Profiling benchmarks to characterize the failures for approximate memory

Anish Tondwalkar Haina Li
University of Virginia
{tondwalkar,hainali}@virginia.edu

1. Introduction

Traditional techniques for providing memory reliability have a one-size-fits-all approach to error correction, which is inefficient and costly because all memory are treated the same way. Recent research has found that not all data are equal, and that applications, depending on the nature of their computation, can have a high tolerance for data errors [5]. For example, Li and Yeung's work [3] on application-level correctness investigated definitions of program correctness based on the application user's point of view instead of that of the system or architecture. They found that all the programs they studied exhibit some fault resilience when you only measure application correctness based on the user-perceived output. Such new approaches aims to trade DRAM reliability for power or performance, providing memory protection only for error-sensitive data.

To help decide with types of data are more fault-resistant than others, this project will focus on characterizing memory failures in both precise and approximate applications by mapping memory access patterns and locations of failure. Granular knowledge of memory access patterns will help reveal how data corruption affects data types and and data structures.

2. Problem Statement

Current works have shown that it is possible to trade off DRAM reliability for power or performance [8, 4, 5]. Some applications can tolerate failures and consequently, can leverage approximate memory to improve system power and performance [5, 3]. In this work, we will examine both precise and approximate applications and characterize the behavior of memory failures on different data types, data structures, locations, and memory access patterns.

We characterize the memory soft-error recoverability of regions of applications so that it might be moved onto less reliable but cheaper and more efficient banks of memory. We use a fault injection and program monitoring framework to inject random single- and multi-bit flip errors, while we log the response of the program — Did it execute correctly and without crashing? — and the region in which we injected the fault.

3. Related work

Characterizing Application Memory Error Vulnerability
The work closest to ours is [5], in which the authors perform a preliminary profiling of application memory error vulnerability. This is based on the work of [3] which showed that application-level correctness was useful property. Luo et al. [5] find exploitable differences in memory error recoverability (both hard and soft) between heap, stack and private memory. We intend to focus on soft errors and extend this to dependence on memory access patterns, data structures, and data types.

Saving DRAM Refresh-power using Hetrogenous RAM
Several papers [4, 8] have noted that differences in memory error recoverability in different parts of applications allow one to partition data in a way that critical data is stored on more reliable memory, while much can be offloaded to less reliable DRAM. These rely on programmer annotations, typically obtained by manually profiling, to note which parts of the program have greater error recoverability. They significantly increase energy efficiency at low cost to application-level reliability, but at significant cost in programmer time.

Software Recovery of Hardware Faults
We primarily discuss data reliability from the software's point of view. We can do this, because significant research [9, 2, 10] has been successful in handing off hardware soft error recovery to the software, allowing it to control the recovery process. Methods that make software more resistant to memory failure, and therefore improves its soft error recoverability have been very successful [10], which bodes well for our work. de Kruijf et al [2] argues that this new approach to memory fault simplifies hardware design and helps technology scaling, achieving a 20% energy efficiency improvement with minimal changes to the code and hardware design in their architectural framework for software recovery of hardware faults. This tries to solve a similar problem to ours, using hardware-software-co-design to increase efficiency, but this still work relies on explicit programmer annotation.

4. Methodology

We will first locate regions of memory failure by examining application memory access patterns and the location of memory failure. Starting with the approach used by Luo et al [5], we will define an address A's *unsafe duration* as the sum of time across an application's execution time between each *read* and previous memory reference to A. Similarly, An address A's *safe duration* is the sum of an application's execution time between each write and previous memory reference to A. The *safe ratio* is the *safe duration/(safe duration + unsafe duration)*.

A *safe ratio* close to 1 means that there's more chance that an error at A will be masked, as it's more frequently written than read. Following the same logic, a *safe ratio* close to 0 means that an error at A will be consumed, as it is more frequently read than written. With this approach, we could clearly identify safe regions of memory by computing the *average safe ratio* of the region's memory address.

Armed with the memory access patterns and the location of memory failures, we could next characterize how these failures are affecting data types and data structures in both precise and approximate applications.

We intend to perform these experiments using `gdb` to inject errors. We emulate single- and multi- bit soft errors. Since modern DRAM can suffer from arbitrary bit flips [6], we simply replace randomly picked bits with their complement. We record the location of the injected fault, and the location at which the fault is exhibited, if any. We proceed as follows:

1. We start the application in the error injection and monitoring framework.
2. We inject errors and note their symbolic location.

3. We run the workload of interest — connecting clients to server applications.
4. If the application crashes, we record the location of the crash, and begin anew.
5. Otherwise, we compare the output of the computation with the expected result and note it.
6. We begin anew, restarting the program and logging fault injection locations, crash locations, and errors until the experiment is over.

