

PRUNERS: Providing reproducibility for uncovering non-deterministic errors in runs on supercomputers

The International Journal of High Performance Computing Applications 2019, Vol. 33(5) 777–783
© The Author(s) 2019
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1094342019834621
journals.sagepub.com/home/hpc



Kento Sato^{1,2}, Ignacio Laguna¹, Gregory L Lee¹, Martin Schulz³, Christopher M Chambreau¹, Simone Atzeni^{4,5}, Michael Bentley⁴, Ganesh Gopalakrishnan⁴, Zvonimir Rakamaric⁴, Geof Sawaya⁶, Joachim Protze⁷ and Dong H Ahn¹

Abstract

Large scientific simulations must be able to achieve the full-system potential of supercomputers. When they tap into high-performance features, however, a phenomenon known as non-determinism may be introduced in their program execution, which significantly hampers application development. PRUNERS is a new toolset to detect and remedy non-deterministic bugs and errors in large parallel applications. To show the capabilities of PRUNERS for large application development, we also demonstrate their early usage on real-world production applications.

Keywords

Debugging, testing, reproducibility, non-determinism, MPI, OpenMP

1. Introduction

Software bugs, or errors, are so prevalent and so detrimental that they cost the US economy alone an estimated US\$59.5 billion annually, or about 0.6% of the gross domestic product, according to a study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST) in 2002 Tasse (2002). Finding and fixing bugs, or debugging, however, is an arduous task, significantly adding to this cost. Every time a bug arises, the programmer must manually diagnose and repair it, stepping through potentially millions of lines of code and examining hundreds or thousands of program state variables.

High-performance computing (HPC) that uses supercomputers has become critically important for research and development (R&D) efforts, engineering applications, and scientific discovery. Supercomputers typically contain very high numbers of compute cores, and applications running on such systems must rely on multiple communication and synchronization mechanisms as well as compiler optimization options to effectively utilize the hardware resources. These complexities often produce errors that occur only occasionally, even when run with the exact same input on the same hardware. These so-called non-deterministic bugs are remarkably challenging to catch due in large part to difficulty in reproducing them. Some errors may not even

reproduce when being debugged, as the act of debugging may perturb the execution enough to mask the bug.

To find and fix these errors, programmers currently must devote a large amount of effort and machine time. One of our recent case studies indicates that resolving a single non-deterministic bug can require three person-months worth of programmers' effort and 19 years of compute-core time in total (Sato et al., 2017). As supercomputers can cost hundreds of millions of dollars to procure and equal sums in energy and support costs to run, expenditure quickly adds up when debugging. Furthermore, the time a user spends debugging is the time taken away from utilization of a major capital investment and the science that the user is trying to conduct. Thus, tools that can detect and remedy these defects are highly valuable.

¹ Lawrence Livermore National Laboratory, Livermore, CA, USA

² RIKEN Center for Computational Science, Livermore, USA

³ Technische Universität München, Munich, Germany

⁴ University of Utah, Salt Lake City, UT, USA

⁵ NVIDIA, Utah, USA

⁶ Oculus Research, Salt Lake City, UT, USA

⁷ RWTH Aachen University, Aachen, Germany

Corresponding author:

Kento Sato, Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, California 94550-5507, USA.

Email: kento.sato@riken.jp

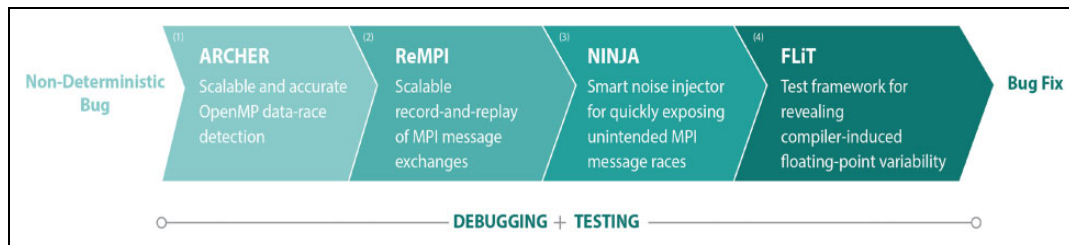


Figure 1. The PRUNERS toolset increases non-determinism coverage for debugging and testing workflows.

