LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-643347

Package Management Practices Essential for Interoperability: Lessons Learned and Strategies Developed for FASTMath

M. C. Miller, L. Diachin, S. Balay, L. C. McInnes, B. Smith

September 5, 2013

First Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE)
Denver, CO, United States
November 17, 2013 through November 17, 2013

Disclaimer

Package Management Practices Essential for Interoperability: Lessons Learned and Strategies Developed for FASTMath

Mark C. Miller
Applications, Simulations and Quality (ASQ) Division
Lori Diachin
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
[miller86,diachin2]@llnl.gov

Satish Balay, Lois Curfman McInnes, and Barry Smith
Mathematics and Computer Science Division
Argonne National Laboratory
[balay,mcinnes,bsmith]@mcs.anl.gov

Abstract

While open-source software packages for high-performance computing (HPC) are an essential foundation of many scientific applications, challenges arise in coordinating multiple packages that employ diverse development and release processes. This document identifies a few key issues and essential practices for packages to more easily interoperate within the same application. We discuss approaches under way within the multi-institutional FASTMath project, whose members develop robust, efficient, and scalable numerical research software for functionalities such as mesh management, discretization, and solvers. Our perspective and priorities for overcoming common problems—including inconsistent installation processes, inconsistent or missing configuration information, copying or spoofing dependent sources as a means of simplifying package code, and inconsistent/missing versioning—should be of interest to HPC software developers who aim to make their packages easier to interoperate with others.

Combinations of multiple software packages developed by different groups have become essential for large-scale computational science, where the capabilities needed for modeling, simulation, and analysis are broader than any single team has resources to address. For example, numerous scientific applications employ packages developed by members of the FASTMath [2] institute for scalable linear, nonlinear, eigen-, and timestepping solvers as well as for parallel mesh management, partitioning, discretization, and adaptive mesh refinement.

Such numerical packages are part of a rich ecosystem of freely available, open-source software under development throughout the high-performance computing (HPC) community. The often tedious trial-and-error process of obtaining, configuring, and installing any single tool may arguably be manageable. However, from the perspective of an end-user application scientist, handling each tool's installation idiosyncrasies can easily become overwhelming when dealing with several packages in combination. Worse, such problems are compounded by the need for consistency among packages to be used within the same application in terms of compiler, compiler version, exotic compiler optimization options, common third-party packages such as the Message Passing Interface (MPI), and other external functionalities.

This short note discusses only a few key issues and practices essential for packages to more easily "play nice" when used in combination with a multitude of other packages within the same application. Certainly, there are many challenges. We focus on here on a minimal subset of issues that we believe to be fundamental impediments to package interoperability. These include eliminating common compile- and link-breaking obstacles as well as reducing barriers to ease of use by simplifying and unifying installation processes. Specifically, we address:

1. Inconsistent installation processes
2. Inconsistent or missing configuration information
3. Copying sources as a means of managing dependencies
4. Spoofing MPI sources as a means simplifying package code
5. Inconsistent/missing versioning
6. Managed installations on (some) leadership computing facilities (LCFs)

Related topics are discussed in [1, 5, 6] and Chapter 4 of [4].

In FASTMath, our focus and philosophy are driven by resources and priorities in a loose collaboration whose members, spanning four national laboratories and five universities, have diverse preferences, sponsors, and priorities. In such a setting, there can be no central authority. Instead, collaboration relies upon gaining consensus as development teams choose to do what is best for the group. In this way, the approach to HPC package interoperation taken in FASTMath resembles the way the open source community has operated successfully for many years.

We bring together software products with varying degrees of interoperability, maturity, robustness, and user support. These circumstances set the context for the range of choices that can be considered as we improve the quality, ease of adoption, and integration of the many FASTMath software products, with the ultimate goal of maximizing the impact of our scalable numerical algorithms in the application community. The remainder of this document elaborates on the six problems highlighted above and our minimalist approach to address them. In particular, we have established recommended practices that are already reducing barriers to using multiple FASTMath packages in combination in large-scale scientific applications, yet preserve individual developer autonomy and flexibility.

Issue 1: Inconsistent installation processes. The full process of installation—that is, downloading a package, preparing it for compilation, testing, and then installing it for use—lacks uniformity across all FASTMath packages. Some packages are hosted for download directly from their revision control systems (e.g., svn or git); others are hosted for download as release tarballs at specific URLs; still others are hosted for download such that a manual, human-in-the-loop registration step is required before download can proceed.

Once a package has been downloaded, the process of configuration, or preparing it for compilation, also varies significantly. The range of approaches includes using GNU Autotools, employing CMake, relying on manually editing makefiles, using custom tools, and employing a hybrid of these approaches.

When debugging problems in an application that uses a multitude of packages, one often needs to independently confirm that an installed package is operating as expected. Some FASTMath packages have rich and robust test suites; some have relatively simple sanity-check tests; others use for their primary source of testing the applications in which they are most frequently embedded.

