SAND2019-11412C

Toward interoperable and flexible scientific computing libraries: Lessons Learned from xSDK

Keita Teranishi and xSDK Developers

Sandia National Laboratories, Livermore, CA

E4S Forum, September 23, 2019

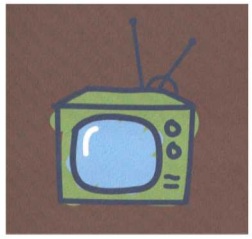
exascaleproject.org xsdk.info

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

xSDK Lessons Learned: General Observation

- **Working toward shared understanding of issues and perspectives is essential and takes time**
  - Need regular opportunities for exchanging ideas, persistence, patience, informal interaction
  - Must establish common vocabulary
- **Lots of fun, too … xSDK: Life is good ☺**

It takes all kinds.

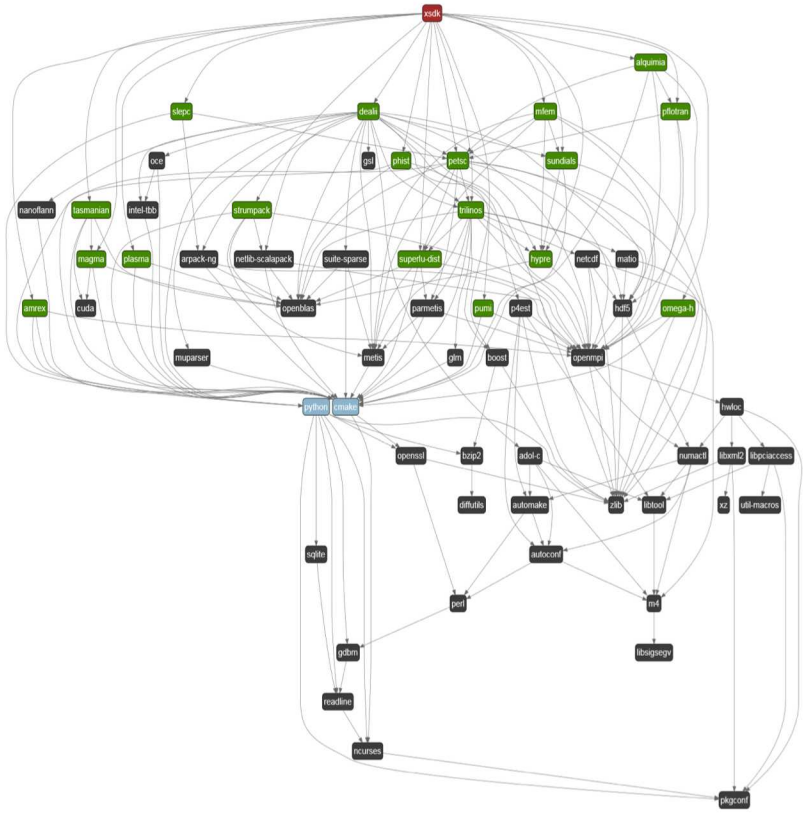

Think outside the box.


Face the bumps with a smile.


The pursuit is the reward.

Complexity of xSDK

- **17+ Packages**
  - Multiple Languages (Fortran, C and C++)
  - Multiple runtimes (OpenMP and CUDA)
- **Diversity of computing platforms**
  - Architecture and Runtime
- **Multiple Compilers and Multiple versions**
  - GCC, Clang and other proprietary compilers
- **Spack builds for us. Nice!**
  - **How to report errors and problems?**



xSDK community policies

We welcome feedback. What policies make sense for your software?

https://xsdk.info/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.

M15. All xSDK compatibility changes should be sustainable.

M16. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

Also **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.

R6. Provide versions of dependencies.

R7. Provide README, SUPPORT, LICENSE and CHANGELOG files or their equivalent.

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.

xSDK

ECP EXASCALE COMPUTING PROJECT

xSDK community policies

We welcome feedback. What policies make sense for your software?

https://xsdk.info/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. R… packa…

M7. C…

M8. F… softw…

M9. U… name…

M10. …

M11. …

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.

M15. All xSDK compatibility changes should be sustainable.

M16. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

Also **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.

