Proxy App Use cases at Sandia

Christian R. Trott ¹, Simon Hammond ², Carter Edwards ¹

¹ 1426 Scalable Algorithms, Sandia National Laboratories

Proxy App Use cases at Sandia

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.

SAND 2015-7000PE
Proxy Apps: Connecting the Pieces

ProxyApp

- Compilers
- Programming Models
- Hardware
- Runtimes
- Applications
- Tools
- Libraries
- Frameworks

Proxy App Usecases at Sandia
Evaluating Programming Models

- Starting 2011 use of proxy apps to experiment with on-node programming models
- MiniFE: 20+ variants in 10+ programming models (Serial, OpenMP, Cuda, OpenCL, Cilk, TBB, Kokkos, ...)
- Use to inform development of Kokkos and perform comparisons with native programming models
- Now: evaluation of inter node tasking models

Develop Transition Strategies

- Prototype algorithms for thread scalability
- Develop and demonstrate incremental approach to adopt new architectures
- Use of miniDrivers to accelerate porting of libraries

Vendor collaboration

- Mini-apps are ideal for early hardware evaluation (simulators and early silicon)
- Utilized in procurements for benchmarks
Ca. 2011:

- Becoming clear that medium term future will not allow MPI-only
- New programming models coming out seemingly every few months: Cuda, OpenCL, OpenACC, TBB, Cilk, C++AMP ...
- Which ones are viable? How do they perform? How portable are they? How hard to use?
- Not possible to evaluate in production codes: need miniApps to give space for rapid experimentation
- Start of DOE CoDesign centers and related miniApp efforts: Mantevo, ExMatEx, ExaCT, CESAR

Ca. 2014:

- Wide range of implementations are available, can run many miniApps on all hardware platforms.
- Understanding of strengths and weaknesses of different approaches exists
- Knowledge has been used to design new programming models for C++ (Raja, Kokkos)
- Using miniApps to validate new models (e.g. I. Karlin, et al. "Lulesh programming model and performance ports overview." Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep (2012).)
Focus on simple Krylov solver (CG)
Think of it as Sandia’s Stream benchmark
If this doesn’t work well no point in going further
Probably most variants of any MiniApp (30+)
Direct connection to LAMMPS (same people, same code style, same algorithm)

Challenging piece of code for compilers and architectures

No single bottleneck, and sensitive to almost all performance characteristics of an architecture

Figure 14: Time for 200 iterations of miniFE-Kokkos CG-solve on the Shannon GPU testbed. The problem size is weak scaled, with 8M elements per device. The solid line represents runs using the GPU-Direct capabilities of MVAPICH2-1.9, while the dashed line shows results with manual deep copies during the communication phase. For each data point the best time out of 12 runs was used.

Figure 15: Total time consumed for running 1,000 simulation steps of a 2,048,000 atom Lennard-Jones simulation with miniMD variants on different test-beds. Note that a horizontal line indicates perfect strong scaling and an upward trend indicates a loss in parallel efficiency. The solid lines represent runs using miniMD-Kokkos, while the dashed lines show results with peer variants. For each data point the average of the best 8 runs out of 12 was used.
Most of MiniMD’s characteristics are perfectly reflected in LAMMPS

But: Xeon Phi significantly different relative performance

Reason are less gathers due to single atom type implementation of MiniMD
Renewed focus on task runtimes

- Dharma project at SNL California: evaluate different tasking models including Legion, Uintah and Charm++
- LLNL summer students ported miniApps for tasking frameworks
- LANL looking at Legion+Kokkos
- Interested in Resilience and Load-Balancing
Time is running out

- Current DOE leadership class machines get replaced in 2016 and 2018
- New platforms use XeonPhi (Trinity @ LANL/SNL, NERSC8 @ NERSC, ... @ ANL) or GPUs (Summit @ ORNL, Sierra @ LLNL)
- Threading is required for good performance
- Need to start porting now: Decisions have to be made
- Sandia: MPI + OpenMP and Kokkos; medium term Kokkos to get to Coral platforms

Guiding application teams

- Minimal threading experience in application teams
- Need to run education programs
- Need to develop guidelines for porting
- Incremental approaches necessary
Part of a TriLab (LLNL, LANL, SNL) CoDesign Milestone
Our take: demonstrate phases of adopting Kokkos aligned with machine lifecycle
5 Variants were written: Minimal CPU, Minimal GPU, Opt 1-3
Demonstrate performance improvements related to necessary code changes

Developed Steps for Migrating Apps:
- Add Kokkos::parallel_for with capture by reference
- Mitigate write conflicts with atomics
- Capture by value
- Move data structures to use Kokkos allocations
- Introduce Multi-Dimensional Views with Layouts and Traits
- Use new algorithms where necessary
Initial introduction of Kokkos

- Support CPU like architectures only
- Replace loops with `parallel_for` and `parallel_reduce`
- Resolve write conflicts with atomic operations
- No change to data structures
- Not portable!

