The latest in Tpetra: Trilinos’ parallel sparse linear algebra

Mark Hoemmen
Sandia National Laboratories
23 Apr 2019
SAND2019-4556 C (UUR)
Tpetra: parallel sparse linear algebra

- “Parallel”: MPI + Kokkos
- Tpetra implements
  - Sparse graphs & matrices, & dense vecs
  - Parallel kernels for solving $Ax=b$ & $Ax=\lambda x$
  - MPI communication & (re)distribution
- Tpetra supports many linear solvers
- Key Tpetra features
  - Can manage > 2 billion ($10^9$) unknowns
  - Can pick the type of values:
    - Real, complex, extra precision
    - Automatic differentiation
    - Types for stochastic PDE discretizations
Over 15 years of Tpetra

2004-5: Paul Sexton starts Tpetra

2005-10: ill-tested research-ware

Mid 2010: I start work on Tpetra

Kokkos 2.0

Stage 1 rewrite of Tpetra to use Kokkos 2.0

Assemble new Tpetra team; gather requirements

Deprecate & purge for Trilinos 13

2008: Chris Baker redesigns Tpetra; 2009: Kokkos 1.0

Fix bugs; rewrite kernels; deploy Tpetra-based multigrid in app (no threads)

Stage 2: purge old interface, more kernels, some thread-parallel fill

Improve GPU+MPI performance & remove sequential bottlenecks (ongoing)
Tpetra is a team now

- Current team (2017-)
  - Karen Devine
  - Geoff Davidson
  - Tim Fuller
  - Mark Hoemmen
  - Jonathan Hu
  - Kyungjoo Kim
  - Chris Luchini
  - William McLendon
  - Chris Siefert
  - Christian Trott

- Plus contributions by
  - Victor Brunini
  - James Elliott
  - Si Hammond

- & code reviews by many (thanks!)
How does Tpetra use Kokkos?

- Tpetra uses most of Kokkos
- Data structures
  - View: 1-3 D, scalars & integers, device & host
  - KokkosSparse::CrsMatrix (developed 2013 for Tpetra)
  - DualView, StaticCrsGraph, UnorderedMap, UniqueToken
- Algorithms: sort, random
- Computational kernels (kokkos-kernels)
  - Dense BLAS & key sparse kernels
  - Kokkos-kernels started as part of Tpetra
- Many parallel loops (for, reduce, & scan)
  - Set up sparse data structures; (un)pack MPI buffers
  - “Long tail” of thread parallelization
Tpetra performance improvements

- **GPU + MPI performance**
  - SPARC linear solver communication 7x faster on multiple GPUs
  - Aria AFF test ~10x faster on 2 GPUs than ~1 year before

- **Highlights of some issues fixed**
  - 4626: Avoid slow CUDA allocations in iterative linear solvers
    - Reuse allocation via static memory pool (eventually: write allocator?)
  - 4630: Make sure that local kernels use cuBLAS where possible
    - Aria AFF solve 2x faster overall; MultiVector $X^T \cdot Y$ time 23.7s to 1.2s
    - Made P100 np1 slightly faster than Broadwell np36 for full nonlinear iteration (76s vs. 84.5s at the time)
  - 4417,4418: Avoid unneeded atomic updates in MultiVector unpack
    - Improved Tpetra solve times on multiple GPUs
    - Good idea for any parallel Kokkos back-end
Aria AFF performance

AFF Performance Test Comparison

Results as of Sierra 4.51.8 (13 Apr 2019);
GPU solves about 10x slower before
Lessons learned

- CUDA, UVM, & host-pinned View allocations are **SLOW**
  - 100-20,000x (1 ms!!!) slower than HostSpace View allocations
  - (HostSpace only ~ 5x slower than raw new / delete)
  - ~30x slower than pairwise MPI_I{recv,send} exchanges on node
  - we already keep static allocations; considering memory pool

- Overexposing implementation hinders optimization
  - If users can get Kokkos::DualView out of MultiVector, then MultiVector can’t lazily allocate device memory (see above)

- MPI: “CUDA-aware” != fast

- Most benefit: Careful measurement & tedious details
  - e.g., “Are you actually calling cuBLAS?”
  - Kokkos Profiling immensely helpful
Upcoming deprecations

- **LO, GO, & Node (3 of 4) template parameters**
  - Simplify Trilinos’ build logic
  - Improve build times & sizes

- **DynamicProfile (in favor of StaticProfile)**
  - \{E,T\}petra’s sparse graphs & matrices had 3 options historically:
    - DynamicProfile: Expand storage per row on demand, unboundedly
    - StaticProfile: Strict upper bound on # entries / row, at constructor
    - Const graph: No structure changes post construction
  - We will remove DynamicProfile
  - Reduce code complexity (inherited from Epetra)
  - Remove impediments to thread parallelization
  - CrsGraph Export / Import will resize as needed
Tpetra’s next steps

- GPU performance improvements
  - Minimize device-host data movement
  - Minimize device & host-pinned memory allocations
  - Ensure correctness w/ CUDA_LAUNCH_BLOCKING off
- Improve asynchrony & latency hiding
  - Interface to let users exploit CUDA streams
  - Overlap MPI communication with other work
- Simplify access to Tpetra objects’ local data
  - Easy parallel iteration over all (rows, entries, etc.)
  - Scope-based model of accessing local data
Kokkos wish list

- Finish execution space instance (CUDA stream) support
  - See Christian’s intro talk this morning
  - Interfaces often exist in Kokkos, but ignore stream argument
- Improve build times & sizes
  - 2-10 (!) hours for all of Trilinos
  - Decade-long running battle
- Help w/ trade-offs: When is it worth...
  - ... sync’ing data to device? (CUDA)
  - ... launching parallel kernel? (CUDA, OpenMP)
  - Is this an abstraction on top of Kokkos?
- Team sort
Thanks!
Extra slides
Obstacles

- Finding & making time to measure performance
  - Increasing pressure for solver features
  - Fragile build & multiplying configurations
- Trilinos & application build times