Making Correctness Testing and Performance Autotuning Integral Parts of Software

X. Sherry Li, Lawrence Berkeley National Laboratory. xsli@lbl.gov

Keeping software updated is a challenge. Sustainability refers to the ability to maintain the scientifically useful capability of a software product over its intended life span, including understanding and modifying a software product's behavior to reflect new architectural advances. To this end, an important task in the development cycle is to design and implement a comprehensive set of testing strategies with a continuous integration harness. In addition, the software needs to deliver good performance when moving to a new hardware environment.

Nowadays, more and more scientific software has adopted good methodology to increase sustainability. For example, the following methods are widely used: 1) *Automatic build system.* The old-school way of editing makefile leaves much burden to the users and is error-prone when moving to a new system. Using CMake/Ctest greatly helps increase build-test productivity and robustness, and easier to manage dependencies on third-party software. CMake supports builds for both Linux and Microsoft Windows. 2) *Open source repository.* The earlier svn tool was good enough to coordinate code contributions within the development team but proved more difficult when we want to incorporate contributions from the user community. Git is a better tool to incorporate distributed contributions, making the code truly open-source.

However, the following elements are often overlooked, and become more and more important, especially for numerical software targeted at high performance computers:

- **Correctness testing.** An automated regression test suite is a crucial task for any major code. For example, we implemented the following testing strategy in SuperLU sparse direct solver software.
    - Collected small- to medium-sized test matrices with different numerical properties and sparsity structures.
    - Wrote the coverage unit testing code with all possible combinations of input parameters and matrices to invoke all user-callable solver routines. For each test, we check backward and forward error bounds.
    - Set up nightly regression test harness for Linux desktop and Mac laptop. The nightly test results are displayed on the CDash dashboard using the CMake/CTest facility.
    - Hooked up with Travis CI for continuous integration testing, which is triggered after each commit to the git repo, so any new code can be tested automatically and immediately.

  The developers execute the testing code regularly to make sure the newly added features do not introduce any errors in the production software.
  The users also have a responsibility: they should execute the testing code at the installation stage to make sure the library delivers correct results and performs as

intended. Passing the installation test increases users confidence in the software. Moreover, when the user code encounters errors, the testing code of each component (e.g., libraries) can greatly help the user isolate the search space and pinpoint potential source of errors.

- **Performance autotuning.** Given a myriad of diverse computer architectures, achieving optimal performance and performance portability of the scientific codes has become an increasing challenge. Each of these codes has a number of tuning parameters that may be difficult to choose to optimize performance, e.g., minimize runtime, memory use or energy consumption.

  We are working on building a general autotuning framework applicable across many applications. Autotuning strategies are concerned with picking the right set of parameters to optimally solve a particular problem on a given architecture. To this end, we are investigating a Bayesian black-box optimization method based on multi-output Gaussian process. The execution of the application code is treated as a function evaluation with a set of exposed performance-sensitive parameters. Specifically, we use multitask and transfer learning to exploit the correlation among the multiple function evaluations at different parameters to build the learning model which can choose the parameter setting for the unseen task. It is also possible to incorporate algorithmic and hardware performance models in this autotuning framework.

We believe that making the correctness testing and performance autotuning the integral parts of the software release would greatly increase the productivity of the developers as well as the users. It would increase the sustainability of the software going into the exascale computing era when node architecture is highly heterogeneous.