The Trilinos Project Exascale Roadmap

Michael A. Heroux
Sandia National Laboratories

Contributors: Mark Hoemmen, Siva Rajamanickam, Tobias Wiesner, Alicia Klinvex, Heidi Thornquist, Kate Evans, Andrey Prokopenko, Lois McInnes

trilinos.github.io
Outline

- Challenges.
- Brief Overview of Trilinos.
- On-node parallelism.
- Parallel Algorithms.
- Trilinos Products Organization.
- ForTrilinos.
- The xSDK.
Challenges

- On-node concurrency expression:
  - Vectorization/SIMT.
  - On-node tasking.
  - Portability.

- Parallel algorithms:
  - Vector/SIMT expressible.
  - Latency tolerant.
  - Highly scalable.

- Multi-scale and multi-physics.
  - Preconditioning.
  - Software composition.

- Resilience (Discussed tomorrow morning MS40).
Trilinos Overview
What is Trilinos?

- Object-oriented software framework for…
- Solving big complex science & engineering problems.
- Large collection of reusable scientific capabilities.
- More like LEGO™ bricks than Matlab™.
Optimal Kernels to Optimal Solutions:
- Geometry, Meshing
- Discretizations, Load Balancing.
- Scalable Linear, Nonlinear, Eigen, Transient, Optimization, UQ solvers.
- Scalable I/O, GPU, Manycore

- 60+ Packages.
- Other distributions:
  - Cray LIBSCI.
  - GitHub repo.
- Thousands of Users.
- Worldwide distribution.

Each stage requires greater performance and error control of prior stages: Always will need: more accurate and scalable methods, more sophisticated tools.
# Trilinos Package Summary

<table>
<thead>
<tr>
<th>Objective</th>
<th>Package(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discretizations</strong></td>
<td></td>
</tr>
<tr>
<td>Meshing &amp; Discretizations</td>
<td>STK, Intrepid, Pamgen, Sundance, ITAPS, Mesquite</td>
</tr>
<tr>
<td>Time Integration</td>
<td>Rythmos</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td>Automatic Differentiation</td>
<td>Sacado</td>
</tr>
<tr>
<td>Mortar Methods</td>
<td>Moertel</td>
</tr>
<tr>
<td><strong>Services</strong></td>
<td></td>
</tr>
<tr>
<td>Linear algebra objects</td>
<td>Epetra, Tpetra, Kokkos, Xpetra</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Thyra, Stratimikos, RTOp, FEI, Shards</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Zoltan, Isorropia, Zoltan2</td>
</tr>
<tr>
<td>“Skins”</td>
<td>PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika</td>
</tr>
<tr>
<td>C++ utilities, I/O, thread API</td>
<td>Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx, Trios</td>
</tr>
<tr>
<td><strong>Solvers</strong></td>
<td></td>
</tr>
<tr>
<td>Iterative linear solvers</td>
<td>AztecOO, Belos, Komplex</td>
</tr>
<tr>
<td>Direct sparse linear solvers</td>
<td>Amesos, Amesos2, ShyLU</td>
</tr>
<tr>
<td>Direct dense linear solvers</td>
<td>Epetra, Teuchos, Pliris</td>
</tr>
<tr>
<td>Iterative eigenvalue solvers</td>
<td>Anasazi, Rbgen</td>
</tr>
<tr>
<td>ILU-type preconditioners</td>
<td>AztecOO, IFPACK, Ifpack2, ShyLU</td>
</tr>
<tr>
<td>Multilevel preconditioners</td>
<td>ML, CLAPS, Muelu</td>
</tr>
<tr>
<td>Block preconditioners</td>
<td>Meros, Teko</td>
</tr>
<tr>
<td>Nonlinear system solvers</td>
<td>NOX, LOCA, Piro</td>
</tr>
<tr>
<td>Optimization (SAND)</td>
<td>MOOCHO, Aristos, TriKota, Globipack, Optipack</td>
</tr>
<tr>
<td>Stochastic PDEs</td>
<td>Stokhos</td>
</tr>
</tbody>
</table>
Unique features of Trilinos

