SAND2017-3945C



Thread parallelism in Trilinos' sparse algebra interfaces & linear solvers

Mark Hoemmen
Sandia National Laboratories
28 Apr 2017

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. SAND2017- XXXX C

Outline

Sandia National Laboratories

- This is a talk about rewriting software to use threads
  - Don't expect performance graphs or convergence plots
- Tpetra, a Trilinos package, implements sparse linear algebra
- Kokkos is a shared-memory parallel programming model
- We rewrote Tpetra & downstream solvers to use Kokkos
- What is & isn't (yet) thread parallel in Trilinos' linear solvers
- Retrospective on sparse linear solver library development
  - You should care because you may be asked to do this at some point
  - "To a modern mathematician, design [or software development] seems to be a second-rate intellectual activity" (George Forsythe)

Background & interests

Sandia National Laboratories

- Academic background
  - UIUC (math & computer science)
  - TU Berlin (math)
  - UC Berkeley (computer science)
  - Numerical linear algebra, esp. iterative linear solvers
- Research interests:
  - Communication-avoiding algorithms
  - Algorithmic fault tolerance
  - Parallel programming models
  - Sparse matrix data structures & computational kernels

Software projects at Sandia

Sandia National Laboratories

- Started at Sandia (ABQ) in 2010
- Recently moved from
  - Center for Computing Research to
  - Engineering Sciences (more application-oriented)
- Tpetra product owner since 2011
- Led Trilinos' Scalable Linear Algebra Services capability area
- Led Trilinos tutorial development & gave external & internal tutorials
- Kokkos developer
- Cofounder of KokkosKernels

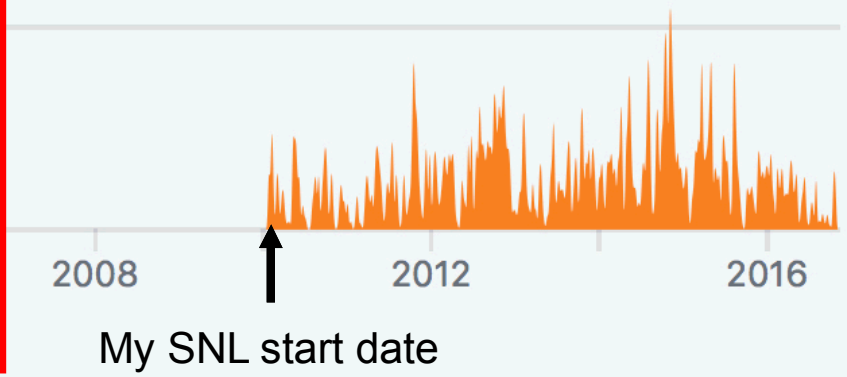
Top Trilinos contributor

mhoemmen

4,629 commits / 1,329,437 ++ / 1,114,087 --

#1

- Top # commits overall, since FY16, & since FY14
  - Overall: 4629 (Ross 4246)
  - FY16-: 835 (Tobias 680)
  - FY14-: 2634 (Andrey 1133)



Includes some Kokkos & many KokkosKernels commits; both belonged to Trilinos before.

What is Trilinos?

Sandia National Laboratories

- Object-oriented software framework
- Solves big complex science & engineering problems
- More like LEGO™ bricks than Matlab™
- More like a big extended family than a software project
- $ ls Trilinos/packages | wc –w -> 60

What is Tpetra?

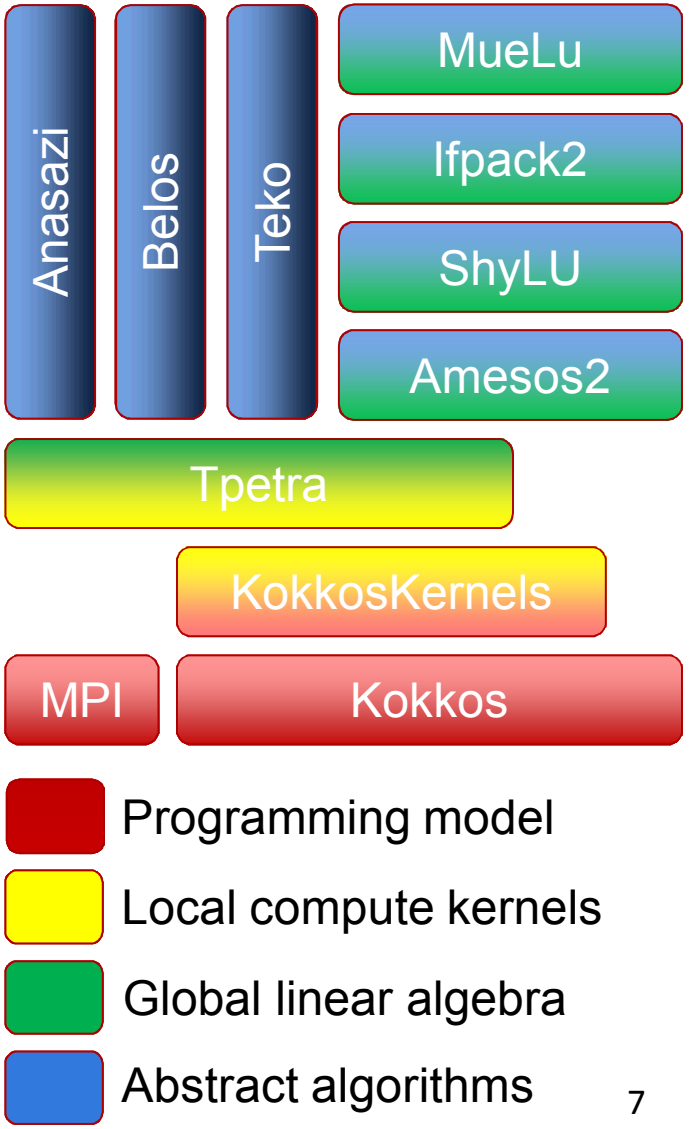
Sandia National Laboratories

- Tpetra, a Trilinos package, implements
  - Sparse graphs, (block) sparse matrices, & dense (multi)vectors
  - Kernels for solving Ax=b & Ax=λx (non-MPI part now in KokkosKernels package)
  - MPI communication & (re)distribution
  - {Create, modify} a {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

Trilinos' sparse linear solver packages

- Local compute kernels (KokkosKernels)
- Global sparse linear algebra (Tpetra)
- Krylov solvers (Belos, Anasazi)
- Sparse direct solvers (Amesos2)
- Domain decompositions, relaxations, incomplete factorizations (Ifpack2)
- Sparse direct + iterative (ShyLU)
- Block preconditioners (Teko)
- Algebraic multigrid (MueLu)

Nothing is standalone!
Everything above Tpetra depends on it.



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

Tpetra development goals

- "Performance portability"
  - 1 implementation, runs everywhere
  - Perform well on many platforms
  - Limited developer time (1-2 FTEs)
- Maintain backwards compatibility
  - Don't break users' builds (interface) or tests (semantics)
  - Trilinos only allows breaking it at major releases (1-2 years)
  - Apps have been using Tpetra for about 5 years
    - Internal users at Sandia & other US Department of Energy labs
    - External users through apps like deal.II, Nalu, & CASL VERA
    - Many Trilinos packages depend on Tpetra – esp. algebraic multigrid
  - Hard balance between performance, user support, & research

Why MPI + threads?

