Scientific Programming 20 (2012) 83–88
DOI 10.3233/SPR-2012-0355
IOS Press

A new overview of the Trilinos project

Michael A. Heroux [*] and James M. Willenbring
Sandia National Laboratories, Albuquerque, NM, USA

Abstract. Since *An Overview of the Trilinos Project* [*ACM Trans. Math. Softw.* **31**(3) (2005), 397–423] was published in 2005, Trilinos has grown significantly. It now supports the development of a broad collection of libraries for scalable computational science and engineering applications, and a full-featured software infrastructure for rigorous lean/agile software engineering. This growth has created significant opportunities and challenges. This paper focuses on some of the most notable changes to the Trilinos project in the last few years.

At the time of the writing of this article, the current release version of Trilinos was 10.12.2.

Keywords: Software libraries, software frameworks

1. Introduction

The Trilinos project is a community of developers and a collection of reusable software components called *packages*. Trilinos provides a large and growing collection of open-source software libraries for scalable parallel computational science and engineering applications, as well as a software infrastructure that supports a rigorous lean/agile software lifecycle model [13].

The initial goals of the Trilinos Project were to foster research and development in the area of mathematical solver software, and to support production-quality solvers resulting from that effort [1]. The project showed early signs of success and received an R&D 100 award in 2004 [12]. In 2007, the scope of Trilinos expanded beyond solvers to include a broad assortment of algorithms and enabling technologies in the areas of Computational Science and Engineering (CSE) [1].

Presently the project continues to grow, and new elements have been added to retain agility and our commitment to strategic goals. This paper provides an overview of the project, highlighting some of the most recent advances, and the emerging challenges we face at this time.

1.1. Trilinos project beginnings

An inherent element of scientific library development is a natural decomposition of efforts into small teams. Scientific libraries have a natural scope such that one or a few people work closely together on a cohesive collection of concerns. In 1999 the Accelerated Strategic Computing Initiative (ASCI) was funding many new algorithmic and software efforts. These efforts included an explicit requirement to increase the level of formal software engineering practices and processes, including efforts of small development teams. Even before this time, there were numerous disjoint mathematical software efforts underway at Sandia National Laboratories, each with a small team of developers who were producing software intended for research and development, and eventual use within an application.

The Trilinos project was established to address two important needs: (1) bringing teams of library developers together in order to leverage commonalities and produce compatible software components, formally called *packages* and (2) to amortize the cost and efforts associated with more formal software engineering requirements. With a modest level of coordination and without unduly compromising package team autonomy, Trilinos project members could leverage each other's efforts, consolidate commonly needed tools, make packages compatible, and define a common set of software engineering tools and processes.

This two level, federated model is intrinsic to the Trilinos philosophy. It has at times led to redundancy in capabilities, but even these redundancies have strengthened the project in the long run. For example, in the early years of the project, each package had its own way of allowing users to define parameters. Over time, the lack of compatible parameter lists

[*] Corresponding author: Michael A. Heroux, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185-1110, USA. Tel.: +1 320 845 7695; E-mail: maherou@sandia.gov.

1058-9244/12/$27.50 © 2012 – IOS Press and the authors. All rights reserved

across packages forced us to synthesize and consolidate parameter list capabilities. The outcome was a very full-featured and robust parameter list capability that combined the best of features from the existing package parameter list capabilities. In general, this kind of diversity in exploration, unity in standardization has served us very well.

1.2. The Trilinos community

The Trilinos project is not a monolithic or tightly coordinated effort. It is organized around the principle of *subsidiarity*: assigning responsibilities for decisions to the lowest reasonable level. As such, it is a federated system, and most Trilinos developers view themselves more as package developers. For example, the Zoltan team and package, are widely recognized in the scientific community for their contributions to load balancing and partitioning algorithms and software. Trilinos is a delivery vehicle for their work. The Zoltan team brings in its own funding, plans its own activities and actively monitors its responsibilities relative to the Trilinos framework, scrutinizing what is proposed and available at the framework level to make sure that the framework is helping them in their efforts, and not imposing unnecessary overhead or restrictions on its autonomy. This team also monitors new capabilities emerging in other packages that might be useful for their own work. These dynamics are true for most package teams.

The community model has enabled rapid growth in Trilinos capabilities, since package teams are largely autonomous. However, over time we have increased expectations on package teams. In particular, we have specific expectations on software quality and stability, especially for key capabilities upon which other packages depend. This increased level of formality is natural and accepted by package teams, although each increase in formality is heavily vetted and scrutinized, to make sure it is truly essential and useful.

