Overcoming Extreme-Scale Reproducibility Challenges Through a Unified, Targeted and Multilevel Toolset

D. H. Ahn, G. L. Lee, G. Gopalakrishnan, Z. Rakamaric, M. Schulz, I. Laguna

August 14, 2013

First International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering
Denver, CO, United States
November 17, 2013 through November 17, 2013

Overcoming Extreme-Scale Reproducibility Challenges Through a Unified, Targeted, and Multilevel Toolset[*]

Dong H. Ahn
Lawrence Livermore National Laboratory, Livermore, CA
ahn1@llnl.gov

Gregory L. Lee
Lawrence Livermore National Laboratory, Livermore, CA
lee218@llnl.gov

Ganesh Gopalakrishnan
School of Computing, University of Utah, USA
ganesh@cs.utah.edu

Zvonimir Rakamarić
School of Computing, University of Utah, USA
zvonimir@cs.utah.edu

Martin Schulz
Lawrence Livermore National Laboratory, Livermore, CA
schulzm@llnl.gov

Ignacio Laguna
Lawrence Livermore National Laboratory, Livermore, CA
ilaguna@llnl.gov

ABSTRACT

Reproducibility, the ability to repeat program executions with the same numerical result or code behavior, is crucial for computational science and engineering applications. However, non-determinism in concurrency scheduling often hampers achieving this ability on high performance computing (HPC) systems. To aid in managing the adverse effects of non-determinism, prior work has provided techniques to achieve bit-precise reproducibility, but most of them focus only on small-scale parallelism. While scalable techniques recently emerged, they are disparate and target special purposes, e.g., single-schedule domains. On current systems with $O(10^6)$ compute cores and future ones with $O(10^9)$, any technique that does not embrace a *unified*, *targeted*, and *multilevel* approach will fall short of providing reproducibility. In this paper, we argue for a common toolset that embodies this approach, where programmers select and compose complementary tools and can effectively, yet scalably, analyze, control, and eliminate sources of non-determinism at scale. This allows users to gain reproducibility only to the levels demanded by specific code development needs. We present our research agenda and ongoing work toward this goal.

1. INTRODUCTION

Reproducibility, meaning the ability to repeat executions with the same numerical result or code behavior, is a highly desired feature of HPC codes, especially from the perspective of end users. It is also a key desideratum for efficient code development in virtually all HPC code development life-cycles. Unfortunately, non-determinism, e.g., as introduced by concurrency scheduling, has a far-reaching detrimental impact on reproducibility. In the message-passing paradigm, for instance, a wild-card receive can match messages that are sent in different orders [18]; in the shared-memory paradigm, threads can access the same data in different interleavings; and in the multi-task paradigm, tasks are executed in the arrival order, creating a timing dependency. Such non-determinism not only affects the perceived order of computational steps, thus hampering code understanding for developers, but also the numerical results that determine the quality or reliability of the scientific findings.

Computational science and engineering applications require reproducibility for code verification, debugging, and code validation, as well as for comparative performance measurements. For example, if a coding error emerges only in a rare execution path, it is not only hard to reproduce and fix in subsequent runs, but also nearly impossible to distinguish it from a transient fault. Similarly, different execution paths can change the order of floating-point operations, which may not be associative. This impedes numerical reproducibility across runs, making code verification difficult.

As multicore systems are going mainstream, research to provide safety properties, such as determinism, has attained renewed urgency. However, much prior work has focused on bit-precise reproducibility at a scale much smaller than what large scale applications require. Very recently, more scalable techniques have emerged [20, 8], but they often only target a specific domain (e.g., single-schedule domains).

In today's massive amount of non-determinism—introduced from multiple levels of parallelism—disparate, special-purpose tools will not reach the reproducibility level demanded by extreme-scale computing. In contrast, we argue for a *common* toolset that embodies a *unified*, *targeted*, and *multilevel* approach as an effective—yet scalable—means to analyze, control, and eliminate non-deterministic sources.

2. POSITION AND RESEARCH AGENDA

For extreme-scale concurrency, we are skeptical that a one-size-fits-all tool can provide production applications with the required degree of reproducibility. Prevalent HPC trends are already rushing into a multi-dimensional scale explosion of reproducibility concerns. For example, the U.S. Department of Energy is actively determining the key elements of its exascale roadmap [10, 7, 23, 16], and most researchers believe this will deliver machines with over a billion compute cores, building on sharply reduced memory per core and heterogeneous computing or accelerators. Further, these architectural trends will force applications to use mixed parallelism (e.g., MPI+X), or rely on new, often fine-grained task-based systems, both of which introduce more diverse sources of non-determinism [4]. Given these trends, previous research that focused only on controlling small-scale non-determinism for single paradigms become impractical.