…s soon as

…f library

…NGELOG

…K-compatible" package, *and* it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

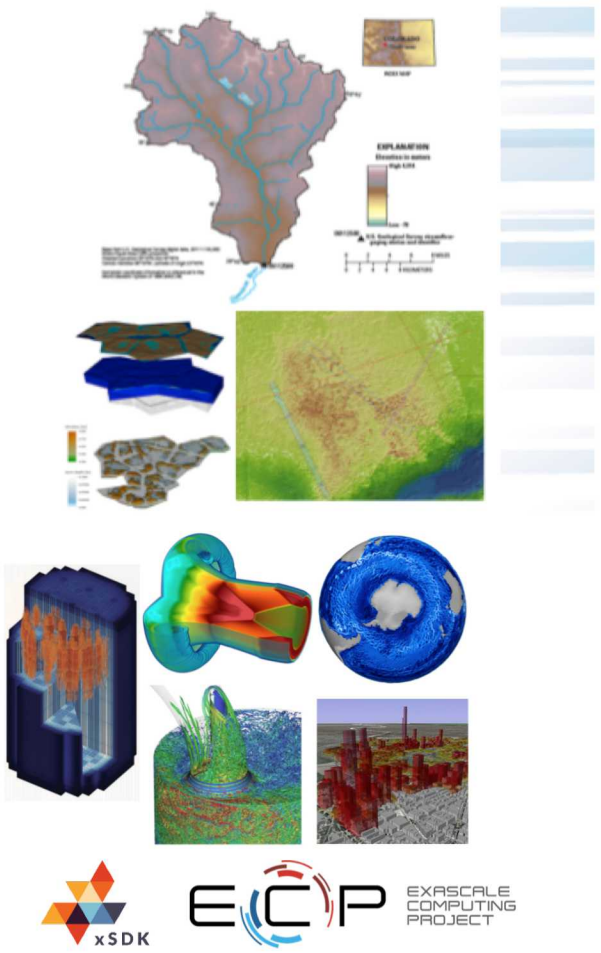
How to check the policy compliance?

xSDK

ECP EXASCALE COMPUTING PROJECT

Applications using xSDK

- PFLOTRAN and Alquimia
  - Multiphysics & multiscale modeling of watershed dynamics
  - Provided as part of xSDK
  - Spack script for individual application packages
- Nalu in ExaWind
  - Call hypre from Trilinos (xSDK Trilinos)
- Laghos in CEED
  - MFEM and hypre
  - Planning to use SuperLU, SUNDIALS and PUMI
- AMPE and Truchas in ExaAM
  - SUNDIALS and hypre
  - Wrote Spack script for AMPE and Truchas



xSDK Lessons Learned: Users' Perspective

- Building the whole xSDK takes time and produces a very large executable.
  - Future releases should allow building a subset
- Need better document for Spack and xSDK
- Application developers want subset of xSDK or special build of xSDK
  - **Some library capabilities are disabled for the interoperability**, but needed for some applications
  - It will be important to provide flexibility through the xSDK to allow users to use their own versions of some xSDK libraries.
  - Need a searchable library collection based on capability
- xSDK member libraries should also pursue improved compatibilities where possible to **avoid for users to have building their own versions**.

xSDK Lessons Learned: Developers' Perspective

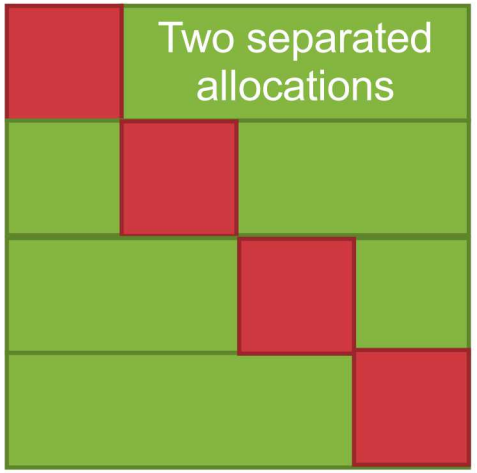
- Requires some code modifications to **eliminate naming conflicts**
  - Namespaces
  - Unique prefix for function names and preprocessor macros
- Maintaining interoperability needs close communication with the developers of other packages
  - Coordination for release scheduling is challenging
  - **Domain-specific version of software-coupling could hurt interoperability**
    - Be generic and follow the best practice!
    - Special version may work effectively for the integration with A, but fails with B and C
- Work toward better, faster, more people-efficient workflow for development and testing is important!
  - Needs for continuous and integrated testing

