An MLP-Aware Leakage-Free Memory Controller

Andrew Vuong
University of Utah
Salt Lake City, Utah
andrewv@cs.utah.edu

Ali Shafiee
University of Utah
Salt Lake City, Utah
shafiee@cs.utah.edu

Meysam Taassori
University of Utah
Salt Lake City, Utah
taassori@cs.utah.edu

Rajeev Balasubramonian
University of Utah
Salt Lake City, Utah
rajeev@cs.utah.edu

ABSTRACT

Timing channels can be exploited to leak information between two virtual machines running on a shared server. Indeed, cache timing channels are important components in the Spectre attack. A shared memory controller can also be leveraged to establish a timing channel. Recent efforts have designed memory controllers that eliminate such timing channels, but that incur throughput penalties of over 2×. This paper advances the state-of-the-art by better matching the memory controller policies to the memory-level-parallelism (MLP) needs of typical applications. Our new memory controller improves upon the performance of the state-of-the-art policy by 14%, while eliminating memory controller timing channels.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**;

KEYWORDS

Security, memory controllers, information leakage, timing channels

ACM Reference Format:
Andrew Vuong, Ali Shafiee, Meysam Taassori, and Rajeev Balasubramonian. 2018. An MLP-Aware Leakage-Free Memory Controller . In *HASP '18: Hardware and Architectural Support for Security and Privacy, June 2, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3214292.3214296

1 INTRODUCTION

Cloud platforms allow consumers to manage their computation and storage at low cost. These consumers also demand high levels of security and privacy for their sensitive data/computations, e.g., electronic health records. To a large extent, low cost is achieved because cloud operators dynamically schedule many workloads, belonging to different clients, on shared hardware. Each client or workload may be assigned its own Virtual Machine (VM), thus isolating and protecting each client. However, when VMs execute on shared hardware, information side channels do exist between the VMs. For example, if two VMs share a hardware cache, the latencies and miss rates experienced by one VM can betray the cache behavior of the other VM. Such a timing channel has been exploited by Ristenpart et al. [10] to crack passwords in an Amazon cloud infrastructure, in spite of various noise and real-world phenomena.

The recent Spectre attack [8] relied on cache timing channels to learn the secret-dependent footprints left behind by speculative execution. A cache timing channel can be eliminated with techniques like cache partitioning [16, 17], which exists in some commercial hardware [9]. But an application's secrets can be leaked through a variety of side channels, including through shared hardware structures such as the memory controller [14] and branch predictor [4]. Modern processors do not currently incorporate defenses against memory controller timing channels. This is an important vulnerability that can be exposed by attacks styled after Spectre.

A concrete attack through the memory controller was demonstrated by Wang et al. [14]. They show that the memory intensity of a VM can correlate with the number of 1s in the private key of an RSA decryption algorithm. Thus, by monitoring the memory intensity of a co-scheduled VM, an attacker VM can narrow the search space and quickly estimate private keys used by co-scheduled VMs.

Another important exploitation of memory-based timing channels is the establishment of a covert channel. Consider a cloud infrastructure set up by a hospital that includes electronic patient health records. The VMs used by the hospital may use state-of-the-art firewalls and encryption to protect their sensitive data. But they may use untrusted third-party software, e.g., a document reader, that has full access to unencrypted data and serves as a trojan. Because of the firewalls, the third-party trojan software can't explicitly send sensitive data outside the firewall. But by establishing a covert channel, it can leak information to another VM that is co-scheduled on the same hardware. In short, the third-party software can modulate the memory traffic rate, which is detected by the co-scheduled VM, thus communicating 1s (high traffic) and 0s (low traffic) to the co-scheduled VM. Recent work by Hunger et al. [5] has demonstrated memory-based covert channels that can leak information between threads at a high rate of 500 Kbps.

Therefore, to protect a sensitive cloud application, it is important to close the timing channel that exists within the memory system or memory controller. In other words, it is important to

HASP '18, June 2, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6500-0/18/06…$15.00
https://doi.org/10.1145/3214292.3214296

design a memory controller where the memory latencies experienced by one VM are independent of the memory activity of other co-scheduled VMs.

