More Robust and Portable Software Builds for the DOE Software Ecosystem

Roscoe Bartlett (rabartl@sandia.gov)*, Daniel Ibanez, and Keita Teranishi
Sandia National Laboratories, NM/CA

December 9, 2021

1 Motivation Deploying scientific computing software for high performance computing (HPC) systems has become increasingly challenging with the rapid growth of the HPC software stack to accommodate a variety of application needs to exploit ever-increasing computing system capability. Under the leadership of the DOE Exascale Computing Project (ECP), this problem has been addressed through (1) the development of a software package management framework (Spack) [3] and (2) the investigation of container platforms [8] to improve the software building and installation process. Along with these software technologies, the ECP has been establishing a sustainable software ecosystem for HPC software stack [1, 4], introducing suites of pre-build software collections and new community policies to advocate stable software building, testing and documenting practices for improving the software quality and the interoperability between them.

The software ecosystem effort by the ECP has resulted in some successful outcomes in productivity improvement at several supercomputing sites and applications. Many of the software packages under the ECP funding have already been incorporated with Spack along with the prevailing use of modern build automation tools such as CMake 3.x in many ECP software packages and applications. However, these successes would not be possible without the maturation of these modern build automation tools. Currently, Spack orchestrates the build processes of individual software packages throughout all software dependencies as well as directly manipulating the package-level build processes themselves. Likewise, both E4S and xSDK policies mandate proper use of build automation tools to enable seamless integration with Spack. Thus, it is essential for ASCR to engage the broader community of modern builds systems so that the community pays more attention to the needs for HPC systems and applications, and includes the HPC features as part of the software ecosystem.

2 Challenge The ecosystem of the modern software build tools has been led by the broader community such as enterprise and cloud computing. Their major interest is supporting popular and emerging programming languages such as C++, Java, Scala and Rust, and adaption to popular computing platforms (Cloud and standard Linux distributions). Some popular languages are accommodated with their official build and deployment tools with clear separation of functionality between the tools, enabling a clean holistic solution at their early stage. Focus on de facto standard configurations of computing systems such as popular Linux OS distributions contributes to the robustness and portability on popular and robust cloud middleware and APIs which further improves the flexibility for the main stream of distributed systems. Moreover, the grassroots efforts of the user community strengthens the ecosystem of these tools.

*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.

In contrast, the HPC community has unique and fragile software ecosystems. HPC software is typically written with a variety of programming languages such as C, C++, Fortran and some HPC language extensions such as OpenMP, CUDA and DPC++. These languages have historically had neither one official build tool nor a holistic solution to embrace their diversity. In addition to this, numerous unique and heterogeneous system configurations makes it hard to achieve portable solutions. Although the recent effort by ECP is addressing this problem, the HPC community still relies on various different build systems. The lack of minimum standards for proper build tool usage results in a number of build system variants. In addition, even with a popular and successful build tool such as CMake, there is a substantial amount of functionality that needs to be added to create complete working build and test systems for HPC machines which has lead to the development of tools to fill this gap [7, 5, 2].

Reducing the number of build systems used by the underlying packages built by Spack helps to address these problems. If standardizing on one underlying build system, CMake looks like the obvious choice currently. It is widely adopted by the ECP packages being built by Spack and it has large adoption and broad support in the C++ community (it is now the most popular build tool for C++ code in the world). However, CMake has some fundamental issues that make it difficult for new users/developers to learn to use it effectively and avoid making mistakes. Just having every package adopt CMake is not enough; they must also use it correctly. Any small deviation and subtle change in the proper usage of CMake within a package's build system can substantially damage robustness, compatibility, and portability — debugging and verifying CMake build systems is still a daunting task for many developers. The lack of common proper CMake usage and community-wise knowledge sharing also expose a huge challenge for seamless multiple-package building and integration through an external tool (e.g. Spack). Another major problem of CMake is its software testing capability (CTest) that can be used for Continuous Integration (CI). Developing a fully functional and robust build and test system for a single package still takes a significant effort with raw-CMake scripting and other scripting code. Customizing the usage of CTest so that it can be run by Spack on various HPC systems involves more heavy-lifting which requires understanding of the advanced features of CMake/CTest[6]. As mentioned above, some CMake-based reusable tools [7, 5, 2] address CI portability issues, but these efforts are lacking coordination and with the ECP and ASCR leadership to participate in better support from the HPC software ecosystem.

3 Proposed Directions

We propose that ASCR and the HPC community should extend the software ecosystem to software build tools with an emphasis on the modern CMake. Closer community involvement will improve the build system capability and its usage of individual packages towards more robust and portable software build process. The outcome of this effort will make a large impact on the entire HPC community and ASCR's projects. Efforts to improve the software ecosystem build processes should include:

- Provide substantial funding and direction to address some of the major core language issues with CMake that impede initial adoption by new users and developers and complicate maintenance of CMake build systems.
- All future DOE contracts for new HPC systems should include some funding for the system vendors to make CMake functional under their system configurations, and address any CMake issues on systems as a first order user on the system. This includes the capability to locate, test and detect the version/build information of compilers, runtime and proprietary libraries (e.g. Intel Math Kernel, Libraries, HDF5).

- Define minimal standards and policies for the usage of modern CMake to create package build systems to enable portable package integration, especially on HPC systems with ease.
- Provide training to DOE package developers in the usage of modern CMake and minimal standards for usage on HPC systems.
- Encourage the continued transition of the DOE software ecosystem being built under Spack to a consistent compatible usage of modern CMake for all packages built from source.
- Provide support and encourage collaboration and interoperability between CMake/CTest HPC support tool providers that fill in the gap between raw CMake/CTest and tools like Spack.
- Create a clear separation of the functionality between Spack and CMake. We suggest that Spack should confine its roles to the orchestration of building multiple packages, version compatibility management (and patching), module/container deployment, and support of build-cache management to improve build and installation performance and robustness. Alternatively, the portability, robustness, and fine-level details (i.e. exact compiler options) of building individual packages should be left up to CMake.

References

[1] R. A. Bartlett, I. Demeshko, et al. xsdk foundations: Toward an extreme-scale scientific software development kit. *Supercomput. Front. Innov.*, 4(1):69–82, 2017.

[2] R. A. Batlett et al. TriBITS. `https://tribits.org`. Accessed: 2021-11-30.

[3] T. Gamblin, M. LeGendre, et al. The spack package manager: Bringing order to hpc software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15. Association for Computing Machinery, New York, NY, USA, 2015. ISBN 9781450337236.

[4] M. Heroux et al. E4S Project. `https://e4s-project.github.io/index.html`. Accessed: 2021-11-30.

[5] D. Ibanez and B. Granzow. CApp. `https://github.com/sandialabs/capp`. Accessed: 2021-11-30.

[6] C. Scott. Professional CMake, A Paractical Guide, 10th edition. `https://crascit.com/professional-cmake`. Accessed: 2021-11-30.

[7] C. White et al. BLT. `https://github.com/LLNL/blt`. Accessed: 2021-11-30.

[8] A. J. Younge, K. T. Pedretti, et al. A tale of two systems: Using containers to deploy HPC applications on supercomputers and clouds. In *IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2017, Hong Kong, December 11-14, 2017*, pp. 74–81. IEEE Computer Society, 2017.