Among all of these phases of the package installation process, configuration is often the most challenging for users. Expecting FASTMath users to master the configuration interfaces of all different packages is unrealistic. In addition, it is desirable for the installation of any FASTMath package to be scriptable—that is, to be able to write scripts that download, configure, and install a package without requiring a human in the loop.

Resolution. To provide consistency in the configuration phase of the installation process for FASTMath packages, we have adopted an approach of "uniformity of interfaces" rather than "uniformity of implementations." For downloads we request that each package provide a URL to obtain a file (typically a tarball) with the entire contents of the release.

To accomplish the common configuration interface, we have adopted the GNU Autoconf command-line options style of interface as the common approach for the FASTMath packages.

```
./configure --prefix=<installation point> --with-mpi-dir=<>  --enable-debug ...
make
make install
make check
```

Note that all packages continue to be free to implement their configuration processes any way they like and are not required to use GNU Autoconf. For packages that use an edited file-based approach for configuration, a simple script must be provided with default values for all options that accepts the user's command-line options and then automatically edits the requisite file(s).

The two categories of options, `--with-XX` and `--enable-YY`, have fundamentally different purposes that can be easily confused. Options of the form `--with-XX` are used to specify dependencies on *external* packages, while options of the form `--enable-YY` are used to turn on (or off) optional features *internal* to the package being configured. For example, `--with-cc=gcc` indicates the C compiler to be used to compile the package, while `--enable-static` indicates that the package should be built for static linkage. As another example, the hypre solver package uses `--enable-bigint` to configure hypre to use 64-bit integers for indexing and sizing of vectors and matrices, while PETSc uses `--enable-64-bit-indices` for the same purpose. This situation introduces another issue we face: different packages use different option names for the same feature. We are now determining appropriate consistent names for these common features. For backward compatibility, packages are free to retain their previous option names as well. For dependence on BLAS and LAPACK we have adopted the options `--with-blas-dir` and `--with-lapack-dir` to indicate the directory location of the package or `--with-blas-lib` and `--with-lapack-lib` to indicate the exact BLAS and LAPACK package names to link.

We do not concern ourselves with installing pre-compiled binary versions of FASTMath packages. Installing a given package, A, with an option to use an external package, B, requires compiling A with appropriate information about the location of B's installation. For example, if PETSc is to be used with hypre,

then PETSc must be installed in the presence of an installed hypre; thus, we cannot provide a single PETSc installation that both supports and does not support hypre interfacing. Because FASTMath includes many packages, we could never provide all combinations as pre-compiled binaries. More important, all FASTMath packages are configurable at compilation time. For example, should the packages use 32-bit indices (suitable for most simulations) or 64-bit indices (needed for very large runs but requiring more memory usage)? Because of the huge number of possible combinations of configurable options, providing pre-compiled versions of all combinations is infeasible.

Issue 2: Inconsistent or missing configuration information. Even with a common approach to configure and compile packages, additional functionality is needed to install new packages to work with previously installed packages. For example, if one wishes to install PETSc to use a previous installation of hypre, one needs to know what compilers and other options were used to install hypre so that PETSc can be installed with compatible options. Currently most FASTMath packages provide no scriptable way to obtain this information.

Resolution. The pkg-config helper tool [3], which works on all Unix systems (including Linux and Mac OS) as well as Microsoft Windows, provides a unified interface for querying installed packages for the purpose of compiling software from its source code. Pkg-config works by having the package installer generate a file of a particular format that contains configure information; then a small utility that may be called by a script (called pkg-config) queries an installed package for information about the package configuration. Most Linux open-source projects use pkg-config, as do many HPC software projects, including NetCDF, MPICH, and PETSc. We hope to extend this capability to most if not all of the FASTMath packages. Note that pkg-config imposes no constraints on how a package creates the pkg-config file, which may be created by a shell script or in PETSc's case by a python script that works with the PETSc configure tools.

Issue 3: Copying sources as a means of managing dependencies. BLAS and LAPACK are long-established numerical packages for common vector and dense matrix linear algebra operations. For many HPC package developers, it is often convenient to handle dependence on BLAS and LAPACK by including copies of them in their own packages. Then, when application developers obtain the package, they need not worry about having to separately obtain, compile, and install BLAS and LAPACK. This approach works well for any given *single* package. However, when multiple packages need to be combined in the same executable, as is the case for many scientific applications that employ multiple FASTMath packages, this approach can introduce several problems. First, link-time errors can occur because of multiple definitions of BLAS and LAPACK symbols. Worse, in the case of dynamic linking, this practice can lead to the wrong implementation of a given BLAS or LAPACK function being used without any warning other than an application that behaves differently from expected, with no explanation as to the cause.