A few recent papers have designed such memory controllers. Wang et al. [14] introduced the first leakage-free memory controller. They leverage the concept of *Temporal Partitioning (TP)* that essentially allocates the memory controller to one VM at a time in round-robin fashion. During its *turn* to use the memory controller, a VM first issues memory requests, then enters a dead time when no new requests are issued. Thanks to the dead time, the VM's requests do not pose resource conflicts with requests issued by the next VM. While this eliminates leakage between VMs, it introduces significant performance slowdown. This is because of bandwidth underutilization and long queuing delays stemming from a VM's long wait for its turn.

A follow-up project by Shafiee et al. [12] improved upon the performance of the TP design. It introduced a *Bank Triple Alternation (BTA)* mechanism that reduced the turn lengths for each VM. This approach imposes constraints on the banks that can be accessed in each turn, thus ensuring that accesses by one VM do not conflict with accesses by another VM.

Wang et al. [15] then introduced a solution, SecMC-NI, that represents the state-of-the-art in leakage-free memory controllers. SecMC-NI grows the turn length for each VM, but imposes scheduling constraints to maximize the memory requests issued in that turn. However, while better than prior work, this design only achieves 48% of the throughput achieved by a non-secure memory controller, i.e., the slowdown is significant and the room for improvement remains high.

In this work, we build upon the state-of-the-art in two ways. We first modify SecMC-NI with additional constraints similar to BTA. This helps reduce the turn length. It is also a better match for the memory-level parallelism (MLP) exhibited in most applications. Second, we observe that such constrained memory controller policies work better when requests are spread across many ranks. To promote such spreading, we employ a memory system that employs many rank subsets [1, 19]. Our techniques bring the throughput levels closer to the non-secure baseline, while being implemented entirely in hardware and not increasing the burden on the OS.

2 BACKGROUND

2.1 Threat Model

As summarized in Section 1, we are targeting a cloud infrastructure where a single socket executes multiple different and unrelated VMs. These VMs may belong to different clients and do not trust each other. In fact, one of these VMs may be malicious and is attempting to extract information from other VMs. It does this by performing periodic memory accesses and recording average latencies for those accesses. Because of contention for the shared memory controller, these latencies reflect the memory intensity of co-scheduled VMs. This information leak either (i) betrays access patterns of co-scheduled threads, e.g., revealing the content of keys [10], or (ii) can be used to establish a covert channel between VMs. As described in Section 1, a covert channel can be used by a trojan application in one VM to betray secrets to a cohort in another VM [5].

Note that such timing channel attacks are performed without compromising the OS and without requiring physical access to the hardware. It can be performed by a remote user simply by asking the cloud operator to execute the user's VM.

There may be several side channels in a system, e.g., timing channels in a shared cache, timing channels in a branch predictor, leakage through addresses visible on the memory bus, etc. To fully protect a VM, each of these side channels will have to be closed with piecemeal solutions. Therefore, the solutions in this paper that address the timing side channel in the memory controller must be combined with other orthogonal solutions, e.g., cache partitioning [16].

2.2 Memory Controller Basics

A memory controller manages the memory modules connected to a single memory channel. A channel is connected to multiple *ranks* and each rank is itself partitioned into multiple *banks*. Many timing parameters dictate when operations can be issued to each rank and bank. For a detailed rundown of relevant timing parameters, please refer to the descriptions in prior work [12, 14]. Here, we'll focus on the bottomline timing constraints imposed by those parameters.

The memory controller can issue a request to one rank, and follow it with a request to a different rank (so that both requests can be performed in parallel). For the DRAM simulation parameters described in Section 4, requests to different ranks can be issued with a minimum gap of 6 memory cycles. Shafiee et al. [12] show that if two accesses to different ranks are issued with a 7-cycle gap, they are guaranteed to not interfere with each other, regardless of whether the accesses are reads or writes.