1.3. Scope and strategic goals

Although the Trilinos project encompasses a very broad collection of activities, we do have specific limits and can list specific goals that apply to all activities. First, Trilinos is a *libraries* project. We do not produce a stand-alone software product such as a fully compiled and linked executable program. We of course produce many executable programs as part of the compilation and testing of Trilinos, but these are produced at the service of the libraries. By focusing on libraries we actually embrace a larger, more complicated set of requirements. Our software must be easily embedded into an application under many conditions, and we have many interfaces to support. This open exposure demands a great deal of testing and documentation, and a very flexible configuration environment.

Second, we can succinctly state our strategic goals (see Fig. 1). We use these goals as an important litmus test for whether or not any particular activity should be conducted as part of the project. We expect that every Trilinos community member can strongly assent to at least one of the goals.

Algorithmic goals:

- *Scalable computations:* As problem size and processor counts increase, the cost of the computation will remain nearly fixed.
- *Hardened computations:* Execution will never fail unless the given problem is essentially intractable, in which case we diagnose failure and provide a reliable measure of error.
- *Full vertical coverage:* Provide leading edge enabling technologies across the entire technical application software stack: from problem construction to solution, analysis and optimization.

Software goals:

- *Universal interoperability:* All Trilinos packages, and important external packages, will be interoperable, so that any combination of packages and external software (e.g., PETSc, Hypre) that makes sense algorithmically will be possible within Trilinos.
- *Universal accessibility:* All Trilinos capabilities will be available to users of major computing environments: C++, C, Fortran, Python and from portable computers to the latest scalable systems.
- *Self-sustaining:* All Trilinos software will be clearly written, have thorough testing coverage and adequate documentation so that future development, as well as post-development refactoring by 3rd party developers can be confidently done.

Fig. 1. Trilinos strategic goals. These goals are meant to be unachievable, providing a metric for ever-improving capabilities. All Trilinos community members should be able to identify with one or more of these goals.

2. Trilinos capability areas

To help manage the growth of Trilinos, seven *capability areas* were initially defined and a leader was chosen to be a resource for both users and developers and to coordinate development efforts and lead strategic planning for each area. Two additional capability areas have been added after strategic opportunities were identified during developer planning sessions. The nine Trilinos Capability Areas are summarized in Fig. 2 and discussed below. More details are available on the Trilinos website [3].

2.1. *Framework and tools*

Most parts of Trilinos that are not associated with package capabilities are included in this capability area. This includes the CMake [6] build system, testing infrastructure, Trilinos documentation, and utilities provided by Trilinos packages primarily for the benefit of other packages.

2.2. *Software engineering technologies and integration*

This area includes support for software engineering concerns, such as scalability, interoperability, and integration across all Trilinos packages. It is not necessarily focused on any specific package, or the Trilinos framework, but is concerned with the assessment, exploration and adoption of new software processes, practices and tools. Efforts in this area have helped identify many of the new tools and process that allow us to increase framework efficiencies as the number of Trilinos packages continues to increase.

Strategic capability areas:

- Framework and tools
- Software engineering technologies and integration
- Discretizations
- Geometry, meshing and load balancing
- Scalable linear algebra
- Linear and eigensolvers
- Nonlinear, transient and optimization solvers
- Scalable I/O
- User experience

Fig. 2. Trilinos strategic capability areas. Each area has a leader whose responsibility is to set and coordinate Trilinos project strategic directions in the area. This responsibility cuts across package teams, assuring that gaps and redundancies are being addressed for the project as a whole.

2.3. *I/O support*

The Trilinos project added this capability area to cover the various aspects of I/O, to address the needs of new application areas for which large-scale data objects were required. The initial I/O capabilities in Trilinos were designed for small scale testing and were not sufficiently scalable. Specific capabilities include support for converting certain Trilinos objects to netCDF, hdf5 and Exodus file formats. Additional capabilities not associated specifically with other Trilinos packages include object serialization, and application-directed checkpoint and restart.

2.4. *Discretizations*

The packages in this area focus on the numerical solution of partial differential equations. Finite element, finite volume, and finite difference discretizations are all supported, and assembly of finite-element data into a linear system of equations.

2.5. *Meshes, geometry and load balancing*

Tools for creating, accessing, and manipulating mesh and matrix data are central to this capability area. Meshing efforts currently include inline meshing, efficient parallel reading and distribution of meshes, and efficient storage and management of mesh data. Load balancing capabilities include general partitioning and repartitioning capabilities for a variety of data, with notable emphasis on matrix partitioning.

2.6. *Scalable linear algebra*

Support for scalable sparse and dense linear algebra computations have always been part of Trilinos. Capability offerings have evolved with the state-of-the-art from serial support, and parallel support via MPI, to include other parallel computation models and standards, such as Pthreads [11], OpenMP [8], CUDA [7] and TBB [5].

