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

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

Outline

Sandia National Laboratories

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

Challenges

Sandia National Laboratories

- 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).

Sandia National Laboratories

Trilinos Overview

What is Trilinos?

Sandia National Laboratories

- Object-oriented software framework for…
- Solving big complex science & engineering problems.
- Large collection of reusable scientific capabilities.
- More like LEGO™ bricks than Matlab™.

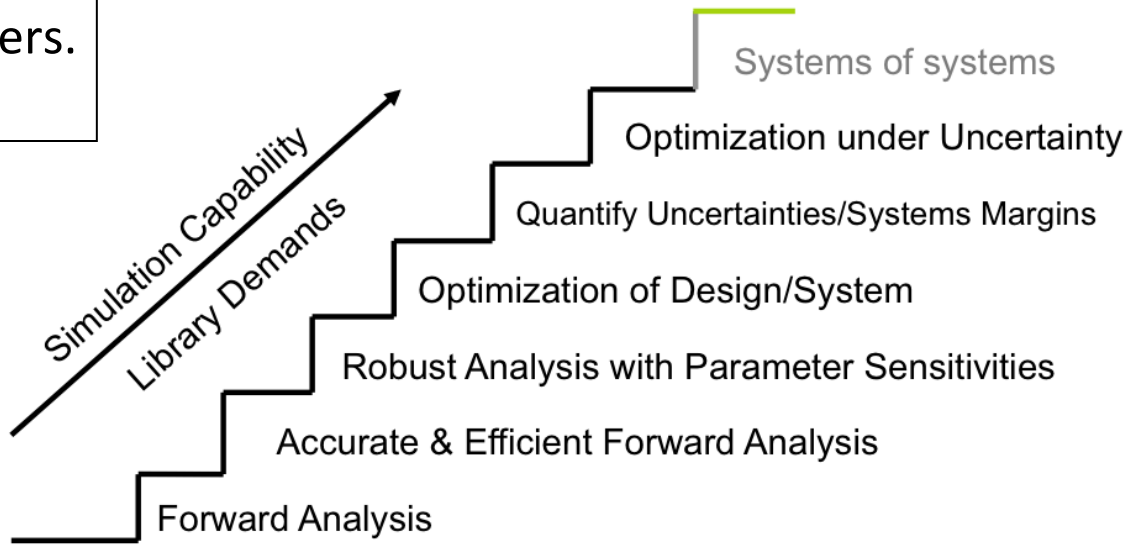
Trilinos

R&D 100

Optimal Kernels to Optimal Solutions:

- Geometry, Meshing
- Discretizations, Load Balancing.
- Scalable Linear, Nonlinear, Eigen, Transient, Optimization, UQ solvers.
- Scalable I/O, GPU, Manycore

Laptops to Leadership systems



- 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

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

Unique features of Trilinos

Sandia National Laboratories

- 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

Anasazi Snapshot

https://trilinos.github.io/anasazi.html

Sandia National Laboratories

Solver	Gen.	Prec.	Complex Herm.	Non-Herm.	Interior
BKS	yes	no	yes	yes	yes
Block-Davidson	yes	yes	yes	no	yes
Generalized-Davidson	yes	yes	no	yes	yes
LOBPCG	yes	yes	yes	no	no
RTR	yes	yes	yes	no	no
TraceMin family$	yes	yes	no^	no^	yes

