Exploring a Brink-of-Failure Memory Controller to Design an Approximate Memory System

Meysam Taassori   Niladrish Chatterjee[†]   Ali Shafiee   Rajeev Balasubramonian
University of Utah, [†] NVIDIA

Abstract

Nearly every synchronous digital circuit today is designed with timing margins. These timing margins allow the circuit to behave correctly in spite of parameter variations, voltage noise, temperature fluctuations, etc. Given that the memory system is a critical bottleneck in several workloads, this paper attempts to safely push memory performance to its limits by dynamically shaving the timing margins inherent in memory devices. This is implemented with an adaptive memory controller that maintains timing parameters for every bank and gradually pushes the memory system towards the brink of failure. Each bank may be handled differently, depending on the extent of parameter variation. Such an approach may lead to occasional run-time errors. Additional support for ECC or chipkill may help the memory system recover from errors that are introduced by an overly aggressive memory controller. This is a much stronger capability than the limited form of memory over-clocking that can be deployed today. We believe that such a brink-of-failure memory controller can form the basis for an approximate memory system. Memory timing parameters can be easily adapted per memory region or per memory operation, enabling easy tuning of the performance-precision trade-off for approximate computing workloads. The preliminary analysis in this paper serves as a limit study to understand the impact of memory timing parameters on application throughput.

1 Introduction

Commercial computer systems are designed to provide reasonably high levels of reliability. The commercial viability of a system that crashes every day is zero. Hence, systems are over-provisioned in many regards so they are not operating on the brink of failure. For example, power supplies on a board or power delivery networks on a chip can handle a little more than the maximum expected power draw. In a similar vein, nearly every synchronous digital circuit on a processor or memory chip today is designed with timing margins. These timing margins allow the circuit to behave correctly in spite of parameter variations, voltage noise, temperature fluctuations, etc.

There is however one niche market segment that operates the system near the brink of failure to eke out very high performance. Gaming enthusiasts frequently resort to processor and memory "over-clocking" [20, 40, 52]. Processor and memory vendors expose a few parameters that can be set at boot-up time in the BIOS to allow a system to operate at frequencies higher than those in the specifications. For example, memory over-clocking today allows a change to the memory bus frequency, the DIMM voltage, and three DRAM timing parameters (tRP, tRCD, tCL) [52]. This is an effective coarse-grained approach to shrink timing margins and boost performance, while trading off some reliability.

In this work, we attempt to bring the *brink-of-failure (BOF)* approach to mainstream computer systems in a safe and architecturally controlled manner, with a primary focus on the main memory system. We propose an adaptive memory controller that can use more aggressive timing parameters for various regions of memory or for specific memory instructions. The memory system can also be augmented with some form of error detection/correction support to track error rates and recover from errors when possible. A more aggressive memory controller yields higher performance while introducing a few errors. The timing parameters are adjusted based on observed error rates and application requirements. This enables fine-grained control of the memory system and the performance-precision trade-off.

The proposed BOF memory controller has two primary applications. It serves as an important component in an approximate computing system [44]. It also helps extract the highest performance possible from memory systems that suffer from large parameter variations [29].

This project is at a very early stage. As a first step, this paper quantifies the impact of various timing parameters on application performance, thus showing the potential room for improvement. We believe that this is an important area for future work, requiring support from the hardware, operating system, programming models, and applications to fully realize its potential.

2 Background

In modern server systems, a single processor socket has up to four memory controllers. Each memory controller drives a single DDR (double data rate) memory channel that is connected to one or two DIMMs. DRAM chips on a DIMM are organized into ranks; memory controller commands are sent to all DRAM chips in one rank and all the DRAM chips in a rank together provide the requested data.

DRAM Timing Parameters

A DRAM chip has little logic or intelligence; for the most part, it simply responds to the commands received from the memory controller. The memory controller has to keep track of DRAM state and issue commands at the right times.