Sandia National Laboratories

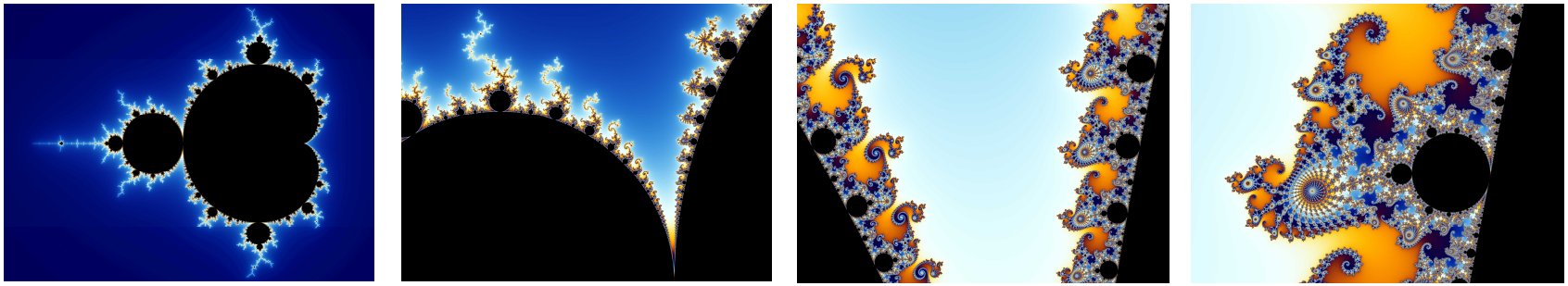
Why MPI + threads?

Norway Lemming
(can swim; are migrating, not suiciding)

- "Everybody's doing it"
- "Boss/funding/vendor said"
- Today: benefit mainly on specialized architectures
- Threads have issues too
  - Too much sharing (MPI "discourages nonlocality")
  - Fork-join overhead
- Better Q: Why 2 levels?
  - MPI + X vs. MPI only
  - "MPI+MPI" also 2 levels (2nd MPI uses shared memory)

MPI assumes computers are fractal

Sandia National Laboratories



- Code 100,000 processes (sort of) like 1,000
- Pretend that networks are flat
  - Physical problems have natural locality
  - Hardware-aware load-balancing software (e.g., Zoltan)
  - Subcommunicators (use MPI to express hierarchy)
- 1-process case
  - Good: Optimize for memory locality, vectorization, ILP, …
  - Bad: Ignore performance, pretend it's 1995

Zooming in too far breaks MPI only

Konrad Zuse
Phantasie, 1967

- Fine-grained parallelism
  - Vector lanes != MPI procs
  - Gather-scatter, shuffle, etc.
- Library too much overhead; need compiler
  - Vectorization
  - Language extensions
- Fuzzy zone between MPI processes & vector lanes
  - Where MPI+threads help or at least don't hurt
  - e.g., hyperthreads on KNL

How MPI + threads?

Sandia National Laboratories

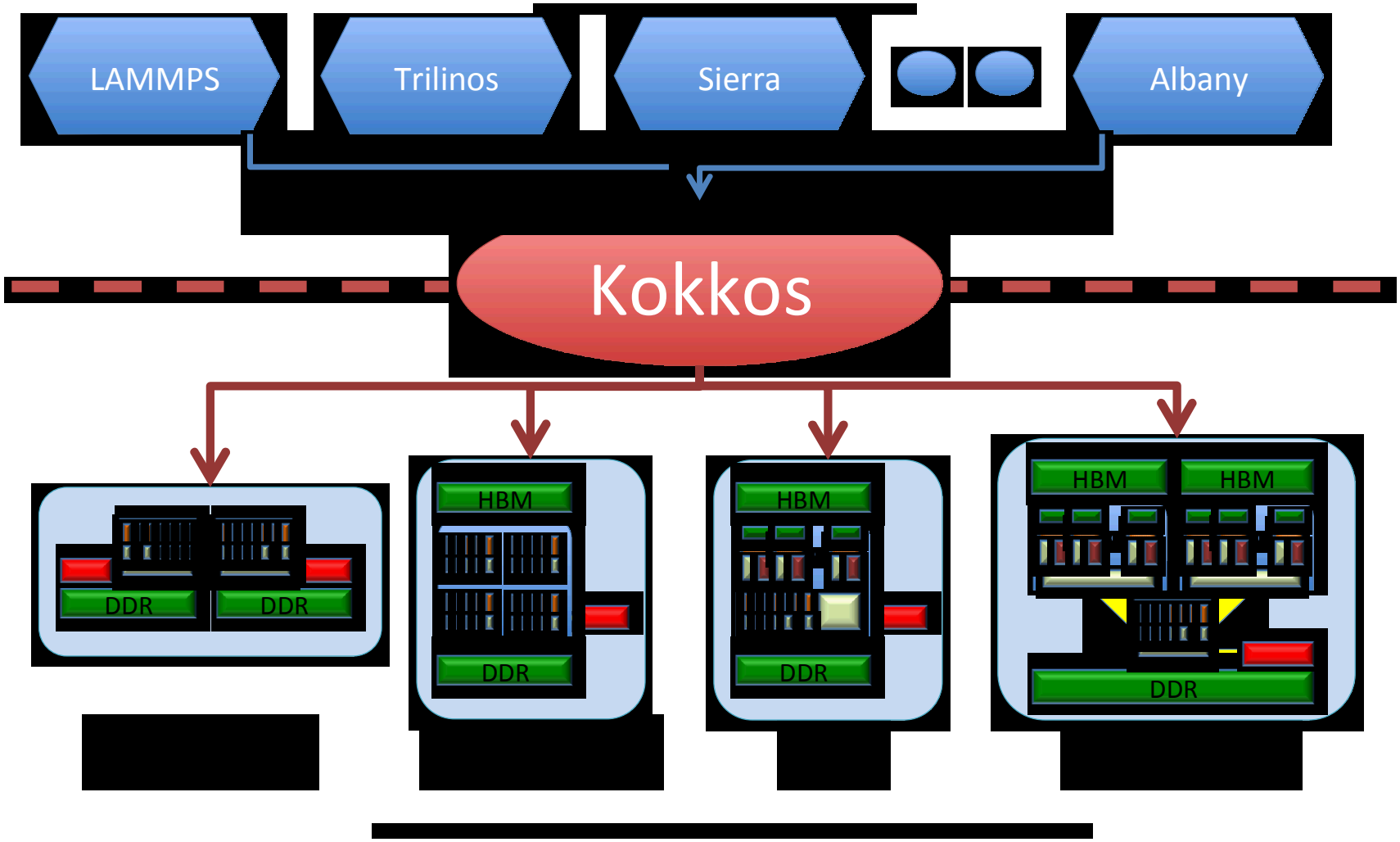
Kokkos: Common C++ - based programming model for thread parallelism on GPUs, CPUs, …