PRUNERS is the only software toolset that is designed specifically to solve these challenges of debugging and testing for non-deterministic software bugs with the scalability, accuracy, composability, and portability demanded by today's high-end systems. In addition, early usage on real-world bugs at Lawrence Livermore National Laboratory (LLNL) proves that the effectiveness of this coordinated toolset for fixing non-deterministic software bugs and errors.

2. The PRUNERS toolset

Sources of non-determinism are multilevel—arising from multiple levels of the software stack (e.g. applications, libraries, and/or system) and from different programming models and paradigms (e.g. message passing vs. shared memory). A single tool does not fit all required levels of reproducibility. Our solution to the challenge is the PRUNERS toolset (Figure 1). A reason for having a toolset is to flexibly control sources of non-determinism and provide required levels of reproducibility depending on specific code development needs by users—levels of reproducibility versus performance/scalability.

The PRUNERS toolset offers four novel debugging and testing tools that can significantly assist programmers with detecting, remedying, and further preventing these errors in a coordinated fashion. The core objective of these tools is effectiveness in scalable detecting, controlling, and testing targeted, system-software-introduced sources of non-determinism. Our toolset specifically aims at the non-determinism introduced by the use of today's two most dominant parallel programming models, the Message Passing Interface (MPI) and OpenMP, as well as major compilers. The development and source code of these programming models and of compilers are outside the control of the programmers. Therefore, a non-deterministic error caused by their use is extremely difficult to diagnose without proper tools. This is sharply contrasted with application-level sources where programmers have explicit control over the application and visibility inside of the source code. The PRUNERS toolset meets our objective by being split into the following four interoperable individual tool components: Archer (Atzeni et al., 2016), ReMPI (Sato et al., 2015), NINJA (Sato et al., 2017), and FLIT (Sawaya et al., 2017).

To ensure wide applicability and flexibility, these components are designed to complement each other while each

individual component also works effectively as a single, stand-alone tool. PRUNERS provides a common multilevel toolset that unifies complementary and targeted tools that can build on the strengths of one another.

2.1. ARCHER

A data race condition is inarguably the most malignant form of parallel interaction among threads in OpenMP applications. This condition occurs when two or more threads can access shared data without proper synchronization and at least one of them modifies the data. The result can change depending on which thread wins this race. This leads to an undefined application behavior and thus introduces latent bugs.

Data races are not unique to OpenMP. On the contrary, it is one of the most common errors in many shared-memory programming models, and automatic detection of this condition has been widely studied. Nonetheless, efficient and scalable race checkers of OpenMP applications are scarce. Our previous study (Protze et al., 2014; Atzeni et al., 2016) shows that no existing solution offers the needed capability with requisite scalability, accuracy, and portability. Mature open-source data race detectors like Helgrind and ThreadSanitizer (TSan) (Serebryany and Iskhodzhanov, 2009) only target the low-level portable operating system interface (POSIX) threads and do not support high-level programming models such as OpenMP. Intel[®] Inspector INT (As of Feb 6th, 2018) has arguably the best-in-breed OpenMP support, but it still suffers from accuracy and overhead problems (Atzeni et al., 2016) in addition to portability problems due to hardware vendor lock-in.

Archer is a scalable and accurate OpenMP data-race detection tool (Figure 2). Archer's static analysis passes classify the given OpenMP code regions into two categories: guaranteed race-free and potentially racy. Its dynamic analysis then applies state-of-the-art data-race detection algorithms to check only the potentially racy OpenMP regions of code. The static/dynamic analysis combination is central to the scalability (while maintaining analysis precision) of Archer (Atzeni et al., 2016).