Memory is designed to be an upgradeable commodity. When a DIMM fails or when more memory capacity is required, the user can pull out a DIMM from the motherboard and replace it with another new DIMM. A single memory controller must work correctly with all possible DIMMs. JEDEC is an industry consortium that specifies a number of standard timing parameters (e.g., tRC, tRP, tRFC) that govern every memory device. These specifications are referred to as the JEDEC standard. Processor companies then design memory controllers that manage each of these timing parameters. When a DIMM is plugged in, the memory controller reads in the values for each timing parameter from the DIMM. These values are then used to appropriately schedule commands to the DIMM. For example, the JEDEC standard specifies that tRFC is the time taken to perform one refresh operation. After issuing a refresh operation, a memory controller is designed to leave that memory device alone for an amount of time equaling tRFC. Some DIMMs have a tRFC of 160 ns, some have a tRFC of 300 ns; this is determined by reading a register on the DIMM at boot-up time. There are more than 20 such timing parameters.

The JEDEC standard has been defined to facilitate easy adoption by all memory and processor vendors. It therefore errs on the side of being simple, while potentially sacrificing some performance. For the refresh example above, in reality, the memory controller can safely schedule some memory operations as the refresh operation is winding down, i.e., the memory controller can resume limited operation before the end of tRFC. JEDEC could have specified a number of additional timing parameters to capture this phenomenon and boost performance. But putting this in the JEDEC standard could complicate every memory controller, even the simplest ones on embedded devices.

DRAM Commands and Microarchitecture

The memory system offers a high degree of parallelism. Each channel can perform a cache line transfer in parallel. Each DRAM chip is itself partitioned into eight independent banks. Therefore, the collection of DRAM chips that form a rank can concurrently handle eight different operations. To fetch data from a bank, the memory controller first issues an Activate operation – this fetches an entire row of data into the bank's row buffer (a collection of sense-amps). The memory controller then issues a Column-Read command to fetch a 64-byte cache line from the row buffer. The memory controller can also fetch other cache lines in the row buffer with additional Column-Read commands; these low-latency operations are referred to as row buffer hits. Before accessing a different row of data in the same bank, the memory controller has to first issue a Precharge command that clears the row buffer and readies the bitlines for the next Activate command.

Memory Controllers

When there's a miss in the processor's last level cache, memory transactions are queued at the memory controller. The memory controller maintains separate read and write queues. Reads are given higher priority. Writes are drained when the write queue size exceeds a high water mark. The memory controller has a sophisticated scheduler that re-orders the transactions in the queue to improve response times and throughput. The scheduler attempts to maximize row buffer hits and bank utilizations, hide refreshes and writes, and prioritize high-throughput threads while maintaining fairness. Once a request is selected, it is decomposed into the necessary Precharge, Activate, Column-Read sequence and the commands are issued as long as no timing parameter is violated.

Memory Reliability

Most server memory systems include support to detect and recover from hard and soft errors in memory. Such servers typically use ECC-DIMMs, where the DIMM is provisioned with additional DRAM chips that store error correction codes. The most common error correction code is SEC-DED (single error correct, double error detect). SEC-DED adds an eight bit code to every 64-bit data word that enables recovery from a single bit failure in that word. To handle failure of multiple bits in a 64-bit data word, stronger codes are required. One example is chipkill, that allows recovery from complete failure in one DRAM chip. Early chipkill implementations [1, 15, 37] incurred high overheads. More recently, new chipkill algorithms and data layouts have been developed that greatly reduce these overheads [24, 25, 53, 56].

Parameter Variation

It is well-known that parameter variation grows as device dimensions shrink [39]. Over the last decade, a number of papers have proposed techniques to cope with parameter variations in the processor and SRAM caches (e.g., [4, 7, 13, 17, 32, 34, 51]). Various schemes have been used to detect at run-time that timing margins in processors have shrunk and that voltage/frequency adjustments are required. For example, Razor [17] uses a delayed latch, Lefurgy et al. [32] monitor critical paths, and Bacha and Teodorescu [7] use functional unit ECCs to detect timing violations. Few papers have examined parameter variation in DRAM even though it has been identified as a significant problem [6, 16, 21, 27, 29, 33, 36, 57].