Sandia National Laboratories

- Parallel {for, reduce, scan} w/ user loop body
- Exposes different levels of parallelism
  - Flat [0,N), or hierarchical (team, thread, vector)
  - Task parallelism, data parallel inside (on GPUs too!)
- Control where data live & code executes
- Enable "hybrid" (host + GPU) parallelism
- Multidimensional arrays (Kokkos::View) w/ slices
  - Decouple array layout (row/column-major, tiled, …) from app
  - Default layout optimized for the architecture (SoA / AoS)
  - Unified interface to shared memory, texture fetch, atomic access, …
- Thread-scalable data structures (e.g., hash table), sort, etc.
- Write once, parallel everywhere! github.com/kokkos

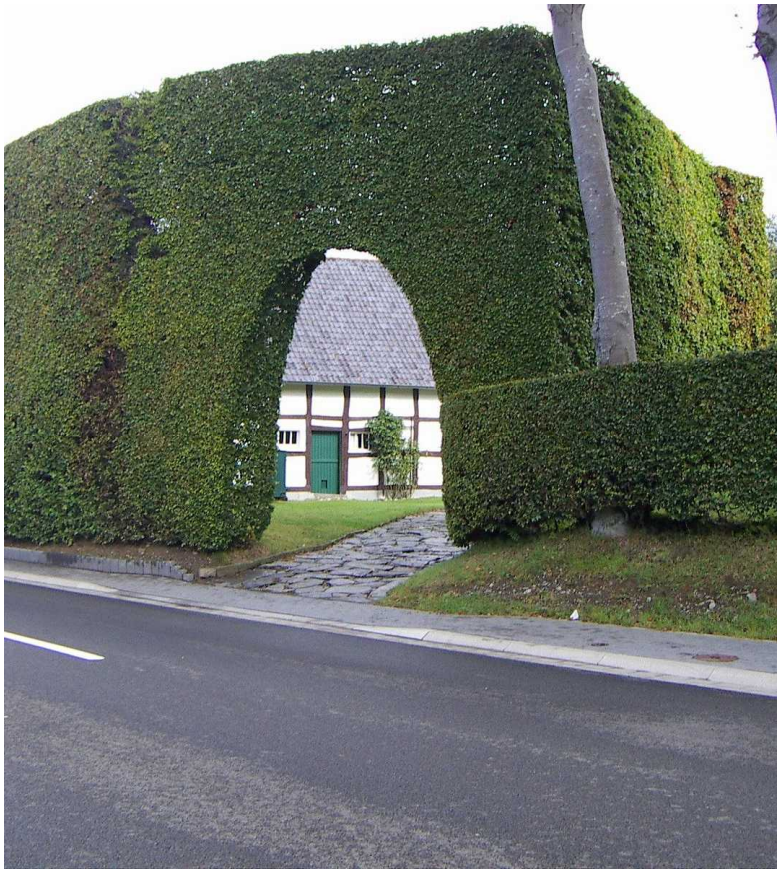
Kokkos: *Performance, Portability, & Productivity*

Sandia National Laboratories



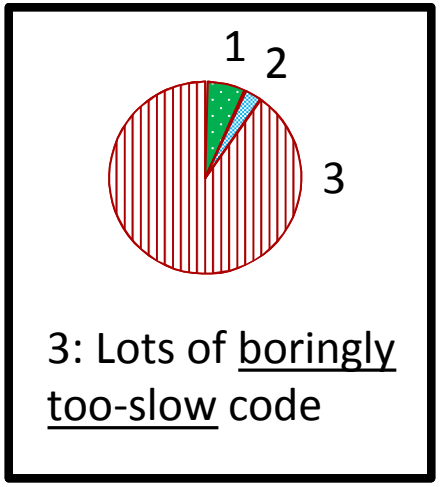
Kokkos as hedge against…

- Hardware divergence
- Parallel programming model
  - OpenMP, OpenACC, CUDA, TBB, Pthreads, Qthreads, …
- Traditional shared memory
  - vs. PGAS / distributed shared
- Threads at all
  - Kokkos' semantics require vectorizable (ivdep) loops
  - Works w/ explicit SIMD
- Kokkos protects our HUGE time investment in Tpetra threading



Performance portability

- "Nowhere not too slow"
  - Track hardware features that scale
  - ➔ Parallelism (MPI, threads, vectors)
- 3 parts of your code
  1. Most performance-critical
  2. Not worth optimizing
  3. "Amdahl's long tail"



- For critical fraction, accept cost of nonportable solutions
  - Hardware-specific optimizations; vendor / 3rd-party library
  - Self-contained computational kernels that make great papers
- Performance portability: all about "everything else"
  - Not self-contained; solution not easily buyable / publishable
  - Custom code: e.g., pack MPI message buffers for sparse C := A*B

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 **	Builds by default in develop
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, well along
BCRS (block) Jacobi (s. & s.)	Y	N	Y	OPT in progress, well along

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	Y*	MueLu setup; off by default
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	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)	Y	Y	N	In progress
Line smoothing	N	N	Y	Many tridiagonal solves
BCRS line smoothing	N	N	N	In progress; well along
Domain decomp. setup	N	N	N	Computing overlap

