Sandia National Laboratories

Exceptional service in the national interest

SAND2016-10846C

Threads & CUDA status of Tpetra & downstream solvers

Mark Hoemmen
Sandia National Laboratories
25 Oct 2016

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

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 NO. 2011-XXXXP

Outline

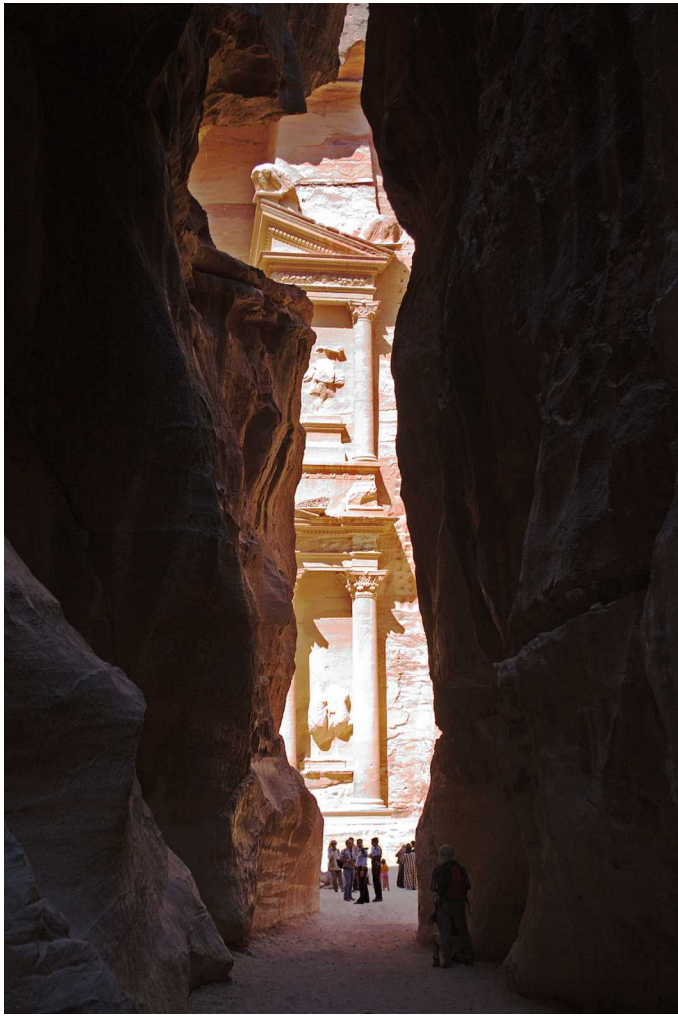
Sandia National Laboratories

- Tpetra's hardware & feature requirements
- Brief history of Tpetra
- Status: Thread-parallelizing solvers & preconditioners
- Status: Support for multiple memory & execution spaces
- Status: Thread-parallel fill

What is Tpetra?

Sandia National Laboratories

- Tpetra, a Trilinos package, implements
  - Sparse graphs, (block) sparse matrices, & dense (multi)vectors
  - Parallel kernels for solving Ax=b & Ax=λx
    - "Local" thread-parallel kernels mostly moved to new package KokkosKernels
  - MPI communication & (re)distribution
  - Fill: {Create,modify} {graph,matrix,vector}
- Key Tpetra features
  - Can solve problems w/ > 10^9 unknowns
  - Can pick the type of values (real, complex, automatic differentiation, ensemble, …)
  - MPI + X (threads) parallelism



Must support > 3 architectures

Sandia National Laboratories

- Systems here (or nearly)
  - Trinity (Intel Haswell & KNL)
  - Sierra (CORAL): NVIDIA GPUs + IBM multicore CPUs
  - Clusters, workstations, etc.
- 3 different architectures
  - Multicore CPUs (big cores)
  - Manycore CPUs (small cores)
  - NVIDIA GPUs
- MPI only, & MPI + threads
  - Threads don't always pay on common CPU architectures
  - Don't slow down MPI-only case in legacy codes

