C++ Abstraction Layers – Performance, Portability and Productivity

Sandia National Laboratories
D. Dinge[1], S.D. Hammond[2], C.R. Trott[2] and H.C. Edwards[2]
1) Scientific Apps & User Support, 2) Center for Computing Research, Sandia National Laboratories, NM, USA

Sandia National Laboratories

Problem Description and Context

Programming applications for performance and portability across modern computing architectures is extremely challenging for many application developers. Hardware-native programming models may provide much higher performance but may only run on one vendor or even one machine. Standardized cross-platform models provide greater porting potential but often much lower performance.

In this work we evaluate the use of the Kokkos C++ programming model in providing a performance portable implementation of the LULESH hydrodynamics mini-application. We compare the performance and programmer productivity of selecting Kokkos over using OpenMP directives. For this purpose we evaluate programmer productivity from the perspective of the number of source code lines changed and the number of places ("sites") in the code where changes have to be made.

Our performance results use the LULESH Zones/Second Figure-of-Merit (FOM) metric for Intel Haswell, Sandy Bridge and Knights Corner, IBM POWER8, ARM64 and NVIDIA Kepler GPUs

Conclusion: Kokkos can provide performance portability with similar amounts of code changes to directive-based solutions

Case Study: The LULESH Hydrodynamics Mini-Application

LULESH is a shock hydro-dynamics mini-application developed by Lawrence Livermore National Laboratory (LLNL) for exploring future computer architectures and advanced programming models. Several versions of the code have been developed including versions which utilize OpenMP, CUDA, MPI and other novel programming models.

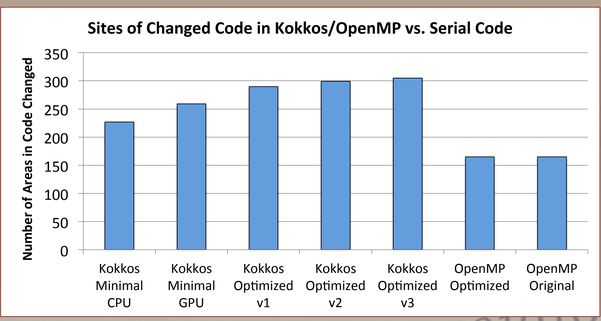
The code is available from http://codesign.llnl.gov/

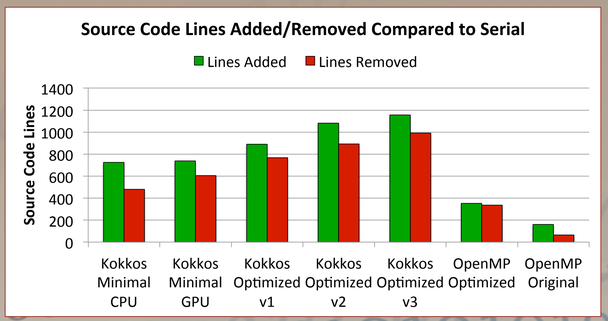
Original OpenMP

```
#pragma omp parallel for firstprivate(length, rho0, q_cut)
for (Index_t i = 0 ; i < length ; ++i){
   Index_t elem = regElemList[i];

   if ( delvc[i] <= Real_t(0.) ) {
      Real_t ssc = ( pbvc[i] * e_new[i]
      + vnewc[elem] * vnewc[elem] * bvc[i] * p_new[i] ) / rho0 ;

      if ( ssc <= Real_t(.1111111e-36) ) {
         ssc = Real_t(.3333333e-18) ;
      } else {
         ssc = SQRT(ssc) ;
      }

      q_new[i] = (ssc*ql_old[i] + qq_old[i]) ;

      if (FABS(q_new[i]) < q_cut) q_new[i] = Real_t(0.) ;
   }
};
```

Kokkos Implementation using C++11 Lambda Functions

```
Kokkos::parallel_for (length, [=] (const int i) {
   Index_t elem = regElemList[i];

   if ( delvc[i] <= Real_t(0.) ) {
      Real_t ssc = ( pbvc[i] * e_new[i]
      + vnewc[elem] * vnewc[elem] * bvc[i] * p_new[i] ) / rho0 ;

      if ( ssc <= Real_t(.1111111e-36) ) {
         ssc = Real_t(.3333333e-18) ;
      } else {
         ssc = SQRT(ssc) ;
      }

      q_new[i] = (ssc*ql_old[i] + qq_old[i]) ;

      if (FABS(q_new[i]) < q_cut) q_new[i] = Real_t(0.) ;
   }
});
```





What is the Kokkos Programming Model?

Kokkos implements a programming model in C++ for writing performance portable applications targeting all major HPC platforms. The model provides abstractions for parallel execution dispatch and data management. As a backend and compile target Kokkos can use OpenMP, UNIX p-threads and CUDA allowing it to generate efficient, hardware-native executable.