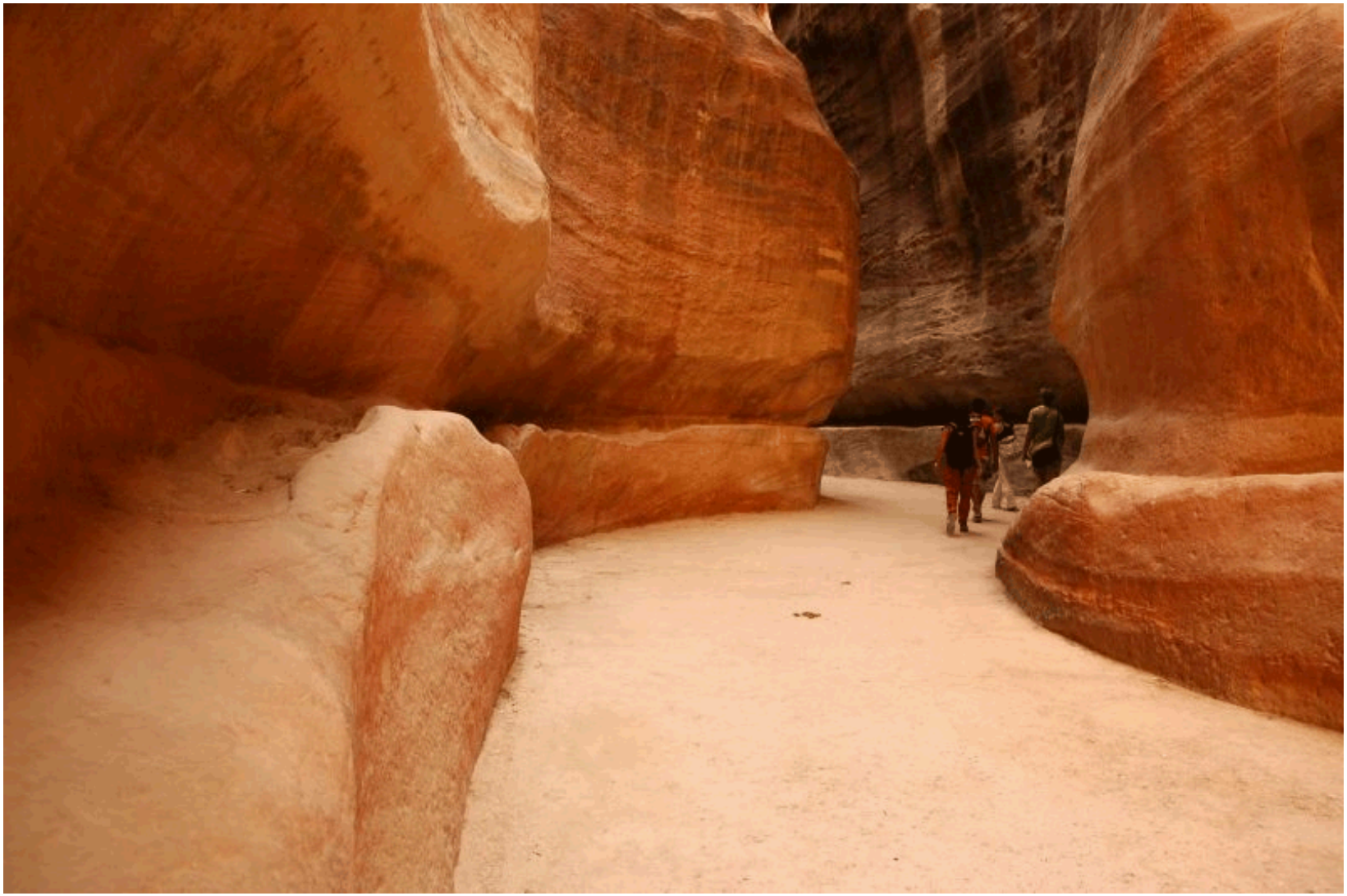
Future work (hard stuff)

Sandia National Laboratories

Kernel / solver name	TPK	OPT	DPL	Notes
FastILU (SPAI ILU)	Y	Y	N	Fine-grained parallel setup. Needs algorithm research.
Algebraic multigrid aggregation (part of setup)	N	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!				
(not even a complete list)				

What made thread parallelization hard?

Sandia National Laboratories

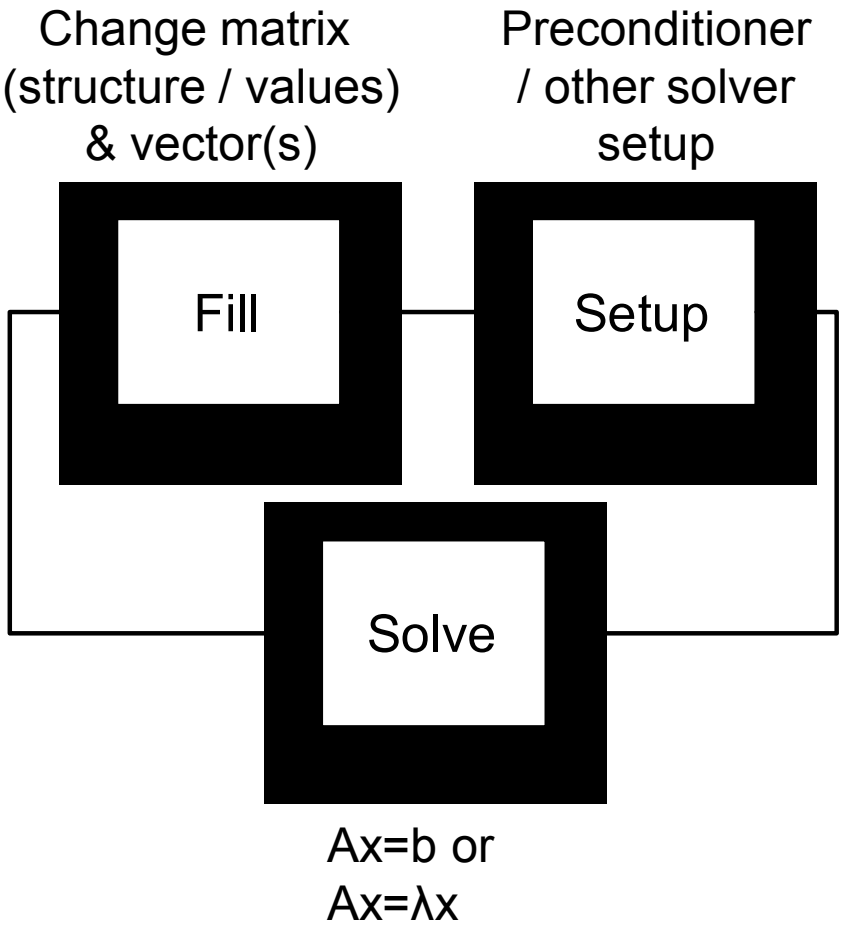


Sparse linear algebra use pattern

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), …
  - Software interfaces affect performance A LOT
- Setup for solve (e.g., build preconditioner)
- Solve linear system(s), eigenvalue problems, etc.
  - Coarse-grained computational kernels (e.g., sparse mat-vec)
  - Software interfaces affect performance less
- Repeat (nonlinear iteration, time steps, parameter study, …)
  - Trilinos data structures & solvers optimized for reuse, e.g., of
    - Data structures (graph, basis vectors, allocations) &/or
    - MPI communication patterns (where to send / receive what)

