DOE Centers of Excellence Performance Portability Meeting 2017

August 22 - 24, 2017

http://www.lanl.gov/asc/doe-coe-mtg-2017.php

Crowne Plaza Downtown Denver

ABSTRACTS

Tuesday

Session 1 - Performance Portability, Best Practices, Words of Wisdom

Setting expectations for Performance Portability between Companion Accelerator and Manycore Systems
Author: John M Levesque, Cray

Of course it is always possible to have a portable application that runs "well" on numerous target architectures. The real question is what if any performance can be given up on one of the architectures in order to be portable across all the targets. There have been several excellent studies done on this subject in Europe and this talk will discuss the conclusions of that work. The four systems of Interest will be a hosted GPU system, such as that to be delivered to LLNL and ORNL, the many-core KNL system delivered to NERSC and LANL/SNL, the state-of-the-art Xeon multi-core system and the upcoming ARM based systems. While three of these systems are very similar, the hosted GPU system has some characteristics that require optimizations that will hinder performance of the other three.

When Performance Portability is less than Perfect: Matching Applications to Architectures
Author: Mark O'Connor, ARM, Allinea

We're entering an era of increasing diversity in both system architectures and application workloads. In this heterogeneous environment performance portability is a constant challenge. Wherever it is less than 100% successful there remains significant value in making sure the right applications are run on the right machines. This talk looks at user-friendly ways to characterize application performance and advise on its suitability for certain architectures. In it we examine existing generic solutions and discuss extensions allowing sites to build their own "architecture suitability" report that estimates an application's relative efficiency on specific clusters available to them.

Exascale Computing Without Templates
Authors: Karl Rupp, Richard Mills, Barry Smith, Matthew G. Knepley, and Jed Brown (ANL, Rice University, University of Colorado, Boulder)

We argue, giving technical details, that the use of C++ template metaprogramming comes with several signicant disadvantages for a composable and maintainable software stack. These problems may only became apparent as a project grows and matures, and therefore it is especially important for software libraries at the bottom of the software stack to preserve runtime customizations, facilitate debugging, and play nicely with other libraries. As an alternative path forward, we propose a strategy that we believe is superior in the long run: We discuss the value of adopting professional software development practices such as coding standards and rigorous testing, the need for application programmers to develop data structures and interfaces that are completely understood by the application development team and are designed for the continuous evolution of the application, the willingness to develop small architecture-specic kernels where necessary, and the reuse of existing libraries.

Implications of a Metric for Performance Portability: Necessity of Specialization and Application-Specific Abstractions

Authors: John Pennycook, Jason Sewall, Victor Lee, Intel

As noted in the final report from the 2016 DOE COE PP meeting, there is not yet a universally accepted definition of "performance portability", or even consensus that such a definition is necessary or desirable. In this talk, we review a number of previously proposed definitions and discuss our ongoing efforts to unify them into a quantitative metric that captures the needs and desires of the performance portability community. We demonstrate the utility of our current metric in: comparing different software development methodologies; guiding application optimization efforts; and distilling the roles of different stakeholders.

Achieving high values of performance portability (as measured by our metric) requires an application to achieve both high performance and high portability. Unfortunately, accomplishing this with a single source code is very likely intractable: specializing code for all different inputs, workflows, algorithms, hardware, memory hierarchies, instruction sets and more is a problem that cannot be solved by a compiler alone. However, the alternative -- specialization performed manually by the application developer for each of these eventualities -- has historically been unpopular because of its impact on programmer productivity and code maintainability. We demonstrate the necessity for some degree of specialization using simple examples and discuss the difficulties we have encountered in expressing specialization using existing performance portable programming models and languages. Our experiences are used to motivate proposed extensions to Kokkos, parallel C++ and OpenMP.

Session 2 - Memory Hierarchy (on-node)

Memory Management Extensions for OpenMP 5.0

Stephen Olivier, Center for Computing Research, Sandia National Laboratories, slolivi@sandia.gov

Current and emerging architectures include heterogeneous memory components with varying functional and performance characteristics. To address this trend in a portable way, members of the OpenMP Language Committee and its Affinity Subcommittee authored a 2017 technical report (OpenMP TR5). They have continued to develop the ideas proposed in TR5 toward inclusion in the OpenMP 5.0 specification. This talk will give an overview of the major concepts, and distinguish the two broad feature sets: core API elements that have been the focus of near term efforts for the OpenMP 5.0 Preview 2 technical report, and additional functionality that will build on the core after 5.0 Preview 2.

The proposed extensions include new constructs, clauses, and runtime library routines designed to meet the needs of diverse memory designs and application use cases thanks to contributions from vendors, DOE labs, and other HPC centers. However, more feedback is needed, particularly for the later extensions whose development is less mature.

Deep Copy and Unified Memory in OpenMP

Tom Scogland, Lawrence Livermore National Laboratory

Early experiences with OpenMP 4.0, as well as other directive-based offload models, have shown that deep copy is a key challenge to porting complex applications to offload directives. Without a flexible deep-copy mechanism, pointer-based data structures are at best difficult to manage, particularly when shared memory between the host and device cannot be assumed. Despite the importance of the issue, and the considerable effort expended by vendors, standards bodies and users, no solution has emerged as the clear choice. This talk will discuss the current state of deep copy in OpenMP as well as our plans and designs for the next step. Specifically our plans for making

unified memory usage standards compliant as well as an extension that combines a restricted compiler-assisted deep copy with a mechanism for users to register their own custom mapping implementations for true deep copy on distributed memory systems.

A declarative approach to managing memory properties
Author: CJ Newburn, NVIDIA
Once there is a separation of concerns between 1) application developers who specify the semantics for how data collections are used as operands and declare those operands' usage properties and 2) tuners who manipulate additional properties as they map those semantics onto different target architectures, then codes can be more portable, more maintainable, and they can be more easily tailored to emerging features in target architectures. This talk describes a declarative approach that draws on buffer design ideas from the hStreams and HiHAT projects, and shows how the design is relevant to memories of different kinds, layers and locations. Examples will be offered for Xeon Phi, POWER and GPUs. Preliminary results will be shared from the HiHAT project.