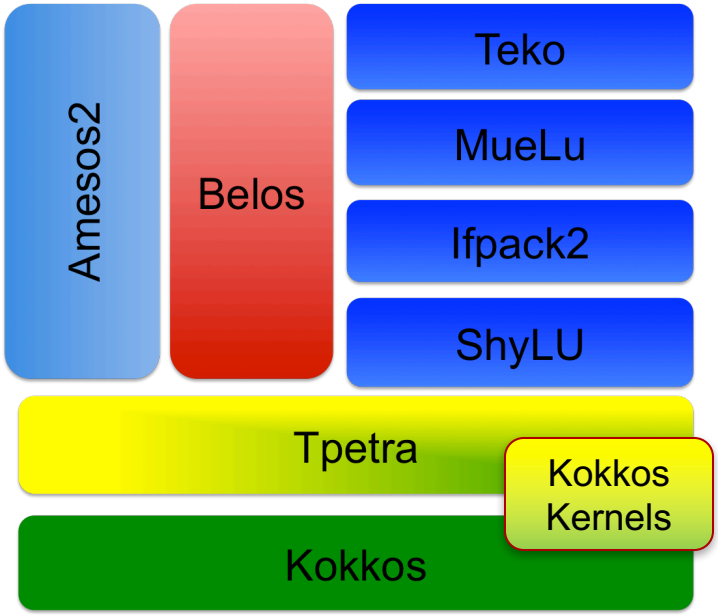
- Abbreviations key
  - "Gen." = generalized eigenvalue system.
  - "Prec:" = can use a preconditioner.
  - BKS = Block Krylov Schur.
  - LOBPCG = Locally Optimial Block Preconditioned Conjugate Gradient.
  - RTR = Riemannian Trust-Region method.
- ^ denotes that these features may be implemented if there is sufficient interest.
- $ denotes that the TraceMin family of solvers is currently experimental.

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

Works with:
- Epetra
- Tpetra
- Custom

Trilinos linear solvers

Sandia National Laboratories



- 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)

Sandia National Laboratories

On-node data & execution

Must support > 3 architectures

Sandia National Laboratories

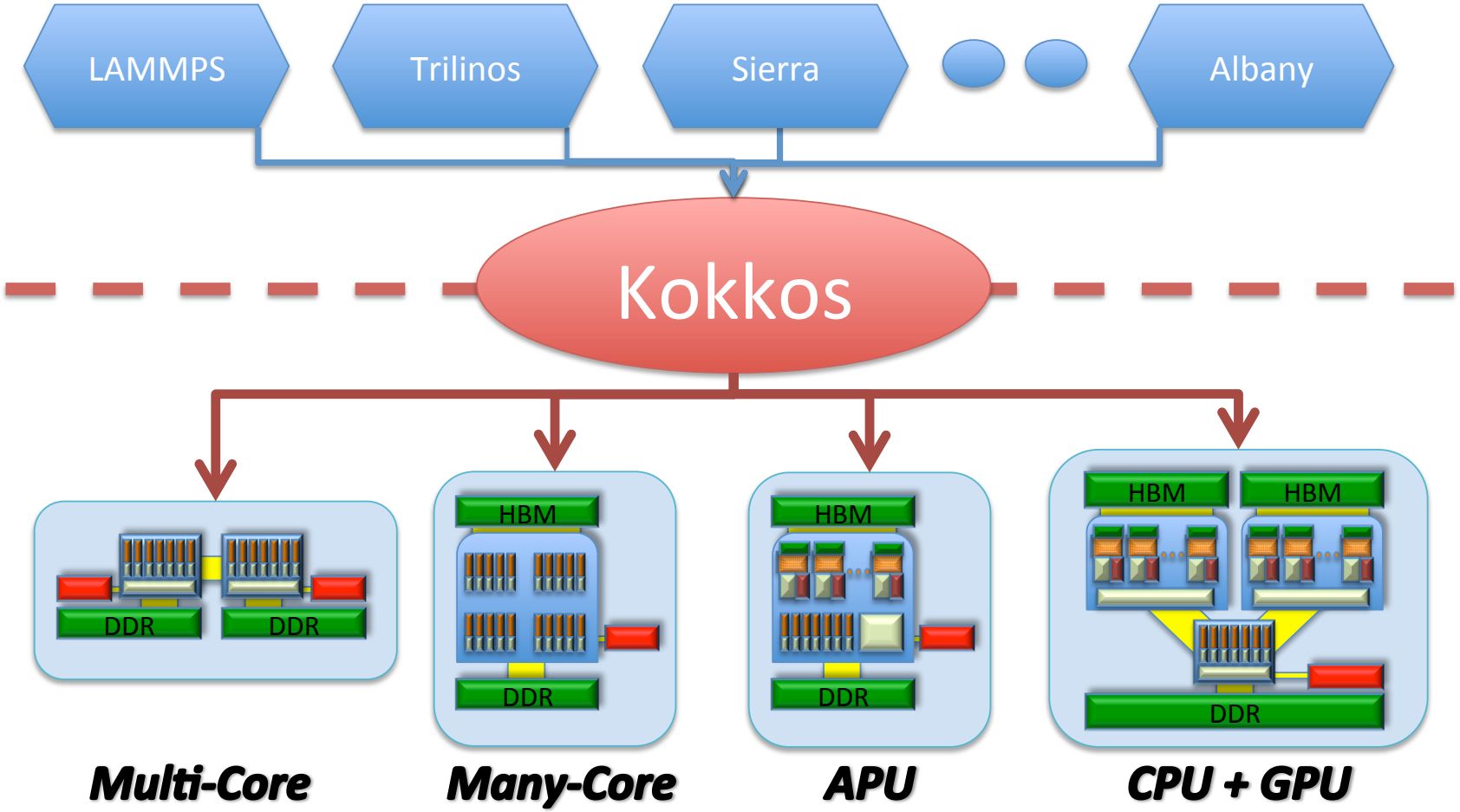
- Coming systems to support
  - Trinity (Intel Haswell & KNL)
  - Sierra: NVIDIA GPUs + IBM multicore CPUs
  - Plus “everything else”
- 3 different architectures
  - Multicore CPUs (big cores)
  - Manycore CPUs (small cores)
  - GPUs (highly parallel)
- MPI only, & MPI + threads
  - Threads don’t always pay on non-GPU architectures today
  - Porting to threads must not slow down the MPI-only case

