Obtaining Threading Performance Portability in SPARTA using Kokkos

Stan Moore
DOE COE Performance Portability Meeting
April 20, 2016
SPARTA

(Stochastic Parallel Rarefied-gas Time-accurate Analyzer)

- Direct Simulation Monte Carlo (DSMC) code
- Models rarefied gas flows using particles
- Features *in situ* meshing and visualization
- Core developers are Steve Plimpton and Michael Gallis (Sandia)
SPARTA (cont.)

- Relatively **new** code—first public release in July 2014
- Written using **object-oriented C++** (~45,000 lines)
- Parallelized using **MPI** and domain decomposition
- Easily extensible using **C++ virtual inheritance**—reduces code duplication
- **No third-party libraries**
Kokkos

(Greek for *kernel* or *grain*)

- Provides *abstractions* (in C++) for both *parallel execution* of code and *data management*
- Designed to target complex node architectures with *N*-level memory hierarchies and *multiple types of execution resources*
- Currently can use *OpenMP*, *Pthreads*, and *CUDA* as backend programming models
- Core developers are *Carter Edwards* and *Christian Trott* (Sandia)
- Open-source, [https://github.com/kokkos/kokkos](https://github.com/kokkos/kokkos)
Kokkos (cont.)

- In practice, Kokkos allows SPARTA to:
  - Use threading on top of existing MPI parallelization (MPI + X)
  - Run and give reasonable performance on multithreaded CPUs, Xeon Phis, and GPUs
- Kokkos abstractions only require a single C++ code base
Initial Porting Strategy

- Leverage (reuse) the original SPARTA code instead of a complete rewrite
- Keep the Kokkos version as similar to the original MPI code as possible
- Initially parallelize kernels using simple parallel loops (no thread teams)
- Find and optimize bottleneck kernels, adding more complexity to the Kokkos code if necessary (ongoing)

“Premature optimization is the root of all evil” –Donald Knuth
Initial Porting Strategy (cont.)

- Kokkos package is an optional add-on to SPARTA
- Uses C++ virtual inheritance and functions to reduce code duplication
- First such optional package in SPARTA
- Patterned after the LAMMPS molecular dynamics code, which has 61 optional packages (including a similar Kokkos package)
- So far, particle moves without complex surfaces have been ported, along with the collide routine
Kokkos Porting Workflow

1. **Profile** code to find bottlenecks
2. **Identify** a kernel to be threaded (in SPARTA: typically loops over particles or grid elements)
3. Change kernel data structures to Kokkos **views**
4. Change kernel loop to Kokkos **parallel for, reduce, or scan**
5. Make changes, if necessary, to ensure the kernel is **thread-safe** (modify kernel or use atomics)
6. Test code on **CPU/Xeon Phi**
7. Add in **memory transfer** between **host** and **device**
8. Test code on **GPU**
Incremental Approach

- Kokkos *dual views* contain a reference to data in *device* (e.g. GPU) memory as well as a mirror copy on the *host* (e.g. CPU)
- Can easily *sync* between host and device copies
- Non-Kokkos code runs on the *host*
- SPARTA uses *primitive* memory allocation (no std::vector)
- *Data structures* are allocated using Kokkos, and pointers to the data structures in non-Kokkos code are *set to* point to the Kokkos *host view*
- This allows non-Kokkos portions of the code to still run with zero or little modification (may need some memory transfer for GPU)
Results

- Collisional benchmark with 10 million particles for strong scaling
- Used 1 MPI task (1 Sandy Bridge CPU) per K20X GPU
Challenges—Code Maintenance

- Since Kokkos threading package is an optional add-on to SPARTA, need to prevent divergence between original MPI and Kokkos code versions
- Must periodically synchronize changes and bug fixes
- Regression testing can help catch changes to main SPARTA that break the Kokkos package
- Without Kokkos, would still need a CUDA version and an OpenMP version of SPARTA
Challenges—Specialization

Case study: atomics

- **Statistics**, such as number of collisions, number of cell crossings, etc. are **collected** inside thread parallel loops.
- For thread safety, can either use a **parallel reduction** over threads or an **atomic** fetch-and-add to global variables.
- On K20X GPU, atomics are 10% **faster** than parallel reduction (overall for a collisional benchmark problem).
- On BGQ, atomics are 7% **slower** than parallel reduction.
- How much code **complexity** is a 10% gain in performance worth? (However, little differences add up.)
- Chosen solution: add **command line option** to toggle between parallel reduction and atomics.
Conclusions

- Using Kokkos in SPARTA gives reasonable threading performance on multiple platforms.
- Kokkos allows one to leverage existing C++ code—a complete rewrite isn’t necessary and an incremental porting approach is possible.
- Some code specialization for different platforms will probably always be necessary to gain maximum performance.
Questions?