2.7. *Linear and eigensolvers*

Initially, solvers were the primary focus of Trilinos and remain an integral subset of Trilinos capabilities. Iterative and direct solvers are available, as well as numerous types of preconditioners, including block, ILU and multilevel.

2.8. *Embedded nonlinear analysis tools*

A broad array of research areas are included in this capability area, including the solution of nonlinear equations, time integration, optimization, uncertainty quantification, bifurcation tracking, parameter continuation and automatic differentiation. This capability area is sometimes considered to be the highest-level capability area, in that it builds on top of most of the other capability areas.

2.9. *Usability*

The newest capability area focuses on reducing the factors that make Trilinos hard to use and are not intrinsic to the science on which Trilinos is built. Planned future efforts include adding and improving documentation, refactoring code and creating "EZ Trilinos", which will illustrate the construction and solution of $Ax = b$ in Trilinos using very few lines of code.

3. Focus on enhancing the Trilinos community model

After the significant expansion of scope, the focus of most Trilinos-level initiatives has turned to enhancing the Trilinos community model. In this section, we will discuss two significant changes that we have started implementing. These changes and others (see [2]) will increase the usability of Trilinos for segments of the Trilinos community in very different ways. We also discuss how to contribute to the community, either directly through Trilinos, or through compatible efforts.

3.1. *BSD licensing*

Trilinos packages initially were licensed under the GNU Lesser General Public License (LGPL and LGPL-compatible licenses). This model of licensing is too restrictive for some users, most commonly those working in an industry setting. We have started moving to a BSD [9] or BSD-compatible license wherever possible. New packages are applying for BSD licensing, and existing packages are being converted. With the current release, Trilinos 10.12, about sixty percent of Trilinos packages have BSD-compatible licensing. Eventually we anticipate that all, or very nearly all of Trilinos will be BSD-compatible. If some parts of Trilinos cannot be converted, the Trilinos Team is considering a BSD-only distribution that excludes LGPL licensed code.

3.2. *Trilinos.org*

We have started to move the primary Trilinos web presence to trilinos.org. Trilinos.org is a new generation of the Trilinos web portal, built primarily on Trac [14] and focused on improving collaborative opportunities for both users and developers. Increased access to issue tracking and more open and dynamic web content are cornerstones of the new portal. The more open and dynamic content creation offers a good opportunity to collaborate using more modern and effective methods than using mail lists in many cases.

3.3. *Contributing to Trilinos*

There are a few ways to contribute to the Trilinos Project. Most simply, a bug fix or enhancement for an existing package can be sent to the package development team for consideration. The development team also welcomes well-constructed feedback illustrating reproducible issues with the code. At a higher level, it is possible to contribute an entire Trilinos package, or create a package that is interoperable with Trilinos, but maintain the package separately. Before a package can be added to Trilinos, the new package must be proposed and approved using a formal policy.

A package that utilizes the Trilinos build system, but is not formally part of Trilinos can be included in the Trilinos build process using the Trilinos_EXTRA_REPOSITORIES option. While a package built in this way can utilize Trilinos capabilities, it is not possible for packages inside Trilinos to use the capabilities in such packages, due to current build system restrictions. The ability to leverage the existing Trilinos build system makes it much easier to develop new capabilities. The Trilinos build system is also familiar to thousands of Trilinos users, making packages developed in this way more easily accessible.

A large majority of Trilinos users will find that the best way for them to obtain Trilinos is through release distribution tarballs. However, users interesting in submitting patches, or those interested in becoming developers or close collaborators who do not have access to the Trilinos repository, may prefer to use the read-only Trilinos public repository. To learn where to find and how to use the public repository, go to the Trilinos website [3], and click on "Public Git Repository" in the lower right.

4. Trilinos packages

As discussed throughout this document, Trilinos is composed of numerous packages that comprise its core functionality. Presently there are over 50 packages

Table 1

Summary of Trilinos packages

	Objective	Package(s)
Discretizations	Meshing and discretizations	STK, Intrepid, Pamgen, Sundance, ITAPS, Mesquite, Panzer
	Time integration	Rythmos
Methods	Automatic differentiation	Sacado
	Mortar methods	Moertel
Services	Linear algebra objects	Epetra, Jpetra, Tpetra, Xpetra, Kokkos
	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
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi, Rbgen
	ILU-type preconditioners	AztecOO, IFPACK, Ifpack2
	Multilevel preconditioners	ML, CLAPS, Muelu, ShyLU
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA, Piro
	Optimization (SAND)	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos

Note: Details are available at trilinos.org.

in Trilinos, covering a broad spectrum of functionality. We have a reached a point where a new application based on Trilinos components can be primarily focused on specification of the computational model that is peculiar to the problem being solved. All other setup functionality: meshing, discretization, partitioning, load balancing and problem formulation can be done using scalable Trilinos capabilities (see the Albany Project, mentioned in [10], for an example). Furthermore, solution support includes state-of-the art linear, nonlinear, eigensystem and transient solvers with further capabilities for embedded optimization and uncertainty quantification. I/O support and many programming facilities such as parameter lists, interfaces to common third-party libraries and support for generic manycore parallelism also aid developers in writing efficient, portable code. The summary in Table 1 shows the current list of Trilinos packages.