Simplified Interface to Complex Memory (SICM)
Authors: Sean William, Latchesar Ionkov, Michael Lang (LANL)
This work, part of ECP, deals with finding a common abstraction to enable portability for runtime software and applications on emerging hardware with complex memory hierarchies. Our research concerns the problems of portable software abstraction for heterogeneous memory, based on various expectations of what such heterogeneous memory systems will actually look like. One candidate is the Intel Xeon Phi, which has high-bandwidth memory that can be configured in several ways. Cache-based modes do not require application or runtime manipulation, while other modes require usermanagement of the high-bandwidth memory. Manual control of its high-bandwidth memory in flat and sub-NUMA modes is exposed using existing NUMA subsystems. This represents a distortion of the original intention of NUMA, in which there is always a single, "natural" priority ordering for memory placement. That is, in the traditional view of NUMA, memory should be placed on the closest node, based on an integer distance emblematic of latency. The Xeon Phi high-bandwidth memory breaks this assumption: if HBM has lower distance, then all allocations will be placed there, even those that are not bandwidth-sensitive; if HBM has higher distance, then allocations will almost never be placed there automatically, but instead must be explicitly placed there by use of memory policy (e.g., mbind, numactl). Intel chose the latter solution, which means, in noncache modes, high-bandwidth memory must be used explicitly, and this use must begin from specific knowledge of how HBM is exposed via NUMA.

This is a very nonportable solution and could become arbitrarily complex and nonstandard in the future as other technologies enter, e.g., IBM's and Nvidia's NVLink and Intel's byte-addressable 3D Xpoint. We assert that "true" portability in these situations will require hardware vendors to expose generic information about memory devices such that high-level orderings can be generated, e.g., a prefer-bandwidth ordering, a prefer-capacity ordering, a normal/default ordering. To get the ball rolling on that effort, we are modifying the Linux kernel to add additional memory policies to support the definition of such orderings, so that, even though we will still rely on mechanisms like mbind and numactl, those system calls will be far more portable, and be based on higher-level, intent-based descriptions. This approach, more broadly, is based on an expectation that the Xeon Phi will serve as a model for the presentation of future heterogeneous memory systems. Our presentation will cover our work in the SICM project to date and the status of the kernel modifications.

Umpire: Resource Management for Heterogeneous Memory Hierarchies
Authors: David Beckingsale, LLNL
To achieve good performance on platforms with heterogeneous memory hierarchies, applications must carefully control data placement and movement within those hierarchies. Advanced architectures provide multiple levels of memory with varying capacities, access timing rules, and visibility to different compute resources. Applications must intelligently allocate data in these spaces. Depending on the total amount of memory required, applications may also be forced to move data between different parts of the memory hierarchy. Furthermore, applications using multiple packages must coordinate effectively to ensure that each package can use the memory resources it needs.
In this presentation, will will present Umpire, a library to address these challenges from an application-oriented perspective. Specifically, Umpire will provide support for: querying memory resources, provisioning and allocating memory, and memory introspection. This project will allow computer scientists and computational physicists to efficiently program the memory hierarchies of current and future high-performance computing architectures, without tying their application to specific hardware or software. Development of Umpire will focus on developing an API to allow applications to easily access other memory technologies, and build up capabilities based on application needs. We will present the current software design and roadmap, and highlight some initial application needs that Umpire will aim to address.

Session 3 - Applications Experience 1

Kokkos port of CoMD mini-app for Trinity-class systems and NVIDIA Pascal nodes
Authors: David Gunter , Adetokunbo Adedoyin (Los Alamos National Laboratory)
We present a user case study in porting the CoMD mini-app to Kokkos. We focused on porting the two core kernels for the MD force calculations, the Lennard-Jones potential and that of the embedded-atom method. We will discuss issues that arose as part of the porting effort as well as present performance changes, highlighting before vs. after results. Results we be gathered from runs on Cray/Intel KNL and Haswell systems, and single-node results from NVIDIA Power8+ and Pascal based systems.

SW4Lite: Performance Portability using RAJA
Authors: Ramesh Pankajakshan, Bjorn Sjogreen (LLNL)
SW4Lite is a proxy app for SW4 (Seismic Waves 4^{th} Order), a 3D seismic modeling code that solves the elastic wave equations using a fourth order accurate finite difference approximation on Cartesian and curvilinear grids with explicit fourth order accurate time stepping. The main computational work consists of stencil evaluations with a stencil of size 5^3 = 125 points at most grid points. The baseline version uses MPI and OpenMP for parallelism and delivers good performance on SMP clusters. Furthermore, SW4lite provides Cuda versions of the most computationally intense routines for computing on GPU enabled machines. The Cuda version of SW4lite runs well on Coral EA machines. The proposed talk will describe use of RAJA to construct a portable version of SW4lite
An overview of the memory management issues needed to achieve portability and performance using RAJA will be presented. This will be followed by performance data for a representative test case on CTS-1, ATS-1(KNL) and Coral-EA nodes. The source of any performance discrepancies will be discussed along with possible ways to mitigate them without significant impacts along the portability axis.

Performance Portability Experiments with the Grid C++ Lattice QCD Library
Authors: Alejandro Vaquero, University of Utah

The Lattice QCD ECP application team is formulating a new data-parallel API to support exascale calculations. It is essential that any implementation of the API have performant portability between multi/many-core and GPU architectures. We are evaluating "Grid" as a candidate implementation of the API. Grid is a C++ data-parallel framework written for Lattice QCD simulations. It has been optimized to run efficiently on multi/many-core CPU architectures with its SIMD-friendly data layouts, while the C++ abstraction layer makes the architecture-specific implementations invisible to the end users. Its data-parallel design can in principle also work well with GPUs. Started with a trimmed-down version of Grid, we have investigated different approaches (OpenACC, Just-In-Time compilation and CUDA) for performance portability in this C++ library. In this talk, we will discuss the pros and cons of each approach, and present the performance of a QCD benchmark code on various architectures.

Experiences Porting a Multiphysics Code to GPUs
Authors: Brian Ryujin, LLNL
Ares is a massively parallel multiphysics ALE-AMR code at LLNL. It is written in C and C++ with over 700k lines of code. It is currently making the transition to GPU architectures, focusing mainly on Sierra. For this transition, we have chosen to use RAJA and Unied Memory as our performance portability strategy. In this talk, we will present our experiences with our approach. Preliminary performance results indicate that we can achieve up to 12X speedup when utilizing the GPUs over utilizing only the CPUs on the node while preserving performance characteristics for the CPU only code.