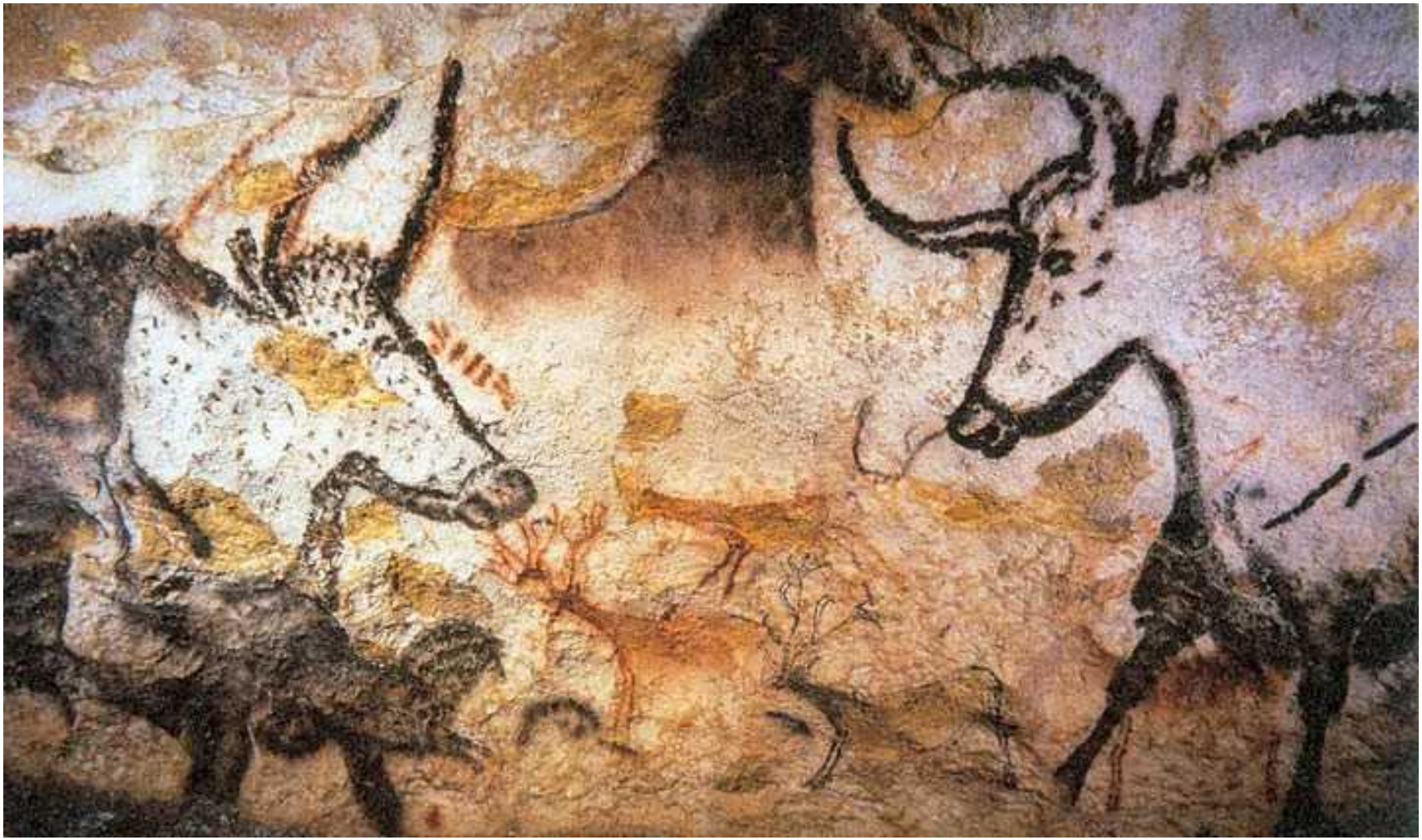
CORAL COLLABORATION
OAK RIDGE • ARGONNE • LIVERMORE

SIERRA

Kokkos as hedge against…

- Hardware divergence
- Parallel programming model
  - OpenMP, OpenACC, CUDA, TBB, Pthreads, Qthreads, …
- Traditional shared memory
  - vs. PGAS / distributed shared
  - e.g., MPI 1-sided
- Threads at all
  - Kokkos' semantics require vectorizable (ivdep) loops
- Kokkos protects our HUGE time investment of porting Tpetra

History of Tpetra



4 Tpetra eras, 3 Kokkos versions

Sandia National Laboratories

- Mike Heroux, Paul Sexton, Kris Kampshoff: 2002-5
  - “Kokkos” (0.x) (package for computational kernels)
  - 2004: Had sparse matrix & dense vector; MPI worked
- Chris Baker, Alan Williams: 2008-10
  - 2008: Massive rename; use Teuchos::RCP etc.
  - 2009: “Kokkos (1.0) Node API”
  - Preconditioners: Ifpack2 (2009), MueLu (late 2010)
- “Productionization” (team effort): 2011-2013
  - Focus on use in an internal application, without threads
  - Fix bugs & improve non-threaded performance of solvers & fill
- Kokkos (2.0) refactor (Hoemmen, Trott): Late 2013 – present
  - Preserve interface, replace data structures & kernels
  - Change interface for thread-parallel fill; finish kernels

Status of solver-related kernels

Sandia National Laboratories

Categories of completeness

Sandia National Laboratories

- Thread-parallel kernel (TPK) exists
  - Our Kokkos kernel, &/or third-party libraries
  - Runs in parallel, correctly, on GPU
- Optimized (OPT)
  - Publications, brief measurements, or “not worrisome”
  - Multicore CPUs, KNL, & GPUs
  - Efforts underway now to make this more systematic
- Deployed (DPL) in Trilinos solvers
  - Not just a prototype code or unattached “bare” kernel
  - Can call it through Amesos2, Belos, Ifpack2, MueLu, etc.
  - If deployed but not thread-parallel yet, still “DPL”

Kernels & solvers done or nearly so

Sandia National Laboratories

Kernel name	TPK	OPT	DPL	Notes
CRS sparse mat-vec	Y	Y	Y	
Dot, Norm, Axp(b)y (update)	Y	Y	Y	(Multi)Vector; for Krylov
GEMV (GMRES orth.)	Y	Y *	Y	Depends on opt'd BLAS
Chebyshev (setup & solve)	Y	Y	Y	
Jacobi (setup & solve)	Y	Y	Y	
Gauss-Seidel (setup & solve)	Y	Y	Y **	Doesn't build by default yet
Sparse triangular solve	Y	Y	N	HTS (OpenMP) in ShyLU now; cuSPARSE for CUDA
BCRS (block) sparse mat-vec	Y	N	Y	OPT in progress
BCRS (block) Jacobi (s. & s.)	Y	N	Y	OPT in progress

TPK: Thread-parallel kernel exists
OPT: Optimized (CPU & GPU)
DPL: Deployed in Trilinos' solvers

Kernels & solvers in progress

Sandia National Laboratories

Kernel name	TPK	OPT	DPL	Notes
Sparse matrix-matrix multiply	Y	Y	N	MueLu setup; DPL FY17
Explicit sparse transpose	N	N	N	MueLu setup (maybe)
Sparse (un)pack for comm	N	N	N	MueLu setup
CRS fillComplete	Y/N	Y/N	Y/N	MueLu setup; some done
{Ex,Im}port setup	N	N	N	MueLu setup
Sparse Cholesky	Y	Y	N	In progress
Sparse LU	Y	Y	N	In progress
ILU(k)	?	?	N	In progress
Line smoothing	N	N	Y	Many tridiagonal solves
BCRS line smoothing	N	N	N	In progress (FY17)
Domain decomp. setup	N	N	N	Computing overlap

Future work (hard stuff)

Sandia National Laboratories

Kernel / solver name	TPK	OPT	DPL	Notes
FastILU (SPAI ILU)	Y	?	N	Fine-grained parallel setup. Needs algorithm research.
Algebraic multigrid aggregation (part of setup)	?	N	N	Needs algorithm research & lots of software work.
{Ex,Im}port execution (MPI comm. from multiple threads)	N	N	N	Depends on MPI support. Rewrite for 1-sided?
Space for your kernels here!				