CORAL COLLABORATION
OAK RIDGE • ARGONNE • LIVERMORE

SIERRA

Kokkos: *Performance, Portability, & Productivity*

Sandia National Laboratories



Kokkos Programming Model

Goal: One Code gives good performance on every platform

Sandia National Laboratories

- 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

Sandia National Laboratories

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

Sandia National Laboratories

```
#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.

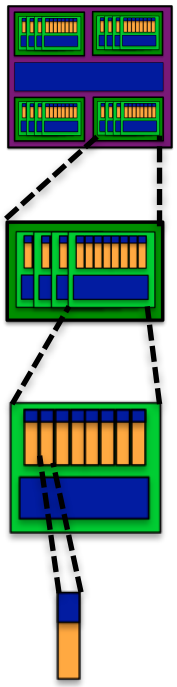
Kokkos is our hedge

Sandia National Laboratories

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

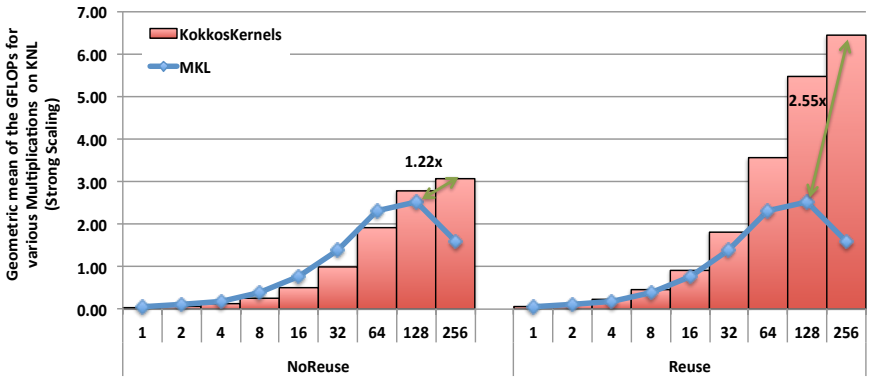
Hyper Thread

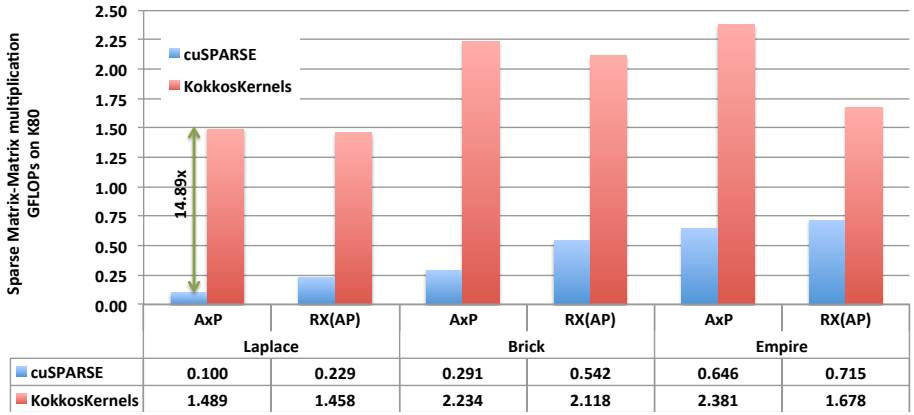
- Vector Parallelism, Synch free, e.g. Matrix Row x Vector

Vector Lane

- Elemental Functions, Serial, e.g. 3x3 DGEMM

Kokkoskernels: Sparse Matrix-Matrix Multiplication (SpGEMM)



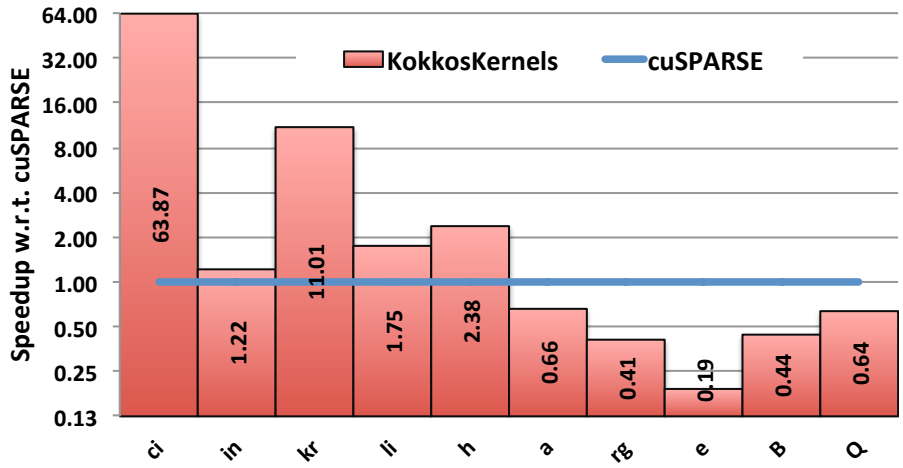


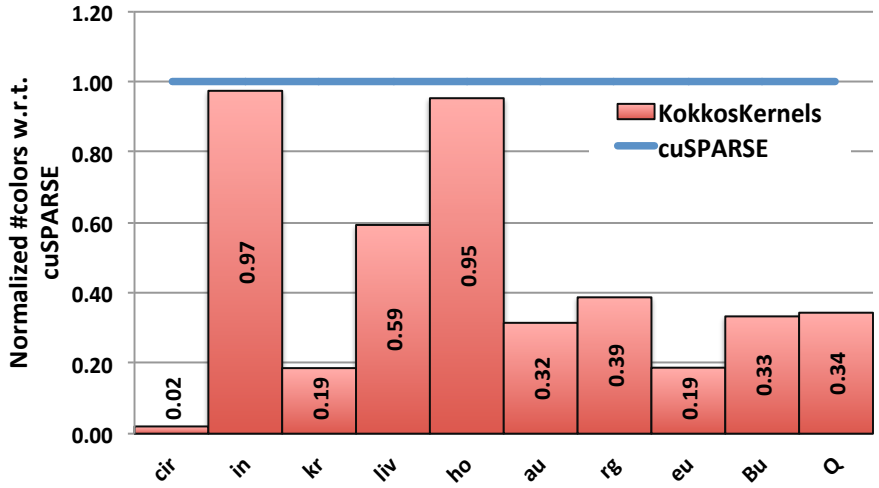
	Laplace		Brick		Empire	
	AxP	RX(AP)	AxP	RX(AP)	AxP	RX(AP)
cuSPARSE	0.100	0.229	0.291	0.542	0.646	0.715
