Software Sustainability Considerations for a Performance Library

A white paper for the 2019 Collegeville Workshop on Sustainable Scientific Software (CW3S19)

Sarah Knepper, Shane Story; Intel Corporation

sarah.knepper@intel.com, shane.story@intel.com

Software sustainability is a multi-faceted topic. Just as it is important that developers are able to maintain, extend, and improve on a software project, so too is it important that the resulting software is usable, both now and in the future. While all software projects must make tradeoffs and prioritize aspects of software sustainability, there are certain challenges and benefits that a vendor performance library like the Intel® Math Kernel Library (Intel® MKL) faces. Intel MKL features highly optimized, threaded, and vectorized math functions that maximize performance on each processor family, using industry-standard C and Fortran APIs. It is a multi-decade, closed-source, free library covering functionality spanning dense and sparse linear algebra, fast Fourier transforms, vector math, random number generators, and more. We will consider a few facets of software sustainability and their considerations in Intel MKL as they impact user and/or developer experience.

APIs

The application programming interfaces (APIs) are one of the first experiences a new user has with a library. Whenever and wherever possible, to improve portability it is desirable to build off de facto standards, such as the BLAS and LAPACK for dense linear algebra and FFTW for fast Fourier transforms. Using industry-standard APIs also allows compatibility and eases the burden on application developers. For instance, the `dgemm` function in BLAS is widely understood to perform the following Double precision GEneral Matrix-Matrix product: $\mathbf{C} := \text{alpha} * \text{op}(\mathbf{A}) * \text{op}(\mathbf{B}) + \text{beta} * \mathbf{C}$, where $\text{op}(\mathbf{X})$ is either $\mathbf{X}$ or $\mathbf{X}^T$. By structuring an application's code to call `dgemm`, an application developer can swap in any library that provides an implementation of the BLAS. While a newcomer may initially find the function names esoteric, one quickly discovers the common naming conventions used in the BLAS and LAPACK function names and can learn to decipher them with ease.

While designing APIs, one needs to consider multi-language support. A relatively flat API, that minimizes the use of higher-level objects, may be easier to use in a different language or library. For instance, the popular Eigen and Trilinos libraries contain wrappers to core Intel MKL functionality, allowing users to take advantage of the higher-level APIs while still benefiting from optimizations in Intel MKL.

Linking

To improve usability of a software project, it should be buildable and usable with as many tools as possible. For a library like Intel MKL, this means that a large range of compilers, operating systems, linking models, and even threading paradigms should be supported. Additionally, the automatic dispatching in Intel MKL allows the library to detect the processor being used and run code that specifically targets that processor. As a result, so long as the link line does not change and backwards compatibility is maintained, an application that dynamically links to Intel MKL can just drop in a new version of the library without needing to re-build. However, sometimes disruptions are unavoidable, and a trade-off analysis must be performed. For instance, quite a few years ago, Intel MKL introduced a layered model concept to support different interfaces, threading models, and core computations. This required a change to the link line, which halted adoption for a while. However, the layered model concept has proven to stand the test of time, supporting new library features like TBB threading support that did not exist when the link line change was first made. Thus, it is imperative to anticipate change and attempt to get it right the first time when introducing incompatibilities, to avoid unnecessary disruptions.

Features

While it is tempting to support every feature request that is received, it is not sustainable to do so. There is always the initial burden of implementing, testing, and documenting a feature; then there is the ongoing support required to maintain the feature going forward, ensuring its compatibility as other features are added. For a project with a large customer base like Intel MKL, it is not feasible to continually add and subsequently remove features; the deprecation and ultimate removal of features is not taken lightly. The need to avoid disruptions to users requires that the decisions of what functionality to include or deprecate are highly scrutinized, with decisions documented internally. Once a library reaches a certain level of maturity, instead of taking a “if you build it, they will come” approach, it may be better to ensure that any added feature would be immediately useful for multiple users and expected to remain useful for years to come.

Along these lines, the ability to customize the library to meet the specific needs of a user is highly desirable. Open source software is more naturally customizable, so a closed-source product has more challenges to achieve this aspiration. The custom dynamic-link libraries/shared objects building capability in Intel MKL allows users to customize the library to their needs. For instance, an embedded application more have more library size limitations than a high-performance computing application, and so the embedded application developer may want to include only those parts of a library used in the application. It is easier if this sort of need can be anticipated and architected in from the start, rather than cantilevered in later.

Development

Being a performance library, the performance impact of a code change may be weighed heavier than the readability or maintainability impact. This can lead to large hand-written assembly files that require a high level of expertise to understand. On the other hand, given the large range of processor families, operating systems, and programming languages that are supported, maximizing code re-use is absolutely necessary. This can be achieved in multiple ways, including relying on the compiler to provide optimizations, writing generic code that can use processor-specific block sizes, or even developing a generator that produces different output tuned for a particular problem size or instruction set.

For software projects that are driven by a large corporation like Intel, the tools used are often chosen for business reasons outside of the control of the developers, to have consistency within the corporation and reduce the support burden on the IT staff. This can lead to situations where the history of source code changes is lost or difficult to locate if the version control system changes, say from CVS or SVN to Git. Similarly, if the defect tracking system changes, history of bugs may also be obfuscated. A tool change can sometimes provide an unexpected forcing function to thoroughly analyze the current state of the code and processes used during development, ultimately improving the quality.

Since Intel MKL ships as part of a broader package of useful libraries and tools, the schedule can sometimes drive decisions that may impact design or implementation choices. However, this ensures a consistent release cycle, allowing the latest features and bug fixes to reach users in a timely manner. An established product often has a list of release criteria that must be met, including items like no violations from a static analyzer tool or attaining certain pass rates for different types of tests. The regular cadence of releases with weekly analysis of the current status of release criteria helps to catch issues relatively quickly after their introduction.

While Intel MKL is a closed-source library, the online forum provides an opportunity for users to raise, comment on, and track issues. There is significant community involvement on the forum, with both Intel and non-Intel contributors providing feedback, going back over 15 years.

Summary

This white paper has touched on a number of software sustainability topics that impact projects of all shapes and sizes, focusing on a decades-old, widely-adopted performance library from Intel: Intel MKL. It has considered sustainability both from the user experience as well as from the software developer experience. As with all projects, tradeoffs are needed to balance resources, objectives, timelines, and priorities, while ensuring the continued long-term success and viability of the library.