Need thread-parallel fill



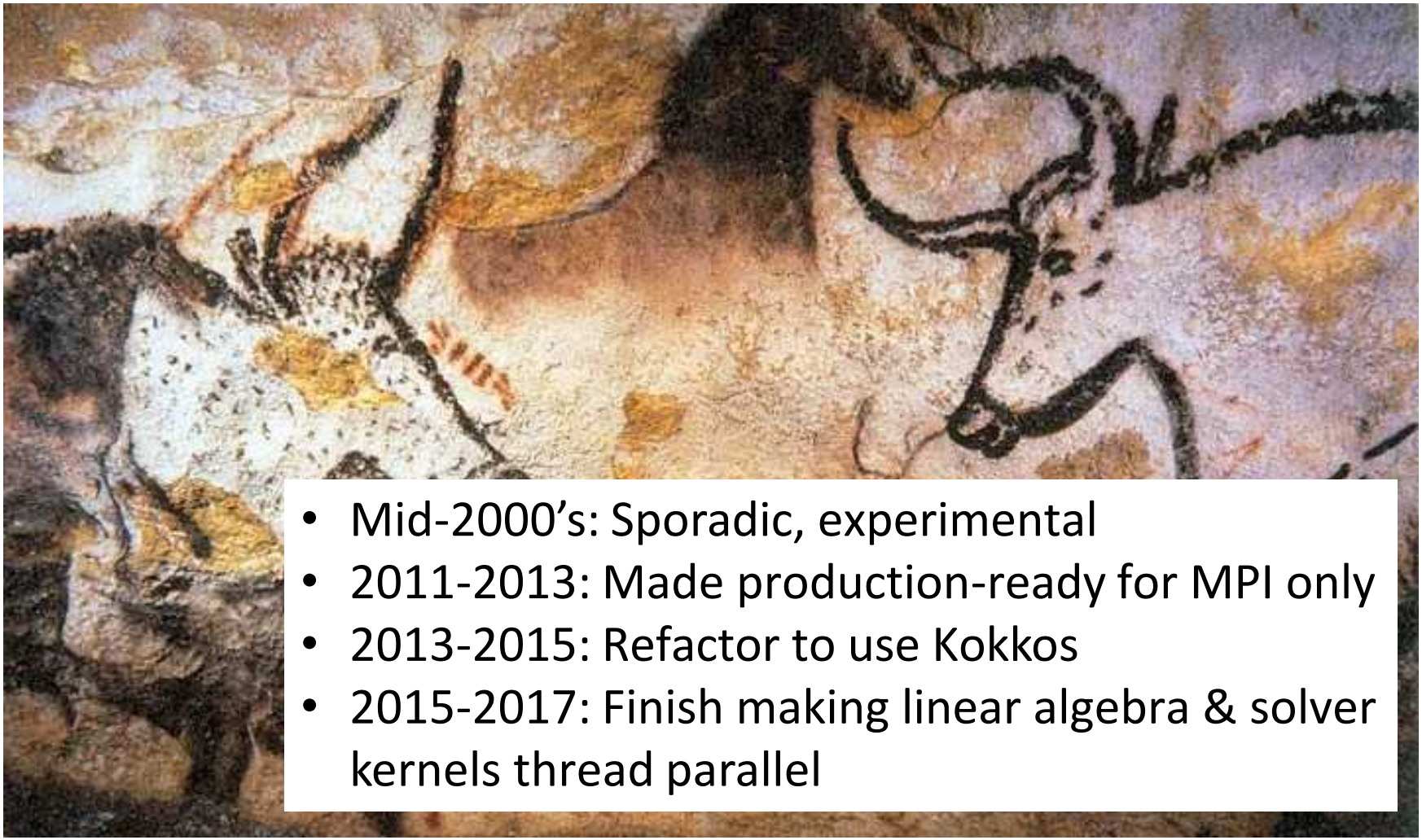
- Fill & setup not free
- Some solves are cheap, so fill & setup time matter
- Amdahl's Law:
  - Threading just solves makes cute, easy-to-publish papers
  - Solves do take most app time
  - But: 90% time w/ 1 thread ➔ 50% time w/ 10 threads
- Preconditioners create sparse matrices, so they also need fill

Fill challenges thread parallelization

Sandia National Laboratories

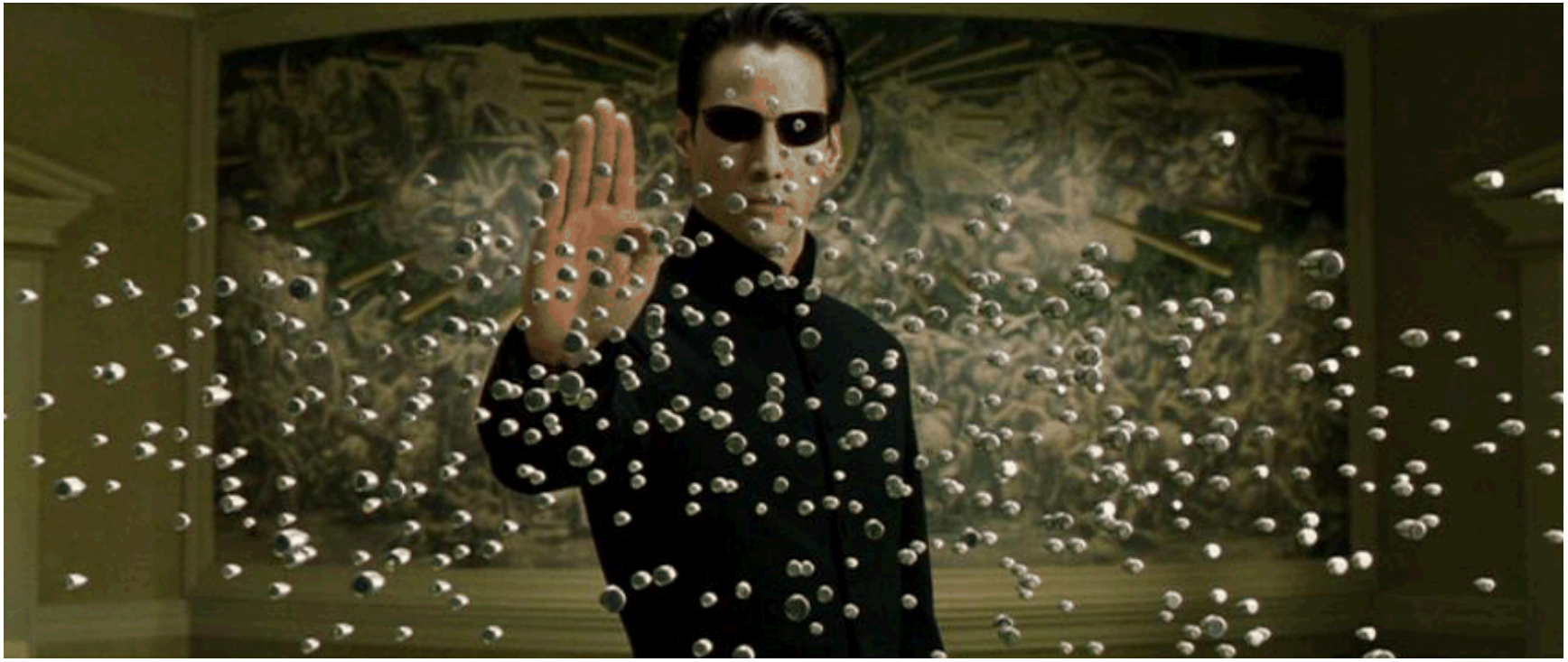
- Implications for linear algebra data structures
  - Fill interfaces affect thread safety & performance
  - Don't care that y := A*x 5% faster w/ your funny new data structure
- Users use slow, unparallelizable interfaces
  - For nearly 20 years, huge difference between best practice for global assembly in apps that use Trilinos, & Trilinos' official examples
  - Users imitate examples; give them good ones
- Lesson learned: Stay out of the fill business
  - Tpetra had/has complicated, stateful fill interfaces
  - Maintaining current interface's fill semantics was main cost
    - Added >= 1 year to refactor effort
    - Hinders further refactoring, but no performance advantage to apps
  - Better to let users build local data structures, & hand them off

History of Tpetra



- Mid-2000's: Sporadic, experimental
- 2011-2013: Made production-ready for MPI only
- 2013-2015: Refactor to use Kokkos
- 2015-2017: Finish making linear algebra & solver kernels thread parallel

What is working on Tpetra like?