2.2. ReMPI

All data races are unsafe. Thus, accurately detecting the offending source-code sites would be sufficient to fix the bug. This is not the case for non-deterministic sources in

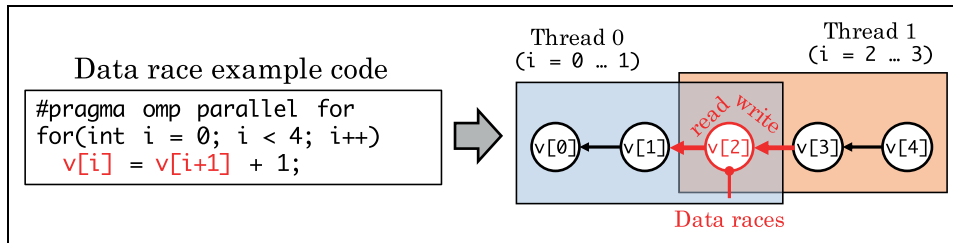


Figure 2. Archer: the same index of array v is read and written by multiple threads without mutual exclusion, that is, data races. Archer detect such racy access.

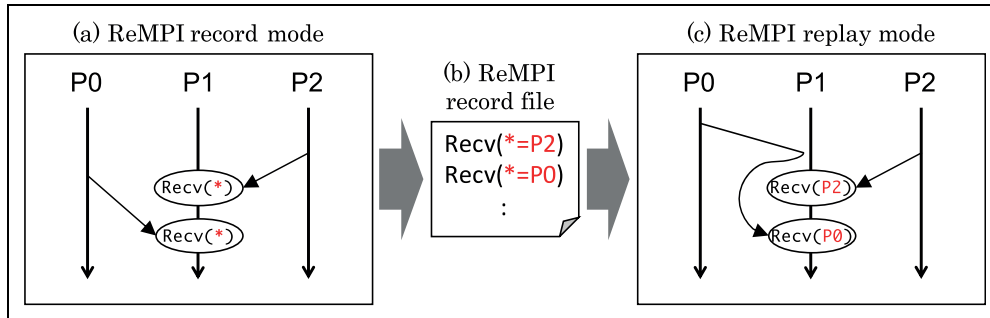


Figure 3. ReMPI: (a) ReMPI record mode) an MPI process (P1) issues two wildcard receives (Recv(*)) and receives a P2 message followed by a P0 message; (b) ReMPI record file) ReMPI records the order of these received messages; (c) ReMPI replay mode) instead of issuing the two Recv(*), ReMPI issues Recv(P2) followed by Recv(P0) based on the record to replay the exactly same message receives. Even if P0 message arrive first, ReMPI can pass P2 message first to P1.

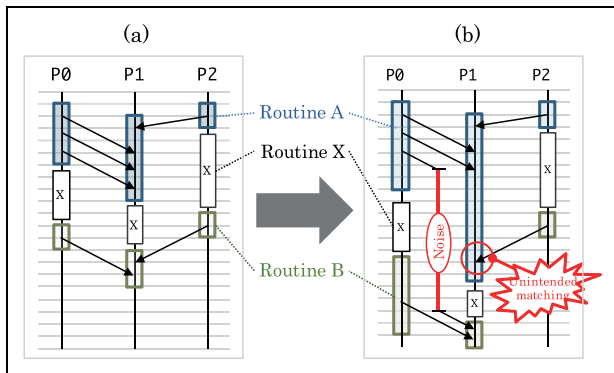


Figure 4. NINJA: (a: “Intended” message matching) Routine A and B (interleaved by Routine X) are unsafe communication routines, that is, Routines using the same MPI tag without any synchronization between them. Routine A and B are isolated by Routine X so that the message races rarely occurs. (b: “Unintended” message matching) NINJA injects delay to messages to expose such rarely-occurring message races.

MPI. By contrast, MPI explicitly allows non-deterministic message exchanges for high performance so that applications can wait for messages and process those that arrive in any order. Thus, an ability to detect where the non-deterministic MPI functions are invoked or called is insufficient in and of itself.

