RETROSPECTIVE: Detailed Design and Evaluation of Redundant Multithreading Alternatives

Shubhendu S. Mukherjee[a], Michael Kontz[b], and Steven K. Reinhardt[c]

[a]SiFive, Inc. [b]Hewlett-Packard Enterprise [c]Microsoft

Context

This paper was part of a stream of work in the area of soft error analysis and tolerance techniques done by two of the co-authors (Mukherjee and Reinhardt) in the early 2000s. This effort started with an ISCA 2000 paper on redundant multithreading (RMT) [1] followed by this paper two years later [2]. A few other papers were published with additional collaborators [3,4], culminating in Mukherjee's book on the topic [5].

The original impetus for this collaboration came in 1998 after Mukherjee joined the Alpha architecture group at Compaq. Compaq had already bought Tandem, the famed manufacturer of fault-tolerant computers, the prior year. At that time, Tandem's primary hardware strategy used lockstepped pairs of off-the-shelf microprocessor devices to detect hardware faults in the CPUs. With both Tandem and Alpha under the Compaq umbrella, there was an effort to use Alpha processors in this lockstepped configuration.

Unfortunately, cycle-by-cycle lockstepping requires deterministic execution, which is difficult to achieve using large complex devices with features such as on-chip ECC-protected caches that occasionally require extra cycles for error correction. In addition, lockstepping has an inherently high resource overhead. Mukherjee recognized that simultaneous multithreading (SMT) [O27, O28],[1] a then-new technology being explored by the Alpha group at Compaq, provided an opportunity to introduce redundancy without these drawbacks.

Meanwhile, Reinhardt was an assistant professor in the EECS department at the University of Michigan, Ann Arbor. As part of his group's research on SMT processor design, he and his students (primarily Steve Raasch) had added SMT support to the SimpleScalar simulator [6]. (This work was a precursor to M5 [7] / gem5 [8].) Given their friendship as graduate students at the University of Wisconsin-Madison, Mukherjee approached Reinhardt to leverage the latter's SMT and simulator work, and together they began to explore and define the concepts underlying RMT, leading to their initial publication [1].

Mukherjee and Reinhardt then sought to push RMT towards productization within Compaq. The next logical step was to re-do the initial investigation in the context of a real industrial SMT processor design, in this case the Alpha EV8 which was under development at Compaq at that time. Reinhardt began consulting for Compaq and recruited a Michigan graduate student, Michael Kontz, to go to Compaq and work with Mukherjee as a summer intern. This paper is the result of that internship.

Redundant Multithreading Concepts

Most of the key concepts of RMT were introduced in our earlier paper [1].[2] RMT runs two identical instruction streams and compares the outputs of these streams to ensure both are executing correctly. Because an RMT processor implemented with SMT replicates the streams logically, rather than physically as in lockstepping, it is more difficult to identify which hardware structures are covered by the replicated instruction streams and which are not. The concepts we developed to help reason about these issues are now widely used across academia and industry.

Sphere of Replication. The sphere of replication represents a logical boundary. All activity and state within the sphere is replicated, in time and/or in space, providing fault detection coverage. Values that cross the boundary of the sphere of replication are the outputs and inputs that require comparison and replication, respectively. Structures outside the sphere do not gain any coverage from replication and must employ other fault coverage techniques (e.g., error codes). The size of the sphere (how much is replicated) provides a trade-off between performance, area, and fault coverage.

Input Replication. Input replication is the process by which inputs to the sphere are propagated to the redundant

[1] [O*n*] refers to reference [*n*] in the original paper.

[2] That paper is among the 50 most cited ISCA papers from the first 50 years of ISCA [9].

execution streams in such a way that the replicated streams continue to execute in an identical fashion (i.e., do not diverge). Different types of inputs (e.g., cached load data, uncached load data, interrupts) may require different replication techniques.