Status of support for multiple memory & execution spaces

>1 memory or execution spaces

- Our upcoming platforms
  - Trinity (KNL): 2 memory spaces (HBM, DDR4)
  - Sierra: 2 exec & mem spaces (GPUs + multicore CPUs)
- Common hardware features
  - 2 memory spaces: “fast & small” vs. “slow & big”
  - Can access any mem space from any exec space (NUMA)
  - Limited “fast” (<1GB/core)
- Not just Kokkos’ problem: Tpetra, solvers, &/or apps must get involved

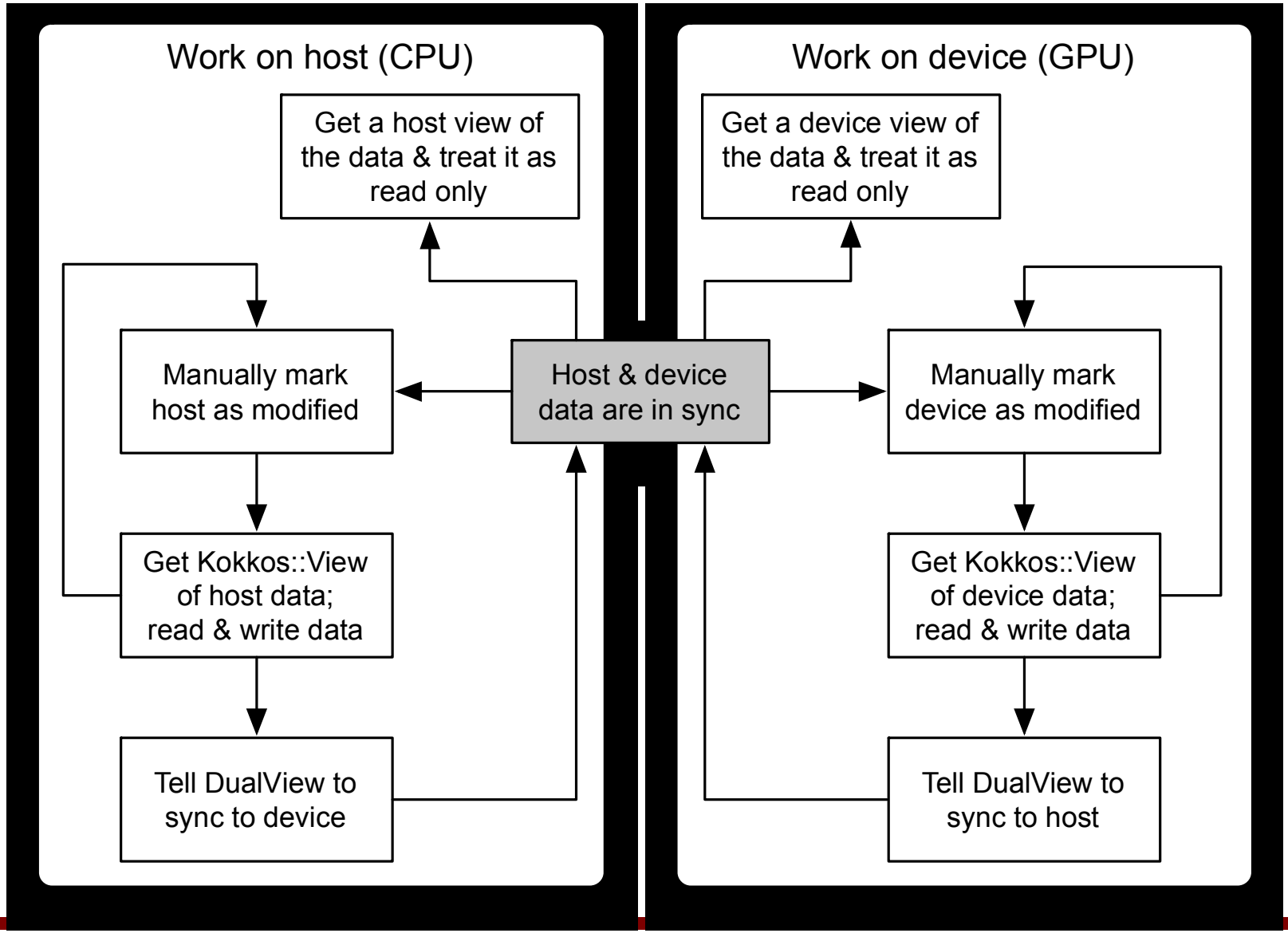
- Support 3 use cases
  1. Gradual porting (mix new Kokkos-ized code with legacy host code)
  2. Use “fast” memory as temp work space (page in / out as needed)
  3. Concurrently use 2 exec spaces (e.g., MPI pack on host, compute on device)

Strategies for 3 use cases

Sandia National Laboratories

- Dual view semantics: (1) & (2)
  - Handles gradual porting (1) & temp workspace (2) use cases
  - Successful use in LAMMPS (interactions btw user vs. GPU modules)
  - Only explicitly manages use of 1 execution space at a time
    - Tpetra *prefers* executing where most recent version of data live
    - Tpetra *may* execute in another space (e.g., overlap pack & compute)
- Concurrent execution in >1 exec spaces? (3)
  - Extension of how Kokkos handles data parallelism in tasks
  - Optional "execution space instance" argument to kernels
  - Exec space instance behaves like "stream" in CUDA
  - Would only expose capability; exploiting it is harder
    - Tpetra could overlap MPI-related (un)pack w/ host compute
    - Solvers & apps could overlap different kinds of work

Dual view semantics



Example of DualView use

Sandia National Laboratories

- Tpetra::Vector<…, Cuda> X (map);
- { // Some code that works on CUDA
  - X.sync<Cuda> (); // copy changes to CUDA device only if needed
  - X.modify<Cuda> (); // we're changing in CUDA space
  - auto X_lcl = X.getLocalView<Cuda> ();
  - parallel_for (X.getLocalLength (), [=] (const LO i) {X_lcl(i) = …});
- } // done w/ CUDA. Don't sync unless will change on host.
- { // Some code that works (only) on host
  - X.sync<Host> (); // copy changes to host memory only if needed
  - X.modify<Host> (); // we're changing in host space
  - auto X_lcl = X.getLocalView<Host> ();
  - parallel_for (X.getLocalLength (), [=] (const LO i) {X_lcl(i) = …});
- } // done changing on host. Skip sync & modify if only 1 space.