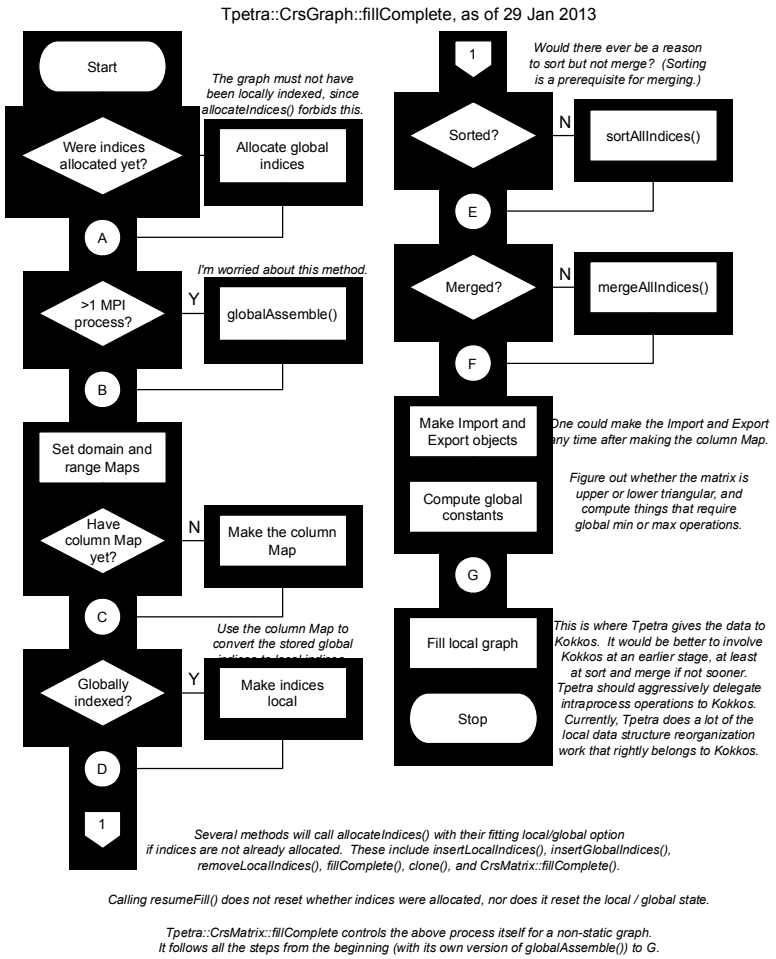


I wish it felt like this once in a while

What is working on Tpetra like?

It usually feels like like sewer diving: Technical, gross, possibly necessary

What working on Tpetra is like



- Flow chart of 1 function in Tpetra's sparse graph class
- Tpetra's sparse data structures are "stateful," because of the many ways to create & modify them
- Statefulness hinders optimization & threading, but changing it would break both downstream Trilinos packages & users' code
- Backwards compatibility, but high development & maintenance cost

What working on Tpetra is like

- Example: Dot product of 2 vectors
- Vector: Dense 1-D array, distributed over MPI processes
  - {double* data_, int lclLen_, MPI_Comm comm_;}, right?
- `double dot (const Vector& x, const Vector& y);`
  - That's all it is, right?
- How little time I have to show you how much more it is....

Tpetra dot: 1 of 9 (naïve version)

Sandia National Laboratories

```
class Vector {
public:
  Vector (/* ... */);
private:
  double* data_;
  int lclLen_;
  MPI_Comm comm_;
};

double dot (const Vector& x, const Vector& y) {
  double lclSum = 0.0;
  for (int i = 0; i < x.lclLen_; ++i) {
    lclSum += x.data_[i] * y.data_[i];
  }
  double gblSum = 0.0;
  MPI_Allreduce (&lclSum, &gblSum, 1, MPI_DOUBLE,
                 MPI_SUM, x.comm_);
  return gblSum;
}
```

Tpetra dot: 2 of 9 (reference count)

Sandia National Laboratories

```
class Vector {
public:
  Vector (/* ... */);
private:
  Teuchos::ArrayRCP<double> data_;
  MPI_Comm comm_;
};

double dot (const Vector& x, const Vector& y) {
  double lclSum = 0.0;
  for (int i = 0; i < x.data_.size (); ++i) {
    lclSum += x.data_[i] * y.data_[i];
  }
  double gblSum = 0.0;
  MPI_Allreduce (&lclSum, &gblSum, 1, MPI_DOUBLE,
                 MPI_SUM, x.comm_);
  return gblSum;
}
```

Tpetra dot 3 of 9: template Scalar

Sandia National Laboratories

```
template<class Scalar>
class Vector {
public:
  typedef Scalar scalar_type;
  Vector (/* ... */);
private:
  Teuchos::ArrayRCP<scalar_type> data_;
  MPI_Comm comm_;
};

template<class Scalar>
typename InnerProductSpaceTraits<Scalar>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  typedef InnerProductSpaceTraits<Scalar> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  for (int i = 0; i < x.data_.size (); ++i) {
    lclSum += IPT::dot (x.data_[i], y.data_[i]);
  }
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  const int mpiCount =
    MpiTypeTraits<dot_type>::getCount (lclSum);
  MPI_Datatype mpiDt =
    MpiTypeTraits<dot_type>::getType (lclSum);
  MPI_Allreduce (&lclSum, &gblSum, mpiCount, mpiDt,
                 MPI_SUM, x.comm_);
  return gblSum;
}
```

Tpetra dot 4 of 9: MPI abstraction

Sandia National Laboratories

```
template<class Scalar>
class Vector {
public:
  typedef Scalar scalar_type;
  Vector (/* ... */);
private:
  Teuchos::ArrayRCP<scalar_type> data_;
  std::shared_ptr<const Tpetra::Comm> comm_;
};

template<class Scalar>
typename InnerProductSpaceTraits<Scalar>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  typedef InnerProductSpaceTraits<Scalar> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  for (int i = 0; i < x.data_.size (); ++i) {
    lclSum += IPT::dot (x.data_[i], y.data_[i]);
  }
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  allReduce (lclSum, &gblSum, TPETRA_SUM, *x.comm_);
  return gblSum;
}
```

5 of 9: Introduce Kokkos & threads

Sandia National Laboratories

```
template<class Scalar, class Device>
class Vector {
public:
  typedef Scalar scalar_type;
  typedef Device device_type;
  Vector (/* ... */);
private:
  Kokkos::View<scalar_type*, device_type> data_;
  std::shared_ptr<const Tpetra::Comm> comm_;
};

template<class Scalar, class Device>
typename InnerProductSpaceTraits<Scalar>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  typedef InnerProductSpaceTraits<Scalar> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  Kokkos::parallel_reduce (x.data_.dimension_0 (),
    [=] (const int i, dot_type& inout) {
      inout += IPT::dot (x.data_[i], y.data_[i]);
    }, lclSum);
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  allReduce (lclSum, &gblSum, TPETRA_SUM, *x.comm_);
  return gblSum;
}
```