Output Comparison. Output comparison is the process by which outputs from the redundant execution streams inside the sphere are compared to verify their correctness. Equivalent outputs must be paired for comparison before they propagate outside the sphere, where unchecked results would cause potentially erroneous values to leak into the rest of the system. Typically, a larger sphere requires fewer output comparisons, but increases the latency before the error is caught, which may complicate error recovery.

Contributions of This Paper

At a high level, this paper is interesting as an example of how a largely academic concept and evaluation [1] translates to a detailed commercial design. While the EV8 was sadly never produced, with or without RMT, the environment in which this work was performed was aimed at productization and staffed not by academics and graduate students but by engineers who had the experience of multiple generations of successful products.

We found that the concepts from the original paper carried over well, but many of the details had to be adjusted or even completely redone to apply in a different and more detailed base design. For example, our original work assumed a FIFO queue for load value replication, but maintaining out-of-order load execution in the trailing thread turned out to be important for performance, so this structure had to be redesigned to support multiple associative probes using a correlation tag. At the extreme, our simple concept of forwarding branch targets between threads had to be completely redesigned to work with the instruction cache line prediction scheme that drove the EV8 fetch stage. As far as performance, we found that while our initial study indicated that a unified 64-entry store queue was adequate for both threads, the EV8 design required per-thread store queues to avoid unacceptable stalls. Even then, the simulated overhead of running a redundant thread increased from 21% to 30%.

The more robust EV8 modeling infrastructure also allowed us to expand the scope of our studies. Our earlier paper looked at only one design point, a single-core SMT processor replicating a single logical execution thread using two hardware thread contexts, a design we termed Simultaneous Redundant Threading (SRT). Here we also looked at the impact of running two logical threads on four hardware contexts. More significantly, we also looked at the option of using a dual-core chip multiprocessor (as the EV8 was designed to be) to cross-couple two pairs of redundant threads across cores, an option we termed Chip-level Redundant Threading (CRT). This paper also coined the term RMT as an umbrella description covering all these techniques.

Impact and Future Directions

In 2001, Intel acquired the Alpha architecture division of Compaq. Intel sought more cost-effective alternatives for soft error protection for the Xeon processor line. This led to the development of the architecture vulnerability factor (AVF) methodology [3, 4] to help determine the relative need for fault coverage across microarchitectural structures.

The SRT/RMT papers along with the AVF work inspired some follow-on research. Some researchers have extended RMT to include fault recovery [O29]. RMT can be exposed to software control, allowing redundancy to be switched on and off as needed, based on application resiliency requirements and/or AVF analysis [10].

Although we are not aware of a commercial RMT implementation yet, we believe that the flexibility of RMT may motivate its use in new areas. For example, the need to meet differing reliability standards for automotive subsystems, as defined by ISO's ASIL (Automotive Safety Integrity Level) standards, might benefit from the lower overhead and dynamically controllable nature of RMT.

References

[1] S. K. Reinhardt and S. S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," Proc. 27th Int'l Symp. on Computer Architecture (ISCA), June 2000.

[2] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed Design and Implementation of Redundant Multithreading Alternatives," Proc. 29th Int'l Symposium on Computer Architecture (ISCA), May 2002.

[3] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," 36th Annual International Symposium on Microarchitecture (MICRO), December 2003.

[4] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," Proc. 31st Int'l Symposium on Computer Architecture (ISCA), June 2004.

[5] S. S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann, 2008.

[6] D. A. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report #1342, University of Wisconsin-Madison Computer Sciences Department, June 1997.

[7] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, 26(4):52-60, July-Aug. 2006.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," ACM SIGARCH Computer Architecture News, 39(2), May 2011.

[9] G. Upasani, M. Sinclair, A. Sampson, P. Ranganathan, D. Patterson, S. Shah, N. Parthasarathy, and R. Jain, "Fifty Years of ISCA: A data-driven retrospective on key trends," arXiv:2306.03964, June 2023.

[10] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, S. S. Mukherjee, "Software-Controlled Fault Tolerance," ACM Transactions on Architecture and Code Optimization, 2(4):366–396, December 2005.