Parameter variations in logic and DRAM processes are typically attributed to changes in the effective gate length that are caused by systematic lithographic aberrations, and changes in threshold voltage that are caused by random doping fluctuations [6, 39, 57]. Most prior works on DRAM parameter variation have focused on how it impacts cell retention time [21, 27, 33, 36]. Correspondingly, techniques have been proposed to refresh different rows at different rates [35, 36]. Wilkerson et al. [55] lower the refresh rate in eDRAM caches and overcome errors in weak cells with strong error correction codes. The work of Zhao et al. [57] develops a parameter variation model for 3D-stacked DRAM chips and quantifies the expected variation in leakage and latency in different banks. They show that parameter variation can cause most banks to have data read latencies that fall in the range of 12-26 cycles. The authors then propose a non-uniform latency 3D-stacked DRAM cache. With the variation profile described above, a commodity DRAM chip today would have to specify a conservative uniform latency of 30 cycles even though several requests can be serviced in half the time. Another study with an older DRAM process shows that latencies in different banks can vary by 18 ns because of parameter variation [30]. A more recent paper from Hynix shows that for 450 DRAM samples, the delay variation for circuits within a single wafer is almost 30% [29]. The delay variation grows as voltages shrink, implying that future technologies will likely see more variations.

The above data points argue for more intelligent memory controllers that can automatically detect and exploit variations in DRAM timing parameters. In addition to the manufacturing variations described above, there are other sources of predictable and unpredictable variations at runtime because of the operating environment. Timing parameters vary predictably as temperature changes. The timing for some operations may vary unpredictably because of voltage supply noise. Some voltage fluctuations are predictable, either caused by static IR-drop [48] or dynamic LdI/dt [26]. The timing for every operation can also vary based on other simultaneous activities on the DRAM chip because some critical resources are shared (most notably, charge pumps and the power delivery network [48]).

Approximate Computing

In recent years, multiple papers [8–10, 14, 18, 31, 38, 42–45, 49, 50, 54] have made the argument that applications in certain domains like media/image processing, computer vision, machine learning, etc. can tolerate a small degree of imprecision. Typically, imprecision can be tolerated in some data structures, but not in the control logic [8, 18, 44]. The underlying hardware exploits the energy/precision or performance/precision trade-off, although, most prior work has exploited the former. A recent example, Truffle [18], splits the processor into an instruction control plane and a data movement/processing plane. The former has to be precise, while the latter can be selectively imprecise depending on the instruction type. A compiler (e.g., EnerJ [44]) is expected to designate instructions as being approximate or not. The processor operates at a single frequency, but the data movement/processing plane can operate at a high or low voltage. At low voltage, the pipeline consumes significantly lower energy, but may yield occasional errors because the circuits are slower and may not meet the cycle time deadlines.

While prior work has examined the microarchitecture for approximate processing and SRAM caches [18], support for approximate memory is more limited. Flikker [50] relies on the application to identify memory pages that can tolerate imprecision; such pages are refreshed less frequently to save energy. A more recent paper by Sampson et al. [45] assumes an MLC PCM main memory and selectively uses a less precise write process or allocates approximate data to rows that are riddled with hard errors. It thus trades off precision for faster writes and better endurance. Apart from Flikker [50], no prior work has examined an approximate DRAM memory system by varying DRAM timing parameters.

3 Proposal

The previous section provides background on memory systems, parameter variation, and approximate computing. For many reasons, we believe that the time is right to develop sophisticated memory controllers that can shave timing margins in an architecturally controlled manner:

1. Parameter variations will increase as process technologies and voltages shrink [29, 39]. While traditional over-clocking is limited by the slowest circuit on the chip, architectural support can exploit the higher speeds for several circuits on the chip.

2. Timing variations in the memory system can be exploited with relatively simple updates to timing parameters in the memory controller.

3. Recent advancements in chipkill algorithms [24, 25, 53, 56] have lowered their overheads and made them a viable addition to a brink-of-failure system.

4. Most recent papers in approximate computing have focused on approximate processors. The work in this project will help define the missing piece – the approximate main memory subsystem.

5. The memory wall continues to escalate – processor pin counts are not increasing [22] and workloads are accessing growing in-memory datasets [2, 41, 46, 47].

We next define a few essential pieces that will together form the BOF memory system.

Organizing a Rank. Every DRAM chip on a DIMM is expected to have similar worst-case timing parameters. However, a number of banks within each DRAM chip may be able to operate faster. The latency to fetch a cache line from bank-3 is determined by the latency of the slowest bank-3 among the DRAM chips that form a rank. This requires that ranks be created in a manner that is aware of parameter variations on each chip. A chip with a fast bank-3 should be ganged with other DRAM chips with fast bank-3 to form a rank. Building a rank with few DRAM chips helps reduce the negative impact of a slow chip. This argues for using wider-IO DRAM chips or mini-ranks [5, 58]. Run-time re-organization of DRAM chips into ranks may also be possible with a smart buffer chip on the DIMM. The above techniques will help create a memory system that exhibits high non-uniformity – some banks that operate at the typical specified speed and many other banks that can operate at faster speeds.