We continue testing until a significant number of errors has occurred in each of our regions of interest.

5. Plan

5.1. Steps

5.1.1. Milestone 1 - March 22, 2016 .

By the first milestone we should have found a set of programs that provide us with a diverse range of applications, and that we should be able to use to characterize the memory soft error recoverability of each type of region according to data structures used, it's location in the program or program memory, and its access pattern.

We should be able to characterize specific regions in these programs, and should be able to provide a preliminary analysis of what kinds of program, memory structures, and access patterns we will look for with greater specificity.

5.1.2. Milestone 2 - April 12, 2016 .

By the second milestone we will have completely characterized the regions of interest in each of our benchmark programs and will have begun the experiments.

By this point we should have identified the areas in which we expect to see the biggest differences, and have completed a small set of pilot experiments that tell us which characteristics are likely to give us significant results.

We should have also began to gather and analyze the data which will form our main result.

5.1.3. Report - May 6, 2016 .

By the report, we will have finished the experiments, gathered and analyzed the data, and have come to conclusions as to the differences in soft error recoverability between regions due to their singled-out characteristics.

If we have the time, we will also build tools to help statically and dynamically identify these characteristics, so program memory regions can be moved between heterogeneous blocks of physical memory based on those characteristics.

5.2. Goals

5.2.1. Finished Product - May 6, 2016 .

We aim to characterize the memory error recoverability of regions of a large set of test programs, based on data structures, data types, location, and memory access patterns. We'll have classified the regions of interests, have performed experiments that map injected hardware soft memory errors onto software recovery, application-level error, or crash, and characterize each region.

If we have more time we want to proceed to build a tool to provide annotations without programmer intervention, and possibly one to dynamically move more recoverable structures to less reliable DRAM and more fragile structures into a more expensive, more reliable DRAM, say, with ECC.

5.2.2. Venue .

Completion of this project, including the more ambitious goals, would result in a significant contribution that may be publishable at a conference. We note that the central thrust of this project seems well-adapted for submission to the Special Interest Group on Measurement and Evaluation (SIGMETRICS), but also suitable for the Workshop on Energy-Efficient Design (WEED), or, especially with extensions, to the International Symposium on Computer Architecture (ISCA) or Architectural Support for Programming Languages and Operating Systems (ASPLOS) if we develop a tool.

References

[1] R. A. Ashraf *et al.*, "Understanding the propagation of transient errors in HPC applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 72. Available: http://dl.acm.org/citation.cfm?id=2807670

[2] M. de Kruijf, S. Nomura, and K. Sankaralingam, "Relax: An Architectural Framework for Software Recovery of Hardware Faults," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 497–508. Available: http://doi.acm.org/10.1145/1815961.1816026

[3] X. Li and D. Yeung, "Application-Level Correctness and its Impact on Fault Tolerance," in *IEEE 13th International Symposium on High Performance Computer Architecture, 2007. HPCA 2007*, Feb. 2007, pp. 181–192.

[4] S. Liu *et al.*, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: ACM, 2011, pp. 213–224. Available: http://doi.acm.org/10.1145/1950365.1950391

[5] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2014, pp. 467–478.

[6] T. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *Electron Devices, IEEE Transactions on*, vol. 26, no. 1, pp. 2–9, Jan 1979.

[7] S. Nomura *et al.*, "Sampling + DMR: Practical and Low-overhead Permanent Fault Detection," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 201–212. Available: http://doi.acm.org/10.1145/2000064.2000089

[8] A. Sampson *et al.*, "EnerJ: Approximate Data Types for Safe and General Low-power Computation," in *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11. New York, NY, USA: ACM, 2011, pp. 164–174. Available: http://doi.acm.org/10.1145/1993498.1993518

[9] H. Schirmeier *et al.*, "Fail*: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance," in *Dependable Computing Conference (EDCC), 2015 Eleventh European*, Sept 2015, pp. 245–255.

[10] A. Sharma *et al.*, "Towards analyzing and improving robustness of software applications to intermittent and permanent faults in hardware," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, Oct 2013, pp. 435–438.

[11] L. Tan, Z. Chen, and S. L. Song, "Scalable Energy Efficiency with Resilience for High Performance Computing Systems: A Quantitative Methodology," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, no. 4, p. 35, 2015. Available: http://dl.acm.org/citation.cfm?id=2822893

[12] L. Yu *et al.*, "Quantitatively Modeling Application Resilience with the Data Vulnerability Factor," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 695–706. Available: http://dx.doi.org/10.1109/SC.2014.62