Kokkos: Enabling Manycore Performance Portable Applications and Libraries

Hybrid parallel architectures offer the opportunity for terascale workstations, petascale clusters, and exascale supercomputers. Developers of “MPI+X” computational applications and libraries are challenged by the diversity of manycore devices with architecture specific programming models and performance requirements. This challenge has traditionally been addressed by developing and maintaining multiple versions of architecture-tuned versions of codes. Redundant code development and maintenance is not an acceptable strategy. Instead we created the Kokkos library to enable applications and libraries to have a single code version that is portable across multicore CPUs, Intel Xeon Phi, and NVidia Kepler GPUs; and achieve no less than 90% performance of the architecture-tuned version of their code. Kokkos is unique in that it only uses standard C++ (no compiler extensions) and provides portable array data structures with optimal, architecture-tuned, memory access patterns hidden behind a portable API.

Portable Data Parallel Execution
- Dispatch work (computation+data) to a device
- Parallel reduction is just this simple, for a user

```
// Execute the DOT C++ functor on the device
// using a parallel reduction driver
parallel_reduce(nwork, DOT<device>(x,y), result );
```

```
template < class Device >
struct DOT {
    typedef Device device_type; // execute on device
    typedef double value_type ; // reduction type
    // Called thread-parallel with iw ∈ [0..nwork):
    void operator()(int iw, value_type & val) const
    { val += X(iw) * Y(iw); }
    // Initialize thread-private temporary values:
    void init( value_type & val) const { val = 0 ;}
    // Join two thread-private temporary values:
    void join( volatile       value_type & val ,
              volatile const value_type & in )const
    { val += in ; }
    // Views to arrays:
    const View< const double*, device_type > X, Y ;
};
```

```
// Views to arrays:
const View< const double*, device_type > X, Y ;
```

```
// Using C++1998 standard and “functor” pattern to maximize portability
// Using C++2011 standard with lambda feature would result in simpler syntax
```

```
// Read array data through GPU texture cache
View< const double*[3][8], Device, RandomRead > x;
```

```
// Correct versus wrong array layout
// Using and not using GPU texture cache
```

```
// Read array data through GPU texture cache
View< const double* [3], Device, RandomRead > x ;
// Device = Kepler GPU then use texture cache
// otherwise use normal data access
```

```
// Views to arrays:
const View< const double*, device_type > X , Y ;
```

```
// Requiring C++1998 standard and “functor” pattern to maximize portability
// Using C++2011 standard with lambda feature would result in simpler syntax
```

```
// Performance of miniMD force computation
// Correct versus wrong array layout
// Using and not using GPU texture cache
```

```
// Performance of miniMD force computation
// Correct versus wrong array layout
// Using and not using GPU texture cache
```

FUNDING AGENCY: ASC / Computational Systems and Software Environment (CSSE)

FUNDING ACKNOWLEDGEMENT: 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.

CONTACT: H. Carter Edwards, SNL, hedwar@sandia.gov [SAND2013-8091P]

The Kokkos library is publically available through the Trilinos project at trilinos.sandia.gov.