Status of dual view semantics

Sandia National Laboratories

- Tpetra & downstream packages currently assume UVM
  - HostMirror::memory_space == memory_space == CudaUVMSpace
    - Haven't yet built Tpetra & downstream w/ 2 different memory spaces
    - Need to fix for using Tpetra for explicit HBM management on KNL
  - A little Tpetra & most downstream code assume host access
    - Forces CUDA_LAUNCH_BLOCKING=1, hindering task parallelism
    - sync() calls fence() ➔ correct use of dual view semantics would relax this
- Kokkos: CUDA ok, HBM (KNL) nearly ready
- MultiVector & Vector done (have dual view semantics)
- Next: BlockCrsMatrix , CrsMatrix, CrsGraph (last)
  - This reflects Petra preferred reuse case
  - Dynamic graph structure changes may be host only for a while
  - Recommended: build Kokkos data structures & pass off to Tpetra

Status of thread-parallel fill

Sandia National Laboratories

Sparse linear algebra use pattern

Sandia National Laboratories

- Fill: Create / modify matrix & vector data structures
  - As many ways to do this as there are applications
  - e.g., iterate over rows, entries, mesh points, elements (FEM), volumes (FVM), aggregates (AMG), …
- Setup for solve (e.g., build preconditioner)
- Solve linear system(s), eigenvalue problems, etc.
  - Coarse-grained computational kernels (e.g., sparse mat-vec)
- Repeat
  - e.g., nonlinear iteration, time stepping, parameter study
  - Petra Object Model & Trilinos solvers optimized for reuse, e.g., of
    - Data structures (graph, basis vectors, allocations) &/or
    - Communication patterns

Thread-parallel fill justifies refactor

Sandia National Laboratories

- Typical use pattern for sparse linear algebra
  - Fill into matrix & vector data structures
  - Setup for solve (e.g., build preconditioner)
  - Solve linear system(s)
  - Repeat
- Need thread-parallel fill because fill & setup not free
  - Lessons from 2011-2013 refactor for applications:
    - Some solves are cheap, so fill & preconditioner setup time matter
    - Most actual runs use few MPI processes, so node performance matters
  - Amdahl's Law: Even if solves take 90% of time with 1 thread, if you have 10 threads, now (sequential) fill & setup are 50%
  - Preconditioners create matrices, so they need fill too

Thread-parallel fill still primitive

Sandia National Laboratories

- Stage 1 plan: “If you want thread-parallel fill, get Kokkos widget out of Tpetra object, & fill into that”
- Advantages
  - Preserve Tpetra interface backwards compatibility
  - Force app devs to buy in (they need to rethink algs anyway)
- Disadvantages
  - Appears to defeat a major purpose of switching to Tpetra
  - “Kokkos widget” is yet another public interface to support
  - Tpetra already hard enough to use (e.g., ~10 LoC to Map)

Thread-safe Tpetra fill partly works

Sandia National Laboratories

- Done for CrsMatrix & (Multi)Vector, for methods that
  - Don't change graph structure (no "insert" yet)
  - Don't cause MPI communication (sumIntoGlobal for off-process rows)
- Return error code / success count; don't throw on error
- No more internal temp array dynamic allocation
- Don't touch Teuchos::RCP reference count
  - Don't call any method that returns RCP, like get*Map()
- sumInto, transform: Atomic update option
  - Default: Use atomic updates if not Serial
- sumInto, replace, transform: Take Kokkos::View / raw ptr
  - Avoids Teuchos::ArrayView debug mode reference count issues

Next steps

Sandia National Laboratories

Next steps

Sandia National Laboratories

- Thread-parallelize all needed solver / preconditioner kernels
- Ensure correct parallelization & good performance of kernels
- Improve interfaces & user experience for thread-parallel fill
- Support concurrent execution in multiple execution spaces
- Think about the right way to do MPI_THREAD_MULTIPLE

Thanks!

Sandia National Laboratories

- Kokkos refactor of Tpetra has been & is a HUGE effort
- Tpetra is only the beginning – lots of solver effort too
- At least half of the building has contributed in some way