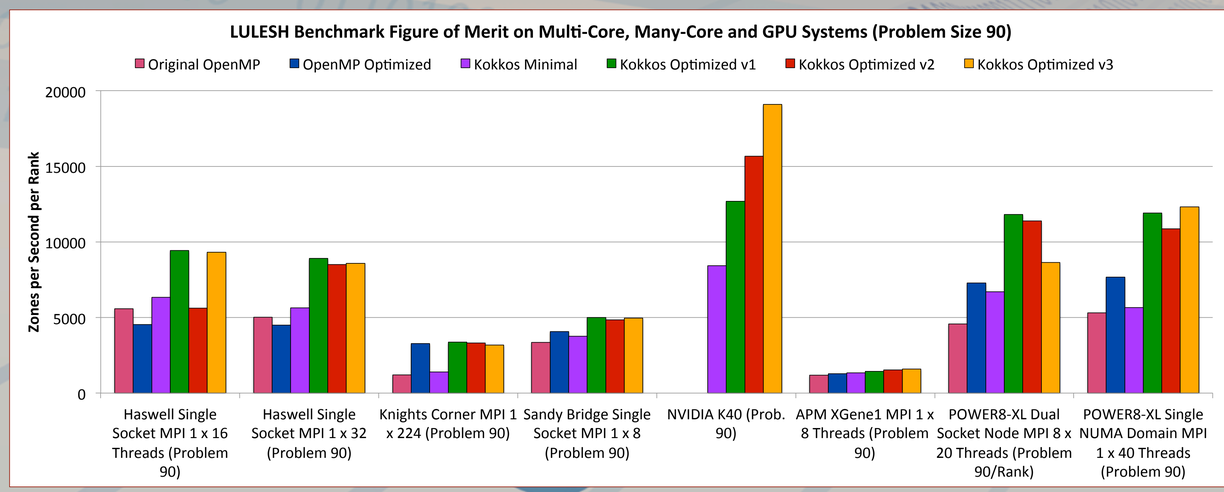
Kokkos can be downloaded from: http://www.github.com/kokkos

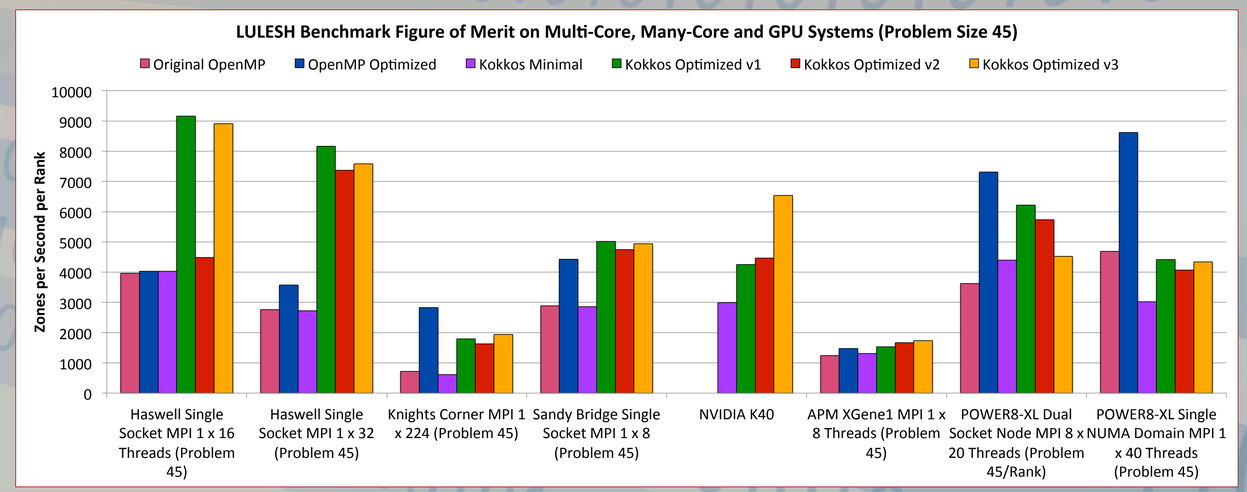
Optimizations to Kokkos LULESH Implementations

Optimized Version 1.0
- Use buffers for temporary arrays
- Reduce register pressure in hourglass by limiting call to 1-dimension
- Atomics in hourglass

Optimized Version 2.0
- Use RandomAccess Views for domain
- Switch some structures to 2D views
- Use team parallelism for hourglass calculation

Optimized Version 3.0
- Fuse kernels for energy calculations
- Fuse kernels for EOS evaluation for elements





National Nuclear Security Administration
U.S. DEPARTMENT OF ENERGY

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.