KokkosKernels	1.489	1.458	2.234	2.118	2.381	1.678

- 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.

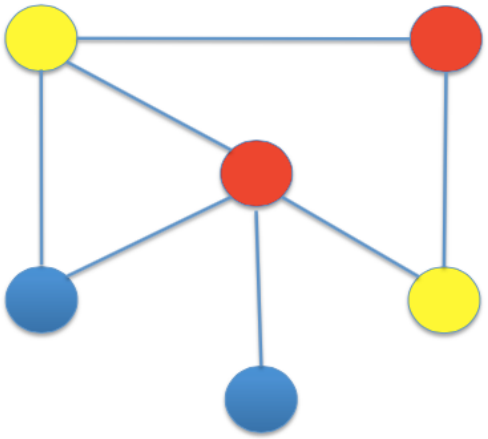
- **Up is good.**

Kokkoskernels:
Graph Coloring and Symmetric Gauss-Seidel



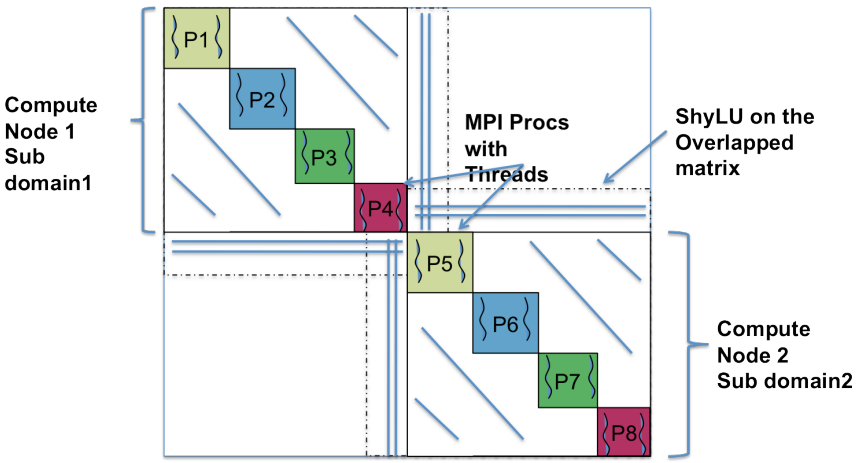


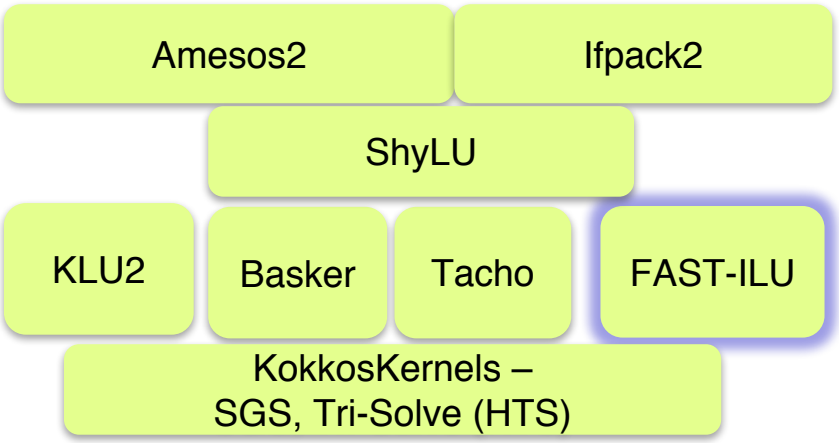
- Goal: Identify independent data that can be processed in parallel.
- Performance: Better quality (4x on average) and run time (1.5x speedup) w.r.t cuSPARSE.
- Enables parallelization of preconditioners: Gauss Seidel: 82x speedup on KNC, 136x on K20 GPUs



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)

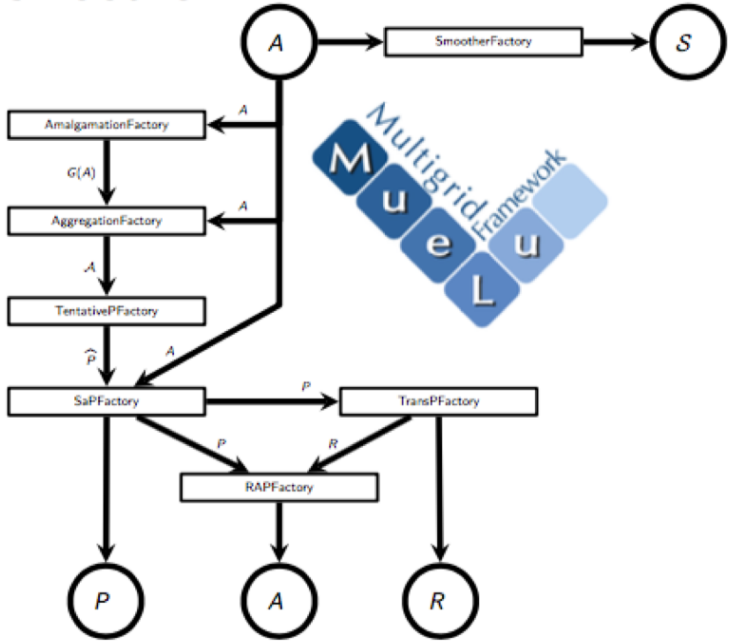
MueLu – The Trilinos Multigrid framework

Sandia National Laboratories

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

Example: Building blocks for transfer operators and level smoother



`www.trilinos.org/packages/muelu`

Object Construction Modifications:
Making MPI+X truly scalable (lots of work)

Sandia National Laboratories

- 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