6 of 9: Kokkos vs. std::complex

Sandia National Laboratories

```
template<class Scalar, class Device>
class Vector {
public:
  typedef Scalar scalar_type;
  typedef typename ScalarTraits<scalar_type>::val_type
    impl_scalar_type;
  typedef Device device_type;
  Vector (/* ... */);
private:
  Kokkos::View<impl_scalar_type*, device_type> data_;
  std::shared_ptr<const Teuchos::Comm> comm_;
};

template<class Scalar, class Device>
typename InnerProductSpaceTraits<
  typename Vector<Scalar,
Device>::impl_scalar_type>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  typedef InnerProductSpaceTraits<impl_scalar_type> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  Kokkos::parallel_reduce (x.data_.dimension_0 (),
    [=] (const int i, dot_type& inout) {
      inout += IPT::dot (x.data_[i], y.data_[i]);
    }, lclSum);
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  allReduce (lclSum, &gblSum, TPETRA_SUM, x.comm_);
  return gblSum;
}
```

7 of 9: Lambdas vs. CUDA / old GCC

Sandia National Laboratories

```
template<class Scalar, class Device>
class Vector {
public:
  typedef Scalar scalar_type;
  typedef Device device_type;
  typedef typename ScalarTraits<scalar_type>::val_type IST;
  Vector (/* ... */);
private:
  Kokkos::View<IST*, device_type> data_;
  std::shared_ptr<const Tpetra::Comm> comm_;
};

template<class Scalar, class Device>
class LclDotFunctor {
public:
  typedef typename ScalarTraits<Scalar>::val_type IST;
  typedef InnerProductSpaceTraits<IST> IPT;
  typedef typename IPT::dot_type dot_type;
  LclDotFunctor (const Kokkos::View<const IST*, Device>& x,
                 const Kokkos::View<const IST*, Device>& y)
    : x_ (x), y_ (y) {}
  KOKKOS_INLINE_FUNCTION void
  operator() (const int& i, dot_type& inout) const {
    inout += IPT::dot (x_(i), y(i));
  }
private:
  Kokkos::View<const IST*, Device> x_;
  Kokkos::View<const IST*, Device> y_;
};

template<class ImplScalar, class Device>
typename InnerProductSpaceTraits<ImplScalar>::dot_type
lclDot (const Kokkos::View<const ImplScalar*, Device>& x,
        const Kokkos::View<const ImplScalar*, Device>& y)
{
  typedef LclDotFunctor<ImplScalar, Device> functor_type;
  typedef InnerProductSpaceTraits<ImplScalar> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  Kokkos::parallel_reduce (x_.dimension_0 (),
                           functor_type (x, y), lclSum);
  return lclSum;
}

template<class Scalar, class Device>
typename InnerProductSpaceTraits<
  typename Vector<Scalar, Device>::IST>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  typedef typename Vector<Scalar, Device>::IST IST;
  typedef InnerProductSpaceTraits<IST> IPT;
  typedef typename IPT::dot_type dot_type;
  const dot_type lclSum = lclDot (x.data_, y.data_);
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  allReduce (lclSum, &gblSum, TPETRA_SUM, *x.comm_);
  return gblSum;
}
```

8 of 9: Lambdas work w/ CUDA now

Sandia National Laboratories

```
template<class Scalar, class Device>
class Vector {
public:
  typedef Scalar scalar_type;
  typedef Device device_type;
  typedef typename ScalarTraits<scalar_type>::val_type IST;
  Vector (/* ... */);
private:
  Kokkos::View<IST*, device_type> data_;
  std::shared_ptr<const Tpetra::Comm> comm_;
};

template<class ImplScalar, class Device>
typename InnerProductSpaceTraits<ImplScalar>::dot_type
lclDot (const Kokkos::View<const ImplScalar*, Device>& x,
        const Kokkos::View<const ImplScalar*, Device>& y) {
  typedef InnerProductSpaceTraits<ImplScalar> IPT;
  typedef typename IPT::dot_type dot_type;
  dot_type lclSum = ScalarTraits<dot_type>::zero ();
  Kokkos::parallel_reduce (x.dimension_0 (),
    KOKKOS_LAMBDA (const int i, dot_type& inout) {
      inout += IPT::dot (x(i), y(i));
    }, lclSum);
  return lclSum;
}

template<class Scalar, class Device>
typename InnerProductSpaceTraits<
  typename Vector<Scalar, Device>::IST>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y) {
  auto lclSum = lclDot (x.data_, y.data_);
  typedef decltype (lclSum) dot_type;
  dot_type gblSum = ScalarTraits<dot_type>::zero ();
  allReduce (lclSum, &gblSum, TPETRA_SUM, *x.comm_);
  return gblSum;
}
```

9 of 9: Explicit template instantiation

```
// Tpetra_Vector_decl.hpp:

template<class Scalar, class Device>
class Vector {
public:
  typedef Scalar scalar_type;
  typedef Device device_type;
  typedef typename ScalarTraits<scalar_type>::val_type IST;
  Vector (/* ... */);
private:
  Kokkos::View<IST*, device_type> data_;
  std::shared_ptr<const Tpetra::Comm> comm_;
};

template<class SC, class Device>
typename InnerProductSpaceTraits<SC>::dot_type
lclDot (const Kokkos::View<const SC*, Device>& x,
        const Kokkos::View<const SC*, Device>& y);


template<class Scalar, class Device>
typename InnerProductSpaceTraits<
  typename Vector<Scalar, Device>::IST>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y);

// Tpetra_Vector_def.hpp:

template<class Scalar, class Device>
Vector<Scalar, Device>::Vector ( /* ... */ ) { /* ... */ }

template<class ImplScalar, class Device>
typename InnerProductSpaceTraits<ImplScalar>::dot_type
lclDot (const Kokkos::View<const ImplScalar*, Device>& x,
        const Kokkos::View<const ImplScalar*, Device>& y)
{ /* ... */ }
```

```
template<class Scalar, class Device>
typename InnerProductSpaceTraits<
  typename Vector<Scalar, Device>::IST>::dot_type
dot (const Vector<Scalar>& x, const Vector<Scalar>& y)
{ /* ... */ }

// Use this in a header file to make an explicit
instantiation.

#define TPETRA_VECTOR_DEF( SC, DT ) template Vector<SC,
DT>;
```

Not shown: Lots of CMake code:

- Decide what Scalar, etc. types get enabled
- Generate .cpp files automatically using the above macro, 1 per file, so builds use less memory & take less time

Next steps

Next steps

Sandia National Laboratories

- Thread-parallelize all needed solver / preconditioner kernels
  - That's mostly not my job – colleagues are supposed to do that
  - It's really hard to get them to go from prototypes to production
- Give better examples for thread-parallel fill
  - If apps must own fill, we need to show them what to do
  - Interesting for graphs with less predictable structure
- Support execution in multiple concurrent tasks
  - Different CUDA streams
  - Task-parallel programming models
- Rethink MPI / thread interactions
  - Avoid sync between (un)packing data & MPI communication
  - More communication / computation overlap

Thanks!