Instead, PRUNERS approaches MPI sources by explicitly and scalably controlling them. ReMPI (Sato et al., 2015) records the non-deterministic message exchanges in one record run and then controls subsequent runs to follow the

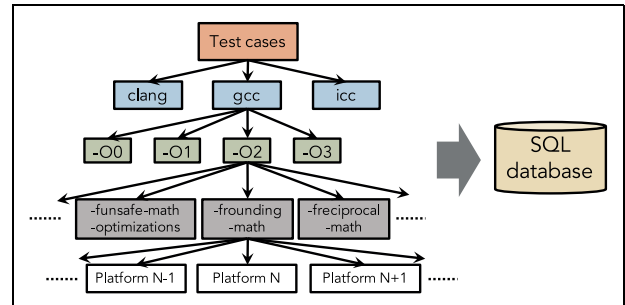


Figure 5. FLiT: A framework for testing floating-point arithmetic variability in different compilers, optimization flags, compiler options and platforms. These results are stored in SQL database for further analysis.

same message exchange sequence in the following replay runs (Figure 3). Under ReMPI’s control, programmers can deterministically replay MPI message exchanges, thereby, reproducing a target bug, even under the control of a parallel debugger, for further root-cause analysis. With a new compression algorithm called Clock Delta Compression (CDC), ReMPI enables scalable MPI record-and-replay (Sato et al., 2015).

2.3. NINJA

A common cause of non-deterministic bugs is unintended message races, a condition in which two or more “message

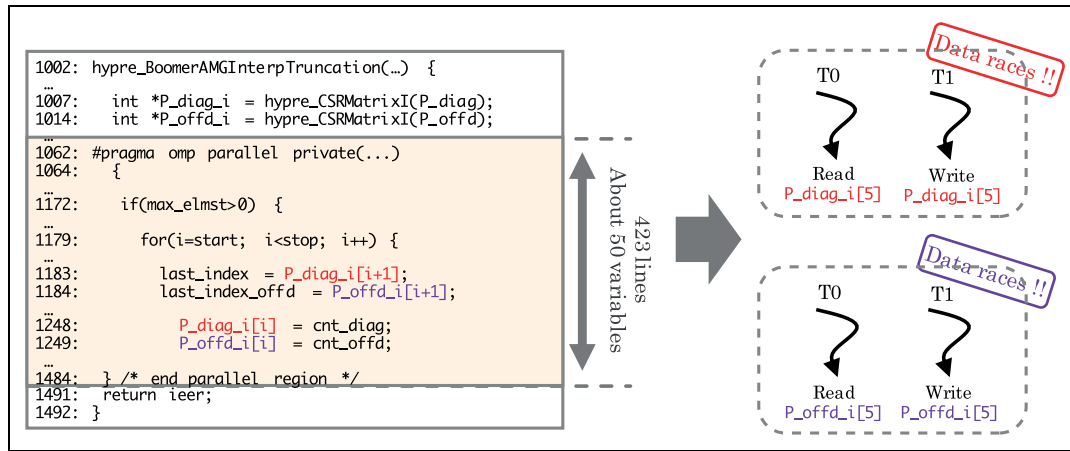


Figure 6. ARCHER identified the OpenMP data races causing HYDRA.

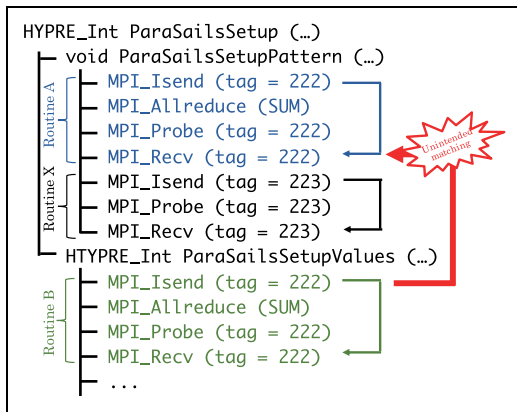


Figure 7. Real message-race code in Hypre 2.10.1 (depicting only MPI functions): Routine A and B are unsafe communication routines isolated by a different communication routine (Routine X). NINJA successfully manifested this real message races which are not uncovered without NINJA.

sends” race to match with a “message receive” and at least one of them is unintended. Unintended message races can significantly increase development cost because programmers must often run the application many times until the races finally manifest. In fact, observing the bug itself, especially under a debugging tool’s control, is often even more challenging than finding the root cause of the bug for unintended message races. Further, these tools typically exhibit noticeable runtime overhead, which distorts and thereby potentially masks message race bugs.