Extra slides

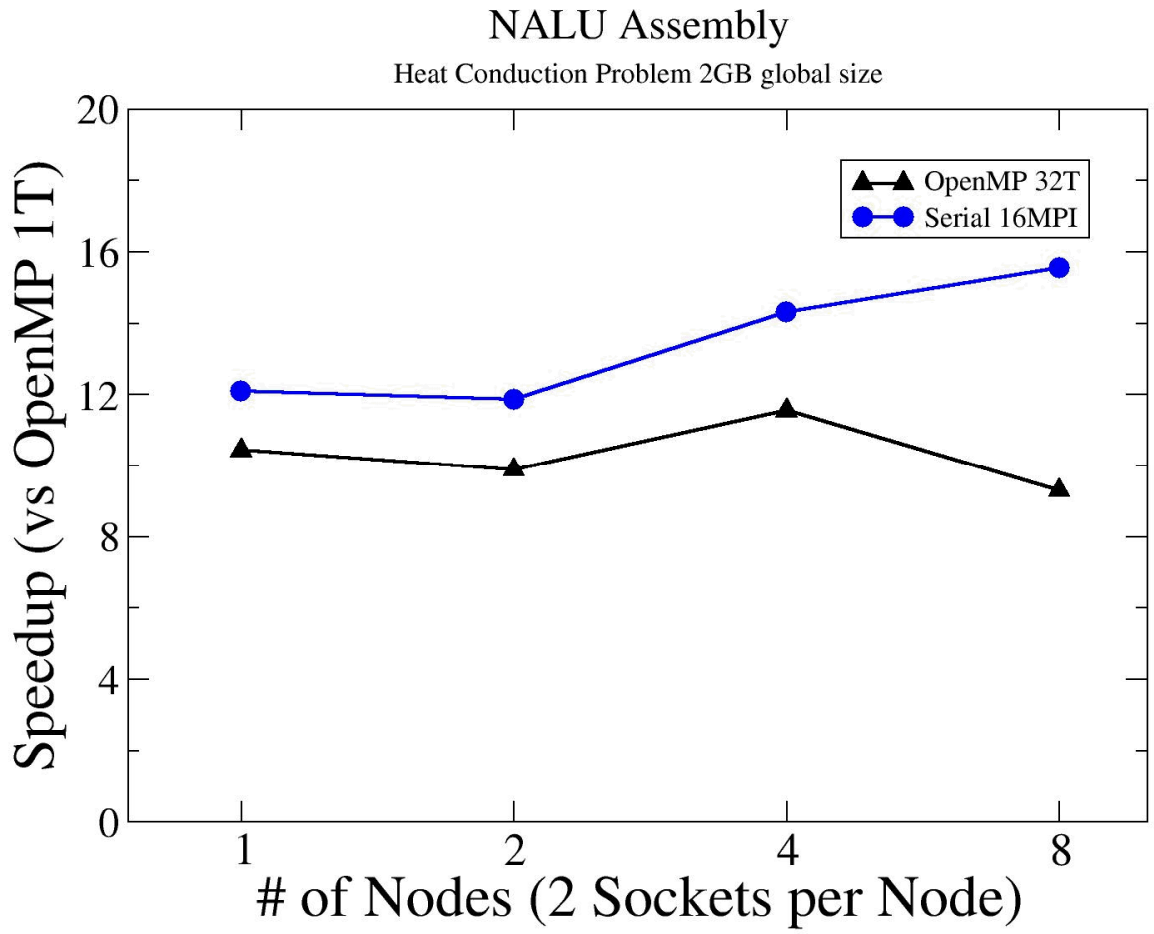
Sandia National Laboratories

Old OpenMP fill perf. results

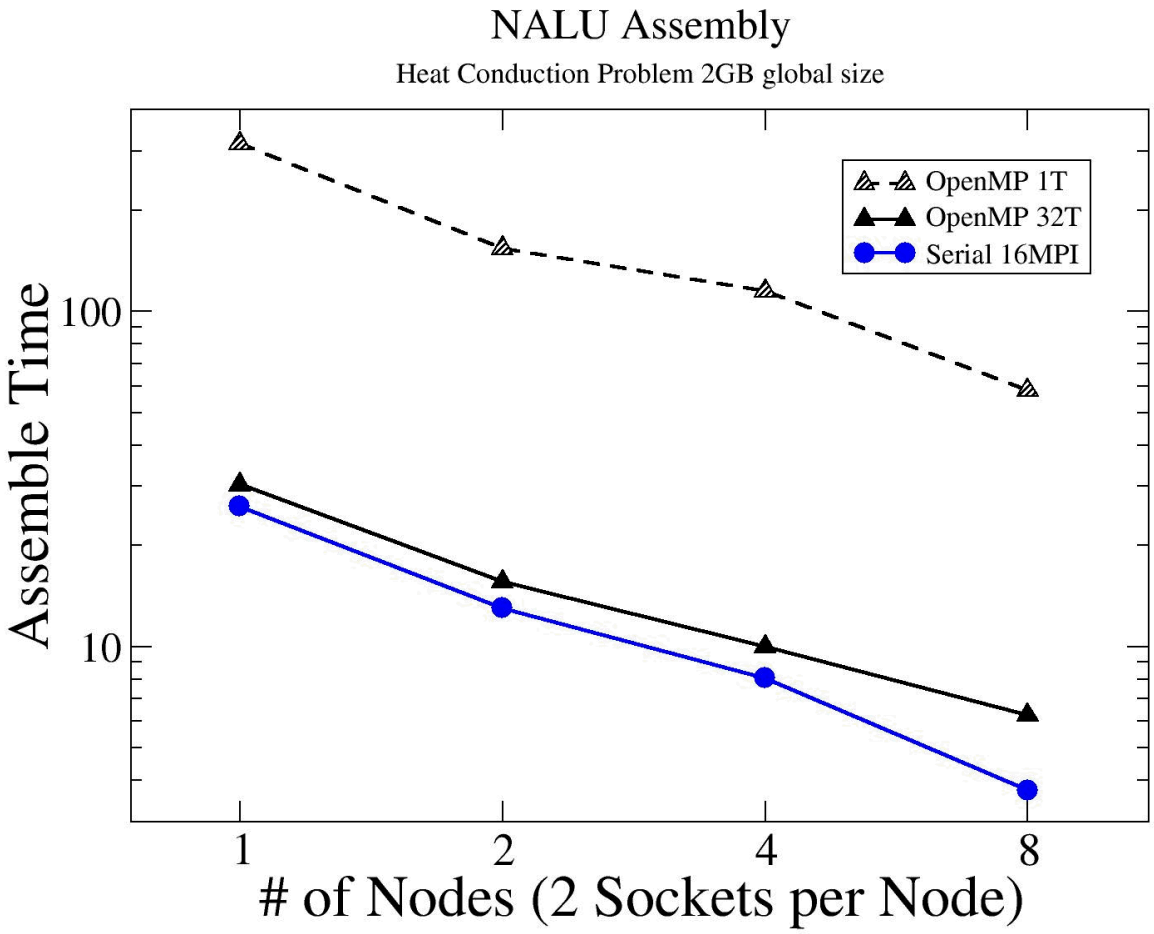
Sandia National Laboratories

- Nalu (available on Github) heat conduction problem
- Exercises new Tpetra fill interface w/ atomics
  - MINIMAL changes to Nalu assembly!
- Shepard testbed: Haswell
  - 16 cores / socket, 2 hyperthreads / core, 2 sockets / node
- Strong scaling of 2GB test problem
  - With OpenMP: 1 MPI process per socket
  - OpenMP uses hyperthreads; MPI-only did not
  - Hard for OpenMP to beat MPI-only here; doesn't yet, but scales well
- 32 or 64 threads/MPI process is what we plan for KNL
  - MPI-only works fine on Haswell, but not so well on KNL
  - MPI-only won't work on GPU (CORAL/Sierra)