[*]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-CONF-642354). The Utah group was supported by NSF Awards OCI 1148127 and CCF 1241849.

When combating extreme-scale reproducibility challenges, formal methods such as dynamic concurrency analysis and control mechanisms, which work well at small scales, will break. Simply put, there will be too much non-determinism coming from too many sources. Furthermore, tools that can effectively control schedules on a single-schedule domain can lose their efficacy on hybrid parallelism when non-deterministic behavior is observed only as a result of an interplay between multiple schedule domains.

In short, reproducibility on extreme-scale computing is a grand challenge: we must perform more complex and expensive operations while needing to scale them up to orders-of-magnitude larger concurrency levels. Our proposed solution to this challenge is a common *multilevel* toolset that *unifies* complementary and *targeted* tools that can build on the strengths of one another. Clearly, a broad research agenda will be required to achieve this goal.

What are the important levels of reproducibility for application development? We view reproducibility as a continuum. Depending on specific code development needs, users may require only the final result to be *externally deterministic* [4]. Further, users may require bit-precise reproducibility in a scale-invariant fashion (e.g., independent of MPI process or thread counts), or only less strict statistical guarantees. Other users may require intermediate results to be reproducible through consistent schedules between runs. However, because there are multiple intermediate levels, *internally deterministic* [4] behaviors increase complexity. Thus, we must understand common application properties that can benefit from various reproducibility levels.

How to identify and limit the targeted sources of non-determinism? Not all non-determinism is relevant to the required level of reproducibility. We must develop ways to identify and limit the relevant sources of non-determinism while letting loose other sources to minimize performance and scalability impact.

What should be the complementary capabilities and composability attributes of tools for effective search-space reduction? Some tools must scalably analyze the severity levels of non-determinism and reduce the search space; other tools must provide detailed analysis on the reduced space; yet, others must analyze the various manifestations of non-determinism, e.g., quantifying the impact on floating point arithmetic operations. In order for these tools to build on one another, we must understand a variety of use cases and workflows.

How to coordinate tools in multiple-schedule domains effectively? A growing trend towards hybrid parallelism adds yet another dimension: how to combine tools with the same capabilities in different schedule domains, e.g., combining an MPI-schedule-controlling tool with a thread-schedule-controlling tool. Would it still be effective to explore alternative schedules of one domain while permitting non-determinism in other domains?

How to evaluate the effectiveness of the toolset? Setting up controlled experiments in the presence of non-determinism is challenging on its own right. Thus, one needs to research systematic ways to introduce or remove non-determinism into applications for evaluation.

3. CURRENT PRACTICE AND BENEFITS

It is well known that software debugging and testing are

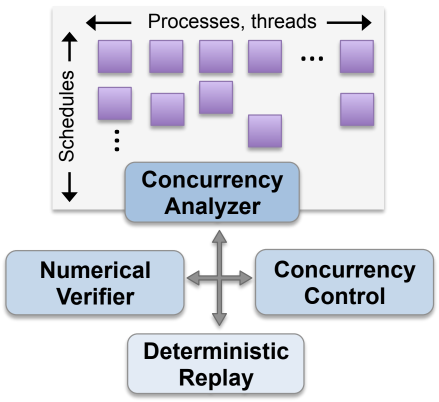

Figure 1: Main building blocks of our approach

expensive. For example, the National Institute of Standards and Technology (NIST) estimates that software errors cost the U.S. economy $60 billion annually, and half of the cost is due to developers' time since they overall spend about 25% of their time for debugging and testing. Our experiences indicate that there is a large gap in the amount of time required to address various classes of bugs. In particular, concurrency-related non-deterministic errors are far more costly to fix; the larger the concurrency, the more expensive the debugging process becomes.

Further, computational science and engineering applications are often in continuous development—they have a frequent feedback loop with experiments; the result of an experiment, for instance, may lead to changes in physics theories and practices, or may require new code to be developed for new physics regimes. In fact, for applied science organizations like Lawrence Livermore National Laboratory, code development is often on the critical path to their mission.