Breakout Day 1

Session 1 & 2 (2 parallel sessions):

Multi-Level Memory Current Experiences
Session Leads: Ian Karlin (LLNL), CJ Newburn (NVIDIA)
Current and emerging architectures include heterogeneous memory components with varying functional and performance characteristics. Currently on systems users have two options to handle data motion. They can rely on hardware features, such as unified memory on GPU systems or cache modes on CPU systems. Alternatively, they can manage data motion explicitly using vendor specific memory management techniques and OpenMP map directives.

In this breakout we will lead a discussion on what users are doing today, where the shortcomings, successes and open issues are, and what they hope to use in the future. In particular, we hope to understand if any techniques being used today are portable between KNL and GPU machines and what kinds of SW infrastructure could facilitate that portability. Dealing with NVM and other emerging memory technologies that enrich the memory hierarchy is also a challenge. We will outline key challenges and discussion potential solutions.

- **Joining the two sub-sessions into one for the first 30 minutes** of the 90-minute breakout
- Asking a representative author of the 5 presentations of session 2 (memory hierarchy) and 2 presentations of session 6 (hierarchy memory, off node) do the following on a **single diagram** that shows how the work we're presenting relates to one another, hopefully as a layered Venn diagram
    - start with a **panel** where reps each get 3 minutes, in a combined slide deck, to highlight
        - **where your work fits** in the diagram

        - where we think there's **agreement**
        - where we think there are **opens**
        - how we think we should **move toward closure** of those opens
- After this opening section, we'll split into halves for discussion
    - **Audience reactions** on the above topics and panelist positions
    - Discuss what has been done and needs to be done to make solutions **target multiple architectures**
    - Examine **disruptions from technology trends**, such as the increasing bandwidth gap between HBM and DDR, or what happens when NVM is added to the mix.
    - Wrap up

Session 3

Performance, Portability and Productivity: Definitions & Metrics
Session Lead: John Pennycook, Intel
Adoption of a set of common definitions and metrics for performance, portability and productivity is a necessary step towards productive debate, discussion and collaboration between application developers, the developers of libraries/frameworks and hardware vendors.

This breakout session aims to answer the following questions:
- Are shared definitions and metrics useful to the community? Why (or why not)?
- What do performance, portability and/or productivity mean to you?
- Are you/your site currently tracking performance, portability and productivity of codes? How?
- How do metrics and definitions inform solutions to these problems in a software sense?
- How do metrics and definitions inform solutions to these problems in a hardware sense?

Session 4

Productivity Issues in Preparing Large Codes for Performance Portability
Session Lead: Anshu Dubey, Argonne National Laboratory
Note Taker: Emily Simpson, NNSA
In obtaining performance portability, large multiphysics codes that were developed during the age of distributed memory bulk synchronous parallel model face some challenges that are similar to legacy codes, and some others that are unique to them. Many of these codes contain bits of legacy codes in them. However, the biggest fear for such codes is the durability and support for the abstractions they adopt, because even non-invasive modifications to the code could be quite large. Thus choosing the right abstractions is critical for their productivity. Other challenges include formulation of a process that allows gradual ramp-on of the abstractions, verification during transition, and co-existence of production with transition. All of these challenges relate to productivity during adoption rather than research into abstractions. I am proposing a break-out session to discuss these issues where participants can share their experiences, bring their own concerns to the discussion, and perhaps find a few workable solutions.

Wednesday

Session 4 - Abstractions, DSL

Title: The Flexible Computational Science Infrastructure (FleCSI)
Author: Ben Bergen, LANL
FleCSI is a compile-time configurable framework designed to support multi-physics application development. As such, FleCSI attempts to provide a very general set of infrastructure design patterns that can be specialized and extended to suit the needs of a broad variety of solver and data requirements. Current support includes multi-dimensional mesh topology, mesh geometry, mesh adjacency information, n-dimensional hashed-tree data structures, graph partitioning interfaces, and dependency closures.
FleCSI also introduces a functional programming model with control, execution, and data abstractions that are consistent with both MPI and state-of-the-art task-based runtimes such as Legion and Charm++. The FleCSI abstraction layer provides the developer with insulation from the underlying runtime, while allowing support for multiple runtime systems, including conventional models like asynchronous MPI. The intent is to give developers a concrete set of user-friendly programming tools that can be used now, while allowing flexibility in choosing runtime implementations and optimizations that can be applied to architectures and runtimes that arise in the future. The control and execution models in FleCSI also provide formal nomenclature for describing poorly understood concepts like kernels and tasks. To provide a low barrier to entry, the FleCSI data model does not lock developers into particular layouts or data structure representations, thus providing a low-buy-in approach that makes FleCSI an attractive option for many application projects. This presentation will cover recent accomplishments of applications using FleCSI.

HiHAT, a way forward to perf portability with retargetable infrastructure
Author: CJ Newburn, NVIDIA
Performance portability tends to denote the ability to migrate code from one platform to another without change. Some argue that such a thing is impossible, on the basis that tasks may need to be specialized for each target, and that the layout for data that tasks operate on may need to be customized across targets. In this talk, I'll claim that this is not a contradiction. Perf portability is achievable by retaining a common SW architecture across targets, by using common heuristics in target-agnostic schedulers that use different cost models for each target, and by having the output of the schedulers be customized sequences of a common set of target-agnostic primitives for managing data, moving data, invoking work, coordinating work, and querying the platform. This will be illustrated in an initial implementation of a retargetable infrastructure called HiHAT, for Hierarchical Heterogeneous Asynchronous Tasking. We'll show that overheads for the HiHAT abstraction are low, that there is a clean mapping from HiHAT's target-agnostic APIs to targets like CUDA that enables porting with low effort, and that HiHAT's retargetability enables performance improvements for sample applications like Molecular Orbitals on alternate architectures like GPUs. HiHAT is a community-wide effort that aspires to target all HW platforms and runtimes relevant to this community, for example to enable a common code base that executes well on Intel CPUs, IBM POWER CPUs, ARM CPUs, and NVIDIA GPUs.