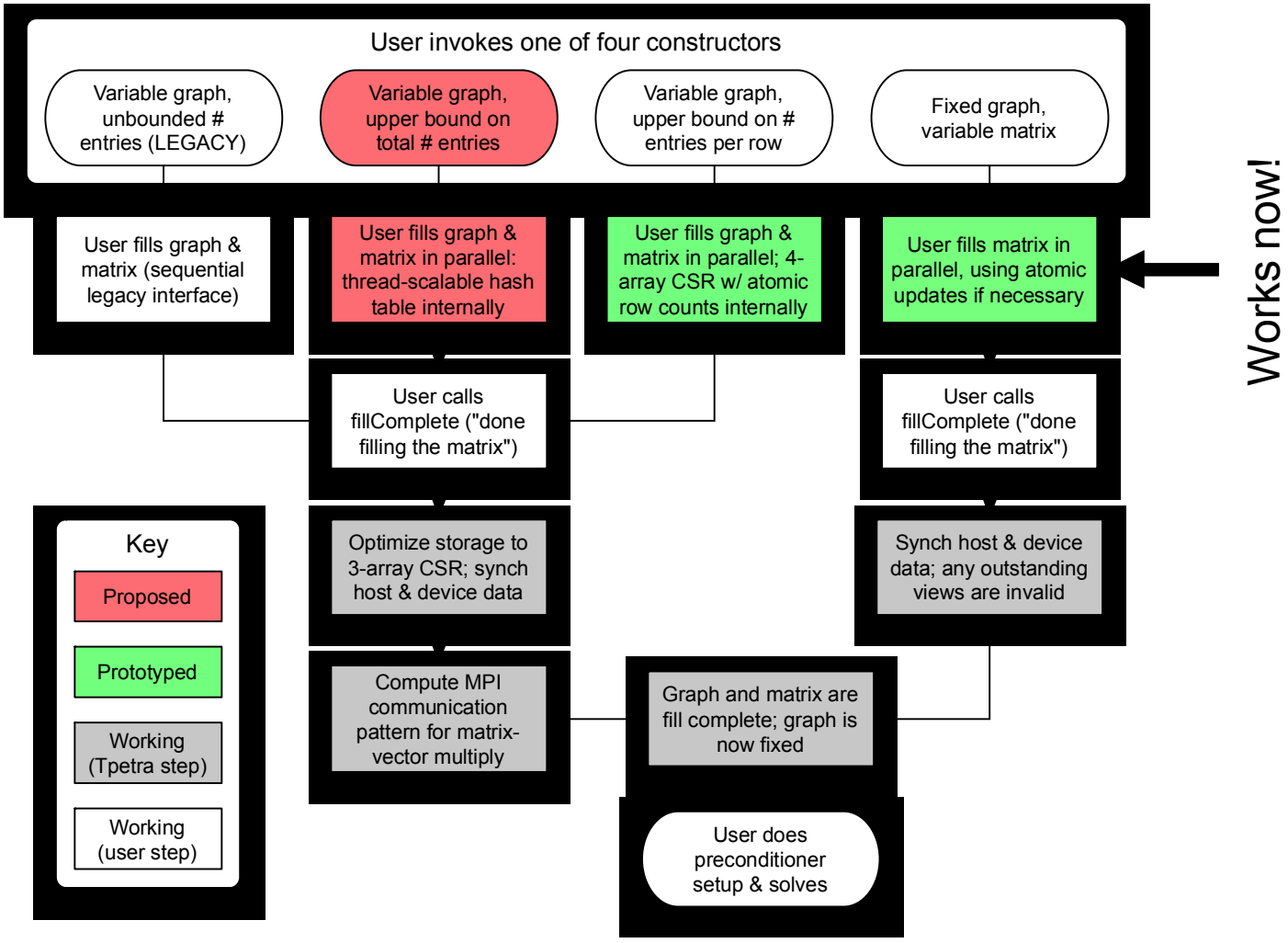
Thread-safe fill: assembly speedup



Thread-safe fill: assembly time



4 ways to create a graph / matrix

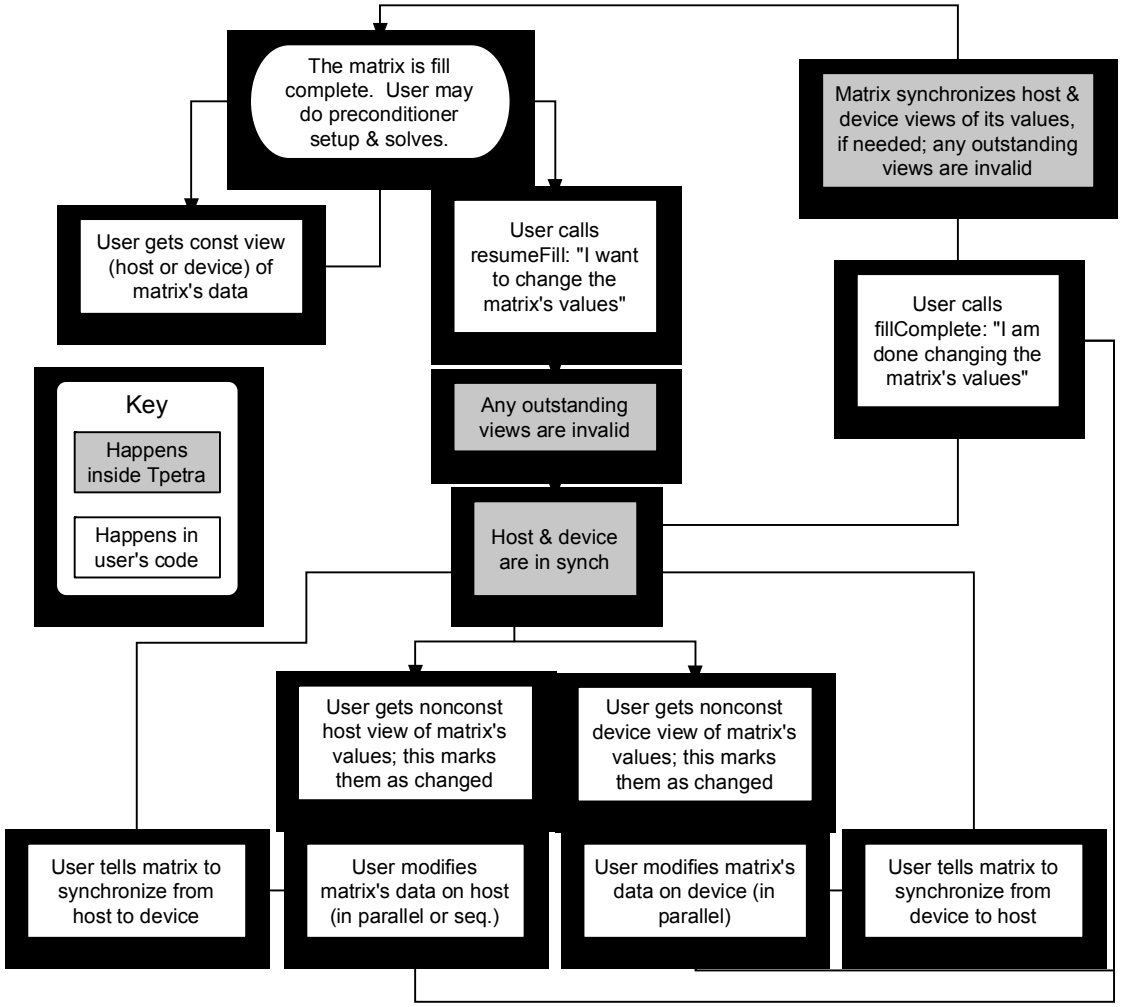


Thread-parallel graph construction

Sandia National Laboratories

- Kokkos FENL example prototype; want directly in Tpetra
- Thread-parallel graph construction in Tpetra
  - Merge entries w/ same (row,column) at insert, not at fillComplete
    - Avoid unnecessary dynamic resizing; less need for DynamicProfile
  - No more lazy allocation of graph / matrix storage
    - If your graph has no entries on that process, you'll just have to pay for the row offsets array – you've made the row Map anyway so why not?
  - No more "lazy choice" of local / global index storage
    - Make graph / matrix w/ column Map ➔ Locally indexed
  - Restricted DynamicProfile & nonlocal insert/sumInto
    - Max *total* (not per row) number of entries (on that process)
    - Repeat (count, allocate, fill) until correct local upper bound

