RETROSPECTIVE: BugNet: continuously recording program execution for deterministic replay debugging

Satish Narayanasamy
University of Michigan

Gilles Pokam
Intel

Brad Calder
Google

I. CONTEXT

BugNet was published in ISCA 2005. When we started this line of research in early to mid-2000, processor industry was at an inflection point. Performance was no longer the only consideration in processor design. Dennard scaling was breaking down, ushering in an era of multi-cores. The increasing complexity of software (e.g., Windows XP had 40-50 million lines of code) was already significantly straining programmer productivity. With the advent of multi-core processors, there arose an even more daunting challenge for most programmers: learning parallel programming, a task that was typically left to specialists. Consequently, there was a need within the computer science community to improve debugging as we transitioned to multi-core processors, given the programming challenges they presented.

In this context, we asked ourselves: what can processors do to help improve programmer productivity? Processors had performance counters to analyze performance issues, but lacked sufficient support for program correctness reasoning and debugging, except for a limited number of watchpoint and breakpoint registers.

Given that a significant fraction of a developer's time is spent on debugging, we then asked: what if we could reserve a small memory space, and log some useful information during a program's execution that could later help programmers debug in the event of a program failure? These questions led to further investigation that resulted in BugNet.

II. BUGNET - A MINIMALIST DESIGN

The concept of time travel debugging (TTD) was known then in the systems and software engineering community, but no production software offered that feature. We defined a variant of TTD which we referred to as Deterministic Replay Debugging (DRD). DRD requires an ability to continuously record all sources of non-determinism during a program's execution, so that the execution can be deterministically replayed to investigate a failure.

We can classify sources of an execution's non-determinism into two categories: input and races. While there were promising solutions for recording uniprocessor executions (e.g., ReVirt at Michigan), there were no feasible solutions for logging all (synchronization and data) races, which is necessary to reason about shared-memory multi-threaded program executions on a multi-core processor.

As we investigated this problem further, we discovered the work of Xu, Bodik and Hill [O24][1], who had just published a paper named Flight Data Recorder (FDR) that advocated for hardware-assisted multi-processor replay. FDR sought to replay the full-system. Their approach requires about six on-chip hardware buffers (one for every source of non-determinism such as system input, DMA, interrupts, races, etc.), mechanisms to transitively reduce coherence message logs, and a hardware-assisted full-system checkpointing [O21].

In retrospective, what distinguishes BugNet from FDR and many follow-up work on hardware-assisted replay including ours [6], is its minimalist design. We realized that if the goal is to help programmers debug their code, it is sufficient to enable replay of just the user-code, without the need to do full-system replay.

The revised problem statement led us to a fairly simple solution: checkpoint registers (similar to what processors already had for handling branch speculation), and then log a memory location's value when it is first read after the checkpoint. BugNet efficiently determines a first read by tracking a metadata bit per L1 cache block. This bit is reset when the cache block is fetched on a cache miss, and set after it is logged. This solution allowed us to easily handle all sources of input non-determinism by simply resetting these bits on a context switch (including interrupts and systems calls). The end result was a solution that was independent of operating system APIs – an essential property for a hardware-assisted solution.

Another important property of this solution is that it guaranteed deterministic replay of each thread in a multi-threaded program independently *without* recording its race logs. This was a huge win as this meant we did not have to wrestle with the coherence mechanism, which to this day is one of the most complex mechanism within a modern processor.

When debugging multi-threaded programs, it remains crucial to replay the synchronization and data races that occur between threads. This problem, however, can be solved fairly efficiently in software. To achieve this, we can record the order of synchronization operations by instrumenting these operations, which are relatively infrequent. In our software

[1] [O..] refers to bibliography in the original paper

implementation of BugNet [7], we discussed one solution that involves logging a sequencer containing a global timestamp whenever a thread performs a synchronization operation. Consequently, we can replay the threads based on the observed happens-before order due to synchronizations. To further reason about data-races in a deterministically replayed execution, we can explore complementary schedules [8] or employ an SMT-based schedule constraint solver [4].

One of the less well-known results of BugNet is its attempt at quantifying *crash latency*. For several software bugs we studied, we showed that it is sufficient to replay a tenth of a second that preceded the crash in order to debug it. This meant that reserving a small amount of memory space, perhaps just a fraction of an on-chip cache, is sufficient.

III. Influence on Industry Replay Tools

After BugNet was published, Harish Patil (Intel) reached out to us to understand if its replay solution can be realized in software using Intel's Pin dynamic instrumentation tool. Intel's challenge then was in studying user-level performance across different OS platforms using their architectural simulators. Cristiano Pereira joined this effort, and helped us realize PinSEL [7]. PinSEL would later become PinPlay and additional features for debugging such as DrDebug were added. PinPlay continues to be used today. It was recently included as part of Intel® Software Development Emulator (SDE).