- Huge library of algorithms
  - Linear and nonlinear solvers, preconditioners, …
  - Optimization, transients, sensitivities, uncertainty, …

- Growing support for multicore & hybrid CPU/GPU
  - Built into the new Tpetra linear algebra objects
    - Therefore into iterative solvers with zero effort!
  - Unified intranode programming model: Kokkos
  - Spreading into the whole stack:
    - Multigrid, sparse factorizations, element assembly…

- Support for mixed and arbitrary precisions
  - Don’t have to rebuild Trilinos to use it

- Support for flexible 2D sparse partitioning
  - Useful for graph analytics, other data science apps.

- Support for huge (> 2B unknowns) problems
### Abbreviations key
- “Gen.” = generalized eigenvalue system.
- “Prec.” = can use a preconditioner.
- BKS = Block Krylov Schur.
- LOBPCG = Locally Optimial Block Preconditioned Conjugate Gradient.
- RTR = Riemannian Trust-Region method.
- ^ denotes that these features may be implemented if there is sufficient interest.
- $ denotes that the TraceMin family of solvers is currently experimental.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BKS</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Block-Davidson</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Generalized-Davidson</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>LOBPCG</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>RTR</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>TraceMin family$</td>
<td>yes</td>
<td>yes</td>
<td>no^</td>
<td>no^</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Core team:
- Alicia Klinvex
- Rich Lehoucq
- Heidi Thornquist

### Works with:
- Epetra
- Tpetra
- Custom

[https://trilinos.github.io/anasazi.html](https://trilinos.github.io/anasazi.html)
Trilinos linear solvers

- **Sparse linear algebra**
  (Kokkos/KokkosKernels/Tpetra)
  - Threaded construction, Sparse graphs, (block) sparse matrices, dense vectors, parallel solve kernels, parallel communication & redistribution

- **Iterative (Krylov) solvers** (Belos)
  - CG, GMRES, TFQMR, recycling methods

- **Sparse direct solvers** (Amesos2)

- **Algebraic iterative methods** (Ifpack2)
  - Jacobi, SOR, polynomial, incomplete factorizations, additive Schwarz

- **Shared-memory factorizations** (ShyLU)
  - LU, ILU(k), ILUt, IC(k), iterative ILU(k)
  - Direct+iterative preconditioners

- **Segregated block solvers** (Teko)

- **Algebraic multigrid** (MueLu)
On-node data & execution
Must support > 3 architectures

- Coming systems to support
  - **Trinity** (Intel Haswell & KNL)
  - **Sierra**: NVIDIA GPUs + IBM multicore CPUs
  - Plus “everything else”

- 3 different architectures
  - Multicore CPUs (big cores)
  - Manycore CPUs (small cores)
  - GPUs (highly parallel)

- MPI only, & MPI + threads
  - Threads don’t always pay on non-GPU architectures today
  - Porting to threads must not slow down the MPI-only case
Kokkos: Performance, Portability, & Productivity

- LAMMPS
- Trilinos
- Sierra
- Albany

Kokkos

Multi-Core

Many-Core

APU

CPU + GPU
Kokkos Programming Model

Goal: One Code gives good performance on every platform

• Machine model:
  • N execution space + M memory spaces
  • NxM matrix for memory access performance/possibility
  • Asynchronous execution allowed

• Implementation approach
  • A C++ template library
  • C++11 now required
  • Target different back-ends for different hardware architecture
  • Abstract hardware details and execution mapping details away

• Distribution
  • Open Source library
  • Soon (i.e. in the next few weeks) available on GitHub

• Long Term Vision
  • Move features into the C++ standard (Carter Edwards voting committee member)
Abstraction Concepts