BOF Memory Controller. The next step is to design a memory controller that can track many sets of timing parameters. A memory controller maintains transaction queues for pending read and write operations. A complex scheduling algorithm is used to select operations from these read and write transaction queues (e.g., TCM [28]). The selected transaction is then expanded into smaller commands (e.g., Precharge, Activate, Column-Read) that are placed in per-bank command queues. The command at the head of the command queue is issued when it fulfils a number of timing constraints. The primary change in the BOF memory controller is that it maintains a separate set of timing parameters for each bank. At run time, error rates are tracked for every execution epoch (say, 100 million cycles). If the error rate for a given bank is lower than some threshold, the timing parameters are made more aggressive for the next epoch. If the error rate exceeds the threshold, the timing parameters are made less aggressive. Some hysteresis can also be provided to avoid oscillations.

Programming Models. Different thresholds can be used for each bank, allowing each memory region to provide a different point on the performance-precision trade-off curve. These thresholds are exposed to the OS so that the OS can map pages appropriately. For example, OS pages are mapped to the most precise memory regions, while pages designated as being approximate by the application are mapped to the less precise regions. The application is also given the ability to set thresholds for memory regions used by the application. Further, when an approximate load instruction is being issued by the memory controller, it can choose to be even more aggressive than the parameters listed in its table. The error rates for a bank are exposed to the application so that it can choose to throttle the bank up/down or move a page to a better home. As part of future work, we will explore programming models that can exploit such hardware capabilities and such feedback from the hardware.

Error Correction Support. Every DRAM chip is different because of parameter variations. As the memory system is pushed closer to the brink of failure, one DRAM chip in the rank is likely to yield errors before the rest. Therefore, chipkill support can help recover from a large fraction of BOF-induced errors [53]. Udipi et al. [53] show that the LOT-ECC chipkill mechanism introduces relatively low overheads in terms of performance and power (under 5%, relative to a memory system with basic SEC-DED support). Chipkill support is only required if the memory system is trying to provide high performance *and* high precision. For an approximate memory system or the approximate regions of memory, even basic SEC-DED support is enough to detect most errors and provide feedback to the application or OS. SEC-DED codes pose no performance overhead (since code words are read in parallel with data words), but introduce a storage and power overhead of 12.5% in the common case. In several domains, SEC-DED support is a requirement, so this may not represent a new overhead. When uncorrectable errors are encountered, the approximate application is allowed to proceed even though a subset of data bits are corrupted. Applications must therefore be designed to tolerate high imprecision in a few data values.

4 Methodology

The project is in its early stages. Our evaluation will ultimately require an empirical analysis of timing margins in actual DRAM chips. In this paper, we simply show the potential for improvement as DRAM timing parameters are varied.

For our simulations, we use Windriver Simics [3] interfaced with the USIMM memory simulator [11]. USIMM is configured to model a DDR4 memory system (bank groups, DDR4 timing parameters, etc.). Simics is used to model eight out-of-order processor cores. These eight cores share a single memory channel with two ranks. Simics and USIMM parameters are summarized in Table 1. Our default scheduler prioritizes row buffer hits, closes a row if there aren't any pending requests to that row, and uses write queue water marks to drain many writes at once [12].

Processor	
ISA	UltraSPARC III ISA
CMP size and Core Freq.	8-core, 3.2 GHz
ROB size per core	64 entry
Fetch, Dispatch, Execute, and Retire	Maximum 4 per cycle
Cache Hierarchy	
L1 I-cache	32KB/2-way, private, 1-cycle
L1 D-cache	32KB/2-way, private, 1-cycle
L2 Cache Coherence Protocol	4MB/64B/8-way, shared, 10-cycle Snooping MESI
DRAM Parameters	
DRAM Frequency	1600 Mbps
Channels, ranks, banks	1 channel, 2 ranks/channel, 16 banks/rank
Write queue water marks Read Q Length	40 (high) and 20 (low), for each channel 32 per channel
DRAM chips	32 Gb capacity
DRAM Timing Parameters (DRAM cycles)	t_{RC} = 39, t_{RCD} = 11 t_{RAS} = 28, t_{FAW} = 20 t_{WR} = 12, t_{RP} = 11 t_{RTRS} = 2, t_{CAS} = 11 t_{RTP} = 6, t_{DATA_TRANS} = 4 t_{CCD_L} = 5, t_{CCD_S} = 4 t_{WTR_L} = 6, t_{WTR_S} = 2 t_{RRD_L} = 5, t_{RRD_S} = 4 t_{REFI} = 7.8μ s, t_{RFC} = 640 ns

