3-Way Software Integrity for HPC

Markus Schordan, Dan Quinlan
Lawrence Livermore National Laboratory
{schordan1,dquinlan}@llnl.gov
May 4, 2015

Trustworthy supercomputing depends on advanced program optimizations to deliver high performance and ways to determine how modifications to a program impact its behavior. The first is important to allow its application on supercomputers, the latter important for cybersecurity. We must be able to track and identify any changes to programs and analyze what the source of the change was. This might be manual changes, or changes introduced by a compiler (in an optimization), or a combination of both.

For program optimizations, polyhedral compilers have become increasingly important in the HPC community as they allow to optimize cache performance and to introduce parallelism automatically. This systematic approach, using the polyhedral model [1], has paved the road to successfully map affine stencil computation to a variety of CPUs, GPUs and FPGAs and also allows to address scalability of HPC kernels. However, verifying an entire polyhedral compiler engine itself, e.g. the Polyhedral Compiler Collection (PoCC), which is the result of 8 years of multi-institution development is out of reach: the compiler is actually around 0.5 million lines of code, making the effort of producing a certification of these compiler optimizations in a manner similar to Leroy’s CompCert work extremely high. However, restricting the verification to certain aspects of a program has allowed to verify kernels of scientific codes. For example, recently some optimization variants have been successfully verified for all benchmarks in the Polybench/C suite [3].

To ensure scientific data integrity, checks on data must be performed, to ensure that no data corruption has occurred (e.g. correctness of repository contents, signatures, etc.).

Thus, to be able to ensure HPC Compiler Integrity, HPC Optimization Correctness, and HPC Data Integrity, we face the problem of repeatability and reproducibility. This problem has become an important issue for computer science in general [2]. We can target systematically the above three challenges for HPC integrity by separating the concerns into

(i) Are we able to repeat the computation at any point in time?
(ii) Are we able to reproduce the computed results?
(iii) Are we able to provide proof that the original input data has not been tampered since any point in time?
(iv) Are we able to provide proof that the program that computes the data has not been tampered since any point in time?

If the input data or HPC application was tampered then we cannot reproduce the results. We may be able to repeat (rerun) the computation, but the results may be different. If we cannot repeat the computation (e.g. a simulation that was running for many weeks on hardware that is not readily available) we rely on (iii) and (iv). (i) and (ii) are also required to test the quality of a model with input data from the past and how well the present, or an other point in time for which data has been gathered, would be predicted.
One crucial aspect is to differentiate between code development and the necessary adaptions of the code and code tampering. We propose a 3-way verification, targeting three different kinds of application properties:

(a) Quality properties of the code (static analysis)

(b) Data integrity properties (static and dynamic analysis)

(c) Platform-specific run time behavior properties (monitoring)

Quality properties of the code are to be never compromised. No version or variant of the HPC application is allowed to not properly maintain the quality properties. For example, we verify for all array accesses in the code that no past the array bound access (or exception) can occur. No change of the source code is allowed that would change that property.

Data integrity properties can be verified at compile time as assertions or through run time verification. Some data integrity can only be performed once the input data is available, therefore both, compile time and run time verification is necessary to verify data integrity.

Platform-specific run time behavior properties are monitored through hardware counters and specified based on previous profile runs. This allows to ensure that an application performs within defined platform-specific bounds. For example, we can specify that the cache behavior of a given code can only divert from a given profile (computed by previous runs) within a specified range. This data is platform specific, but could also be used for an abstract machine that is specifically designed for checking HPC integrity. If such ranges are specified, changes to the software can be identified through a change in behavior that is beyond the capabilities of source-code and binary analysis.

With (a) we address the detection of modifications exposing information through undefined behavior of applications. It also ensures that corner-cases (e.g. past array bound access) that usually are difficult for many analysis and verification tools to handle, are avoided. With (b) we address and hope to specifically tailor assertion verification for HPC. With (c) we “learn” about the expected behavior of an application on a certain platform.

By establishing and combining the maintenance and monitoring of these three properties in the development life cycle of HPC applications, software integrity can be improved dramatically within the next decades, but can also leverage much of the excellent work that has already been done within the past decades. With this 3-way verification approach we can achieve a 3-way software integrity for HPC.

Based on this information we are also able to assess the impact of changes that have been done by an individual person. If this information (the difference of the old to the new version) is stored, we can also create valuable account specific (personalized) information for later analysis, to analyze what impact a specific person had on a code base.

Acknowledgments. LLNL-CONF-670401. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC.

References