Adding `parallel_for` (34x)

```cpp
// replace
for (Index_t i = 0; i < numElem; ++i) {
    ...
}

// with
Kokkos::parallel_for (numElem, [&] (const Index_t& i) {
    ...
});
```
Using atomic_add for scatter add (27x)

// replace
for (Index_t lnode = 0; lnode < 8; ++lnode) {
    Index_t gnode = elemToNode[lnode];
    domain.fx(gnode) += fx_local[lnode];
    domain.fy(gnode) += fy_local[lnode];
    domain.fz(gnode) += fz_local[lnode];
}

// with
for (Index_t lnode = 0; lnode < 8; ++lnode) {
    Index_t gnode = elemToNode[lnode];
    Kokkos::atomic_add(&domain.fx(gnode), fx_local[lnode]);
    Kokkos::atomic_add(&domain.fy(gnode), fy_local[lnode]);
    Kokkos::atomic_add(&domain.fz(gnode), fz_local[lnode]);
}

Taking Error Checks out of Loops (8x)

// replace
for (Index_t i = 0; i < numElem; ++i) {
    ...
    if (domain.v(i) <= Real_t(0.0))
        MPI_Abort(MPI_COMM_WORLD, VolumeError);
}

// with
int check_error = 0;
Kokkos::parallel_for (numElem, [&] (const Index_t& i) {
    ...
    if (domain.v(i) <= Real_t(0.0)) {
        check_error = 1;
    }
});
if (check_error)
    MPI_Abort(MPI_COMM_WORLD, VolumeError);
Adding GPU Support

- Use capture by value
- Make member functions const
- Replace std::containers with other data structures and use Kokkos::malloc for allocations

Classes need const members and function markup

```c++
// replace
class Domain {

    Real_t &x(const Index_t idx) { return m_x[idx]; }
};

// with
class Domain {

    KOKKOS_INLINE_FUNCTION Real_t &x(const Index_t idx) const { return m_x[idx]; }
};
```

Replace std::vector with Kokkos::vector

```c++
// replace
class Domain {

    Kokkos::std<Real_t> m_x;
};

// with
class Domain {

    Kokkos::vector<Real_t> m_x;
};
```
Avoid reallocation of temporary buffers 28x

```c
// replace
Real_t *fx_elem = Allocate<Real_t>(numElem8);
Real_t *fy_elem = Allocate<Real_t>(numElem8);
Real_t *fz_elem = Allocate<Real_t>(numElem8);
...
Release(&fz_elem);
Release(&fy_elem);
Release(&fx_elem);
// with
ResizeBuffer((numElem8*sizeof(Real_t)+4096)*3);
Real_t *fx_elem = AllocateFromBuffer<Real_t>(numElem8);
Real_t *fy_elem = AllocateFromBuffer<Real_t>(numElem8);
Real_t *fz_elem = AllocateFromBuffer<Real_t>(numElem8);
```

Reduce register pressure by separating independent calculations

```c
// replace
...
x[.] = ..;
y[.] = ..;
CalcElemFBHourglassForce(x[.], y[.], zd1, hourgam, coefficient,
                         hgfx, hgfy, hgfk);
// with
...
x[.] = ..;
...
CalcElemFBHourglassForce(x[.], hourgam, coefficient, hgfx);
...
y[.] = ..;
...
CalcElemFBHourglassForce(y[.], hourgam, coefficient, hgfy);
...
Start using Views with Traits and Layout

//add
Kokkos::View<\texttt{const} Real_t *, Kokkos::MemoryTraits<Kokkos::RandomAccess>> m_c_x;

KOKKOS_INLINE_FUNCTION Real_t c_x(\texttt{const} Index_t idx) \texttt{const} { return m_c_x[idx]; }

//replace
Real_t hourmodx = 
x8n[i3] * G.gamma[i1][0] + x8n[i3 + 1] * G.gamma[i1][1]
+ x8n[i3 + 2] * G.gamma[i1][2] + x8n[i3 + 3] * G.gamma[i1][3]
+ x8n[i3 + 4] * G.gamma[i1][4] + x8n[i3 + 5] * G.gamma[i1][5]
+ x8n[i3 + 6] * G.gamma[i1][6] + x8n[i3 + 7] * G.gamma[i1][7];

//with
Real_t hourmodx = 0.0;
for(int j = 0; j < 8; j++)
    hourmodx += x8n(i2,j) * G.gamma[i1][j];

Utilize TeamPolicy where appropriate

//replace
Kokkos::parallel_for("CalcFBHourglass_B", numNode, KOKKOS_LAMBDA(const int gnode) {
    Real_t fx_tmp = Real_t(0.0);
    for (Index_t i = 0; i < count; ++i)
        fx_tmp += fx_elem[cornerList[i]];
    domain.fx(gnode) += fx_tmp;
}

//with
Kokkos::parallel_for("CalcFBHourglass_B", Kokkos::TeamPolicy<>((numNode+127)/128,team_size,2),
    KOKKOS_LAMBDA (\texttt{const} typename Kokkos::TeamPolicy<>::member_type& team) {
    const Index_t gnode_begin = team.league_rank() * 128;
    const Index_t gnode_end = (gnode_begin + 128 < numNode)? gnode_begin + 128:numNode;
    Kokkos::parallel_for(Kokkos::TeamThreadRange(team,gnode_begin,gnode_end),[&] (\texttt{const} Index_t& gnode) {
        reduce\_double3 f_tmp;
        Kokkos::parallel_reduce(Kokkos::ThreadVectorRange(team,count),[&](\texttt{const} Index_t& i,double3& tmp) {
            tmp.x += fx_elem[cornerList[i]]
        },f_tmp);
        Kokkos::single(Kokkos::PerThread(team), [&] () { domain.fx(gnode) += f_tmp.x; });
    });
Merge Kernels

- Reduces data transfers (temporary arrays are replaced with temporary scalars)
- Reduces scheduling overhead