Reproducibility plays a critical role in every aspect of this code development effort; when there are no reliable mechanisms to repeat the code behavior or to achieve the same numerical results, debugging, testing, and verification become increasingly expensive tasks. However, current techniques to managing the effects of non-determinism are largely primitive and ineffective, leading to a significant increase in the development cost. Programmers often implement ad hoc solutions directly in their codes. This is not only error-prone, but also leads to performance or accuracy losses [15, 13] and results in redundant efforts across many codes. As noted before, the ad hoc approach will be ineffective and costly on future systems in which an explosion in concurrency will multiply the effects of non-determinism [18, 19].

By furthering our research agenda, the resulting toolset will significantly improve productivity of programmers across the full application development life-cycle. The research impact is doubled, since the time saved by the toolset will produce extra minutes of other useful code development work. The common multilevel toolset can reduce the redundant efforts across codes, and will provide best performance and accuracy subject to the required reproducibility level.

4. THE PRUNER TOOLSET

We recently launched a project called PRUNER (Providing Reproducibility on Ubiquitously Non-deterministic Environments and Runs) to embody our unified, targeted, and multilevel approach. PRUNER aims to innovate scalable concurrency analysis and control mechanisms for extreme-scale computing. Applications can use these mechanisms to achieve reproducibility of both their execution and their simulation results easily and efficiently. We plan to accomplish this goal by developing a multilevel analysis and control toolset that blends static analysis with run-time techniques to detect, control, and eliminate targeted sources of non-determinism.

Figure 1 shows the main building blocks of our proposed toolset. Our toolset represents a multilevel approach because it will provide multiple levels of reproducibility by controlling and analyzing combined sources of non-determinism introduced from many levels of parallelism, including process-level and thread-level parallelism. We plan to use benchmarks to demonstrate that our toolset scales and to show its effectiveness by integrating it directly into strategies used by mission-critical applications for their testing, debugging, and verification.

To execute our objectives, we organize the project as three research and development thrust areas.

The first thrust advances relevant analysis techniques to detect, analyze, and identify root causes of non-determinism that are introduced by concurrency. However, performing these capabilities exhaustively on all potential code sites of non-determinism within a large application can be intractable. Thus, we will investigate techniques that combine static analysis [6, 17] with automatic, as well as user-guided, dynamic verification techniques [22, 20, 21]. Statically pruning benign sites can ease the pressure on our run-time component, allowing it to scale.

To complement our concurrency analysis and control techniques, we plan to develop a suite of numerical-analysis tools and a control-flow analyzer. Our numerical tools, which quantify the impacts of non-determinism due to different orders of floating-point arithmetic operations, will mix detailed techniques such as GAPPA-based analysis [4] with more scalable approximation methods such as guided random testing. These techniques are essential, as the current explosion in concurrency continues to intensify the adverse effects of this numerical non-determinism—e.g., the growth in numerical errors of a global summation is generally proportional to the degree of concurrency [19].

The control-flow non-determinism analyzer will efficiently compare the control flows between runs to narrow down non-determinism to specific points in execution. To ensure scalability, we will base our techniques on the Stack Trace Analysis Tool (STAT) [2, 12, 1], extending it to perform temporal (e.g., phases), spatial (e.g., processes), and code-space (e.g., functions) reductions to attribute a specific control flow to the observed non-deterministic behaviors.

The second thrust area advances techniques to force non-determinism to surface, as not all non-determinism is easy to be observed. Currently, programmers rely on arduous manual processes and spend days trying to reproduce rarely-occurring concurrency bugs at large scale [11]. These processes often involve ad hoc techniques, such as changing process counts or configurations, choosing different compiler optimizations, or modulating the absolute times at which concurrent events are issued, and are largely ineffective. Thus, we will develop systematic concurrency control techniques that force alternative schedules to expose non-deterministic behavior. Such techniques already exist for limited cases in tools like the Distributed Analyzer of MPI (DAMPI) [20, 21], but the challenge will be to apply similar techniques to other sources of non-determinism at scale. In this context, we will develop novel and highly-scalable distributed algorithms for shared-memory programming models, for example, exploiting the OpenMP tools API [9] to control thread schedules.