In a similar vein, the memory controller can also issue requests to different banks (either in the same rank or in different ranks) in parallel. Shafiee et al. consider all combinations of reads/writes and various timing parameters to show that requests to different banks will never interfere if they are separated by 15 cycles. Further, two requests to the same bank must be separated by 43 cycles so that the first access has enough time to complete (this assumes the worst case of a row buffer miss).

Based on the above discussion, we proceed with the following rules of thumb to ensure that two consecutive memory requests from different VMs do not compete for the same resources:

- Two accesses to different ranks must be separated by 7 cycles.
- Two accesses to different banks in a rank must be separated by 15 cycles.
- Two accesses to the same bank must be separated by 43 cycles.

By following these rules, the memory controller can create a schedule of memory accesses such that two VMs do not interfere with each other and are not vulnerable to memory timing channels.

Note that most high-performance processors have multiple DDR3/DDR4 memory channels. Since each channel is independently controlled, we will focus on a single channel for most of our discussion.

2.3 Temporal Partitioning [14] and Fixed Service [12]

The first attempt at eliminating timing channels in the memory controller uses a policy referred to as Temporal Partitioning (TP) [14]. A VM is assigned a turn for a number of cycles (the turn length); memory requests are initiated in the early part of the turn; the VM does not initiate any requests in the last 43 cycles of the turn (referred to as *dead time*). This guarantees that all memory requests have been completed before the next VM begins its turn. As a result, the next VM views the memory system as a clean slate and its memory latencies are independent of the memory activity exhibited by the previous VM, i.e., timing channels are eliminated.

Shafiee et al. [12] create a deterministic triple-alternation schedule to reduce the dead time (bank triple alternation – BTA). They require that a VM, during a turn, issue requests to (say) bank-ids that are multiples of 3, i.e., bankid modulo 3 = 0. In the next turn, the next VM must issue requests to banks where bankid modulo 3 = 1. In the turn after that, the next VM issues requests to banks where bankid modulo 3 = 2, and so on. By grouping banks in this manner and imposing restrictions on which banks can be accessed in a turn, the memory controller guarantees that consecutive turns will never access the same banks. This makes it easier to create a non-conflicting schedule of memory accesses without a long dead time. Shafiee et al. show that dead time can be reduced to 15 cycles while ensuring that the second VM's accesses are independent of activity in the first VM.

Figure 1 shows an example schedule for both TP and BTA. Note that both of these works recommend issuing a single request per turn to keep the turn length short and not have long queuing delays. Both works also show that if the OS can map each VM to its own set of ranks or banks, it is easier to create non-conflicting schedules and improve performance. However, such approaches are not palatable because of the higher OS complexity, the impact on other cloud management policies, and the relatively small number of ranks per channel. We will therefore only consider techniques in this paper that can be implemented entirely in hardware.

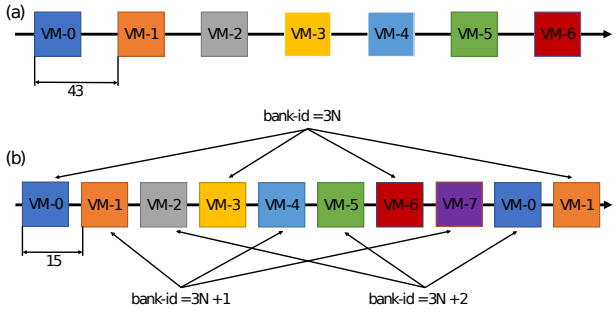

Figure 1: Example schedule for (a) TP and (b) BTA

2.4 SecMC-NI (3R/2B) [15]

In recent work, Wang et al. [15] introduced the state-of-the-art solution, SecMC-NI, with examples shown in Figure 2. Wang et al. show that throughput can be increased by giving a VM a long enough turn to drain multiple requests, by using a short dead time, and by imposing some scheduling constraints that prevent resource conflicts across VMs.

Consider the example shown in Figure 2. A VM is given a 43-cycle turn. Within that turn, the VM gets an opportunity to issue 6 memory requests – 2 requests each from 3 different ranks. All 6 requests are sent to different banks. A deterministic conflict-free schedule is set up by interleaving requests to the three ranks, as shown in Figure 2. Following the rules of thumb described in Section 2.2, consecutive requests to different banks in the same rank are separated by more than 15 cycles; consecutive requests to different ranks are separated by at least 7 cycles.