```c++
// replace
Kokkos::parallel_for("EvalEOSForElems_AA", numElemReg, KOKKOS_LAMBDA(const int i) {
    //do stuff AA
});
Kokkos::parallel_for("EvalEOSForElems_BB", numElemReg, KOKKOS_LAMBDA(const int i) {
    //do stuff BB
});

// with
Kokkos::parallel_for("EvalEOSForElems", numElemReg, KOKKOS_LAMBDA(const int i) {
    //do stuff AA
    //do stuff BB
});
Count sites of modification and number of lines changed

![Source Code Lines Added/Removed Compared to Serial](image.png)

- Lines Added
- Lines Removed
Preliminary data, some of it (in particular on Power) with not yet great Software stack
Some of the testbeds show not yet understood significant performance variances
**FENL: addressing Matrix assembly with Kokkos**

- MiniFE++: focus on Matrix assembly
- Implement gather sum and scatter add algorithms for Matrix assembly
- Use Newton iterations to exercise it.

**Expanding FENL into a miniDriver**

- Utilize Trilinos Tpetra data structures
- Exercise Trilinos Solvers (including AMG)
- Allow for UQ data types
- Now: expand to use Sierra data structures for assembly, and Sierra solvers
Benefits of miniApps

- OpenSource as opposed to export controlled or even classified
- Faster turn around time: an intern can write a new version
- Often small enough to allow run in simulators.
- Little dependencies: can be exercised on new software stacks

Kokkos and the AMD Kalmar Compiler

- Collaboration to develop Kokkos backend for APUs
- Use MiniApps to drive development and test results
- 40 hours ago: got Lulesh-Minimal-GPU and Lulesh-Opt-1 working
- Identified several bugs in compiler as well as new requirements to enable higher optimization levels
Evaluating pre-release versions to head of issues

- Tpetra is Trilinos 2nd generation distributed linear algebra interface
- Started to move to Kokkos in 2013
- Utilizes UVM for NVIDIA platforms to deal with complex data management issues (by ignoring them)
- Performance issues identified as compared to MiniFE which didn’t use UVM - but neither requires data transfer during CG-Solve

**Time for CG-Solve vs number of nodes (Weak scaling problem)**
Replicate issue in MiniFE

- To hard to eliminate use of UVM in Tpetra ¿ add UVM to MiniFE
- Replicate On-Node issue
- Investigate with Profiler
- Finding launch overhead unrelated to data transfer
- Send findings to NVIDIA

Profiler comparison of UVM and noUVM variant
Fix came with driver update

![Graph showing performance comparison between MiniFE and Tpetra with and without UVM support]

- **Time in s**
  - no-UVM
  - 331.22
  - 331.56

- **Applications**
  - MiniFE
  - Tpetra

---

Proxy App Usecases at Sandia

23/24
Proxy Apps are an important tool of collaboration
More nimble than real apps
Allow for experimentation which is hard to do in production code
Develop strategies before starting to modify production codes
But: wider spectrum from skeleton apps to miniDrivers is needed to cover different needs
Proxy Apps are an important tool of collaboration
More nimble than real apps
Allow for experimentation which is hard to do in production code
Develop strategies before starting to modify production codes
But: wider spectrum from skeleton apps to miniDrivers is needed to cover different needs

*ProxyApps are here to stay!*