Load-value logging, similar to BugNet, was also used in Microsoft's iDNA record-and-replay solution [1]. Using iDNA, Microsoft recently enabled time-travel debugging capability within the widely used WinDbg. Microsoft has also built an ecosystem of tools [2] to automatically find a wide range of bugs in executions replayed using iDNA. This includes a data-race analysis that we helped build as part of TruScan [8].

IV. Evolution of Replay Research

Since BugNet, there has been a notable surge in effort and interest to advance record-and-replay solutions within systems, hardware, and programming language communities.

Gilles Pokam joined Intel and led the implementation of hardware-assisted replay. They created QuickRec [10], a prototype built on Intel's FPGA emulation platform, QuickIA. QuickRec's design deviated from BugNet as it utilized coherence hardware to record memory races and modified the Linux kernel to capture input non-determinism. However, QuickRec presented challenges due to coherence tracking issues and the significant overhead of logging input in the Linux kernel. In a subsequent attempt, they developed LDC [9], a simpler design based on BugNet, which has since influenced the development of new data tracing features in Intel PT.

After joining Michigan, Satish Narayanasamy collaborated with Pete Chen, Jason Flinn, and several students, including Dongyoon Lee, to develop new software systems (Chimera [3], Respec [5], DoublePlay [11]) that enabled deterministic record-and-replay with less than 2x performance overhead. This represented a significant improvement of an order of magnitude compared to what was previously achievable without hardware support.

V. Looking Ahead

Time-travel debugging remains a potent tool. Alongside iDNA/WinDbg and PinPlay, several other tools have emerged that enable replay for web browsers, mobile applications, and more. However, these production tools either lack support for logging data races in multiprocessor executions or impose excessive performance costs, as is the case with PinPlay.

Newer hardware features like Intel PT have witnessed increasing adoption due to their ability to trace control flow. Extending it with a BugNet-like feature, which essentially involves logging L1 cache misses, may not entail all the complexities endured by earlier efforts like QuickRec. Such an extension can enable replay for multi-threaded programs with negligible performance costs.

While there has been limited success in harnessing irregular parallelism in traditional sequential applications, remarkable achievements have been made in leveraging data parallelism in modern workloads, such as machine learning. Tracing and debugging machine learning tasks is a promising area for future research, and the extent to which BugNet can contribute to that ecosystem remains to be determined.

References

[1] S. Bhansali, W.-K. Chen, S. de Jong, A. Edwards, R. Murray, M. Drinić, D. Mihočka, and J. Chau, "Framework for instruction-level tracing and analysis of program executions," in *The 2nd International Conference on Virtual Execution Environments (VEE)*, 2006, p. 154–163.

[2] P. Godefroid, "Micro execution," in *36th International Conference on Software Engineering ICSE*, 2014, pp. 539–549.

[3] D. Lee, P. M. Chen, J. Flinn, and S. Narayanasamy, "Chimera: hybrid program analysis for determinism," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2012.

[4] D. Lee, M. Said, S. Narayanasamy, Z. Yang, and C. Pereira, "Offline symbolic analysis for multi-processor execution replay," in *42st Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[5] D. Lee, B. Wester, K. Veeraraghavan, S. Narayanasamy, P. M. Chen, and J. Flinn, "Respec: efficient online multiprocessor replayvia speculation and external determinism," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[6] S. Narayanasamy, C. Pereira, and B. Calder, "Recording shared memory dependencies using strata," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[7] S. Narayanasamy, C. Pereira, H. Patil, R. Cohn, and B. Calder, "Automatic logging of operating system effects to guide application-level architecture simulation," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS/Performance*, 2006, pp. 216–227.

[8] S. Narayanasamy, Z. Wang, J. Tigani, A. Edwards, and B. Calder, "Automatically classifying benign and harmful data races using replay analysis," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.

[9] C. Pereira, G. Pokam, S. Hu, and B. Strong, "Instruction, circuits, and logic for data capture for software monitoring and debugging," in *US Patent*, 2016.

[10] G. Pokam, K. Danne, C. Pereira, R. Kassa, T. Kranich, S. Hu, J. E. Gottschlich, N. Honarmand, N. Dautenhahn, S. T. King, and J. Torrellas, "Quickrec: prototyping an intel architecture extension for record and replay of multithreaded programs," in *The 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 643–654.

[11] K. Veeraraghavan, D. Lee, B. Wester, J. Ouyang, P. M. Chen, J. Flinn, and S. Narayanasamy, "Doubleplay: parallelizing sequential logging and replay," in *16th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.