Software Monsters: Quantifying, Reporting, and Controlling Composite Applications

Justin M. Wozniak, Computer Scientist
Data Science and Learning Division
Argonne National Laboratory
(630) 252-3351 — woz@anl.gov

Challenge

Exciting new scientific applications that are rapidly developed to attack new, critical, and dynamic application spaces are *predominantly* large composite applications (or workflows) that integrate a great deal of software together. New computational paradigms, such as the prevalence of maching learning techniques, uncertainty quantification, and design optimization add to the importance of programming at this level. Applications thus face challenges when integrating the significantly different paradigms of high-performance computing (HPC), big data analysis, and the machine learning toolboxes emerging today. We propose that fundamental software metrics can be brought into innovative programming models to address the construction and execution of scientific applications.

Consider the recent (2020-2021) Gordon Bell finalists for the COVID-19 Special Prize. These applications are generally phrased as workflows, as they combine ensembles of simulations coupled with learning, analysis, and visualization. One such application with an complex machine learning component [1] utilized a custom R installation to perform a novel multi-objective optimization scheme; the R library size to perform these optimizations contains 112 packages totaling 321 MB, and this was for software generally understood and directly used by the developers. More opaque packages such as TensorFlow reside in Python installations totalling 20 GB or more.

These *monstrous* software dependencies and integrations pose a critical challenge to forward progress in scalable scientific applications. They create difficulties maintaining and validating scientific results over non-trivial time scales, and must be better understood.

Opportunity

"If you can't measure it you can't manage it."

The first step toward a scientific study of changes in culture affecting scientific software development and use is quantifying those changes. We propose that a software **score report** can be used to rapidly convey key statistics about the software used by a composite application. But the key observation here is not the static report itself but the *changes over time* to capture developments that involve new risk to an application. Such reports could be automatically generated by a system that understands software dependencies (e.g., Spack [2] or an extension to it).

Traditional software metrics include lines of code, code points, number of packages, the compiled size of packages, and code coverage metrics. Key questions that can be addressed herein include:

1) How many lines of code are written by the user?
2) How many interface points are there between user code and library code?
3) What is the code coverage of the user and library code?
4) What is the code coverage of the user and library code on the current hardware?
5) What is the code coverage of the user and library code on the current application problem?

The measured numbers, however, should not be immediately interpreted as good or bad but monitored for trends over time. These can be collected with respect to changing application structure as components are added or removed and compared against, for example, test suite results. We envision that they could be captured by automated continuous integration tools (e.g. GitLab).

"What just broke?"

A second step toward is the construction of software **structure graphs** using AI-based techniques. Deep learning systems have been used for graph analytics for some time, and can be used to expose underlying features in structure and key latent variables. [3] The graph structure of dependencies and calls in a composite application could reveal metrics for change over time, while also exposing structural qualities (broad vs. deep structures) and the encounter of new structures not seen before, revealing the associated risks and opportunities. Structure graph fingerprints could thus be compiled and exchanged. These graphs allow software metrics from software dependencies to be immediately made available to higher-level packages, thus making score reports easy to generate and understand as software changes.

Software structure graphs are a first step toward AI-driven development tools and best practice identification for scientific software development. Structure graphs collected from ECP-related applications along with user-labeled tags could be analyzed to rapidly reveal software structural practices that are likely to produce good outcomes. At the lowest level, the graph would contain entries for vendor-provided tools for messaging, operating system services, and so on. Structure graphs could be extended to include hardware components, including accelerators or other exotic devices.

Structure graphs could also be used to assist in the automatic generation of auto-tuning schemes for performance, accuracy, or multi-objective schemes [4]. Identifying the most critical and/or sensitive parameters in library software is challenging. Composite application tuning is thus a structured optimization problem in which some subproblems have already been (partially?) solved. If these parameters and knowledge bases could be exposed to higher-level packages, auto-tuning problems could be automatically generated from the structure graphs, while gaining the acceleration from prior dependency optimization runs.

We stress that this kind of reporting could be seen as time-consuming and tedious. To bring AI into the software analysis loop, however, we see that some fundamental metrics must be exposed for automated analysis, and this workshop could be critical in identifying those.

TIMELINESS

"Given enough eyeballs, all bugs are shallow."

This type of software analysis is needed immediately as the rapid adoption and longevity of advanced machine learning libraries developed by small disparate teams and/or corporate giants become critical to scientific applications. These projects must be manageable, and fundamental to that is understanding application structure.

The expected outcome of this approach will ease the development and validation of complex studies that integrate domain science with data science and machine learning techniques. Multiple proposed computer science investigations will result, including fundamentals in AI understanding and information extraction from reports and graphs, as well as software challenges in producing the raw data from existing package managers. The work proposed here will lay the groundwork for future programming models and methods that will allow a wide range of application domains to rapidly adopt hierarchical programming and reduce the complexity and of application-specific testing and evaluation.

This level of programming will become critically important as complex model exploration studies and deep-learning-infused workflows are deployed on exascale systems. In the absence of a comparable systems and methodologies, researchers will have difficulty developing and deploying composite applications. Developments in community software packages will be unmanageable. Software management for advanced scientific computing will be left in the human bandwidth-limited status quo.

REFERENCES

[1] J. Ozik, J. M. Wozniak, N. Collier, C. M. Macal, and M. Binois, "A population data-driven workflow for COVID-19 modeling and learning," *International Journal of High Performance Computing Applications (Finalist for Gordon Bell COVID-19 Special Prize)*, 2021.

[2] T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and W. S. Futral, "The Spack package manager: Bringing order to HPC software chaos," in *Proc. SC*, 2015.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, Aug 2014. [Online]. Available: http://dx.doi.org/10.1145/2623330.2623732

[4] T. Shu, Y. Guo, J. M. Wozniak, X. Ding, I. Foster, and T. Kurc, "Bootstrapping in-situ workflow auto-tuning via combining performance models of component applications," in *Proc. SC*, 2021.