Sandia National Laboratories

Trilinos Product Organization

Product Leaders:
Maximize cohesion, control coupling

Sandia National Laboratories

- 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

https://github.com/flang-compiler

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.

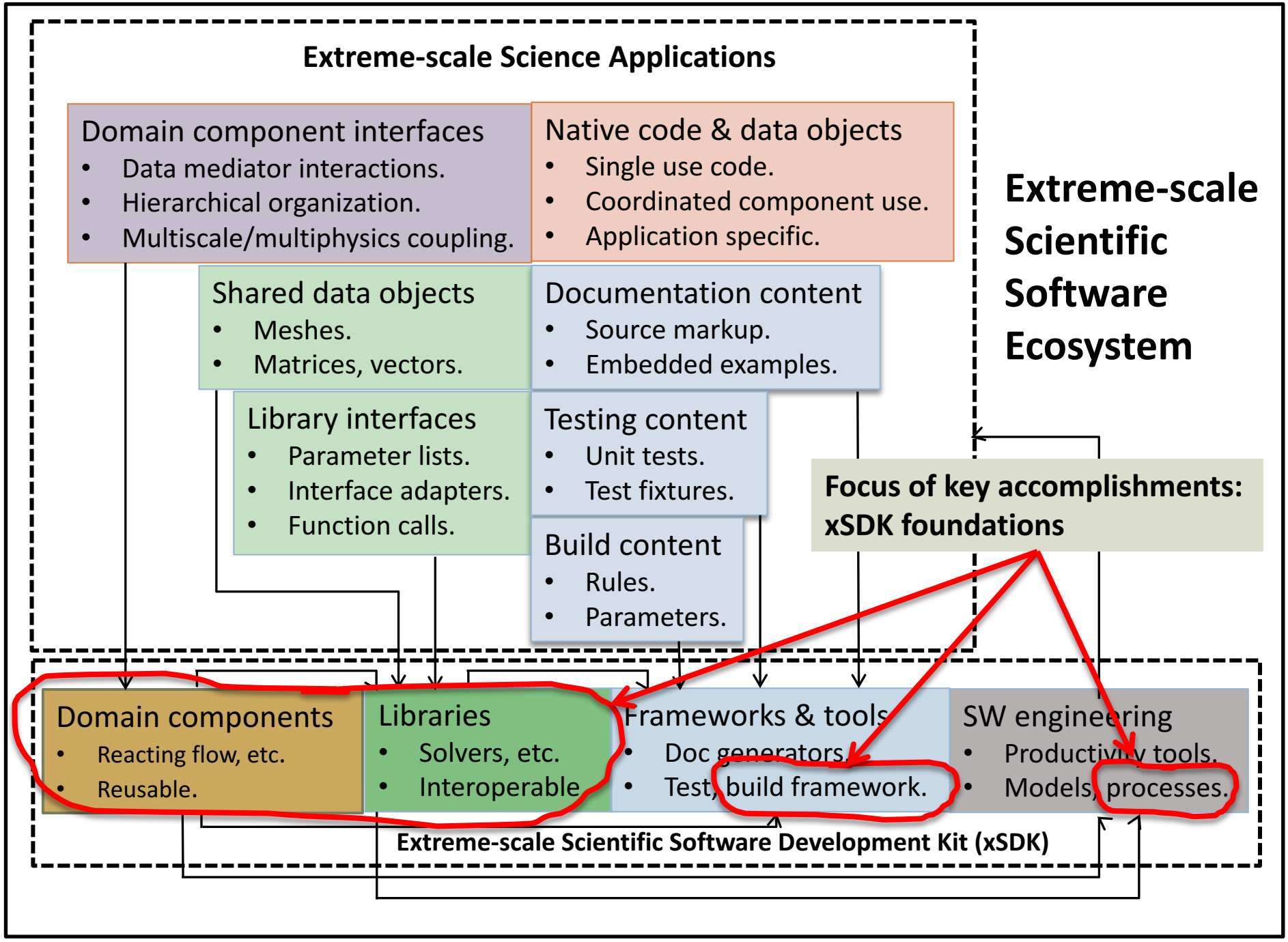
Phase	Description	App Role	Trilinos Role	Approach
1	Basic matrix and parameter list construction, call Trilinos solver	Explicitly construct Trilinos matrix, RHS row-by-row. Define parameters (which preconditioner to user, which iterative method, etc.) Call Trilinos solver.	Solve the problem using the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

U.S. DEPARTMENT OF ENERGY | Office of Science

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

Sandia National Laboratories

The Extreme-Scale Scientific Software Development Kit (xSDK)

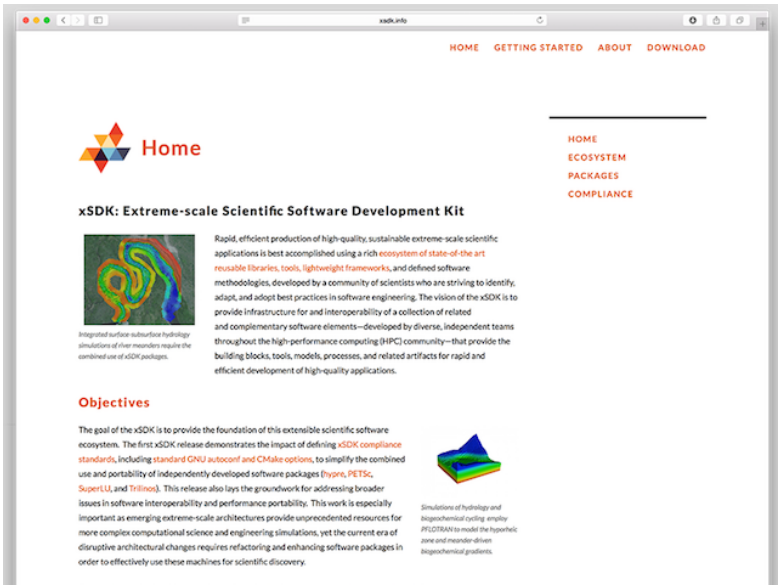

Extreme-scale Science Applications
Domain component interfaces
• Data mediator interactions.
• Hierarchical organization.
• Multiscale/multiphysics coupling.
