Increasing Availability and Impact of Compiler Technology for HPC

Mary Hall, University of Utah

Migrating compiler and programming languages research for HPC into practice has always been difficult, due to a number of factors:

- Compilers are complex software systems, and even standard programming languages such as C++ and Fortran continue to evolve. Therefore, building research compilers that can apply to large-scale applications requires that such compiler efforts track language changes, support a broad set of applications through their lifetimes, and also demonstrate better performance or higher productivity than other available compilers.
- Vendor-specific compilers tend to produce higher-quality code than open source compilers, but they are closed systems for which it is difficult to extend or leverage their capability.
- Open source compilers, and particularly Clang/LLVM which is gaining traction in HPC, have gaps in their capability as compared to research and vendor compilers.
- HPC users are a small market, and therefore, it is difficult to motivate industry to invest in compiler technology beyond what is required to demonstrate performance of new architectures. This required compiler technology may be at a lower level of abstraction than programmers might desire.

The Exascale Computing Program represents a unique opportunity to invest in compiler technology for HPC and migrate the potential performance portability offered by compiler technology and programming systems at a time when architectures are changing radically. However, is this investment sufficient? Instead, should we be leveraging where the large industry investment in compilers is happening, in compilers for deep learning?

This proposed talk will cover three topics. First, what are the opportunities for compiler technology to ease the performance portability challenges of moving to exascale architectures? In particular, we discuss a layered approach that is gaining traction, where a computation's specification is separated from its mapping to hardware, and models or autotuning are used to derive the architecture-specific mapping.

Next, we plan to discuss two paths for building compiler technology for HPC in open source infrastructures. The first path is to continue to develop within the Clang/LLVM infrastructure, as is already a part of the ECP portfolio, including OpenMP and OpenACC support, parallel intermediate representations, loop transformations, a Fortran frontend, just-in-time compilation and better C++ support. This is a pretty extensive list, and the efforts will not be completed by

the end of the ECP program. Nevertheless, it is a collection of support that would have benefits well beyond the DOE community, and therefore it seems likely that these efforts can be done in partnership with others from the Clang/LLVM community.

A second path is to leverage extensive investments in compilers for deep learning applications. Currently, companies including Google, Facebook and Amazon are ramping up large compiler groups to address the needs of deep learning. If one looks closely at the code implementing a neural network, it has abundant data parallelism and data reuse. It has reasonably high arithmetic intensity, but moves significant data through the memory hierarchy that must be carefully managed. Target architectures include not just high-end systems, but also desktop and handheld devices, and therefore, the implementation and even the computation must be tailored to the target architecture. There are also many different frontends that have emerged that should target a common intermediate representation. To solve scalable deep learning problems requires a distributed computing solution. *In fact, the challenges for deep learning compilers have significant overlap with what is needed for the HPC community.*

Among the efforts focused on deep learning compilers, of particular interest is Google's recent introduction of the Multi-Level Intermediate Representation, or MLIR. MLIR is part of the Tensor Flow framework, but has roots in its leadership to LLVM, as well as a capability to lower MLIR to LLVM. A key idea in MLIR is a set of higher-level abstractions (e.g., tensors) that permit MLIR to perform higher-level optimizations more naturally than at the C-like IR abstractions offered in LLVM. Specifically, MLIR is a better representation to reason about array-based computations and parallelism than LLVM. MLIR is new, and therefore, does not have the ecosystem of Clang/LLVM. Most notably, it does not have a C/C++/Fortran frontend, and may not represent all computations of interest to the HPC community. Therefore, it is not a replacement for the Clang/LLVM infrastructure, but there is potential to use it initially in domain-specific tools with an eye towards evolving to support high-level optimization.

For either Clang/LLVM or MLIR, the diversity of exascale architectures will force the HPC community to address performance portability and improvements in open source compiler technology. Engaging a broader community by partnering with industry and other HPC stakeholders may be the only way to remove gaps in open source compilers as compared to state of the art, and move technology forward towards performance portable solutions. A final topic for the proposed talk is to discuss a framework for building this community.