Table 1. Simulator and DRAM timing [23] parameters.

We use a collection of multi-programmed workloads from SPEC2k6 (astar, libquantum, lbm, mcf, omnetpp, bzip2, GemsFDTD, leslie3d) and multi-threaded workloads from NAS Parallel Benchmarks (NPB) (cg, ep, mg) and Cloudsuite [19] (cloud9, classification, cassandra). The SPEC workloads are run in rate mode (eight copies of the same program). SPEC programs are fast-forwarded for 50 billion instructions and multi-threaded applications are fast-forwarded until the start of the region of interest, before detailed simulations are started.

Statistics from the first 5 million simulated instructions are discarded to account for cache warm-up effects. Simulations are terminated after a million memory reads are encountered.

5 Results

To estimate the effect of DRAM timing parameters on performance, we vary the following 13 DRAM timing parameters: tRCD, tRP, tRC, tRAS, tCAS, tCWD, tWR, tRTP, tFAW, tRRDL, tRRDS, tCCDL, tCCDS. Each of these timing parameters is improved in unison by 10%, 20%, 30%, 40%, and 50% (tREFI is increased and the rest are decreased). Figure 1 shows the impact on normalized execution time for each of these configurations. We see that a 50% improvement in all parameters results in a 23% reduction in average execution time. A 30% improvement in all timing parameters yields a 17% improvement in execution time, on average

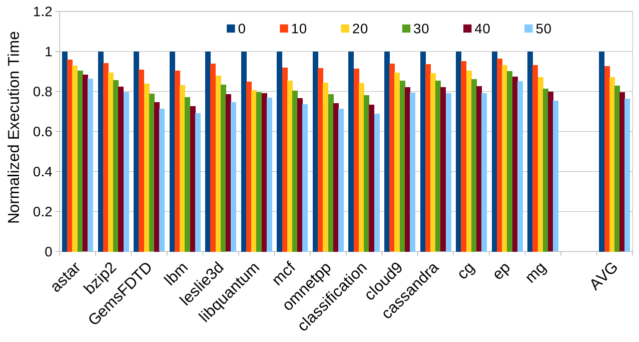

Figure 1. Cumulative effect of lowering different timing parameters on normalized execution time. Results are shown for timing parameters that are 10%,..., 50% better than the baseline.

To understand which timing parameters have the highest impact on performance, we carry out additional experiments that only modify a subset of these timing parameters in each experiment. Each experiment modifies timing parameters that are related to each other. These results are shown in Figures 2-6. We see that tRFC/tREFI (refresh timing parameters) and tCAS/tRAS/tRCD (delays to read data into row buffers) have the highest impact on performance. The effects of tTRANS/tCCDL/tCCDS (delays to move data to DRAM pins) are moderate. Meanwhile, the effects of tRP/tRAS/tRC (bank cycle time) and tFAW/tRRD (parameters to limit power delivery) are minor. We also observe that the effects of these timing parameters tend to be additive. This analysis is useful in designing a complexity-effective adaptive memory controller – to reduce memory controller complexity, it is best to focus on tRFC/tREFI/tCAS/tRAS/tRCD.

6 Conclusions

This paper argues for a brink-of-failure memory system that aggressively tries to shave DRAM timing margins, thus introducing a trade-off between performance

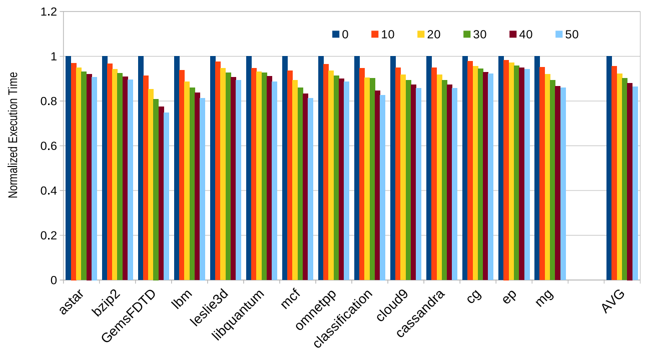

Figure 2. Effect of tRFC and tREFI.