DualView semantics: CrsMatrix



Kokkos refactor release timeline

Sandia National Laboratories

- Started late 2013
  - First step: new Node types that use Kokkos 2.0 inside
- Deprecated Classic at Jan 2015 11.14 release
  - Kokkos Refactor became default
  - Kokkos became required for building Tpetra
  - Using Classic emitted deprecated warnings
- Apr 2015 12.0 release
  - Classic officially no longer existed
  - We retained it “just in case”
- Oct 2015 12.4 release
  - Classic removed completely
  - This fixed Doxygen (partial specialization on Node confused it)

Commit history: “Tpetra 1.0”

Sandia National Laboratories

- Mid 2002: Mike Heroux w/ student Paul Sexton; BLAS/LAPACK
- Late 2002: (Block)ElementSpace, Distributor, {Ordinal,Scalar}Traits
- Early 2003: Vector & VectorSpace
- Mid 2003 (Kris Kampshoff): BLAS & LAPACK wrappers, what later became Teuchos::SerialDenseMatrix (started as Tpetra::DenseMatrix)
- Late 2003: start Tpetra::Comm -> Teuchos; Tpetra Autotools build works
- Late 2003 (Heroux): Kokkos sparse matrix commits
- Feb 2004: Vector usable; SparseMatrix (-> CisMatrix) added
- Mar 2004 (Heroux): Kokkos adds support for permutations
- 2004 (Jason Cross): Jpetra redesigned to imitate Tpetra
- Mid 2004: Tpetra supports MPI; more RefCountPtr use
- Mid 2005: Added DistObject (vs. Object), Export, & MultiVector
- Late 2005 (Marzio Sala): Docs, bug fixes, & tests
- Early 2006 (Heroux, Willenbring): Kokkos::OskiMultiVector
- May 2006 (Ross Bartlett): Operator (2006-7: Anasazi & Belos)
- Jan 2008 (Alan Williams): TBB support (!)

Commit history: “Tpetra 2.0”

Sandia National Laboratories

- Mid 2008 (Chris Baker): Massive Tpetra interface changes & rewrite (into 2009)
  - {Element,Vector}Space -> Map
  - Comm Tpetra -> Teuchos
  - CrsMatrix transpose apply
  - Added RowMatrix & RowGraph
- Mid 2009: “Began moving Tpetra to use Kokkos” (& interface changes); Kokkos Node API
  - Map changed to be passed around as RCP<const Map> instead of being a handle for MapData (as in Epetra)
- Mid 2009 (Heroux): Tpetra_InverseOperator (!) (gone Jan 2010)
- Late 2009 (Williams): CrsGraph Import / Export
- Nov 2009 (Baker): CUDA support in Kokkos (ThrustGPUNode)
- Dec 2009 / Jan 2010 (Baker): Tpetra::HybridPlatform (run-time selection of execution space)
- Mid 2010: Tpetra ETI (Baker, Williams), VBR (Williams)
- Mid 2010 (Kurtis Nusbaum, w/ Chris Siefert): Sparse matrix-matrix multiply
- Mid 2010 (Mark Hoemmen): TSQR
- Late 2010 (Jeremie Gaidamour): Xpetra (called “Cthulhu” initially)

Fun Tpetra history facts

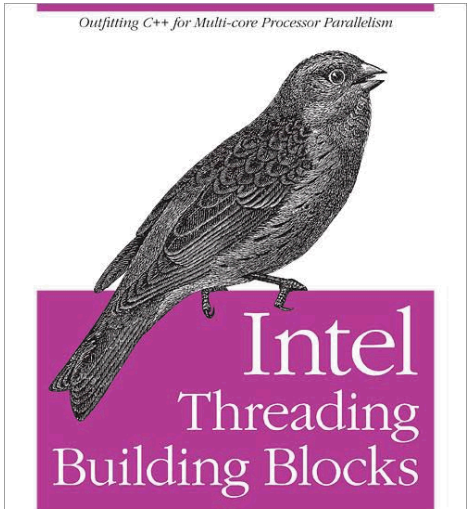
Sandia National Laboratories

- 2003: Kokkos started as a computational kernels package
  - 2014: KokkosKernels package proposal revives this idea
- 2002-5: Many Teuchos classes started in Tpetra
  - Comm, BLAS, LAPACK (though Comm changed afterwards)
  - Lesson: Fight hard over utility classes that affect performance
- Early 2008: Alan Williams (!) added preliminary TBB support
- Mid 2008: Big rename gave Tpetra its current look
- Mid 2009: Baker starts Kokkos Node API effort
  - Later than I had thought, given extensive changes
  - CUDA support added late 2009 / early 2010

Kokkos 1.0 Node API

Sandia National Laboratories

- Chris Baker + Mike Heroux, 2009-10
- Inspired by Intel Threading Building Blocks
- "Node" back-end abstraction
  - CUDA (Thrust), TBB, Pthreads, Serial
  - (OpenMP support came later)
- parallel_{for, reduce} w/ functors
- "Device buffer" allocations
  - Teuchos::ArrayRCP; had to use raw ptrs in kernels
- Reduction Transform Interface (RTI)
  - Required C++0x features (lambda, decltype)
  - Users could apply local entry-wise ops to Tpetra::Vector entries
  - Some "fused" kernels (e.g., AXPY, then norm of result)



Before you hate on Kokkos 1.0…

Sandia National Laboratories

- Good for experiments
  - Uncertainty about GPUs' use for sparse linear algebra
  - Show off mixing precisions & index types
- Tpetra had almost no production use at the time
  - Main reason: No solvers
    - Ifpack2 ("Tifpack") online mid 2009; MueLu end 2010
    - Apps' motivation for using Trilinos: Solvers (or Zoltan)
  - Considerable uncertainty about Tpetra's status in 2010
    - Xpetra (started as "Cthulhu") let MueLu bootstrap w/ Epetra
    - Trilinos devs interpreted Epetra64 in part as lack of confidence in Tpetra
  - Few users ➔ Tpetra == research vehicle
  - Papers & last-minute SIAM minisymposium talks never exercised the bugs that we had to work through in 2011-13