In the first turn, VM-0 decides to issue requests to ranks R-3, R-6, and R-7 (since those ranks have the most pending requests). Accordingly, two requests to R-3, two requests to R-6, and one request to R-7 are placed in the six issue slots (one slot is not utilized). In the second turn, VM-1 decides to issue requests to ranks R-1, R-4, and R-2; it is able to fill all 6 slots. Note that in the two slots assigned to R-4, requests to banks B-3 and B-6 were issued. In the third turn, VM-2 decides to issue requests to ranks R-4, R-5, and R-6. The R-4 requests are to banks B-1 and B-6. Because of the schedule that was adopted for VM-1, VM-2 is forced to delay its access of B-6. This is done to preserve the 43-cycle gap between two accesses to the same bank. Because the banks being accessed in a turn are unique, we are guaranteed to find a correct conflict-free schedule for every turn, in spite of the constraints imposed by the schedule in the previous turn. Thus, a given VM is able to issue any requests it desires in a turn (while conforming to the rule that requests are to different banks in three ranks), regardless of what other VMs are doing.

With the above policy, the schedule may reveal what the other thread was doing. For example, if the previous VM repeatedly accessed rank R-5, the current VM's requests to R-5 typically happen in the later part of its turn. To eliminate this leakage, a VM should be oblivious to the schedule within its turn, i.e., it should not realize that R-5 accesses are later than others. This is enforced by waiting until the end of the VM's turn to return all its requests en masse.

To make the subsequent discussion more intuitive, we will refer to the SecMC-NI policy as *3R/2B*, i.e., a turn schedule that accesses 2 banks each in 3 different ranks.

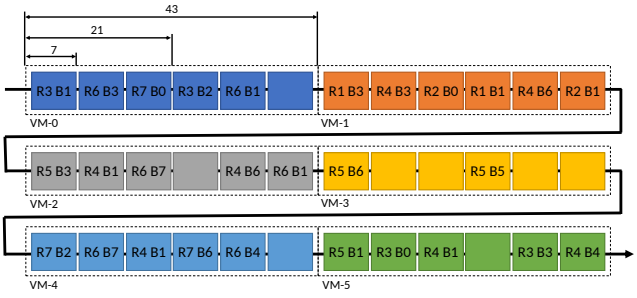

Figure 2: SecMC-NI example schedule.

3 PROPOSED MEMORY CONTROLLER POLICIES

3.1 Drawbacks of SecMC-NI

As shown later in Section 5, the state-of-the-art SecMC-NI (or 3R/2B) solution achieves only 48% of the throughput achieved by a constraint-free non-secure baseline. If the VM does not have 2 accesses each to three different ranks, it will under-utilize the six available slots. Figure 3 quantifies this phenomenon.

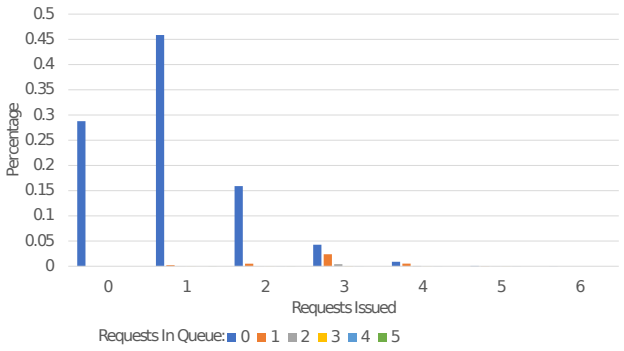

Figure 3: Histogram showing per turn stats for the baseline 3R/2B. Each cluster on the X-axis represents the number of requests issued in a turn; each bar within a cluster represents the number of requests that got left behind in the queue.