Sandia National Laboratories

- Kokkos refactor of Tpetra has been & is a HUGE effort
- Tpetra is only the beginning – lots of solver effort too
- Many Trilinos developers have contributed in some way

github.com/kokkos

github.com/kokkos/kokkos-kernels

trilinos.org

Extra slides

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

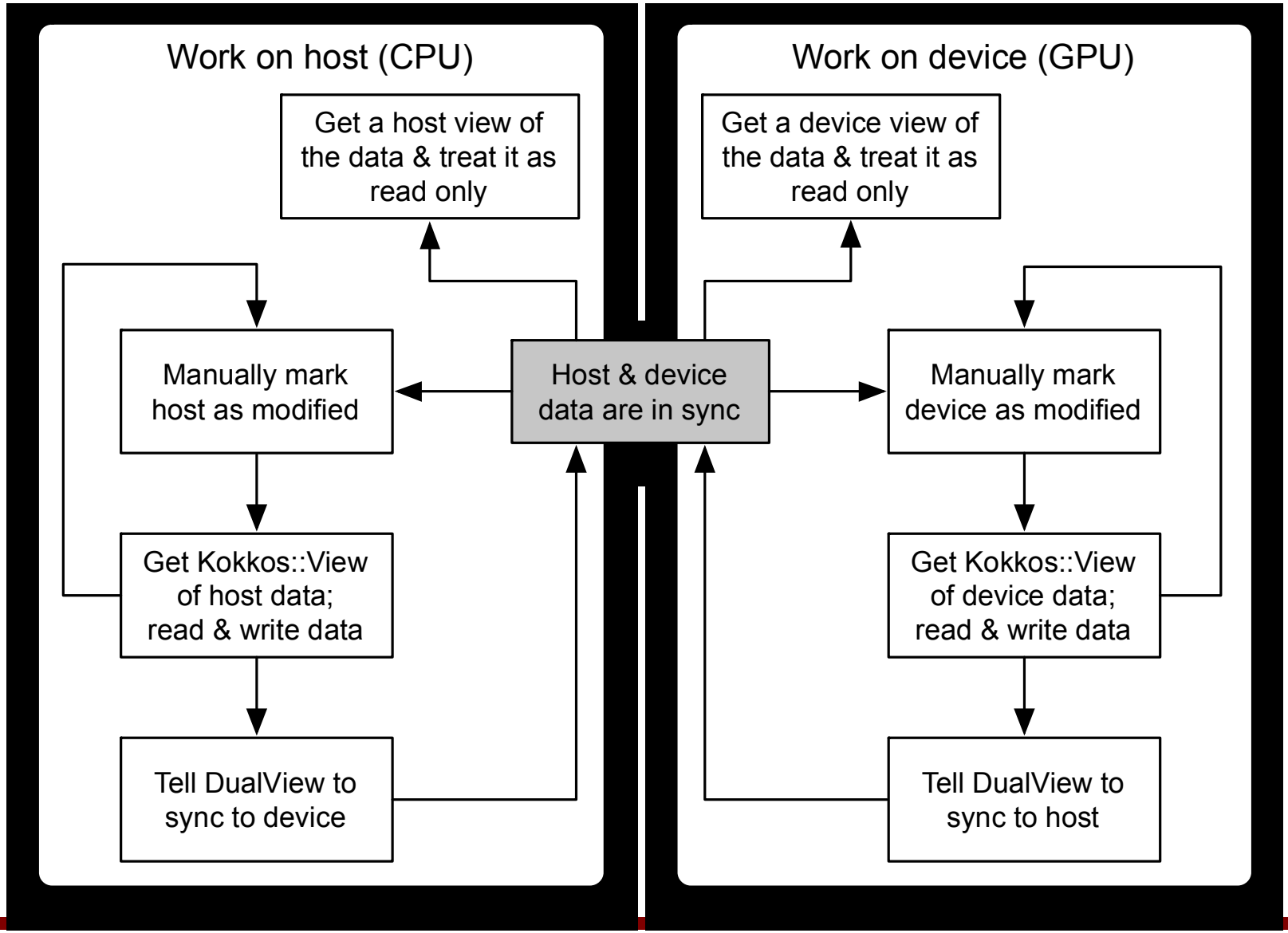
- Support 3 use cases
  1. Gradual porting (mix new Kokkos-ized code with legacy host code)
  2. Use “fast” memory as temp work space (page in / out as needed)
  3. Concurrently use 2 exec spaces (e.g., MPI pack on host, compute on device)

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

- 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)
  - Dynamic graph structure changes may be host only for a while
  - Recommended: build Kokkos data structures & pass off to Tpetra

Thread-parallel fill interface options

Sandia National Laboratories

Coarse-grained (batched)

- Pass many items into linear algebra interface at once
- Library parallelizes inside
- **(+)** Need not be thread safe
- **(+)** Hides complexity
- **(-)** Less backwards compatible
- **(-)** Synchronizes
- **(-)** No cross-kernel reuse
- **(-)** Need enough parallelism to keep whole device busy

Fine-grained

- 1 or few item(s) at a time
- User parallelizes outside
- **(-)** Interface must be thread safe & scalable
- **(-)** Limited parallelism inside
- **(+)** More backwards compatible
- **(+)** Does not synchronize
- **(+)** Users can exploit reuse
- **(+)** Works w/ team/thread

4 Tpetra eras, 3 Kokkos versions

Sandia National Laboratories

- Mike Heroux, Paul Sexton, Kris Kampshoff: 2002-5
  - "Kokkos" (0.x) (package for non-threaded 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": thread-parallel progr. model
  - 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

1999's problems same as today's

- Need to consider fine-grained parallelism
- MPI scaling matters

Solver interfaces ignored fine-grained parallelism

Sandia National Laboratories

- Failure to anticipate end of Dennard scaling
  - 1st hint: Instruction-level parallelism (ILP) in standard HPC processors
  - 2nd hint: Single Instruction Multiple Data (SIMD) ("short vectors")
  - 3rd hint: Multicore consumer processors (2004)
- Too much focus on MPI weak scaling
  - Most apps care about strong scaling on modest process counts
- Failure of solver & app folks to communicate
  - Solver folks want to publish ➔ hard problems & interesting solvers
  - Apps want to minimize software dependencies ➔ only use third-party solver libraries for hard problems
  - Apps only seek help with hard problems, not common problems!
  - Sparse matrix collections are not "representative problems"

1999's problems same as today's

Sandia National Laboratories

- KNL & GPUs expose this cost
  - Branches (e.g., finding value of A(i,j) in a sparse matrix) expensive
  - Many '90s & 00's hardware papers on crazy branch prediction
  - Should have seen this coming!
- MPI scaling still matters
  - Incredible effort to scale algebraic multigrid setup to ~1M processes
  - Trilinos' design hid performance issues by relying on proven software
  - (No "Epetra-based solvers" – they used AztecOO, ML, Zoltan)