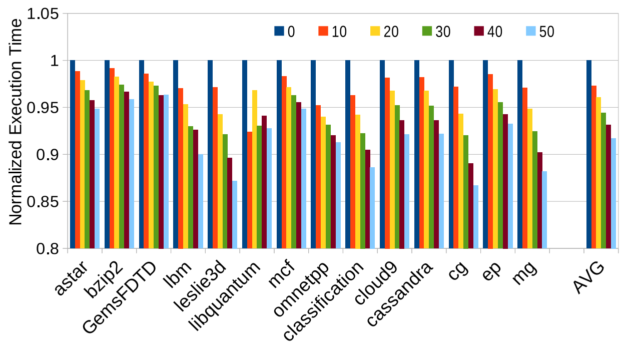

Figure 3. Effect of tCAS, tRAS, tRCD.

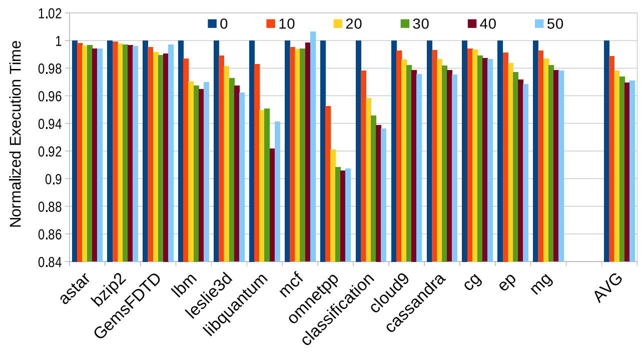

Figure 4. Effect of tRP, tRAS, tRC.

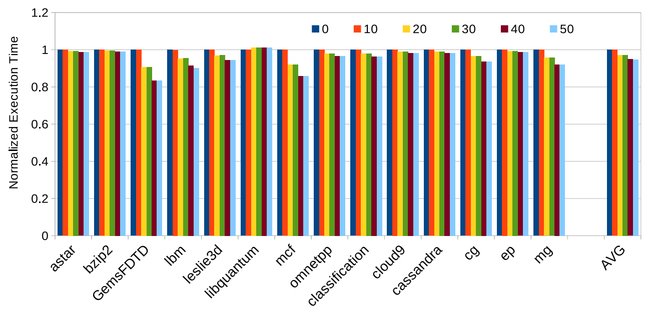

Figure 5. Effect of tTRANS, tCCDL, tCCDS.

and precision. Such a memory system can also exploit the high degree of parameter variation expected in future technologies. We show that a 30% improvement in a set of DRAM timing parameters can yield a 17% improvement in average execution time for a col-

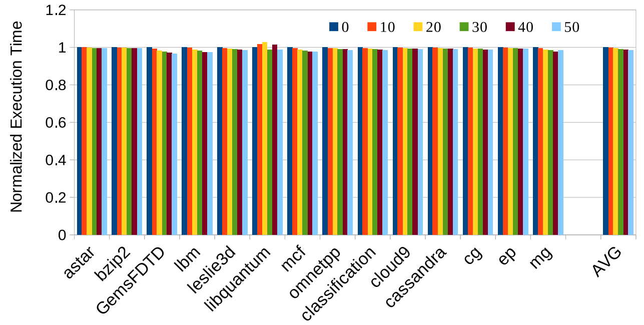

Figure 6. Effect of tRRD, tFAW.

lection of memory-intensive workloads. Much future work remains, including an empirical analysis of variation in DRAM chips, and support from the OS, programming language, and application to exploit an approximate memory system.

References

[1] Advanced Memory Protection for HP ProLiant 300 Series G4 Servers. `http://goo.gl/M2Mqa`.

[2] Memcached: A Distributed Memory Object Caching System. `http://memcached.org`.