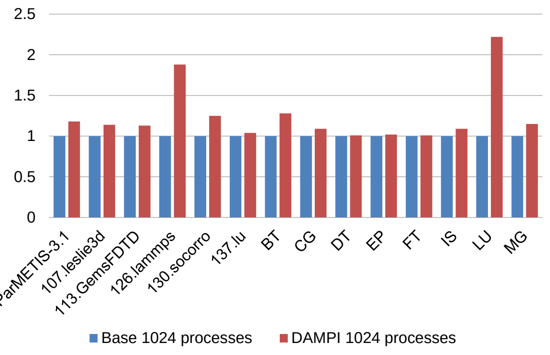

Figure 2: DAMPI overhead at 1K processes

By all means, the exploration space of alternative schedules is large, so we will investigate advanced heuristics that can significantly bound this search space. We will also develop efficient ways to repeat the replay of small regions of execution during exploration, including transactional memory based schedule replay in which many alternative schedules are replayed in transactional memory without having to require a full restart for each alternative schedule. We will also investigate ways to parallelize replays.

The final thrust area advances mechanisms to ensure that internal execution schedules are consistent with those made in prior runs. Consistent schedules have many advantages, including consistent numerical results between runs. However, achieving numerical consistency is becoming a greater challenge, as our applications are increasingly exploiting non-deterministic concurrency patterns such as polling-based MPI communication patterns [3] and threads contributing partial sums in different orders [19, 14]. To gain numerical consistency, applications are currently adopting heavyweight mechanisms [15], often leading to excessive performance or precision losses.

In response, we will investigate deterministic concurrent event ordering. These methods will enable programmers to fix the order of concurrent events (e.g., the message-matching order of a receive site) to achieve the desired level of consistency. We will also investigate multilevel techniques whereby the results of reductions, in conjunction with non-associative floating point operations, provide different reproducibility guarantees: scale-invariant, run-to-run, or statistical (e.g., bounded error growth) reproducibility. Overall, the focus of this thrust is to develop techniques that can maximize performance subject to acceptable reproducibility properties.

For this, scalability is the key aspect, and we plan to develop ways to create a feedback loop with the users so that they can specify a region of the code (e.g., selected functions) or a window of execution (e.g., the final phase) on which to focus. We plan to further explore ways to create equivalence relations among execution paths so that reproduced paths will be *equivalent* instead of *exact.*

Across all three thrusts, we will explore novel trade-off schemes that exchange reproducibility levels with performance and scalability [5]. To minimize the performance impact, we will combine highly-scalable run-time analysis with static analysis. In our previous work, we showed that distributed run-time techniques can incur low overheads and scale well as long as we limit the number of sites that are examined. For example, DAMPI uses a scalable Lamport-clock-based algorithm to detect at run-time those message matches that can occur in any order–i.e., the matches that do not form a *happened-before* relation. Figure 2 shows that medium-to-large benchmarks with limited non-deterministic wild-card MPI receive sites only ran up to ~2x slower under DAMPI at 1,024 MPI processes. In the case of the outliers such as LU, the wild-card receives that DAMPI analyzed as *non-deterministic* turned out to be *deterministic* due to specific uses of message tags. In particular, LU employs a scheme where wild-card receives uses a unique tag for each sender. We will develop various static techniques to extract information such as tag patterns, which can further limit the number of sites our run-time component has to examine.

Finally, evaluating our approaches to identify and eliminate non-determinism requires a reliable test and validation setup. For this, we will pursue the following two directions. First, we will evaluate the overhead incurred by our techniques when applied to benchmarks or applications for which we explicitly know the level of non-determinism. Second, we will develop a series of worst-case benchmarks, expanding the previously developed MPITEST suite [18], to measure the amounts of non-determinism in applications.

5. CONCLUDING REMARKS

Reproducibility plays a pivotal role in computational science and engineering applications. However, neither the current ad hoc approaches nor disparate, special-purpose tools will be powerful enough to provide this essential ability to the level demanded by extreme-scale computing. We propose a novel common multilevel toolset that embodies a *unified*, *targeted*, and *multilevel* approach as an effective—yet scalable—means to analyze, control, and eliminate sources of non-determinism. The notion of multiple levels of reproducibility will open the door for *affordable* reproducibility without solely relying on bit-precise reproducibility. The targeted approach will limit the sources of non-determinism to the required level while letting loose other sources to minimize performance and scalability impacts. The unified approach will enable quick narrowing down of the search space. By furthering a broad research agenda, we hope to produce a common toolset that can significantly improve productivity of programmers across the full code development life-cycle.