**Execution Pattern**: parallel_for, parallel_reduce, parallel_scan, task, …

**Execution Policy**: how (and where) a user function is executed
- E.g., data parallel range: concurrently call function(i) for i = [0..N)
- User’s function is a C++ functor or C++11 lambda

**Execution Space**: where functions execute
- Encapsulates hardware resources; e.g., cores, GPU, vector units, …

**Memory Space**: where data resides
- AND what execution space can access that data
- Also differentiated by access performance; e.g., latency & bandwidth

**Memory Layout**: how data structures are ordered in memory
- provide mapping from logical to physical index space

**Memory Traits**: how data shall be accessed
- allow specialisation for different usage scenarios (read only, random, atomic, …)
Execution Pattern

```c
#include <Kokkos_Core.hpp>
#include <cstdio>

int main(int argc, char* argv[]) {
    // Initialize Kokkos analogous to MPI_Init()
    // Takes arguments which set hardware resources (number of threads, GPU Id)
    Kokkos::initialize(argc, argv);

    // A parallel_for executes the body in parallel over the index space, here a simple range 0<=i<10
    // It takes an execution policy (here an implicit range as an int) and a functor or lambda
    // The lambda operator has one argument, and index_type (here a simple int for a range)
    Kokkos::parallel_for(10, [=](int i){
        printf("Hello %i\n",i);
    });

    // A parallel_reduce executes the body in parallel over the index space, here a simple range 0<=i<10 and
    // performs a reduction over the values given to the second argument
    // It takes an execution policy (here an implicit range as an int); a functor or lambda; and a return value
    double sum = 0;
    Kokkos::parallel_reduce(10, [=](int i, int& lsum) {
        lsum += i;
    },sum);
    printf("Result %lf\n",sum);

    // A parallel_scan executes the body in parallel over the index space, here a simple range 0<=i<10 and
    // Performs a scan operation over the values given to the second argument
    // If final == true lsum contains the prefix sum.
    double sum = 0;
    Kokkos::parallel_scan(10, [=](int i, int& lsum, bool final) {
        if(final) printf("ScanValue %i\n",lsum);
        lsum += i;
    });

    Kokkos::finalize();
}
```
Kokkos protects us against...

- Hardware divergence
- Programming model diversity
- Threads at all
  - Kokkos::Serial back-end
  - Kokkos’ semantics require vectorizable (ivdep) loops
- Expose parallelism to exploit later
- Hierarchical parallelism model encourages exploiting locality

Kokkos protects our HUGE time investment of porting Trilinos

Note: Kokkos is *not* magic, cannot make bad algorithms scale.

Kokkos is our hedge
Parallel Algorithms
Foundational Technology: KokkosKernels

- Provide BLAS (1,2,3); Sparse; Graph and Tensor Kernels
- Kokkos based: Performance Portable
- Interfaces to vendor libraries if applicable (MKL, CuSparse, ...)
- Goal: Provide kernels for all levels of node hierarchy

**Socket**
- Thread Teams, Shared L3, e.g. Full Solve

**Core**
- Thread Parallelism, Shared L1/L2, e.g. Subdomain Solve

**Hyper Thread**
- Vector Parallelism, Synch free, e.g. Matrix Row x Vector

**Vector Lane**
- Elemental Functions, Serial, e.g. 3x3 DGEMM
Kokkoskernels: Sparse Matrix-Matrix Multiplication (SpGEMM)

- SpGEMM is the most expensive part of the multigrid setup.
- New portable write-avoiding algorithm in KokkosKernels is ~14x faster than NVIDIA’s CUSPARSE on K80 GPUs.
- ~2.8x faster than Intel’s MKL on Intel’s Knights Landing (KNL).
- Memory scalable: Solving larger problems that cannot be solved by codes like NVIDIA’s CUSP and Intel’s MKL when using large number of threads.