We developed innovative network noise injection techniques called NINJA to exposing unintended MPI message races (Figure 4). NINJA controls and manipulates the timings of non-deterministic message matches in ways that maximize the chance to expose MPI message-race bugs. With dynamic analysis of communication patterns of applications, NINJA can minimize the overhead by selectively injecting noise into only racy MPI message exchanges Sato et al. (2017).

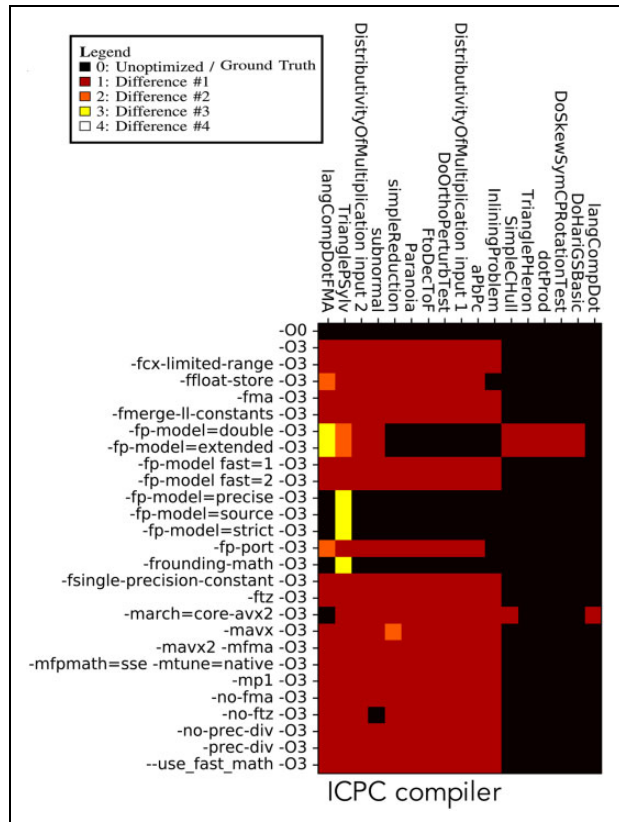


Figure 8. FLiT’s visualization of test divergences under Intel compiler (ICPC).

2.4. FLiT

One way in which a non-deterministic bug manifests itself as a simulation failure is through run-to-run numerical result variation. Floating Point (FP) arithmetic is not associative, that is, $a + (b + c) = (a + b) + c$, and the non-determinism in MPI and OpenMP can randomly change the order of FP arithmetic operations, which then makes certain failures occur only occasionally or also makes verification difficult. This is becoming more significantly complicated when code is ported across different
































	Features				Scalability	Detection Precision/Accuracy	Portability
	Deterministic Replay of MPI Application	Data Race Detection of OpenMP Threads	Ability to Expose Unsafe Message Races	Ease of Detecting Compiler-Induced FP Variability			
PRUNERS	 (Order Replay)	 *6			+ 10K Procs each w/ 10s Threads + 10s of Running Hours	 Strong	 Strong
TotalView Replay Engine*1	 (Data Replay*5)	 (Not Supported)			+ low 100s Procs each w/ 10s Threads + 1~2 of Running Hours	 (Not Supported)	 Weak
Single Process Reverse Debuggers*2	 (Data Replay)	 (Not Supported)			+ Single Process + 1~2 of Running Hours	 (Not Supported)	 Medium
Thread Sanitizer*3	 (Not Supported)		 (Not Supported)		 Strong	 Medium	 Strong
Intel® Inspector*4			 (Not Supported)		 Medium	 Medium	 Weak
Random delay testing	 (Not Supported)			 (Not Supported)	 Weak	 Weak	 Strong

