Software-Defined Memory Controllers: An Idea Whose Time Has Come

Kevin Loughlin
kevlough@umich.edu
University of Michigan

Stefan Saroiu
ssaroiu@microsoft.com
Microsoft

Alec Wolman
alecw@microsoft.com
Microsoft

Baris Kasikci
barisk@umich.edu
University of Michigan

1 INTRODUCTION

As Dennard scaling/Moore's Law have come to an end, architects and programmers have had to find creative ways to improve the performance, power, and reliability of their systems. In the worlds of compute and cache resources, the limits of process scaling are met head-on via innovative hardware-software co-designs. For instance, modern CPUs and their caches expose numerous primitives to enable fine-grained tuning according to system needs. System software controls virtual-to-physical page mappings in the TLB, with the ability to set sizing, access permissions, and other metadata about these mappings. The behavior of loads and stores in the cache hierarchy can be customized via instructions of different operand widths, cacheability, coherence properties, and prefetching effects. Such flexibility enables managers of complex hardware-software systems, such as cloud providers, to make intelligent trade-offs among performance, power, and reliability in compute and cache resources.

In contrast, the main memory system is yet to exploit the same level of hardware-software co-design. Despite over a decade of research proposals [1, 4, 6–10, 13, 14, 16, 18–23, 26–28] demonstrating that main memory would benefit from additional configurability, today's cloud providers have little control over how their servers make use of DRAM. Commodity memory controllers are essentially black boxes, offering little-to-no configurability, further complicated by imprecise or incomplete specifications and documentation. Thus, very little tuning of the main memory system of a server-class machine is possible in practice because (1) memory controllers and BIOS software only expose coarse-grained settings, and (2) the monolithic nature of memory controllers leaves little room for parameterization per address region.

Notably, main memory is an increasingly-important bottleneck for a wide range of applications [3], especially in the age of big data. Furthermore, DRAM reliability continues to decline with process scaling, indicating reliability-insurance mechanisms may limit performance gains in forthcoming DRAM [2, 12, 14]. Finally, as we move towards heterogeneous computing—in which a variety of processing units share main memory—the importance of DRAM configurability to application trade-offs will continue to rise.

Thus, we posit that now is the time for *software-defined memory controllers*, particularly in multi-tenant environments such as the cloud. In our vision for a software-defined memory controller, system software (e.g., a hypervisor) could dynamically fine-tune controller settings and parameters according to the needs of its various applications/guests.

Inspired by prior work in hardware optimizations to the main memory system, we briefly describe key limitations in today's memory controllers and how a cloud provider would benefit from additional configurability in these areas (among many others). Our ultimate goal is to encourage the academic and industry research communities to further pursue innovation in software-defined memory controllers.

2 LET'S ADDRESS ADDRESSING

A key form of memory controller configurability would be fine-grained control of physical address to DDR logical address mappings. Such control would provide a crucial building block for configuring arbitrary memory parameters on a per-DRAM-region basis (e.g., DRAM page/row policies, DDR timings, refresh rates, etc.).

Each physical address in the system maps to exactly one DDR logical address (i.e., a hierarchical address identifying a location in DRAM via a specific channel, module, rank, bank, row, and column). The parallel in the CPU realm is a virtual-to-physical address mapping, which system software typically controls at page-sized granularity. Given control over virtual-to-physical mappings, a cloud provider can easily reason about and configure properties of individual pages.

However, unlike software-defined virtual-to-physical mappings, physical-to-DDR logical mappings are determined via BIOS parameters. Typically, DDR logical addressing is only configurable in terms of a coarse-grained trade-off between *bank-level parallelism* and *row buffer locality* [5], even if selectable at runtime [8]. To exploit bank-level parallelism, consecutive cache lines can be *interleaved* (i.e., spread) across different banks of DRAM, as different banks can be accessed in parallel to increase throughput. To exploit row buffer locality, consecutive cache lines can instead map to the same row of DRAM, as accessing the same row (akin to a cache hit) is faster than accessing another row in the same bank.

Intuitively, different workloads can be limited by different mappings; optimizing for bank-level parallelism can decrease row buffer hits (thereby increasing latency), while optimizing for row buffer hits can decrease bank-level parallelism (decreasing bandwidth). Despite these limits, the mappings in today's servers are largely "one size fits all." For example, with default interleaving enabled on an Intel Skylake server with 192 banks/socket (1 GB per bank), we find that a 2 MB huge page is interleaved across *every* bank in a single socket.