Interoperability Lessons: hypre+Trilinos

- ExaWind calls hypre and Trilinos together.
  - User wants to call hypre through Trilinos APIs
  - hyper and Epetra/Tpetra (Trilinos's sparse matrix package) implement distributed CSR (sparse matrix) format in slightly different manners.
  - Developed xSDK-Trilinos (a special version of Trilinos)
- Development
  - Our team discovered a hypre interface in Trilinos source, but never maintained and tested.
  - The slight difference in sparse matrix implementation requires local data copy between hypre and Trilinos.
  - Trilinos-hypre interface helped for the initial development of ExaWind, eventually phased out for performance reasons
    - Calling hypre's API directly to reduce the overhead.

Sparse Matrix in hypre (PETSc) and Tpetra

hypre/PTESc



Tpetra

- hypre and PETSc separate local sparse matrix into two sparse matrices
- Tpetra supports 1D and 2D partitioning of sparse matrix.
  - Nonzero entries are permuted to separate diagonal and off-diagonal
    - All entries are assigned to a single contiguous allocation
    - Diagonal part comes first
    - With some extra indices.

Interoperability Lessons: SUNDIALS+Trilinos

- SUNDIALS provides abstractions for linear system solver, matrix and vector implementations.
  - Trilinos provide a collection of sparse linear system solvers and preconditioners
  - Runs with MPI+OpenMP/CUDA
  - Sounds great ☺, but … ☹
- Problems:
  - Poor documentation of Trilinos
    - No good example of best use cases
    - No description of the design (class hierarchies)
    - Many deprecated interfaces for backward compatibilities
  - Trilinos's build system is hard to use even with Spack (barely complies xSDK standard)
  - MPI+CUDA works only for OpenMPI

Lessons from Individual xSDK Package Adaptations

xSDK Lessons Learned

Background:

- deal.II was already compliant with almost all of xSDK's Community Policy Compatibilities.
- In particular, it has a very large test suite (10,000+ tests) that covers all of the interfaces we have with other libraries; in some cases, we seem to have better coverage of these external libraries through the interfaces than the package's test suite itself.

Lessons learned:

- Avoid **unprefixed macros** or provide a way to disable them. Avoid unprefixed preprocessor variables.
- Don't use **MPI_COMM_WORLD**, but user provided MPI communicators.

xSDK Lessons Learned

Background:

- hypre had various issues that needed to be addressed.
  - **Name space conflicts** (some functions with simple names)
  - Overlooked prints of error messages
  - No exhaustive test suite that could be run on arbitrary computers

Lessons learned:

- Giving all functions the prefix ‘hypre_’ avoids namespace conflicts.
- Allowing for error messages to only be printed for a higher print level avoids undesired printouts.
- A test suite that allows users to test hypre solvers on any platform and check for errors is now available.

xSDK Lessons Learned

Background:

- MAGMA's solvers rely unconditionally on hardware accelerators (AMD, Intel, NVIDIA).
- Accelerators are optional for xSDK packages.

Lessons learned:

- Having established software practices helps with xSDK integration.
  - Continual maintenance is a must.
    - This is enforced for MAGMA with vibrant accelerator hardware market and frequent product releases.
  - Code documentation is required.
    - Large MAGMA user base made good documentation a must to ease the burden of answering user questions.
- **User contributions might not meet xSDK requirements.**
  - Adjustments were needed for user-contributed Spack package for MAGMA.
- New variant added to support xSDK builds with CUDA present on the installation system.

xSDK Lessons Learned

Background:

- PUMI was not compatible with the requirement for runtime control of output - there were over **700 calls** to functions from the printf 'family'.