Figure 9. Tool comparison table: (*1) Reverse debugging with ReplayEngine TVR (As of March 24th, 2018); (*2) GDB reverse debugging GDB (As of Feb 6th, 2018), rr RR (As of Feb 6th, 2018), UndoDB UND (As of Feb 6th, 2018); (*3) ThreadSanitize Serebryanov and Iskhodzhanov (2009); (*4) Intel inspector INT (As of Feb 6th, 2018); (*5) Data Replay complements PRUNERS. Users would first use NINJA and ReMPI recording to quickly and scalably obtain a record that deterministically reproduces an error. Subsequently, users can combine a ReMPI replay with a data replay tool such TotalView’s ReplayEngine for further root cause analysis. Within the MPI determinism guarantee provided by a ReMPI replay, users apply the data replay tool’s detailed interactive debugging features such as continue-forward and continue-backward on a small subset of processes; (*6) PRUNERS’ Archer extends ThreadSanitizer as its dynamic analysis infrastructure.

hardware platforms and/or optimized using different compilers and their options. While these can also change the FP arithmetic operation order, there are currently no available tools that can help a programmer assess the extent of FP variability. FLiT provides a powerful testing framework to tackle this well-known problem for which systematic solutions are lacking.

FLiT is a test framework for quickly revealing compiler-induced floating-point variability (Figure 5). FLiT provides a large collection of pre-designed tests and test automation facilities, including build scripts (i.e. Makefiles) generation, distribution of the tests on different platforms, and result amalgamation/query supported through database (i.e. SQL) queries. It can also display this compiler-, platform-induced floating-point variability using an intuitive 2-D heat-map style.

3. State-of-the-practice

To ensure that our capabilities work for large application development, we have actively engaged many production code-development teams at the Lawrence Livermore National Laboratory (LLNL) and tested the utility of our tools on large code bases. In this process, these users have challenged our research team with those bugs that they previously deemed intractable, and PRUNERS has consistently proven to be effective on them.

For example, Archer detected and helped fix highly elusive OpenMP data races in HYDRA, a very large multi-physics application for Livermore’s National Ignition Facility (NIF), that caused code crashes that only intermittently manifested themselves after varying numbers of time steps and only at large scales (8192 MPI processes or higher) on Sequoia (Figure 6). Further, as LLNL code

teams increasingly multi-thread their applications, they have begun to integrate Archer directly into their build-and-test systems to catch data-race bugs at testing time, before production runs are conducted.

When reducing novel techniques into production, portability and usability is significantly important for end users. ReMPI is indeed highly portable on any MPI implementations (e.g. MVAPICH, OpenMPI, and others) using a MPI profiling wrapper (PMPI), so that ReMPI can be easily applied to large-scale MPI applications even without recompiling applications by simply preloading the ReMPI shared library, *librempi.so*. ReMPI is also designed to work with other parallel debuggers, TotalView TV (As of Feb 6th, 2018) and DDT DDT (As of Feb 6th, 2018). These ReMPI's capability has been significantly helping ParaDiS (dislocation dynamics application) and Mercury (domain-decomposed particle transport application) debug and test MPI non-determinism.

NINJA has been shown to manifest unsafe message races consistently within LLNL Diablo's (a massively parallel implicit finite element application) use of Hypre-2.10.1 (a scalable linear solvers and multigrid method library). Hypre had a message-race bug that has not been uncovered until recently (Figure 7).

An easy-to-use and community-extensible tester like FLiT, which is designed specifically to capture compiler- and platform-induced FP variability, is also becoming increasingly important in scientific work dependent on supercomputers. Critical supercomputing applications such as the Community Earth Simulation Model, for instance, have yielded inconsistent results when ported across platforms and compilers, which hampered validation effort. Similarly, errors in floating-point calculations have led to inaccurate findings based on data analysis of Large Hadron Collider experiments. Our FLiT test suggests that even a single compiler has a large number of optimization options that can significantly affect the numerical results of FLiT's pre-designed "Litmus testers"(Figure 8). Our study has shown that there are more than 40 compiler options available across four major compilers used in high-end systems, which can affect the numerical simulation results. FLiT offers earlier warnings as to the portability of their code to different compilers.