6. REFERENCES

[1] D. H. Ahn, B. R. de Supinski, I. Laguna, G. L. Lee, B. Liblit, B. P. Miller, and M. Schulz. Scalable temporal order analysis for large scale debugging. In *ACM/IEEE Conference on High Performance Computing (SC)*, 2009.

[2] D. C. Arnold, D. H. Ahn, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz. Stack trace analysis for large scale debugging. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.

[3] T. A. Brunner and P. S. Brantley. An efficient, robust, domain-decomposition algorithm for particle Monte Carlo. *Journal of Computational Physics*, 228(10):3882–3890, 2009.

[4] W.-F. Chiang, G. Gopalakrishnan, Z. Rakamarić, D. H. Ahn, and G. L. Lee. Determinism and reproducibility in large-scale HPC systems. In *Workshop on Determinism and Correctness in Parallel Programming (WoDet)*, 2013.

[5] W.-F. Chiang, G. Szubzda, G. Gopalakrishnan, and R. Thakur. Dynamic verification of hybrid programs. In *European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface (EuroMPI)*, pages 298–301, 2010.

[6] clang: a C language family frontend for LLVM. `http://clang.llvm.org`.

[7] SciDAC Co-Design. `http://science.energy.gov/ascr/research/scidac/co-design`.

[8] CORVETTE: Correctness verification and testing of parallel programs. `http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/corvette`.

[9] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. DelSignore, R. Dietrich, X. Liu, E. Loh, and D. Lorenz. OMPT and OMPD: OpenMP tools application programming interfaces for performance analysis and debugging. Technical report, 2013.

[10] DOE Extreme-Scale Technology Acceleration FastForward. `https://asc.llnl.gov/fastforward/rfp`.

[11] I. Laguna, D. H. Ahn, B. R. de Supinski, S. Bagchi, and T. Gamblin. Probabilistic diagnosis of performance faults in large-scale parallel applications. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 213–222, 2012.

[12] G. L. Lee, D. H. Ahn, D. C. Arnold, B. R. de Supinski, M. Legendre, B. P. Miller, M. Schulz, and B. Liblit. Lessons learned at 208K: towards debugging millions of cores. In *ACM/IEEE Conference on High Performance Computing (SC)*, pages 1–9, 2008.

[13] E. Loh. The Ideal HPC Programming Language. *Communications of the ACM*, 53(7):42–47, June 2010.

[14] M. M. Marinak, G. D. Kerbel, N. A. Gentile, O. Jones, D. Munro, S. Pollaine, T. R. Dittrich, and S. W. Haan. Three-dimensional HYDRA simulations of National Ignition Facility targets. *Physics of Plasmas*, 8(5):2275, 2001.

[15] A. A. Mirin and P. H. Worley. Improving the performance scalability of the community atmosphere model. *International Journal of High Performance Computing Applications (IJHPCA)*, 26(1):17–30, 2012.

[16] 2013 Exascale Operating and Runtime Systems. `http://science.doe.gov/grants/pdf/LAB_13-02.pdf`.

[17] ROSE compiler infrastructure. `http://rosecompiler.org/`.

[18] M. Schulz, D. Kranzlmüller, and B. R. de Supinski. Exploring unexpected behavior in MPI. In *International Conference on High Performance Computing and Communications (HPCC)*, pages 843–852, 2006.

[19] O. Villa, D. Chavarría-mir, V. Gurumoorthi, A. Márquez, and S. Krishnamoorthy. Effects of floating-point non-associativity on numerical computations on massively multithreaded systems. In *Proceedings of Cray User Group Meeting (CUG)*, 2009.

[20] A. Vo, S. Aananthakrishnan, G. Gopalakrishnan, B. R. de Supinski, M. Schulz, and G. Bronevetsky. A scalable and distributed dynamic formal verifier for MPI programs. In *ACM/IEEE Conference on High Performance Computing (SC)*, pages 1–10, 2010.

[21] A. Vo, G. Gopalakrishnan, R. M. Kirby, B. R. de Supinski, M. Schulz, and G. Bronevetsky. Large scale verification of MPI programs using Lamport clocks with lazy update. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 330–339, 2011.

[22] A. Vo, S. Vakkalanka, M. DeLisi, G. Gopalakrishnan, R. M. Kirby, and R. Thakur. Formal verification of practical MPI programs. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 261–270, 2009.

[23] X-Stack Software Research. `http://science.energy.gov/ascr/research/computer-science/xstack`.