Lessons learned:

- Design your library from the beginning with a print statement wrapper so it can run in silent mode, or with various levels of output for performance information, developer level debugging, etc..
  - Use grep/sed to automate replacement of the printf family functions with the wrapper API.
  - The handful of C++ cout/cerr uses were manually replaced with the wrapper functions. In some cases stringstream was used to compose the strings and then those strings were passed into the API.

xSDK Lessons Learned

Background:

- SUNDIALS has interfaces to several external libraries e.g., **PETSc, hypre, KLU, LAPACK**, …
  - Existing CMake options did not align with xSDK policies.
  - Added redundant options that overwrite existing variables to maintain options for current users.
- MFEM has interfaces to SUNDIALS time integrators and nonlinear solvers.
  - Updates to SUNDIALS for xSDK compatibility were introduced along side a new linear solver API.
  - **The new API broke compatibility with MFEM** and required updating the MFEM interface to SUNDIALS.

Lesson learned:

- Packages working toward xSDK compatibility should adopt xSDK conventions early to ease user transition to new options.
- Maintaining interfaces between xSDK packages requires regular communication and testing with in-development versions.

xSDK Lessons Learned

SuperLU

Background:

- SuperLU initially faced challenges with **build system, revision control, namespacing**.

Lessons learned:

- Migration from manual editing make.inc to **CMake/Ctest** increases build-test productivity and robustness.
  - Easier to manage dependencies (ParMetis, machine-dependent files), and platform-specific versions (_MT, _DIST, GPU) and correctness
  - Better accommodate special build requirements (e.g., disable third-party software like ParMetis)
- Migration from svn to git improves distributed contributions and bug fixes. E.g., users have contributed:
  - Working with Windows environment, building as both static and shared libraries simultaneously
- Proper namespacing allows 3 versions of the library (serial, multithreaded and distributed) to be used simultaneously and to be used by other packages in xSDK.
- Improved productivity of new code development:
  - Wrote comprehensive regression unit test code
  - UseTravis CI for continuous integration on each git commit

xSDK Lessons Learned

Background:

- Trilinos had interfaces to both PETSc and hypre, but those interfaces were
  - **Poorly documented** (e.g. – no hypre interface document)
  - **Not tested regularly** (e.g. – the PETSc-Trilinos interface was broken in recent releases)

Lessons learned:

- Interfaces supported for the xSDK require regular testing and clear documentation
- Continual maintenance of code and documentation will be required; occasional fixes are insufficient

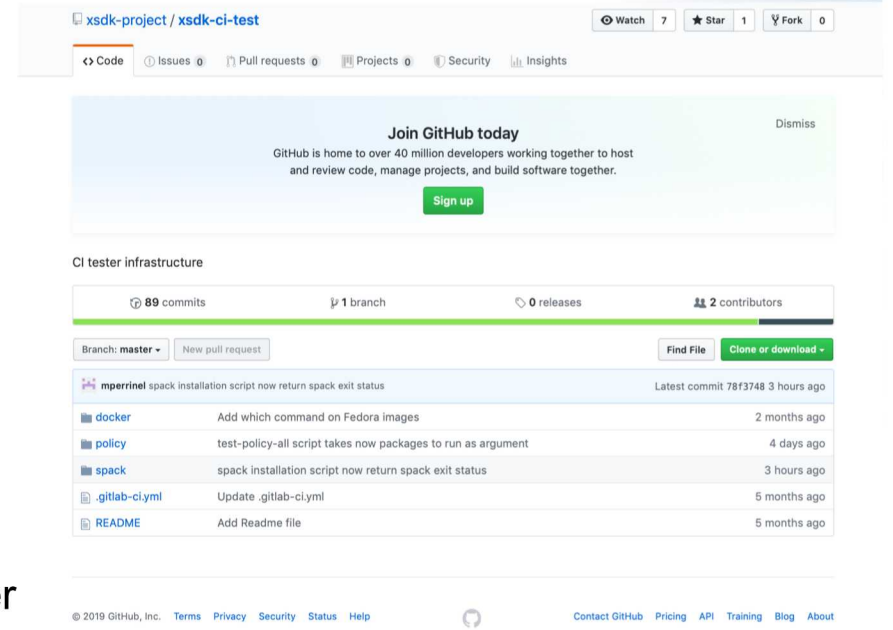
Toward Better Integration

Testing xSDK

- xSDK doesn't contain directly any source code, but only a Spack Package.py file that defines the rules of a *`spack install xsdk`* instruction.
  - Every xSDK package is tested by its own continuous integration (CI) mechanism.
- Testing xSDK means testing whether an individual package is xSDK-compatible.
  - Any xSDK installation must work for as many platforms and compilers as possible
  - Every package must comply with the community's policy
  - Every package must be interoperable