4. Summary

Non-deterministic execution is becoming increasingly common and is particularly difficult for programmers to comprehend and debug. PRUNERS is the first coordinated toolset that is designed specifically to help debug and test for non-deterministic bugs, with features and attributes commensurate for large supercomputers. It was designed specifically with scalability, accuracy, and composability in mind. These features superior to existing debugging tools as shown in Figure 9. PRUNERS has demonstrated early success on real-world bugs and already resulted in cost savings at LLNL.

Acknowledgement

This work was performed under the auspices of the US Department of Energy by LLNL under contract DE-AC52-07NA27344 (LLNL-JRNL-747183).

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

References

- (As of Feb 6th, 2018) Allinea DDT (Distributed Debugging Tool). Allinea Software (Now Part of ARM). Available at: <https://www.allinea.com/products/ddt>.
- (As of Feb 6th, 2018) GDB and Reverse Debugging. GNU. Available at: <https://www.gnu.org/software/gdb/news/reversible.html>.
- (As of Feb 6th, 2018) Intel[®] Inspector 2017. Available at: <https://software.intel.com/en-us/intel-inspector-xe>.
- (As of Feb 6th, 2018) rr. Mozilla. Available at: <https://rr-project.org>.
- (As of Feb 6th, 2018) TotalView for HPC. Rogue Wave Software. Available at: <http://www.roguewave.com/products-services/totalview>.
- (As of Feb 6th, 2018) UndoDB. Undo. Available at: <http://undo.io/products/undodb/>.
- (As of March 24th, 2018) Reverse debugging with ReplayEngine. Rogue wave software.e. Available at: <http://www.roguewave.com/products-services/features/reverse-debugging>.
- Atzeni S, Gopalakrishnan G, Rakamaric Z, et al. (2016) ARCHER: effectively spotting data races in large OpenMP applications. In: *2016 IEEE international parallel and distributed processing symposium (IPDPS)*, pp. 53–62. DOI: 10.1109/IPDPS.2016.68.
- Protze J, Atzeni S, Ahn DH, et al. (2014) Towards providing low-overhead data race detection for large openmp applications. In: *2014 LLVM compiler infrastructure in HPC*, pp. 40–47. DOI: 10.1109/LLVM-HPC.2014.7.
- Sato K, Ahn DH, Laguna I, et al. (2015) Clock delta compression for scalable order-replay of non-deterministic parallel applications. In: *Proceedings of the international conference for high performance computing, networking, storage and analysis, SC '15*. New York, USA, 2015, p. 62. ACM. ISBN 978-1-4503-3723-6. DOI: 10.1145/2807591.2807642.
- Sato K, Ahn DH, Laguna I, et al. (2017) Noise injection techniques to expose subtle and unintended message races. In: *Proceedings of the 22Nd ACM SIGPLAN symposium on principles and practice of parallel programming, PPOPP '17*. New York, USA, 2017, pp. 89–101. ACM. ISBN 978-1-4503-4493-7. DOI: 10.1145/3018743.3018767.
- Sawaya G, Bentley M, Briggs I, et al. (2017) FLiT: cross-platform floating-point result-consistency tester and workload. In: *2017 IEEE international symposium on workload characterization (IISWC)*.

Serebryany K and Iskhodzhanov T (2009) Threadsanitizer: data race detection in practice. In: *Proceedings of the workshop on binary instrumentation and applications, WBI '09*. New York, USA, 2009, pp. 62–71. ACM. ISBN 978-1-60558-793-6. DOI:10.1145/1791194.1791203. Available at: <http://doi.acm.org/10.1145/1791194.1791203>.

Tassey G (2002) The economic impacts of inadequate infrastructure for software testing. Available at: <https://www.bibsonomy.org/bibtex/28a92e32577dcb1f2708367efb2fe406c/hlackner>.