Create Petra sparse matrix, 200X-15

Sandia National Laboratories

- DynamicProfile
  - Start w/ 2-D storage (array of arrays); pack on fillComplete
  - Optional hint for max # entries / row, not enforced (realloc)
- StaticProfile
  - Start w/ 1-D storage (compressed sparse row, + # entries / row array)
  - Pack on fillComplete. Max # entries / row strictly enforced.
- (Option to supply a column Map (lets you use local indices))
- Create using a fill-complete CrsGraph
  - Graph structure fixed (more strict than StaticProfile)
  - Local indices only (CrsGraph must be fillComplete)
  - Fastest possible case, optimized for nonlinear solves w/ fixed mesh

Tpetra fill, 2010-2015

Sandia National Laboratories

- Not much different than Epetra, except:
  - Throw C++ exception on user error / out of space
  - Teuchos::ArrayView for input & output arrays
- Get row views: void get{Local,Global}Row{Copy,View}
  - "Copy": into user-provided storage, converting indices if needed
  - "View": throw if graph/matrix stores indices {Global,Local}
- Modify entries: void {replace,sumInto}{Local,Global}Value(s)
  - Convert indices {Global,Local} if needed, alloc'ing temp storage
  - sumIntoGlobalValues allows modifying rows not owned by calling process (store in std::map, & communicate at fillComplete)
  - Any change other than replace or sumInto, requires getting a view
- Add new entries: void insert{Local,Global}{Indices,Value(s)}
  - Default: arbitrary dynamic allocation via 2-D storage
  - Defer merging duplicate entries until fillComplete (differs from Epetra)
  - Lazy alloc: each process only allocs on first "insert" call

Tpetra fill was not thread-scalable

Sandia National Laboratories

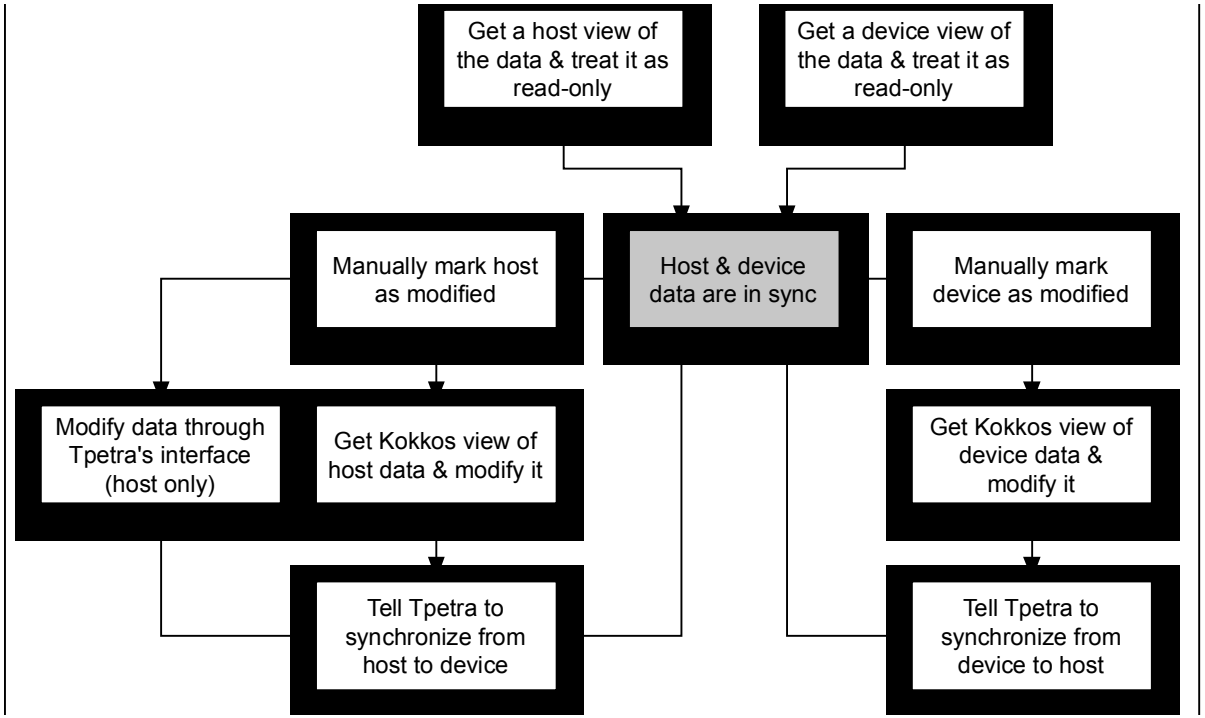
- Dynamic memory allocation ("dynamic profile")
  - Impossible in some parallel models; slow on others
  - Allocation implies synchronization (must agree on pointer)
  - Better: Count, Allocate (thread collective), Fill, Compute
- Throw C++ exceptions on error / when out of space
  - Either doesn't work (CUDA) or hinders compiler optimization
  - Prevents fruitful retry in (count, allocate, fill, compute)
  - Better: Return success / failed count; user reduces over counts
- Unscalable reference counting implementation
  - Teuchos::(Array)RCP: like std::shared_ptr but not thread safe
  - Not hard to make thread *safe*, but updating the ref count serializes!
  - Better: Use Kokkos' thread-scalable count; prefer unmanaged View

Tpetra had no plan for parallel fill

Sandia National Laboratories

- Must fill Tpetra data structures on host, sequentially
  - Common way to access data: host copy ("generalized view")
    - Read-only: Host copy of device data
    - Read-write: Host copy, copies back to device at ref count 0
  - "Device view" was expert mode, never used outside Tpetra
  - Matrix data vanished (into the TPL) at fill complete
- Teuchos::{RCP, ArrayRCP} ref count not thread safe
  - Tpetra stored & returned everything (e.g., Maps, CrsGraph) by RCP
  - In debug mode, even Teuchos::ArrayView ref-counts
  - Can't use device buffers (ArrayRCP) in parallel kernels

DualView semantics (Vector)



Tpetra objects act just like Kokkos::DualView.
Tpetra's evolution of legacy fill interface is host only.
To fill on (CUDA) device, must use Kokkos interface.

If you only have one memory space, you can ignore all of this; it turns to no-ops.

Preferred use with two memory spaces:

1. Assume unsync'd
2. Sync to memory space where you want to modify it (free if in sync)
3. Get & modify view in that memory space
4. Leave the Tpetra object unsync'd