Recent experiences with RAJA nested loop abstractions and CHAI
Author: Adam J. Kunen (Lawrence Livermore National Laboratory)
Recent efforts to port ARDRA to GPU accelerated platforms using a combination of the RAJA programming model, and the Copy Hiding Application Interface (CHAI), have led to some interesting performance and portability issues

related to the interaction between CHAI and the nested-loop constructs in RAJA. We discuss some of these issues in detail, and discuss our reimplementation efforts to address these issues.

Kokkos' Task-DAG Parallel Capabilities & Evolution of Kokkos' Back-ends
Author: H. Carter Edwards, SNL
Kokkos' support for parallelism began with a parallel execution over a simple one-dimensional range of indices. We then added hierarchical thread-team parallelism which provided applications with portable access to GPU's grid/block parallelism and associated shared memory. Our two most recent additions include parallel execution over a multidimensional range of indices (e.g., loop collapse) and
parallel execution of a directed acyclic graph of tasks (task-dag). We present our two new portable task-dag capabilities: parallel execution of an application's kernel on a static work-graph and
parallel execution of a dynamic task-graph where an application may exercise a heterogeneous collection of kernels and dynamically introduce new tasks and dependences as the task-graph executes.

Kokkos defines programming model abstractions necessary for applications to achieve performance portability across diverse next generation platform (NGP) architectures. Kokkos' back-ends map these abstractions to NGP target architectures using potentially vendor-preferred programming mechanisms to obtain the best performance. These back-ends are regularly updated to provide applications with improved abstractions and to leverage improved programming mechanisms; e.g., OpenMP v3 to v4.5 and CUDA v7 to v9. We are able to provide applications with backward compatibility through these updates because Kokkos' programming model abstractions have proven to be future-proof. We review the evolution of Kokkos' abstractions and back-ends, and present plans to incorporate new programming mechanisms to further improve performance.

Session 5 - Applications Experience 2

Performance Portability in SPARC – Sandia's Hypersonic CFD Code for Next-Generation Platforms
Authors: Micah Howard, Andrew Bradley, Steve Bova, Travis Fisher, James Overfelt, Derek Dinzl, Ross Wagnild (SNL)
This talk with focus on the work being done with SPARC to achieve a performance portable hypersonic CFD code. SPARC is being developed to support the aerodynamic design and analysis needs of the nuclear weapons enterprise and run efficiently on NNSA's CTS and ATS platforms. The software design considerations we've encountered and addressed to attain performance portability will be discussed, along with challenges that remain. Performance analyses will be presented for CTS-1, ATS-1 and ATS-2-like systems. A candid briefing of the high and low points of large-scale application code development in the context of performance portability will also be given.

Portability and Performance of the Nekbone Mini-App
Authors: Scott Parker, Sudheer Chunduri, Ronald Rahaman (Argonne National Laboratory)
This talk presents the results to date of a performance comparison of the Nekbone mini-app on systems with CPU and GPU based architectures. Nekbone is a CORAL mini-app that contains the principal computational and communication kernels of the CFD code Nek5000. Nek5000 is a high order, incompressible Navier-Stokes solver based on the spectral element method that has been heavily used at Argonne and other HPC centers. Nekbone solves a standard Poisson equation using a conjugate gradient iteration with a simple preconditioner on a block geometry, which represents one of the core kernels of Nek5000. Nekbone is written primarily in Fortran and has been threaded using OpenMP and tuned for the SIMD architecture on the Intel Xeon Phi "Knights Landing"

processor. The GPU implementation is based on OpenACC and CUDA Fortran of the compute-intensive matrix-matrix multiplication part, which significantly minimized the modification of the existing CPU code while extending the simulation capability of the code to GPU architectures. We present the performance results from both CPU and GPU systems and we compare them with an analytical performance model that allows for fine-grained quantification and analysis of the kernels' performance.

Performance Portability Experiences at NERSC
Authors: Brian Friesen (LBNL), Rahulkumar Gayatri (LBNL), Zahra Ronaghi (LBNL), Thorsten Kurth (LBNL), Brandon Cook (LBNL), Jack Deslippe (LBNL), Balint Joo (JLab), Frank Winter (JLab/U. of Regensburg), Sabbir Khan (ODU), Christian Trott (SNL)
We present experiences, preliminary performance results, and lessons learned from efforts to implement performance portable techniques in various computational kernels derived from the NERSC Exascale Science Application Program (NESAP), which represent a significant portion of the NERSC workload. Among the various kernels we have implemented several performance portable strategies, including compiler directives, libraries, DSLs, and template frameworks. Finally, we discuss the challenges, opportunities, and practical aspects of each approach for potential adopters.

Portability Initiatives for Scientific Computing and Simulation: Molecular Dynamics as a Case Study
Authors: Arnold Tharrington, Ada Sedova (ORNL)
When large application software projects with long expected lifetimes must be significantly rewritten for rapidly evolving HPC architectures, this can result in an unproductive development cycle. There are the obvious extra expenses in terms of man-hours, but in addition, the resulting code becomes more error-prone due to multiple authors and shortened debug-test lifetimes. Furthermore, writing code that is closer to machine-level is a more difficult task, and thus fundamentally more prone to error. Here we discuss some possible solutions and posit best practices developed from our own efforts to create dedicated portable scientific computing applications for HPC, using mini-apps and libraries for molecular dynamics simulations of non-bonded forces as a test case. We explore the application re-design process, incorporation of standard libraries, and the use of the directive-based APIs such as openACC for the short-range non-bonded forces calculation, and share our experiences in the creation of a specialized lower-level portable API library for the Multilevel Summation Method (MSM) long-range electrostatics algorithm.