- Up is good.
Kokkoskernels: Graph Coloring and Symmetric Gauss-Seidel

- **Goal:** Identify independent data that can be processed in parallel.
- **Performance:** Better quality (4x on average) and run time (1.5x speedup) w.r.t cuSPARSE.
- Enables parallelization of preconditioners: Gauss Seidel: 82x speedup on KNC, 136x on K20 GPUs

Kokkos and Kokkos Kernels are available independent of Trilinos: https://github.com/kokkos
ShyLU and Subdomain Solvers: Overview

- **MPI+X based subdomain solvers**
  - Decouple the notion of one MPI rank as one subdomain: Subdomains can span multiple MPI ranks each with its own subdomain solver using X or MPI+X

- **Subpackages of ShyLU**: Multiple Kokkos-based options for on-node parallelism
  - **Basker**: LU or ILU (t) factorization
  - **Tacho**: Incomplete Cholesky - IC (k)
  - **Fast-ILU**: Fast-ILU factorization for GPUs

- **KokkosKernels**: Coloring based Gauss-Seidel (M. Deveci), Triangular Solves (A. Bradley)
MueLu – The Trilinos Multigrid framework

**MueLu multigrid framework:**
- Extensible software layout
- Modularity: Preconditioner layout defined by small building blocks
- Logic: Building blocks connected through logical data dependencies
- Flexible user input system through XML files
- Designed for next-generation HPC systems

www.trilinos.org/packages/muelu
Object Construction Modifications:
Making MPI+X truly scalable (lots of work)

- Pattern:
  1. Count / estimate allocation size; may use Kokkos parallel_scan
  2. Allocate; use Kokkos::View for best data layout & first touch
  3. Fill: parallel_reduce over error codes; if you run out of space, keep going, count how much more you need, & return to (2)
  4. Compute (e.g., solve the linear system) using filled data structures

- Compare to Fill, Setup, Solve sparse linear algebra use pattern
- Fortran <= 77 coders should find this familiar
- Semantics change: Running out of memory not an error!
  - Always return: Either no side effects, or correct result
  - Callers must expect failure & protect against infinite loops
  - Generalizes to other kinds of failures, even fault tolerance
Trilinos Product Organization
Product Leaders: Maximize cohesion, control coupling

- **Product:**
  - Framework (J. Willenbring).
  - Data Services (K. Devine).
  - Linear Solvers (S. Rajamanickam).
  - Nonlinear Solvers (R. Pawlowski).
  - Discretizations (M. Perego).

- **Product focus:**
  - New, stronger leadership model.
  - **Focus:**
    - Published APIs
    - High cohesion within product. Low coupling across.
    - Deliberate product-level upstream planning & design.
ForTrilinos: Full, sustainable Fortran access to Trilinos

**History**

Via *ad hoc* interfaces, Fortran apps have benefitted from Trilinos capabilities for many years, most notably climate.

**Why ForTrilinos**

Trilinos provide a large collection of robust, production scientific C++ software, including state-of-the-art manycore/GPU capabilities. ForTrilinos will give access to Fortran apps.

### Research Details

- Access via *ad hoc* APIs has existed for years, especially in the climate community.
- ForTrilinos provides native, sustainable APIs, including support for user-provided Fortran physics-based preconditioning and user-defined Fortran operators.
- Use of robust auto-generation code and API tools makes future extensibility feasible.
- Auto-generation tools apply to other projects.
- ForTrilinos will provide early access to latest scalable manycore/GPU functionality, sophisticated solvers.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
<th>App Role</th>
<th>Trilinos Role</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic matrix and parameter list construction, call Trilinos solver</td>
<td>Explicitly construct Trilinos matrix, RHS row-by-row. Define parameters (which preconditioner to user, which iterative method, etc.) Call Trilinos solver.</td>
<td>Solve the problem using the solver configuration specified by the user and Trilinos linear, nonlinear functions and stopping criteria.</td>
<td>Partial manual and semi-automated generation of Fortran interfaces, documentation and testing.</td>
</tr>
<tr>
<td>2</td>
<td>Inversion of Control for linear/nonlinear operators, status tests, matrix coefficient values</td>
<td>Register user-provided Fortran functions to compute linear/nonlinear operator, stopping criteria (status tests), or provide matrix coefficient values, in some combination. Define solver parameters.</td>
<td>Solve the problem using the prescribed solver configuration and the Fortran function provided by the user.</td>
<td>Primarily semi-automated generation of Fortran interfaces, documentation and testing.</td>
</tr>
<tr>
<td>3</td>
<td>Support for heterogeneous (host+device memory and execution) solver execution.</td>
<td>Construct data via Trilinos functions (similar to Phase 1) or provide Fortran functions for heterogeneous execution. Define solver parameters.</td>
<td>Solve the problem using the prescribed solver configuration on heterogeneous processors.</td>
<td>Semi-automated generation of code, documentation and testing.</td>
</tr>
</tbody>
</table>

