oconnor at arm.com
Vadlamani	Sirinath	ARM	srinath.vadlamani at arm.com
Govett	Mark	NOAA Earth Sys Research Lab	mark.w.govett at noaa.gov
Sadayappan	P. (Saday)	Ohio State University	psaday at gmail.com
Sukumaran Rajam	Aravind	Ohio State University	sukumaranrajam.1 at osu.edu
Hugues	Maxime	TOTAL	maxime.hugues at total.com
Martineau	Matt	University of Bristol	m.martineau at bristol.ac.uk
Vaquero	Alejandro	University of Utah	alexvaq at physics.utah.edu

Steering Committee

- Hai Ah Nam, LANL (chair)
- Charles Ferenbaugh, LANL (co-chair)
- Rob Neely, LLNL (co-chair)
- Bert Still, LLNL
- Ian Karlin, LLNL
- Mike Glass, SNL
- Rob Hoekstra, SNL
- Tjerk Straatsma, ORNL
- Wayne Joubert, ORNL
- Rebecca Hartman-Baker, NERSC
- Jack Deslippe, NERSC
- Tim Williams, ANL
- Kalyan Kumaran, ANL
- Nick Romero, ANL
- Victor Lee, Intel
- John Pennycook, Intel
- John Levesque, Cray
- Kathryn O'Brien, IBM
- Cyril Zeller, NVIDIA
- CJ Newburn, NVIDIA