Session 6 - Hierarchical Memory, off-node, I/O

SCR and Preparing for Burst Buffers
Author: Elsa Gonsiorowski, LLNL
The burst buffer storage layer presents new challenges and opportunities to HPC applications. These opportunities are most easily realized by the checkpoint / restart use case. SCR is a scalable checkpoint / restart library which augments checkpointing codes. Its main goal is to improve application performance by exploiting and managing multilevel checkpoints. To improve application portability to new systems, the SCR development team is actively implementing support for *all* of the latest burst buffer systems. These implementations will take advantage of vendor-specific technologies thereby providing a unified interface for applications.
This presentation will focus on our experiences with two types of burst buffers:
- the machine-global burst buffer that is exemplified in the LANL Trinity system with Cray Datawarp technology
- the node-local burst buffer that is exemplified by the LLNL Early Sierra systems with IBM BBAPI technology

We will discuss the challenges faced and lessons learned. We will also review general software improvements that have been implemented in the last year and our involvement in the ECP VeloC project.

Toward portable I/O performance by leveraging system abstractions of deep memory and interconnect hierarchies

Author: Francois Tessier, Argonne National Laboratory

Supercomputers are increasingly being architected with deeper and diverse memory subsystems such as on-node SSD, MCDRAM and network-attached burst buffers, and interconnect topologies such as dragonfly and Tori. In order to scale an application's I/O performance, one needs to fully exploit these system characteristics for data movement. Our research tackles this area and a key focus is on effective abstractions of the various memory and storage tiers, as well as the network topology, to ensure scalable and portable performance. We will discuss the efficacy of our approaches on two diverse systems - A Cray XC40 with a Lustre filesystem and an IBM Blue Gene/Q system with a GPFS filesystem.

Session 7 - Languages, Compilers, Frameworks, Tools

Performance Analysis and Optimizations for Lambda-based Applications in OpenMP 4.5

Authors: Carlo Bertolli, IBM; Contributions from: compiler team at IBM and Toronto; Tom Scogland, Ian Karlin, Holger Jones, Bronis de Supinsky, and others at LLNL.

As we are approaching the final passes of CORAL's project delivery for LLNL and ORNL, there is an increased focus in supporting performance portability through high-level frameworks such as Raja. While Raja is being extended to include an OpenMP 4 backend, and it is being used across various applications, we expect to see reports on issues only found at full application scale, which were not addressed when looking at simpler benchmarks.

This talk addresses what are some of the expected issues at full application scale, including support and performance challenges that involve both the OpenMP standard and its implementation in the CORAL Clang/LLVM based compiler for the OpenPower architecture. We will present some examples of these issues that have been shown by the OpenMP community and DoE lab scientists. First, performance of reduction on host and device is greatly limited by inability to use an OpenMP reduction clause in a Raja templated traversal function. Second, handling of Raja lambda parameters requires breaking target-independent feature of the high level application code when running on a GPU device, due to data mapping requirements. Third, there are overheads when using a lambda-based loop body in a traversal method as compared to a vanilla OpenMP program with verbatim loop body.

We will review solutions, describe those that we implemented in our Clang/LLVM compiler with the aim of providing input to OpenMP and C++ communities, and show support and performance improvements that we obtained. Finally, we will show our roadmap towards further expanding performance portability of Raja on OpenPower.

Early Results of OpenMP 4.5 Portability on NVIDIA GPUs

Author: Jeff Larkin, NVIDIA

Over the past year multiple implementations of OpenMP 4.5 have emerged for common HPC architectures. CLANG, XL, Cray, and GCC have all added support for OpenMP offloading to NVIDIA GPUs. In this talk I will present the current state of these implementations based on results using simple benchmark codes. I will discuss the relative performance of each implementation and whether developers can obtain performance portability with the current state of the art compilers and hardware. The talk will focus primarily on the portability between different compilers on NVIDIA GPUs and also portability with traditional CPU architectures.

Adaptive Mesh Refinement for Exascale

Author: Max Katz (NVIDIA); Collaborators: Ann Almgren, John Bell, Brian Friesen, Andy Nonaka, Weiqun Zhang (LBL); Maria Barrios-Sazo, Alan Calder, Don Willcox, Mike Zingale (Stony Brook University); Adam Jacobs (Michigan State University); Chris Malone (LANL)

In this talk we will review performance portability progress for AMReX. AMReX is developed in an ECP Co-Design Center effort to support block-structured adaptive mesh refinement simulations at large scale. Recent development efforts in AMReX have focused on exposing a programming model that is both scalable and straightforward to use on diverse architectures. We will explain how the AMR framework we use can be naturally extended to support these various architectures without requiring substantial changes in the simulation codes that are based on the framework. Finally, we will discuss recent performance results on systems including traditional CPUs, the Intel MIC architecture, and NVIDIA GPUs.

Performance portability via Nim metaprogramming

Author: Xiaoyong Jin, James Osborn (Argonne National Laboratory)

We tackle performance portability via the extensive metaprogramming capabilities of the Nim programming language. High-level Nim code is compiled to C/C++ which provides high performance and great C interoperability, including full access to vector intrinsics, OpenMP pragmas, and even CUDA. Nim's flexible syntax and AST-based metaprogramming system allows one to easily create expressive DSL frameworks that hide the complexity of performance portability, while keeping high maintainability. We are building a high-level framework, QEX (Quantum Expressions), as an implementation of the data parallel API being developed as part of the Lattice QCD ECP application team. In this talk we describe how we use metaprogramming to transform code blocks into CUDA kernels. We will also present our strategy for performance optimizations for CPU and GPU architectures along with some benchmarks.

Portable Heterogeneous High Performance Computing via Domain-Specific Virtualization

Author: Dmitry I. Liakh, National Center for Computational Sciences, Oak Ridge National Laboratory

We present an experimental approach to a performance portable HPC application development based on the concept of domain-specific virtualization of HPC platforms. Our approach summarizes the previous efforts along this direction, adds novel elements, and builds up a formal structure on top. In essence, we describe an extensible framework, called the domain-specific virtual processor, which encapsulates the individual node architecture as well as the HPC system scale. An abstract domain-specific virtual processor provides a portable interface connecting domain-specific algorithms with physical hardware, thus separating algorithm expression from the hardware and system specifics. The abstract domain-specific virtual processor can then be specialized to a concrete computational domain via introducing a concrete virtual processor architecture subsequently mapped to the physical hardware. The concrete virtual processor architecture is static from the point of view of domain algorithms. However, underneath it has a flexibility to customize its structure to the specific node architecture and HPC scale. The latter is encapsulated by imposing a recursive (hierarchical) view on the underlying HPC platform. As a use case, we demonstrate the applicability of the domain-specific virtualization in the domain of numerical tensor algebra.