Michael Heroux (PI, SNL), Kate Evans (co-PI, ORNL), dev team based at ORNL.

https://github.com/flang-compiler
The Extreme-Scale Scientific Software Development Kit (xSDK)
Extreme-scale Scientific Software Ecosystem

Domain component interfaces
- Data mediator interactions.
- Hierarchical organization.
- Multiscale/multiphysics coupling.

Native code & data objects
- Single use code.
- Coordinated component use.
- Application specific.

Shared data objects
- Meshes.
- Matrices, vectors.

Library interfaces
- Parameter lists.
- Interface adapters.
- Function calls.

Documentation content
- Source markup.
- Embedded examples.

Testing content
- Unit tests.
- Test fixtures.

Build content
- Rules.
- Parameters.

Domain components
- Reacting flow, etc.
- Reusable.

Libraries
- Solvers, etc.
- Interoperable

Frameworks & tools
- Doc generators.
- Test, build framework.

SW engineering
- Productivity tools.
- Models, processes.

Focus of key accomplishments: xSDK foundations

Extreme-scale Scientific Software Development Kit (xSDK)
Building the foundation of a highly effective extreme-scale scientific software ecosystem

Focus: Increasing the functionality, quality, and interoperability of important scientific libraries, domain components, and development tools

- **xSDK release 0.2.0: April 2017 (soon)**
  - Spack package installation
    - `./spack install xsdk`
  - Package interoperability
    - Numerical libraries
      - hypre, PETSc, SuperLU, Trilinos
    - Domain components
      - Alquimia, PFLOTRAN
- **xSDK community policies:**
  - Address challenges in interoperability and sustainability of software developed by diverse groups at different institutions

Impact:
- Improved code quality, usability, access, sustainability
- Inform potential users that an xSDK member package can be easily used with other xSDK packages
- Foundation for work on performance portability, deeper levels of package interoperability
# xSDK community policies

**xSDK compatible package:** Must satisfy mandatory xSDK policies:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M1.</strong></td>
<td>Support xSDK community GNU Autoconf or CMake options.</td>
</tr>
<tr>
<td><strong>M2.</strong></td>
<td>Provide a comprehensive test suite.</td>
</tr>
<tr>
<td><strong>M3.</strong></td>
<td>Employ user-provided MPI communicator.</td>
</tr>
<tr>
<td><strong>M4.</strong></td>
<td>Give best effort at portability to key architectures.</td>
</tr>
<tr>
<td><strong>M5.</strong></td>
<td>Provide a documented, reliable way to contact the development team.</td>
</tr>
<tr>
<td><strong>M6.</strong></td>
<td>Respect system resources and settings made by other previously called packages.</td>
</tr>
<tr>
<td><strong>M7.</strong></td>
<td>Come with an open source license.</td>
</tr>
<tr>
<td><strong>M8.</strong></td>
<td>Provide a runtime API to return the current version number of the software.</td>
</tr>
<tr>
<td><strong>M9.</strong></td>
<td>Use a limited and well-defined symbol, macro, library, and include file name space.</td>
</tr>
<tr>
<td><strong>M10.</strong></td>
<td>Provide an accessible repository (not necessarily publicly available).</td>
</tr>
<tr>
<td><strong>M11.</strong></td>
<td>Have no hardwired print or IO statements.</td>
</tr>
<tr>
<td><strong>M12.</strong></td>
<td>Allow installing, building, and linking against an outside copy of external software.</td>
</tr>
<tr>
<td><strong>M13.</strong></td>
<td>Install headers and libraries under <code>&lt;prefix&gt;/include/</code> and <code>&lt;prefix&gt;/lib/</code>.</td>
</tr>
<tr>
<td><strong>M14.</strong></td>
<td>Be buildable using 64 bit pointers. 32 bit is optional.</td>
</tr>
</tbody>
</table>

