Automated, Extensible Correctness Diagnostics for Scientific Computing

Chunhua Liao[1] and Peter Pirkelbauer

Lawrence Livermore National Laboratory

Challenge

Developing scientific simulation codes involve multiple iterations of designing, implementation and refactoring in order to exploit everchanging hardware platforms to answer bigger scientific questions. Any changes to the simulation codes may potentially introduce bugs. Currently, developers often manually insert ad-hoc diagnostic statements into an application to log relevant variable values, at critical steps within an execution. The resulting log information is then used for manual investigation to ensure that the simulation runs as expected. Obviously, this common practice is time-consuming, non reusable, and error-prone.

Opportunity

There is an opportunity to develop extensible and automated techniques to support universal correctness diagnostic capability suitable for reuse across many simulation codes. Ideally, a computational scientist would be able to identify a set of key properties of interests and use standard language constructs, source code annotations or directives to express diagnostic semantics. The HPC programming systems, including compilers and runtime systems, would then take over and automatically implement the corresponding code instrumentation, data logging, and even problem diagnosing and reporting.

In order to achieve this vision, we need to invest in the following research areas:
1. Systematically collecting application semantics which are commonly used by developers to support correctness diagnostics of simulation codes. Example diagnostic semantics can be as simple as which stage a program is currently executing or a range of allowed values for a given variable. Others can be highly domain-specific information such as the number of segments executed in a particle transport program, or convergence rates of iterative methods.

2. Initiating development of standard ways to express such diagnostic semantics and communicate them to a compiler/runtime system, which in turn can automatically instrument code at the right time and trigger the right checking for the relevant properties. Directive-based programming models, such as OpenMP, could be an ideal vehicle to experiment with new annotations for such purposes.

3. It may take years to have matured language and compiler support for automated correctness diagnostics. Library-based approaches should be investigated as early as possible so

[1] Corresponding author: liao6@llnl.gov This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-ABS-829256

experienced developers can leverage libraries to improve diagnostic support. The libraries often form the basis for runtime library support of compilers. There are existing libraries for memory access patterns (e.g. XPlacer[1]) and performance introspection (e.g. Caliper[2]) of HPC software. They could be extended to support correctness diagnostics.

4. Given the diverse and rapid-changing scientific computing needs, it is impossible to anticipate all semantics and properties which are needed in the field. We need to study easy interfaces for users to add customized extensions. Callback functions and user-defined operations (similar to user-defined reduction in OpenMP) may be possible approaches to realize such extensions.

5. Side effects and overhead are major concerns of any code instrumentation, which may disrupt original performance characteristics, impede compiler optimization, and increase time and space complexity of applications. Further research is needed to minimize such negative impact.

In the end, wide collaboration among domain scientists, applied mathematicians, computer scientists, and software engineers is needed to reach the goal of extensible and automated correctness diagnostics of scientific applications.

Maturity

Compiler-based code instrumentation is a mature technique to help inspect behaviors of programs. The resulting information can be used for correctness checking (including data race detection), memory access pattern analysis, and so on. On the other hand, directive-based programming models, such as OpenMP and OpenACC, are commonly used to standardize annotations related to parallelism. Prior studies also successfully explored other types of semantics, including fault tolerance[3]. It is a promising new research direction to incorporate semantics related to correctness diagnostics so users can leverage compilers and runtime to automatically conduct in-depth and standard correctness investigation of scientific computing.

References

1. Peter Pirkelbauer, Pei-Hung Lin, Tristan Vanderbruggen and Chunhua Liao, XPlacer: Automatic Analysis of CPU/GPU Access Patterns, 34th IEEE International Parallel & Distributed Processing Symposium, May 18-22, 2020.
2. David Boehme, Todd Gamblin, David Beckingsale, Peer-Timo Bremer, Alfredo Gimenez, Matthew LeGendre, Olga Pearce and Martin Schulz. Caliper: Performance Introspection for HPC Software Stacks. In Supercomputing 2016 (SC16), Salt Lake City, Utah, November 13-18, 2016.
3. Jacob Lidman, Daniel J. Quinlan, Chunhua Liao and Sally A. McKee, ROSE::FTTransform A Source-to-Source Translation Framework for Exascale Fault-Tolerance Research, Fault-Tolerance for HPC at Extreme Scale (FTXS 2012), Boston, June 25-28, 2012.