[3] Wind River Simics Full System Simulator. `http://www.windriver.com/products/simics/`.

[4] A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, and K. Roy. A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies. *IEEE Transactions on VLSI*, 2005.

[5] J. Ahn and R. S. S. N. Jouppi. Future Scaling of Processor-Memory Interfaces. In *Proceedings of SC*, 2009.

[6] K. Argawal and J. Hayes. Method and Apparatus for Measuring Statistics of DRAM Parameters with Minimum Perturbation to Cell Layout and Environment, 2010. United States Patent, Number US 7,768,814 B2.

[7] A. Bacha and R. Teodorescu. Dynamic Reduction of Voltage Margins by Leveraging On-chip ECC in Itanium II Processors. In *Proceedings of ISCA*, 2013.

[8] W. Baek and T. Chilimbi. Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation. In *Proceedings of PLDI*, 2010.

[9] J. Bornholt, T. Mytkowicz, and K. McKinley. Uncertain¡T¿: A First-Order Type for Uncertain Data. In *Proceedings of ASPLOS*, 2014.

[10] L. Chakrapani, B. Akgul, S. Cheemalavagu, P. Korkmaz, K. Palem, and B. Seshasayee. Ultra-Efficient (Embedded) SOC Architectures based on Probabilistic CMOS (PCMOS) Technology. In *Proceedings of DATE*, 2006.

[11] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti. USIMM: the Utah SImulated Memory Module. Technical report, University of Utah, 2012. UUCS-12-002.

[12] N. Chatterjee, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi. Staged Reads: Mitigating the Impact of DRAM Writes on DRAM Reads. In *Proceedings of HPCA*, 2012.

[13] E. Chun, Z. Chishti, and T. Vijaykumar. Shapeshifter: Dynamically Changing Pipeline Width and Speed to Address Process Variations. In *Proceedings of MICRO*, 2008.

[14] M. deKruijf, S. Nomura, and K. Sankaralingam. Relax: An Architectural Framework for Software Recovery of Hardware Faults. In *Proceedings of ISCA*, 2010.

[15] T. J. Dell. A Whitepaper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. Technical report, IBM Microelectronics Division, 1997.

[16] S. Desai. Process Variation Aware DRAM Design Using Block-Based Adaptive Body Biasing Algorithm. Master's thesis, Utah State University, 2012.

[17] D. Ernst, N. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, T. Mudge, and K. Flautner. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *Proceedings of MICRO*, 2003.

[18] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture Support for Disciplined Approximate Programming. In *Proceedings of ASPLOS*, 2012.

[19] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *Proceedings of ASPLOS*, 2012.

[20] C. Garbella. A Basic Guide to Overclocking and System Building. `http://www.overclockers.com/a-basic-guide-to-overclocking-and-system-building/`.

[21] T. Hamamoto, S. Sugiura, and S. Sawada. On the Retention Time Distribution of Dynamic Random Access Memory (DRAM). *IEEE Trans. on Electron Devices*, 1998.

[22] ITRS. International Technology Roadmap for Semiconductors, 2007 Edition, Assembly and Packaging, 2007.

[23] JEDEC. *JESD79-4: JEDEC Standard DDR4 SDRAM*, 2012.

[24] X. Jian, H. Duwe, J. Sartori, V. Sridharan, and R. Kumar. Low-Power, Low-storage-overhead Chipkill Correct via Multi-line Error Correction. In *Proceedings of SC*, 2013.

[25] X. Jian and R. Kumar. Adaptive Reliability Chipkill Correct (ARCC). In *Proceedings of HPCA*, 2013.

[26] R. Joseph, D. Brooks, and M. Martonosi. Control Techniques to Eliminate Voltage Emergencies in High Performance Processors. In *Proceedings of HPCA*, 2003.

[27] K. Kim and J. Lee. A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs. *IEEE Electron Device Letters*, 2009.

[28] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *Proceedings of MICRO*, 2010.

[29] H.-W. Lee, K.-H. Kim, Y.-K. Choi, J.-H. Shon, N.-K. Park, K.-W. Kim, C. Kim, Y.-J. Choi, and B.-T. Chung. A 1.6V 1.4Gb/s/pin Consumer DRAM with Self-Dynamic Voltage-Scaling Technique in 44nm CMOS Technology. In *Proceedings of ISSCC*, 2011.

