C++ Abstraction Layers – Performance, Portability and Productivity

D. Dinge, S.D. Hammond, C.R. Trott and H.C. Edwards
Center for Computing Research
Sandia National Laboratories
Albuquerque, New Mexico, USA
{dcdinge, sdhammo, crtrott, hcedwar}@sandia.gov

ABSTRACT

Programming applications for performance and portability across modern computing architectures is a critical and challenging goal for HPC developers. A significant programming burden is the need to use hardware-specific programming models to obtain the greatest performance. However, these programming models often do not offer high economic value as development, debugging and performance profiling are expensive endeavors.

Directive-based programming solutions (e.g., OpenMP and OpenACC) have been developed for over two decades as a "lightweight" approach to annotating application source code with declarations for parallel execution and, most recently, data movement. While this approach has been successful at specifying the dispatch of parallel kernels for execution, it has had considerably less success at describing the data structure transformations required for performant execution. Where such transformations have been demonstrated they are often verbose.

Sandia National Laboratories has been investigating an alternative approach to performance portable programming: the Kokkos C++ programming model and library [**?**, **?**]. Kokkos provides abstractions for an application programmer to specify parallel execution dispatch, data movement, and data structure layout. High performance and low overhead is achieved through the use of C++ templates for compile time specifications. Abstractions for dispatch and data layout also provides extreme levels of portability and extensibility ("tune-ability") for each target architecture. In recent benchmarking tests Kokkos-based mini-applications were able to demonstrate greater than 90% the performance of hardware native programming solutions using a single application source code.

In the initial development of Kokkos many of the programming abstractions were seen as intrusive to application source code, a potential downside of the use of C++ library as opposed to a directive-based programming model. In this poster we compare the original OpenMP version of the LULESH benchmark from Lawrence Livermore National Laboratory [**?**, **?**] to a Kokkos-based version. This work evaluates the performance of these versions compared to the original code, as well as a description and quantitative evaluation of the modifications required to the source code. A number of subsequently optimized versions are described with resulting performance improvements.

We show runtime and source code modifications for three important classes of architecture: (1) conventional multi-core CPUs including Sandy Bridge, Haswell and POWER8 processors; (2) state-of-the-art NVIDIA K40 and K80 GPUs, and (3) Intel's Knights Corner (KNC) Xeon Phi. The KNC is being used by Sandia National Laboratories as a prototype system for the forthcoming *Trinity* supercomputer which will contain next-generation many-core Knights Landing (KNL) Xeon Phi sockets. In most cases we are able to demonstrate performance which either matches or exceeds the original OpenMP implementation of the LULESH benchmark that was optimized for execution on the BlueGene/Q architecture in approximately equivalent modified lines of code.

This work has an important contribution in the on-going discussion of performance portable programming models. It is almost unanimously agreed that developers want to see portable and efficient solutions for future generation computing architectures. However, selection of any single programming model is clouded by a lack of clear information regarding the balance between the performance benefits versus the development and maintenance cost associated with porting to that model. The results in this poster show that we are able to provide greater benefits to the programmer with similar or lower overhead than a directive-based strategy.

The C++ language standard is actively investigating new features for on-node parallelism and data structure layout. Our goals for Kokkos are to prototype potential C++ features to insure both performance portability across architectures and application developer productivity, advocate for these features in future C++ language standards, and provide a path forward for our current scientific application portfolio.

1. REFERENCES

[1] H.C. Edwards, D. Sunderland, V. Porter, C. Amsler, and S. Mish. Manycore Performance-Portability: Kokkos Multidimensional Array Library. *Scientific Programming*, 20(2):89–114, 2012.

[2] H.C. Edwards, C.R. Trott, and D. Sunderland. Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014.

[3] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.

[4] I. Karlin, J. Keasler, and R. Neely. LULESH 2.0 Updates and Changes. Technical Report LLNL-TR-641973, August 2013.