Author biographies

Kento Sato is a team leader in the Center for Computational Science at RIKEN (RIKEN R-CCS). His research area is distributed systems and parallel computing, particularly in High Performance Computing (HPC). Major focuses of his research are artificial intelligence, machine learning and deep learning in HPC, application reproducibility (MPI reproducibility, and validation), scalable fault tolerance (scalable checkpoint/restart, fault tolerant MPI, Resilient system design), and I/O optimization (NVRAM, burst buffer, and big data), codesigning, and cloud computing.

Ignacio Laguna is computer scientist at the Center for Applied Scientific Computing at the Lawrence Livermore National Laboratory, California, USA. He develops methods to increase the reliability of HPC applications, including resilience techniques for soft errors and scalable debugging tools.

Gregory L Lee is a computer scientist in Livermore Computing's (LC's) Development Environment Group (DEG) at Lawrence Livermore National Laboratory (LLNL). He is currently the Development Environment User Support Lead. His particular focus areas include debuggers, the Python programming language, compilers, and math libraries. He is the primary developer of the Stack Trace Analysis Tool (STAT), a 2011 R&D100 award winner, and a collaborator on the PRUNERS project, a 2017 R&D 100 finalist.

Martin Schulz is a full professor at the Technical University of Munich, Germany and leads the chair for Computer Architecture and Parallel Systems. He is also a Member of the Directorate of the Leibniz Supercomputing Center (LRZ) of the Bavarian Academy of Sciences and Humanities. His research interests span all aspects of parallel systems, including parallel architectures, programming, tools, and system software.

Christopher M Chambreau supports and develops parallel performance analysis tools in Livermore Computing at Lawrence Livermore National Laboratory.

Simone Atzeni is a compiler runtime software engineer at NVIDIA. His main interests are in OpenMP and OpenACC

parallel programming. Before joining NVIDIA, he was a Computer Science PhD student at the University of Utah. During his PhD, he focused his research on data race detection tools and contributed to the PRUNERS Project as one of the main developer of Archer.

Michael Bentley is a PhD student in computer science at the University of Utah. He is advised by Professors Ganesh Gopalakrishnan and Tucker Hermans and works in collaboration with the Lawrence Livermore National Laboratory. He has focused his research on floating-point reproducibility of compilers and is the primary developer of FLiT as part of the PRUNERS Project.

Ganesh Gopalakrishnan is a professor of computer science in the School of Computing, University of Utah, Salt Lake City, USA. His main interests are in applying formal methods for debugging parallel programs. His recent work has covered data race checking, system resilience, and floating-point non-reproducibility.

Zvonimir Rakamaric is an associate professor in the School of Computing at the University of Utah, where he leads the Software Analysis Research Laboratory. His work aims to improve the reliability and resilience of complex software systems by empowering developers with practical tools and techniques for analysis of their artifacts. To achieve this, his research spans multiple areas, including formal verification, programming languages, software engineering, and security.

Geof Sawaya is a distributed graphics software engineer at Facebook Reality Labs, Redmond, WA. He focuses on distributed computing on many GPUs, in the areas of ray tracing and machine learning especially. He was the original developer of the FLiT tool, which detects floating point result and performance variations due to compiler configurations; this is a contribution to the PRUNERS toolset.

Joachim Protze is research staff at RWTH Aachen University, Germany. His main interests are parallel programming paradigms with a research focus on debugging of parallel applications. He is lead of the OpenMP tools subcommittee and pushes on the OpenMP tools interface to fit for runtime correctness analysis as in the Archer tool of the PRUNERS project.

Dong H Ahn is a computer scientist at the Development Environment Group (DEG) at LLNL. He leads the Next-Generation Computing Enablement (NGCE) project within ASC's Advanced Technology Development and Mitigation (ATDM) program, in which the PRUNERS Toolset has been researched and developed. His current interest includes scalable approaches to managing the adverse impacts of non-determinism in concurrent execution and to high performance computing resource and job management.