Such pervasively-applied interleaving can create performance and security problems between workloads co-located on the same bank(s). For performance, workloads that would otherwise experience high row buffer locality see hit rates decline due to row buffer contention with co-located applications. For security, inter-mixing different trust domains on the same bank can lead to inter-domain Rowhammer bit flips [11] and timing side channels [24].

To combat these problems, a software-defined memory controller could offer "middle grounds" between pervasive system interleaving and optimal row buffer locality, depending on the server's software stack. For example, a hypervisor—whose DRAM footprint arguably benefits more from isolation than bank-level parallelism—could be limited to an exclusive set of banks and thus relatively-isolated from its VMs. Workloads that have inherently-low row buffer hit rates could be similarly isolated to avoid row buffer contention.

One can imagine multiple implementations of such a design, including a memory controller-based address translation unit (akin to TLBs for DRAM caches [17]), or simply additional per-region configuration registers to support finer-grained translation settings (e.g., per DRAM bank group). We encourage additional proposals, designs, and evaluations.

3 GETTING META ABOUT OUR DATA

Today's DDR4 servers include 64 bits of metadata for each 512-bit cache line. These bits are used to implement a line's error correction code (ECC) and are not currently exposed to system software. However, as prior work [19] has shown, applications can benefit from dynamically re-purposing these bits for extra storage, in exchange for decreased ECC protection where acceptable (e.g., fault-tolerant applications).

Especially with DDR5 doubling the amount of metadata per line (i.e., to 128 bits), we propose that the level of metadata configurability should be even broader than re-using ECC for extra storage: system software should configure the use of the bits on a per-application basis. In doing so, software can make efficient use of the sizable "metadata" space according to its own needs (e.g., increasing storage capacity as previously proposed, tracking access count, and defining and enforcing security domains, among arbitrary uses).

4 THE LATENT LATENCY PROBLEM

Over the past decade, there have been significant improvements in the capacity and bandwidths of servers' memory systems. However, latency has essentially remained flat [3].

The aforementioned (§2) fine-grained control of DDR logical addressing offers a simple avenue for improving DRAM latency. As a basic example, workload-friendly addressing could provide decreased latency via better row buffer locality and/or bank-level parallelism.

More broadly, given fine-grained address mappings, cloud providers could tune each region's performance settings according to system needs. For instance, data with a lifetime shorter than a DRAM refresh interval (e.g., 64 or 32 ms) need not be refreshed to preserve its value, avoiding refresh-induced delays of demand reads. Similarly, prior work [15] shows that the DRAM write scheduling policy can be tuned to reduce *write*-caused interference in the performance of demand reads, wherein a read must be delayed due to the data bus processing write(s).

Despite these opportunities, reducing DRAM latency will require further cooperation from DRAM vendors. A software-defined memory controller alone cannot enable many promising ideas, such as tiered-latency DRAM [16], relaxed DDR timing parameters [6, 26], in-DRAM caching [25], and selective data refresh [1, 4]. Nonetheless, we expect such a memory controller to inspire DRAM vendors to expose additional internal knobs to software-level control.

5 LOOKING AHEAD

The systems and architecture communities are increasingly turning to hardware-software co-designs to efficiently, flexibly, and scalably solve problems in a post-Moore's Law world. Software-defined memory controllers would at long last introduce this paradigm into main memory management, enabling cloud providers to better control the performance, power, and reliability of their costly DRAM. It is our hope that the pursuit of software-defined memory controllers will encourage increased collaboration between cloud providers and memory controller vendors, in order to provide the feature set most useful for customer needs. Furthermore, we hope that the observed benefits of software-defined memory controllers will spur additional collaboration with DRAM vendors to achieve even greater gains via new levels of in-DRAM configurability.

REFERENCES

[1] Ishwar Bhati, Zeshan Chishti, Shih-Lien Lu, and Bruce Jacob. 2015. Flexible auto-refresh: Enabling scalable and energy-efficient DRAM refresh reductions. In *ISCA*.

[2] Lucian Cojocar, Kevin Loughlin, Stefan Saroiu, Baris Kasikci, and Alec Wolman. 2021. mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors. *Microsoft Tech Report* (2021).