[30] S. Lee, C. Choi, J. Kong, W. Lee, and J. Yoo. An Efficient Statistical Analysis Methodology and Its Application to High-Density DRAMs. In *Proceedings of ICCAD*, 1997.

[31] L. Leem, H. Cho, J. Bau, Q. Jacobson, and S. Mitra. ERSA: Error Resilient System Architecture for Probabilistic Applications. In *Proceedings of DATE*, 2010.

[32] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tierno, and J. Carter. Active Management of Timing Guardband to Save Energy in POWER7. In *Proceedings of MICRO*, 2011.

[33] Y. Li, H. Schneider, F. Schnabel, and R. Thewes. DRAM Yield Analysis and Optimization by a Statistical Design Approach. *IEEE Trans. on Circuits and Systems*, 2011.

[34] X. Liang, R. Canal, G. Wei, and D. Brooks. Process Variation Tolerant 3T1D-Based Cache Architectures. In *Proceedings of MICRO*, 2007.

[35] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *Proceedings of ISCA*, 2013.

[36] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-aware intelligent DRAM refresh. In *Proceedings of ISCA*, 2012.

[37] D. Locklear. Chipkill Correct Memory Architecture. Technical report, Dell, 2000.

[38] S. Narayanan, J. Sartori, R. Kumar, and D. Jones. Scalable Stochastic Processors. In *Proceedings of DATE*, 2010.

[39] S. R. Nassif. Modeling and analysis of manufacturing variations. In *Proceedings of IEEE Conf. Custom Integr. Circuits*, 2001.

[40] R. Nelson. A Newbie's Guide to Overclocking Memory. `http://www.overclockers.com/a-newbies-guide-to-overclocking-memory/`.

[41] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazieres, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. Rumble, E. Stratmann, and R. Stutsman. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *SIGOPS Operating Systems Review*, 43(4), 2009.

[42] M. Samadi, D. Jamshidi, J. Lee, and S. Mahlke. Paraprox: Pattern-Based Approximation for Data Parallel Applications. In *Proceedings of ASPLOS*, 2014.

[43] M. Samadi, J. Lee, D. Jamshidi, A. Hormati, and S. Mahlke. SAGE: Self-Tuning Approximation for Graphics Engines. In *Proceedings of MICRO*, 2013.

[44] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Proceedings of PLDI*, 2011.

[45] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate Storage in Solid-State Memories. In *Proceedings of MICRO*, 2013.

[46] SAP. In-Memory Computing: SAP HANA. `http://www.sap.com/solutions/technology/in-memory-computing-platform`.

[47] SAS. SAS In-Memory Analytics. `http://www.sas.com/software/high-performance-analytics/in-memory-analytics/`.

[48] M. Shevgoor, J.-S. Kim, N. Chatterjee, R. Balasubramonian, A. Davis, and A. Udipi. Quantifying the Relationship between the Power Delivery Network and Architectural Policies in a 3D-Stacked Memory Device. In *Proceedings of MICRO*, 2013.

[49] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. In *Proceedings of FSE*, 2011.

[50] S.Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn. Flikker: Saving DRAM Refresh-power through Critical Data Partitioning. In *In Proceedings of ASPLOS*, 2011.

[51] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Mitigating Parameter Variation with Dynamic Fine-Grain Body Biasing. In *Proceedings of MICRO*, 2007.

[52] G. Torres. Memory Overclocking. `http://www.hardwaresecrets.com/article/Memory-Overclocking/152/`.

[53] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi. LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems. In *Proceedings of ISCA*, 2012.

[54] S. Venkataramani, V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan. Quality-Programmable Vector Processors for Approximate Computing. In *Proceedings of MICRO*, 2013.

[55] C. Wilkerson, A. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. Reducing Cache Power with Low-Cost, Multi-Bit Error Correcting Codes. In *Proceedings of ISCA*, 2010.

[56] D. Yoon and M. Erez. Virtualized and Flexible ECC for Main Memory. In *Proceedings of ASPLOS*, 2010.

[57] B. Zhao, Y. Du, Y. Zhang, and J. Yang. Variation-Tolerant Non-Uniform 3D Cache Management in Die Stacked Multicore Processor. In *Proceedings of MICRO*, 2009.

[58] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency. In *Proceedings of MICRO*, 2008.