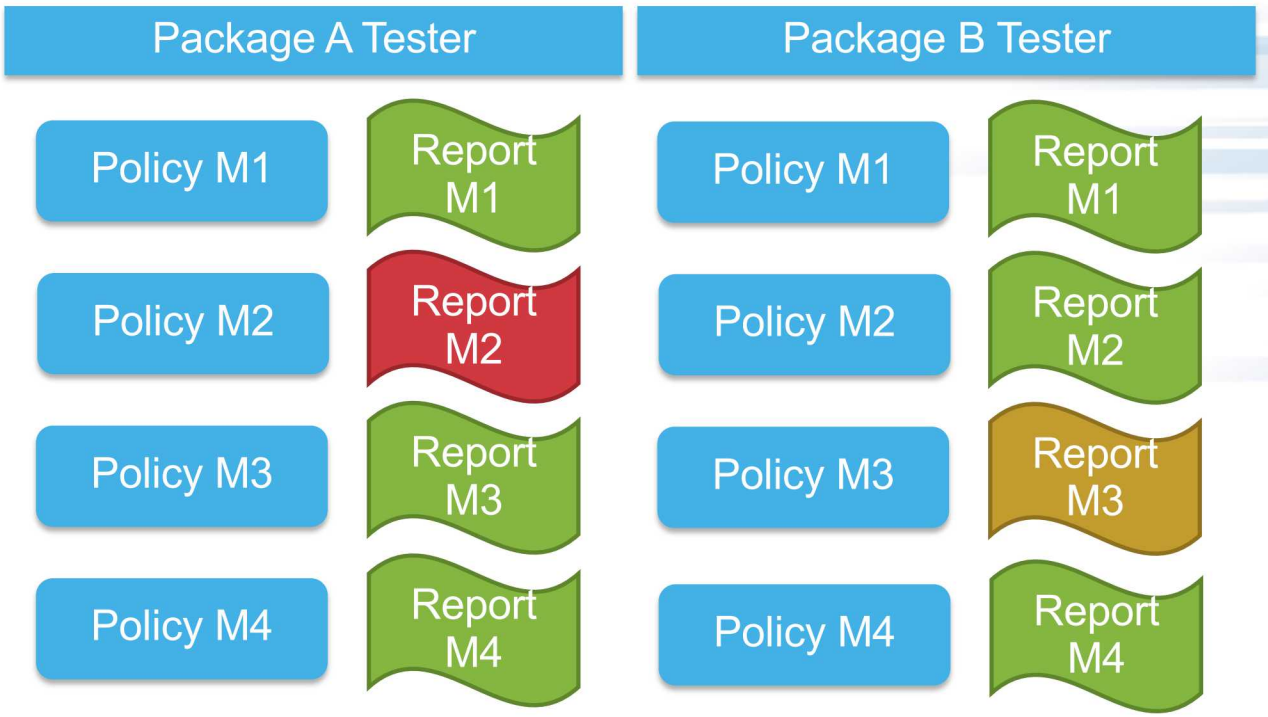
xSDK and Gitlab-CI

- xSDK CI is a Github project that contains:
  - xSDK Spack installation scripts
  - Policy testing scripts
  - Minimal docker images prepared for xSDK testing
- CI pipeline described in gitlab-ci.yml:
  - Pull the xSDK CI project
  - Pull the Spack project
  - Run the xSDK installation scripts
  - Run the policy testing scripts
  - Report all results as a set of artefacts for all xSDK inner packages
  - Repeat for every targeted platform using docker xSDK images
  - (FY20: Built/Run interoperability example programs)