Resolution. If package developers value the convenience of maintaining copies of common dependent packages such as BLAS, LAPACK, or MPI stubs (see below) within their own packages, then users require the ability to disable the compilation and installation of these copies and instead indicate the location of a desired installation of these packages. Package developers might argue that mangling of the symbol names of a copied package such as BLAS or LAPACK should be sufficient to address the concerns raised above. While name mangling may resolve link-time symbol collision, it introduces other problems in debugging and ensuring that a specific implementation of a given package is being used.

Issue 4: Spoofing MPI sources as a means simplifying package code. A somewhat common practice for packages is to maintain their own implementations of portions of MPI for sequential execution. Not only does this practice cause problems when multiple such packages are combined in the same executable (as discussed above in Issue 3), it also can cause problems even when a package is used alone. For example, if a parallel compilation of an application accidentally links against a sequential installation of such a package, the application may crash mysteriously because of calls to the "wrong" MPI.

Resolution. In FASTMath we have decided to focus on supporting interoperability only when all the packages are compiled with a true and complete MPI implementation.

Issue 5: Inconsistent/missing versioning. As a consequence of some FASTMath packages' frequent changes to APIs in support of advancing capabilities in cutting-edge research software, at any given time other packages will work properly only with certain versions of the first package. Installation with the wrong version will result compile-time and/or run-time issues. Another painful truth is that various FASTMath packages may have dependencies on particular versions of common third-party packages, such as HDF5. If

two FASTMath packages require different versions of HDF5, for example, then the two packages cannot interoperate until one is updated to the newer API.

The aforementioned problems exemplify an underlying issue in dealing with large collections of interrelated but independently developed packages having their own release schedules: coordinating resolution of issues manifesting from combinations of packages. Such issues are not resolved by work on any single package. Instead, some minimal processes must be in place to coordinate development on multiple packages.

Resolution. Each FASTMath package will use a version number that uniquely identifies each release. Each package needs to indicate what versions of other packages it can employ and what versions of other packages it requires. Each package will also strive to keep up to date on what versions of other packages it uses. For example, it will be considered socially unacceptable for a package to require the use of an obsolete release of some other package that has several years' worth of more recent major releases. While we do not anticipate "coordinated releases" of all FASTMath packages, we do expect the occasional need to coordinate bug fixes, API updates, and/or feature enhancements among key combinations of packages in support of application needs.

Issue 6: Managed installations on LCFs. It is important for the FASTMath team to develop and maintain the know-how to install the entire suite of FASTMath packages in some standard configurations on some common LCF systems. In this way, we can stay ahead of potential users by encountering and then resolving issues with configuration and installation. In addition, LCF systems are often more exotic and involve more complexity in properly configuring packages to take advantage of available hardware features. Expecting application developers to handle this additional complexity when they may already be overburdened with simply porting their own application is unreasonable.

Resolution. Members of the FASTMath team will periodically install all FASTMath packages on a handful of DOE LCF and Capability Center systems including BG/Q and Cray XC30 or XK7 systems.

Conclusions. We have described several issues that arise from the need to combine in one application multiple software packages that employ diverse development and release practices. Among these, one of the more challenging for end-user application developers is the configuration and installation process. Given limited resources and the need to seek consensus solutions, the FASTMath team has identified some essential requirements for package interoperability: a common and consistent GNU Autoconf-like interface for configuration, the use of pgk-config for configuration information, the ability to disable embedded copies or spoofed third-party APIs, and the use of package versioning and managed installations of FASTMath packages on a handful of LCF systems.

Acknowledgments. Work by Diachin and Miller was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The doc ID is LLNL-CONF-643347. Balay, McInnes, and Smith were supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357.

Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research.

References

[1] R. Bartlett, M. Heroux, and J. Willenbring, *TriBITS lifecycle model version 1.0: A lean/agile software lifecycle model for research-based computational science and engineering and applied mathematical software*, Tech. Rep. SAND2012-0561, Sandia National Laboratories.

[2] L. Diachin (PI), *SciDAC-3 Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) Institute*. `http://www.fastmath-scidac.org/`.

[3] Freedesktop.org, *pkg-config*. `http://www.freedesktop.org/wiki/Software/pkg-config/`.

[4] D. E. Keyes, L. C. McInnes, C. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawlowski, A. P. Randles, D. Reynolds, B. Rivière, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth, *Multiphysics simulations: Challenges and opportunities*, International Journal of High Performance Computing Applications, 27 (2013), pp. 4–83. Special issue.

[5] M. C. Miller, J. F. Reus, R. P. Matzke, Q. A. Koziol, and A. P. Cheng, *Smart libraries: Best SQE practices for libraries with emphasis on scientific computing*, Tech. Rep. UCRL-JRNL-28636, Lawrence Livermore National Laboratory, 2004.

[6] D. E. Post and R. P. Kendall, *Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from ASCI*, Int. J. High Perf. Comput. Applics., 18 (2004), pp. 399–416.