Also specify **recommended policies**, which currently are encouraged but not required:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R1.</strong></td>
<td>Have a public repository.</td>
</tr>
<tr>
<td><strong>R2.</strong></td>
<td>Possible to run test suite under valgrind in order to test for memory corruption issues.</td>
</tr>
<tr>
<td><strong>R3.</strong></td>
<td>Adopt and document consistent system for error conditions/exceptions.</td>
</tr>
<tr>
<td><strong>R4.</strong></td>
<td>Free all system resources it has acquired as soon as they are no longer needed.</td>
</tr>
<tr>
<td><strong>R5.</strong></td>
<td>Provide a mechanism to export ordered list of library dependencies.</td>
</tr>
</tbody>
</table>

**xSDK member package:** Must be an xSDK-compatible package, *and* it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

[https://xsdk.info/policies](https://xsdk.info/policies)
**Status**

ASCRIPT xSDK Release 0.1 and 0.2 gives application teams single-point installation, access to Trilinos, hypre, PETSc and SuperLU.

**Value**

The xSDK is essential for multi-scale/physics application coupling and interoperability, and broadest access to critical libraries. xSDK efforts expand collaboration scope to all labs.

**Research Details**

- xSDK started under DOE ASCR (Ndousse-Fetter, 2014).
- Prior to xSDK, very difficult to use xSDK member libs together, ad hoc approach required, versioning hard.
- Latest release (and future) uses Spack for builds.
- xSDK4ECP will include 3 additional libs.
- xSDK efforts enable new scope of cross-lab coordination and collaboration.
- Long-term goal: Create community and policy-based library and component ecosystems for compositional application development.

---

**Tested on key machines at ALCF, OLCF, NERSC, also Linux and Mac OS X**

More xSDK info

- **Paper: xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit**
  - To appear in *Supercomputing Frontiers and Innovations*, 2017

- **CSE17 Posters:**
  - **xSDK: Working toward a Community Software Ecosystem**
    - [https://doi.org/10.6084/m9.figshare.4531526](https://doi.org/10.6084/m9.figshare.4531526)
  - **Managing the Software Ecosystem with Spack**
    - [https://doi.org/10.6084/m9.figshare.4702294](https://doi.org/10.6084/m9.figshare.4702294)
Final Take-Away Points

- Intra-node parallelism is biggest challenge right now:
  - Kokkos provides vehicle for reasoning and implementing on-node parallel.
    - Eventual goal: Search and replace Kokkos:: with std::
  - Node-parallel algorithms are already available.
  - Fully node parallel execution is hard work.

- Inter-node parallelism:
  - Muelu framework provide flexibility:
    - Pluggable, customizable components.
    - Multi-physics.

- Trilinos Products:
  - Improves upstream planning abilities.
  - Enables increased cohesion, reduced (incidental) coupling.

- Community expansion:
  - ForTrilinos provides native access and extensibility for Fortran users.
  - xSDK provides turnkey, consistent access to growing set of libraries.
    - Contact if interested in joining.
  - EuroTUG 2018: Between ISC and PASC?