Native code & data objects
• Single use code.
• Coordinated component use.
• Application specific.
Shared data objects
• Meshes.
• Matrices, vectors.
Documentation content
• Source markup.
• Embedded examples.
Library interfaces
• Parameter lists.
• Interface adapters.
• Function calls.
Testing content
• Unit tests.
• Test fixtures.
Build content
• Rules.
• Parameters.
Extreme-scale Scientific Software Ecosystem
Focus of key accomplishments: xSDK foundations
Domain components
• Reacting flow, etc.
• Reusable.
Libraries
• Solvers, etc.
• Interoperable
Frameworks & tools
• Doc generators.
• Test, build framework.
SW engineering
• Productivity tools.
• Models, processes.
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

website: xSDK.info



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

IDEAS productivity

xSDK community policies

xSDK compatible package: Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

M5. Provide a documented, reliable way to contact the development team.

M6. Respect system resources and settings made by other previously called packages.

M7. Come with an open source license.

M8. Provide a runtime API to return the current version number of the software.

M9. Use a limited and well-defined symbol, macro, library, and include file name space.

M10. Provide an accessible repository (not necessarily publicly available).

M11. Have no hardwired print or IO statements.

M12. Allow installing, building, and linking against an outside copy of external software.

M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.

M14. Be buildable using 64 bit pointers. 32 bit is optional.

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

R1. Have a public repository.

R2. Possible to run test suite under valgrind in order to test for memory corruption issues.

R3. Adopt and document consistent system for error conditions/exceptions.

R4. Free all system resources it has acquired as soon as they are no longer needed.

R5. Provide a mechanism to export ordered list of library dependencies.

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

IDEAS productivity

xSDK4 Snapshot

Status

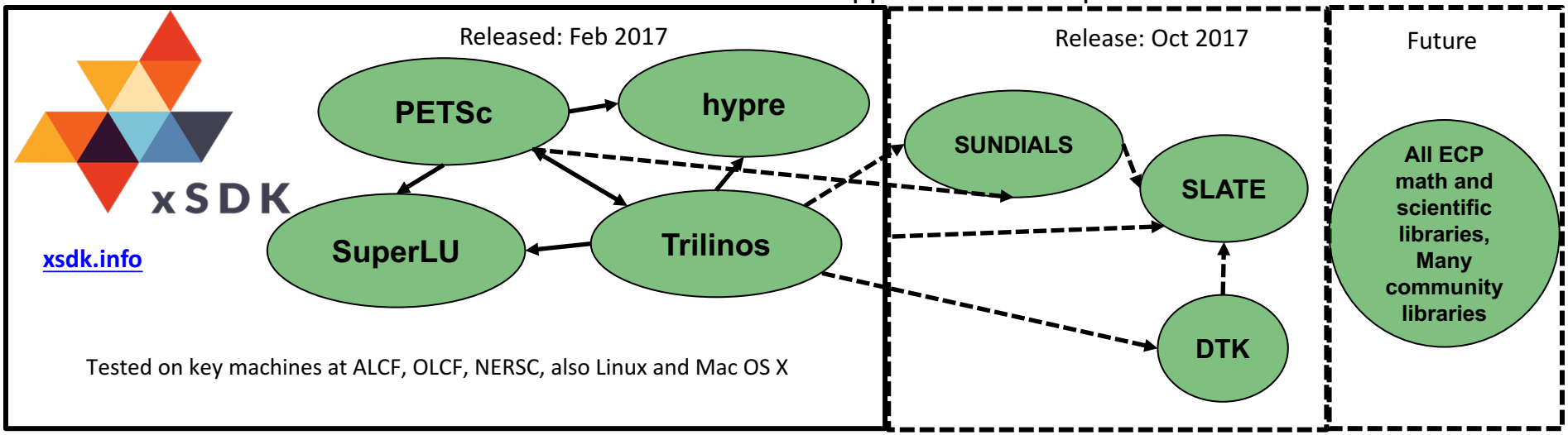
ASCR xSDK Release 0.1 and 0.2 gives application teams single-point installation, access to Trilinos, hypre, PETSc and SuperLU.

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.



Michael Heroux (co-lead PI, SNL), Lois Curfman McInnes (co-lead PI, ANL), James Willenbring (Release lead, SNL)

U.S. DEPARTMENT OF ENERGY | Office of Science

Ref: *xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit,*, Feb 2017, https://arxiv.org/abs/1702.08425, to appear in **Supercomputing Frontiers and Innovations**.

More xSDK info

- **Paper: xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit**
  - R. Bartlett, I. Demeshko, T. Gamblin, G. Hammond, M. Heroux, J. Johnson, A. Klinvex, X. Li, L.C. McInnes, D. Osei-Kuffuor, J. Sarich, B. Smith, J. Willenbring, U.M. Yang
  - https://arxiv.org/abs/1702.08425
  - 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
  - **Managing the Software Ecosystem with Spack**
    - https://doi.org/10.6084/m9.figshare.4702294

Final Take-Away Points

Sandia National Laboratories

- 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?