[3] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. 2019. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. In *ACM SIGMETRICS*.

[4] Miseon Han, Yeoul Na, Dongha Jung, Hokyoon Lee, Seon Wook Kim, and Youngsun Han. 2018. Energy-Efficient DRAM Selective Refresh Technique with Page Residence in a Memory Hierarchy of Hardware-Managed TLB. *IEICE Transactions on Electronics* (2018).

[5] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wenisch, and Aniruddha N Udipi. 2014. Simulating DRAM controllers for future system architecture exploration. In *ISPASS*.

[6] Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Vivek Seshadri, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. 2016. Charge-Cache: Reducing DRAM latency by exploiting row access locality. In *HPCA*.

[7] Marius Hillenbrand and Frank Bellosa. 2017. Putting the OS in Control of DRAM with Mapping Aliases. In *SYSTOR*.

[8] Marius Hillenbrand and Frank Bellosa. 2017. Software-Defined Physical Memory: Putting the OS in Control of DRAM.

[9] Marius Hillenbrand, Mathias Gottschlag, Jens Kehne, and Frank Bellosa. 2017. Multiple Physical Mappings: Dynamic DRAM Channel Sharing and Partitioning. In *APSys*.

[10] Uksong Kang, Hak-Soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, S Jang, and Joo Sun Choi. 2014. Co-architecting controllers and DRAM to enhance DRAM process scaling. In *The memory forum*.

[11] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. 2014. Architectural support for mitigating row hammering in DRAM memories. *CAL* (2014).

[12] Jeremie S Kim, Minesh Patel, A Giray Yaglikci, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. 2020. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *ISCA*.

[13] Seikwon Kim, Wonsang Kwak, Changdae Kim, Daehyeon Baek, and Jaehyuk Huh. 2020. Charge-aware DRAM refresh reduction with value transformation. In *HPCA*.

[14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*.

[15] Chang Joo Lee, Veynu Narasiman, Eiman Ebrahimi, Onur Mutlu, and Yale N Patt. 2010. DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. (2010).

[16] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. 2013. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*.

[17] Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W Lee. 2015. A fully associative, tagless DRAM cache. *ACM SIGARCH CAN* (2015).

[18] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. 2021. Stop! Hammer time: rethinking our approach to rowhammer mitigations. In *HotOS*.

[19] Yixin Luo, Saugata Ghose, Tianshi Li, Sriram Govindan, Bikash Sharma, Bryan Kelly, Amirali Boroumand, and Onur Mutlu. 2017. Using ECC DRAM to adaptively increase memory capacity. *arXiv preprint arXiv:1706.08870* (2017).

[20] Sangkug Lym, Heonjae Ha, Yongkee Kwon, Chun-kai Chang, Jungrae Kim, and Matta Erez. 2018. ERUCA: Efficient DRAM resource utilization and resource conflict avoidance for memory system parallelism. In *HPCA*.

[21] Evgeny Manzhosov, Adam Hastings, Meghna Pancholi, Ryan Piersma, Mohamed Tarek Ibn Ziad, and Simha Sethumadhavan. 2021. MUSE: Multi-Use Error Correcting Codes. *arXiv preprint arXiv:2107.09245* (2021).

[22] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. 2011. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *MICRO*.

[23] Aniruddha N Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P Jouppi. 2010. Rethinking DRAM design and organization for energy-constrained multicores. In *ISCA*.

[24] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer attacks on mobile platforms. In *CCS*.

[25] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi, and O. Mutlu. 2020. FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching. In *MICRO*.

[26] Yaohua Wang, Arash Tavakkol, Lois Orosa, Saugata Ghose, Nika Mansouri Ghiasi, Minesh Patel, Jeremie S Kim, Hasan Hassan, Mohammad Sadrosadati, and Onur Mutlu. 2018. Reducing DRAM latency via charge-level-aware look-ahead partial restoration. In *MICRO*.

[27] Doe Hyun Yoon, Jichuan Chang, Naveen Muralimanohar, and Parthasarathy Ranganathan. 2012. BOOM: Enabling mobile memory based low-power server DIMMs. *ACM SIGARCH CAN* (2012).

[28] H. Yun, R. Mancuso, Z. P. Wu, and R. Pellizzoni. 2014. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *RTAS*.