Title: High Performance Atomics are Critical for Thread Scalability

Authors: H. Carter Edwards, Christian Trott, Simon Hammond (SNL)

Vendors of multicore and manycore next generation platforms (NGPs) must provide high performance atomic operations for their NGP. Both the C and C++ language standards have recognized the importance of atomic operations by standardizing the API for these operations. These operations are critical for performance portable and thread scalable algorithms; and enable simpler and thus more maintainable parallel algorithms. For example, parallelizing a force computation with a simple loop over a spatial discretization requires either replacing sums into

shared variables with atomic sum operation or redesign the algorithm and introduce both code and performance overhead. An alternative, graph coloring, introduces the coloring step and splits the loop into an serial outer loop over smaller single-color loops which is requires deeper changes and often provides lower performance. Another alternative thread-private arrays replace a single large share array with large, redundant per-thread temporary arrays and introduces a reduction step over those arrays. In our experience, parallelizing algorithms with non-trivial data-update or execution patterns is tractable with atomic operations versus far more complex alternatives. For example, we have implemented in Kokkos performance portable and thread scalable hash table and scheduler for executing a directed acyclic graph (DAG) of tasks.

We will present our algorithmic use cases for atomic operations, performance results for these operations on NGPs, and in-progress proposal for the C++20 standard to support atomic operations for high performance computing.

Towards a portable OpenMP data sharing implementation for NVIDIA accelerators in the CLANG/LLVM toolchain
Author: Gheorghe-Teodor Bercea, IBM; Contributors:Carlo Bertolli, Hyojin Sung, Arpith C. Jacob, Alexandre Eichenberger, Tong Chen, Kathryn O'Brien, Kevin O'Brien.
The ability to implicitly share data between threads is a critical portability feature of the OpenMP programming model for both host- and device-side workloads that may target accelerators. In this presentation we tackle the implementation of OpenMP's implicitly shared data model using the shared memory space of NVIDIA GPUs. There are several reasons that make NVIDIA a challenging platform for implementing OpenMP's implicit data sharing: (1) the native CUDA model support assumes the data is local to the thread unless otherwise specified, (2) NVIDIA GPUs do not allow thread-local variables from being shared amongst threads and (3) the lack of native hardware support for a fork-join mechanism similar to that of a CPU. The data sharing support has been implemented as part of the CLANG/LLVM compilation toolchain by adding extensions to: CLANG, the LLVM NVPTX backend and the runtime library libomptarget. The baseline of our implementation, to be later this year, introduces a new CLANG-level code generation scheme for offloading OpenMP target regions to NVIDIA GPUs. The scheme contains a software solution to the abovementioned fork-join issue in which OpenMP regions executed by different numbers of threads are outlined into separate functions. The calls to the outlined functions are controlled using a master-slave approach: the master delegates work to be cooperatively executed by a pool of slave OpenMP threads. For correctness reasons, variables 'shared' across different outlined functions executed by potentially disjoint groups of threads (master and workers) would have to be allocated in either the shared or global memory of the device. For performance, notwithstanding occupancy issues, shared memory has lower latency than global memory. Coupling that with NVIDIA's native support for shared memory variables, the GPU shared memory is ideal for implementing implicit OpenMP data sharing. The drawback of using shared memory is its limited availability. Future work intends to resolve that by using global memory instead.

The LLVM NVPTX backend is architecture specific but it should be independent of the user's choice of programming model for targeting the accelerator be it CUDA, OpenMP or otherwise. The current LLVM NVPTX backend supports the lowering of variables to thread local memory unless explicitly specified otherwise. Since this is only compatible
with the CUDA programming model we extend the LLVM NVPTX backend with the ability to detect shared variables and implicitly lower them to the shared memory space instead of the thread local one. This has been performed by augmenting the LLVM NVPTX backend with a new pass and changing several others. The changes to both CLANG and LLVM are to be upstreamed to their respective public repositories and are currently being reviewed by the LLVM community.

Breakout Day 2

Session 1 & 2 (2 parallel sessions):

OpenMP 5.0 features
Session Leads: Tom Scogland, LLNL / Kevin O'Brien, IBM
This breakout will discuss what functionality is desired from OpenMP and how it would help portability. There will be multiple people from the OpenMP affinity subgroup present to answer questions and collect feedback on desired performance features.

Session 3

SC Facilities Performance Portability Best Practices Website
Session Lead: Jack Deslippe, LBNL
The three Office of Science HPC Facilities--ALCF, NERSC, and OLCF--are working together to provide guidance and best practices on performance portability for our users. We will present a prerelease version of a joint website being developed by the Centers and gather feedback for future improvements. Our intent is to create a living resource, with concrete guidance and interactive tools to guide implementation and optimization for performance portability, updated regularly with evolving HPC software and hardware technology. While targeting our facilities' users, the resource will be publicly available. We will draw from multiple sources, including our respective applications-readiness programs and specific ongoing code exercises concerned with performance portability. We will also document techniques for measuring absolute performance and evaluating how close that performance is to the maximum attainable hardware limits, given the memory and compute characteristics of the code. We hope the discussion will yield constructive criticism of content and approach and identify gaps.

Session 4

ISO/C++17 and Beyond - Parallelism and Concurrency
Session Lead: H. Carter Edwards (SNL)
The ISO/C++ standard is evolving to portably enable on-node parallelism and concurrency.
C++11 added foundational CPU parallel-execution mechanisms such as atomics, threads, mutexes, and asynchronous dispatch. C++17 extended the algorithms library to with policies that permit these algorithms to execute in parallel and allow for vectorization. This evolution will continue with C++20 (hopefully) adding executors that specify how and where parallel execution
will occur, improvements to atomic operations, coroutines with well-defined suspension/resumption semantics suitable for concurrent execution, and (finally) true multidimensional arrays whose memory-layout can be easily and transparently adapted to the
underlying node architecture.

Six DOE laboratories (ANL, LANL, LBNL, LLNL, ORNL, SNL) are participating in the ISO/C++ standard committee and advocating for enhancements that will improve our applications' ability to use C++ for performance-portable on-node parallelism.