5. Conclusion

Trilinos has been growing for roughly one decade, and has evolved. The "Tri" in Trilinos signifies the three packages that comprised the original grand plan for the project. Now there are more than 50. Initially a solvers project, the scope of Trilinos has expanded to include algorithms and enabling technologies from many CSE disciplines. Now the focus has shifted to making Trilinos capabilities more accessible, in multiple ways, to the growing Trilinos community.

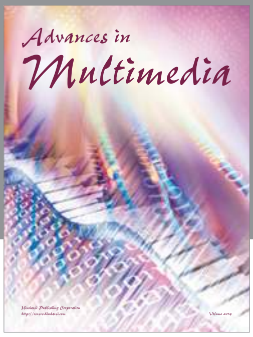
We foresee that Trilinos will continue to grow and adapt. One of the biggest challenges we face over the next few years is the migration to scalable manycore computer systems. These systems represent a disruptive technology change and demand a new abstract machine model and execution model, as well as new algorithms that can efficiently use manycore processors. They also represent a computational capability that opens the door to new modeling and simulation capabilities. Embedded optimization and uncertainty quantification will become increasingly feasible, and the corresponding packages in Trilinos will become more important, as will the integrated design of all Trilinos packages, since these advanced capabilities build on top of existing capabilities.

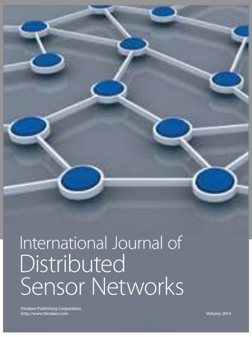
Acknowledgements

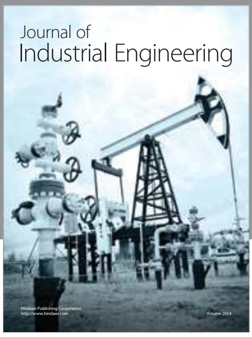
The authors would like to thank the ASC, LDRD and ASCR programs that have funded this work.

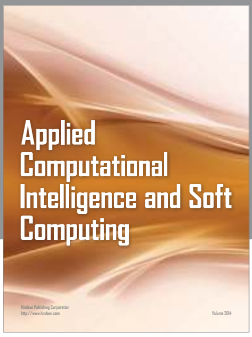
References

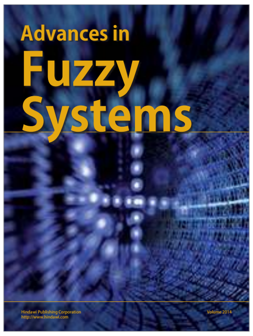
[1] M.A. Heroux, *The Changing Scope of the Trilinos Project*, Sandia National Laboratories, 2007.
[2] M.A. Heroux, *Expanding the Trilinos Developer Community*, Sandia National Laboratories, 2010.
[3] M.A. Heroux, Trilinos homepage, 2011.
[4] M.A. Heroux et al., An overview of the Trilinos project, *ACM Trans. Math. Softw.* **31**(3) (2005), 397–423.
[5] Intel Thread Building Blocks homepage, 2009.
[6] Kitware, CMake – cross platform make, available at: http://www.cmake.org.
[7] NVIDIA CUDA homepage, 2009.
[8] OpenMP.org, 2010.
[9] Open Source Initiative OSI – The BSD License: Licensing/Open Source Initiative, 2010, available at: http://www.opensource.org/licenses/bsd-license.php.
[10] E.R.P. Phipps, A. Salinger, R. Ghanem and R. Tipireddy, Embedded stochastic Galerkin projection and solver methods via template-based generic programming, 2011, available at: http://www.csm.ornl.gov/workshops/applmath11/documents/posters/Phipps_poster.pdf.
[11] pthread.h, 1997.
[12] *R&D Magazine* homepage.
[13] A. Roscoe, M.A.H. Bartlett and J.M. Willenbring, *TriBITS Lifecycle Model*, Version 1.0, Sandia National Laboratories, 2012.
[14] Edgewall Software, The Trac project, 2011, available at: http://trac.edgewall.org/.











Hindawi

Submit your manuscripts at
http://www.hindawi.com