Compliance with xSDK policies

- Using scripted tests to verify compliance
  - Every policy has been analyzed to establish a list of tests that can be automatically verified
  - For every package, the tests produce reports that can be viewed in the CI



xSDK Policy Testing Design

- Every testable mandatory policy has a dedicated testing script which outputs a report.
  - e.g., the first mandatory policy (M1) has test (m1.sh) with corresponding report (report_m1.log)
- All tests are run daily; they can also be manually triggered on the Gitlab-CI
- xSDK Spack installation is done by running *spack install –keep-stage –source xsdk* that keep the source and the build directories of every inner package.
- Source and build directories are used as an input parameter for the policy tests
  - e.g. the m1.sh run for every inner package takes the corresponding source directory as input.

xSDK Policy Testing Design, Examples

M1. Support xSDK community GNU Autoconf or CMake options.	1) Check the CMakeLists.txt or Configure file existence. 2) Interoperability between packages can be tested here with packages inclusion cmake option -DENABLE_<package>. 3) An option —dis/enable-xsdk-defaults must exist.
M3 Employ user-provided MPI communicator.	1) Scan the code and detect if MPI_COMM_WORLD is called which could making this requirement failed. 2) Check for the option of the MPI error-handling prevention
M5. Provide a documented, reliable way to contact the development team.	1) Read the document and parse some information : - Website of the package - Email address of the package owner 2) Ping the website of the package

Conclusion

- Making xSDK work is labor intensive
  - Coordination among xSDK members
  - Interaction with the users
  - Policy compliance
- Great Lessons from the development activities
  - Developers applied some fix to comply xSDK policy
  - Need for automated testing infrastructure
- Automatic testing and integrations are necessary
  - Complement the capability of Spack
  - Policy tester

More info

- **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, *Supercomputing Frontiers and Innovations,* vol 4 No 1 (2017), pp.69-82.
- https://xsdk.info

License, citation and acknowledgements

License and Citation

- This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).
- Requested citation: xSDK Developers, *An introduction to the xSDK, a community of diverse numerical HPC software packages* , ECP 3rd Annual Meeting, Houston, TX, January 15, 2019

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

xSDK

ECP EXASCALE COMPUTING PROJECT


EXASCALE COMPUTING PROJECT


xSDK

Thank you!


U.S. DEPARTMENT OF
ENERGY
Office of
Science


National Nuclear Security Administration

Backup Slides

xSDK Lessons Learned

MFEM

Background:

- MFEM interfaces with numerous libraries in the XSDK including HYPRE, SUNDIALS, SuperLU, PETSc, and STRUMPACK.

Lessons learned:

- Maintaining connections to all existing libraries requires significant effort and communication.
- Synchronizing releases with interoperating libraries that have API changes is difficult.
- Library version interoperability needs to be added to our testing suite.

xSDK Lessons Learned

Background:

- PLASMA relies on BLAS and LAPACK for low-level HPC optimizations.
  - Varied implementations and interfaces across hardware platforms
  - Test suites for BLAS and LAPACK are usually not available in most implementations
  - Additional set of basic routines (Core BLAS) did not gain community acceptance

Lessons learned:

- Must use name space for all interface entry points even if it might be an older project.
- Ease of installation is a necessity. Using established software configuration frameworks (autoconf, CMake, etc.) enforce portable code and adaptation to a variety of software environments.

xSDK Lessons Learned

PETSc/TAO

Background:

- PETSc development was being slowed down by backlog of branches waiting to be fully tested and moved into master.
- Through work on xSDK, we refactored PETSc tests to improve overall test functionality and better support continuous integration testing.

Lessons learned:

- Result: Work toward better, faster, more people-efficient workflow for testing has enabled moving developer branches into master branch without breaking master branch or requiring hand holding.
  - Introduced parallelism (and other techniques) to decrease time for running complete test suite while still providing high coverage
  - Simplified the addition of new tests into suite
  - Introduced finer grain control over what is tested
  - Making testing process more robust to random system, hardware, or software failures
  - Working toward making it easier for anyone to automatically run full PETSc test suite, including dashboard

xSDK Lessons Learned

PHIST

Background:

- PHIST already adhered to most of the xSDK policies, but still required a few changes.

Lessons learned:

- Had to duplicate some CMake flags although Spack is the high-level build system now
- Had to switch off performance optimizations (e.g. no OpenMP, no –march flag) to get it to compile on all platforms
- No easy way to uninstall all xSDK packages via Spack