This breakout will discuss current and anticipated C++ standard features for parallelism, concurrency, and HPC; applications' requirements; and gap analysis to provide guidance for DOE Lab representatives on the ISO/C++ Standard committee.

Thursday

Vendor Performance Portability Panel

Moderator: David Bernholdt, ORNL
Panelist:

- John Levesque, Cray
- Geraint North, ARM
- John Pennycook, Intel
- CJ Newburn, NVIDIA
- Kathryn O'Brien, IBM

Session 8 - Applications Experience 3

Experiences Utilizing CPUs and GPUs for Computation Simultaneously on a Heterogeneous Node
Author: Olga Pearce, LLNL
Much of our porting efforts for Sierra are focused on running the computation on the GPUs, which will comprise >90% of the FLOPS of the system. This talk will describe our approach to utilizing the remaining FLOPS by running a portion of the computation on the CPUs simultaneously with the GPU computation. We will present a proof-of-concept implementation in ARES, a multiphysics ALE-AMR code at LLNL. ARES is written in C/C++ with MPI, and uses RAJA for parallelization within MPI processes. As a portability layer, RAJA enables us to utilize the same source code for both the CPU and the GPU. Our implementation divides the work between the computing resources via domain decomposition, and utilizes all cores of the CPU and all of the GPUs on the node for computation. Load balancing is necessary to use the heterogeneous resources effectively. We will present preliminary results on early delivery pre-Sierra machines at LLNL.

Performance Portable Halo and Halo Center Finding in HACC
Authors: Li-Ta "Ollie" Lo, Jonathan Lee, Adrian Pope (LANL)
Efficiently finding and computing statistics about "halos" (regions of high density) are essential analysis steps for N-body cosmology simulations. The Hybrid / Hardware Accelerated Cosmology Code (HACC) is designed as an MPI+X code and the analysis operators are parallelized for both distributed and shared-memory parallelism. In this talk, we present portable data-parallel algorithms for several variations of halo finding and halo center finding algorithms. These are implemented with the Nvidia's Thrust library of which a subset of the API has been standardized as the parallel algorithm library in C++17. This allows a single implementation to be compiled to multiple backends to target a variety of multi-core and many-core architectures. Finally, we compare the performance of our halo and center finding algorithms against the original HACC implementations on the Moonlight, Stampede, and Titan supercomputers with new timing results on Cori and Theta.

Scaling Post-meshing Operations on Next Generation Platforms
Authors: Roshan Quadros, Brian Carnes, Madison Brewer, Byron Hanks (SNL)
To enable advanced analysis workflows on next generation platforms, it is important to develop scalable in-situ post-meshing capabilities. In the analysis workflow, generally post-meshing operations such as refinement, smoothing, and projection are required after the mesh generation step. In this study, an MPI+ Kokkos [1] hybrid programing model has been used to scale these in-situ post-meshing operations on heterogeneous hardware architectures. Kokkos is a performance portable library that

supports both CUDA and OpenMP runtime libraries for GPU and KNL, respectively.
The refinement operator is used to scale up meshes to higher resolution without writing to disk. The core loop operation splits each element in a mesh into multiple smaller elements using templates. This operator is commonly used in generating a series of refined meshes for solution verification. In this study we compared performance of refinement of block-structured meshes on multiple platforms (GPU and KNP) using MPI+Kokkos in order to determine scalability of refinement and to provide guidelines on its usage.
The smoothing operator is generally used to improve the element quality without changing the mesh connectivity. In this study, we looked at smoothers based on a gradient-based optimization of an element quality metric that is a sum of local element quality metrics. Significant code refactoring was needed to be able to use Kokkos for this algorithm. We present results on shared memory architectures such as KNL and GPU using Kokkos to demonstrate smoothing scalability.
A projection operator is generally used after the refinement and smoothing steps. The projection operator projects the refined and/or smoothed nodes on to the boundary curves and surfaces of the geometric model. In this study, the OpenNURBS library has been used to represent the geometric model. OpenNURBS is well suited for next generation platforms as it is lightweight, easy to port, and provides thread safe query APIs. In this study, mesh nodes were projected on different curve and surface definitions on a Trinity testbed. Both the number of MPI ranks and Kokkos (OpenMP) threads were varied to find an optimal number of MPI ranks per KNL node and number of Kokkos threads per MPI rank. A near linear scaling was observed while varying both MPI ranks and Kokkos threads. A maximum speedup of 77X was observed on a 72 core KNL using hyper threads.
Work is underway to integrate the in-situ post-meshing operations with the SPARC compressible CFD code to simulate reentry problems using structured and unstructured meshes on next generation testbeds. This has begun with refinement and will continue with projection and smoother operators.

Parallelization and Performance of the NIM Weather Model for CPU, GPU and MIC Processors
Author: Mark Govett (NOAA), Jim Rosinski, Jacques Middlecoff, Tom Henderson, Jin Lee, Alexander MacDonald,
Paul Madden, Julie Schramm, Antonio Duarte
The design and performance of the NIM global weather prediction model is described. NIM was designed to run on GPU and MIC processors. It demonstrates efficient parallel performance and scalability to tens of thousands of compute nodes, and has been an effective way to make comparisons between traditional CPU and emerging fine-grain processors. Design of the NIM also serves as a useful guide for finegrain parallelization of the FV3 and MPAS models, two candidates being considered by the NWS as their next global weather prediction model to replace the operational GFS.
The F2C-ACC compiler, co-developed to support running the NIM on GPUs, has served as an effective vehicle to gain substantial improvements in commercial Fortran OpenACC compilers. Performance results comparing F2C-ACC with commercial GPU compilers demonstrate their increasing maturity and ability to efficiently parallelize and run next generation weather and climate prediction models.
This paper describes the code structure and parallelization of NIM using F2C-ACC, and standards compliant OpenMP and OpenACC directives. NIM uses the directives to support a single, performanceportable code that runs on CPU, GPU and MIC systems. Performance results are compared for four generations of computer chips. Single and multi-node performance and scalability is also shown, along with a cost-benefit comparison based on vendor list prices.
https://www.esrl.noaa.gov/gsd/ato/Jan2016_BAMS_NIM_Parallelization_and%20Performance-preprint.pdf

Portability and Scalability of Sparse Tensor Decompositions on CPU/MIC/GPU Architectures
Author: Keita Teranishi, SNL
Tensors have found utility in a wide range of applications, such as chemometrics, network traffic analysis, neuroscience, and signal processing. Many of these applications have increasingly large amounts of data to process

and require high-performance methods to provide a reasonable turnaround time for analysts. In this work, we consider decomposition of sparse count data using CANDECOMP-PARAFAC alternating Poisson regression (CP-APR) with both multiplicative update and quasi-Newton methods. For these methods to remain effective on modern large core count CPU, Many Integrated Core (MIC), and Graphics Processing Unit (GPU) architectures, it is essential to expose thread- and vector-level parallelism and take into account the memory hierarchy and access patterns for each device to obtain the best possible performance. In this presentation, we will discuss the optimization and observed performance of the methods on modern high-performance computing architectures using the Kokkos programming model, overhead incurred by portability, and implications for upcoming distributed solver development.

Melodee: Solving ODEs with platform-specific code generation.

Authors: Robert C Blake III, Tom O'Hara, David F. Richards (LLNL)

Computational cardiac modeling requires solving trivially-parallel systems of ordinary differential equation systems (ODEs) at every point in the tissue. These ODEs are computationally expensive, extremely complex, and under constant scientific revision. The optimizations needed to achieve optimal performance vary substantially from platform to platform, resulting in code that is difficult for developers to port and hard for scientists to modify. To achieve our performance, portability, and usability goals we have developed Melodee, a domain specific language for describing ODEs and generating platform-specific solution code. Our code generators can generate code for IBM BlueGene/Q, Intel Xeon, Intel Knights Landing, and NVidia GPU architectures. This talk will discuss specific per-platform optimization strategies such as layout of data, automatic simd-ization, symbolic manipulation, and automatic rational polynomial generation. We'll also discuss optimizations that don't work as expected. We outline how Melodee can be used outside of cardiac modeling and list a few of the many possible applications.

Poster Session

Progress Porting ALE3D to the GPU

Arlie Capps, capps2@llnl.gov and Peter Robinson, robinson96@llnl.gov)

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.

ECP Kokkos Support Tutorials and Hackathons

Authors: H. Carter Edwards (SNL), Fernanda Foertter (ORNL), Galen Shipman (LANL), Christian Trott (SNL)

The Exascale Computing Project (ECP) is funding a Kokkos Support effort that includes bootcamps / hackathons to accelerate applications' integration of Kokkos for performance portability across diverse next generation platforms (NGPs). Our first bootcamp was held in May'17 with 30+ participants from LANL, ORNL, SNL, LBNL, and NVIDIA who represented 11+ projects. Bootcamp participants had strongly positive feedback for both the tutorial and hackathon sessions. During the hackathon sessions we identified a new parallel abstraction for Kokkos and brainstormed strategies for interoperability with FORTRAN applications. We participated in the ORNL/OLCF organized GPU hackathon at Brookhaven in June'17 where, in five few days, the STAR TPC team used Kokkos to rewrite a performance critical component of their application for performance portable thread parallelism across IBM Power, Intel Haswell & KNL, ARM, and NVIDIA GPU architectures.
This new version can meet its performance requirements executing on a single NVIDIA Pascal GPU (with 2x margin) versus the legacy version that required more than 20 CPU nodes to achieve its throughput needs. In addition our regular tutorials, with hands-on exercises, have been well-received at Supercomputing and GPU Technology conferences.

We will share lessons learned regarding how these tutorials and hackathons have accelerated application teams' ability to transition to Kokkos and develop performance portable code; and resulting plans forimprovements.

Overlapping Data Movement and Compute with XLF and OpenMP 4: Experiences in UMT

Author: David Appelhans, IBM

This talk will describe 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.

SharP: Towards Programming Hierarchical-Heterogeneous Memory based Extreme-Scale Systems

Manjunath Gorentla Venkata, Computer Science and Mathematics Division, Oak Ridge National Laboratory

The exascale systems are expected to have a significant amount of hierarchical and heterogeneous on-node memory, and this trend of system architecture in extreme-scale systems is already seen in pre-exascale systems. Along with hierarchical-heterogeneous memory, the system typically has a high-performing network and a compute accelerator. This system architecture is not only effective for running traditional High-Performance Computing (HPC) applications (Big-Compute), but also running data-intensive HPC applications and Big-Data applications such as data analytics, social network analysis, machine learning, and genomics. As a consequence, there is a growing desire to have a single system serve the needs of both Big-Compute and Big-Data applications.

Though the system architecture supports the convergence of the Big-Compute and Big-Data, the programming models have yet to evolve to support either hierarchical-heterogeneous memory systems or the convergence. Achieving this would require (1) simple, usable, and portable abstraction for hierarchical-heterogeneous memory, (2) unified programming constructs for Big-Compute and Big-Data applications, (3) portability across diverse (GPU-based and Xeon-Phi) memory hierarchy systems, and (4) native support for data-centric abstractions.

The poster describes, SharP, a data-structure based and data-centric programming construct as a solution for these challenges. 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).

Performance portability of numerical time integrators in SUNDIALS library
Authors: Slaven Peles, John Loffeld and Carol Woodward (Lawrence Livermore National Laboratory)
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.

Experiences with performance portability across ARM microarchitectures
Author: Geraint North, ARM
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.

High-performance GPU code generation for high-order stencils: Alleviating register pressure
Authors: Prashant Rawat, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan (Ohio State University)
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.

TTLG: Tensor Transpose Library for GPUs

Authors: Aravind Sukumaran-Rajam, Arjun Suresh, Jyothi Vedurada, Sriram Krishnamoorthy, and P. Sadayappan (Ohio State University)

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.

MACSio Development and Proxy Application Validation

Author: Elsa Gonsiorowski, LLNL

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.

OpenACC/OpenMP4.5 Interoperability

Lixiang Luo (lixiang.luo@ibm.com), IBM Research, ORNL/IBM CoE

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.

AutoPar: Semantics-Aware Automatic Insertion of OpenMP directives
Author: Chunhua "Leo" Liao , LLNL
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.