These results are for simulations with 8 VMs, 8 ranks, and a turn length of 43 cycles. Figure 3 shows a histogram to indicate the common scenarios encountered during a turn. Each cluster on the X-axis represents a count of the requests that were issued in a turn; each bar in a cluster represents the number of requests that were left behind in the queue. We observe that in the most common scenario (occurs in 45% of all turns) 1 request is issued in a turn and 0 requests are left behind in the queue. This is indicative of the low degree of MLP in our benchmark suite. In fact, in nearly 90% of all turns, the benchmark has less than 2 pending requests. This indicates that a turn length that accommodates six issue slots will often be under-utilized. Overall, we observe that 82% of all issue slots in the 3R/2B scheduler go unutilized. Further, the long 43-cycle turn length also increases the wait time for each VM.

Table 2 quantifies the per-benchmark MPKI (last level cache misses per thousand instructions), which ranges from 0.2 to 30.5. This range of MPKIs is similar to the ranges observed in other published workload characterizations [6, 11]. For most out-of-order processors that have a window size under 200, the typical number of outstanding memory accesses is less than a handful for even the most memory-intensive benchmarks.

3.2 Ranked Triple Alternation (RTA)

The observations in the previous sub-section indicate that a 3R/2B baseline schedule leads to under-utilization because an application typically has low MLP. Two accesses to the same bank must be separated by 43 cycles; by having a 43-cycle turn length, the requests issued in one turn are not influenced by the requests issued in the previous turn. But a 43-cycle underutilized turn implies that applications have long wait times until they receive their next turn.

Our goal here is to shrink the turn length to improve utilization and reduce wait times. But if a shorter turn length is used, the banks touched in one turn are not free to receive new requests from the next turn, thus leaking information from one VM to the next. To prevent consecutive turns from touching the same banks, we introduce deterministic scheduling constraints that are application-independent. These scheduling constraints leverage the same concept used in BTA, but applied at the rank level.

We introduce *Rank Triple Alternation (RTA)*, which shrinks the turn length to 14 cycles and allows a VM to issue 2 requests to 2 different ranks in its turn, with the added constraint that *rank-id mod 3 = turn-id mod 3*. In other words, in turn 0, a VM is allowed to touch two different ranks with rank-ids that are both multiples of three; in turn 1, the next VM is allowed to touch two different ranks with rank-ids of the form 3N+1, and so on. After every 3 turns, a one-cycle bubble is introduced so that potential accesses to the same bank are separated by at least 43 cycles. An example of RTA is shown in Figure 4. RTA is also more effective than BTA because the gap between consecutive requests is 7 cycles in RTA and 15 cycles in BTA.

To ensure that a VM can touch different ranks in different turns, the total number of VMs should not be a multiple of 3. If the number of co-scheduled sensitive VMs is a multiple of 3, the OS can introduce a dummy VM or use *Rank Quad Alternation*.

We also observe that RTA can issue two requests to the same rank in a turn as long as those two requests are of the same type (both reads or both writes) and are sent to different banks. Note that all of our schedulers employ an XOR address mapping policy to help evenly distribute requests across all ranks [12, 18].

3.3 Rank Subsetting

The proposed RTA scheduler does not improve performance when two requests are scheduled to different rows in the same bank. This bank-level conflict can cause delays because the second access must wait several turns before being issued. In general, the RTA scheduler has more options when requests are scattered across several banks and ranks. To promote such parallelism, we propose leveraging the concept of rank subsets [1, 19]. Rank subsets deviate from the DDR standard. They require creating a custom DIMM, as is done by certain vendors, e.g., in Power8 systems [13]. Rank subsets, while more expensive than commodity DIMMs, are useful for several reasons – they improve parallelism and reduce energy per access. We make the argument here that rank subsets are also useful in alleviating some of the constraints in leakage-free memory controllers.

Rank Subsetting offers higher memory parallelism by forming smaller parallel ranks. This increases the number of ranks and banks. However, forming smaller parallel ranks increases the data transfer time and reduces the size of the row buffer. The data transfer time, tBURST, increases from 4 cycles to 8 cycles when a rank is partitioned into 2 sub-ranks (referred to as 2-way rank subsetting). We repeated the mathematical analysis of Shafiee et al. [12] for the new timing parameters imposed by rank subsets. This yields new

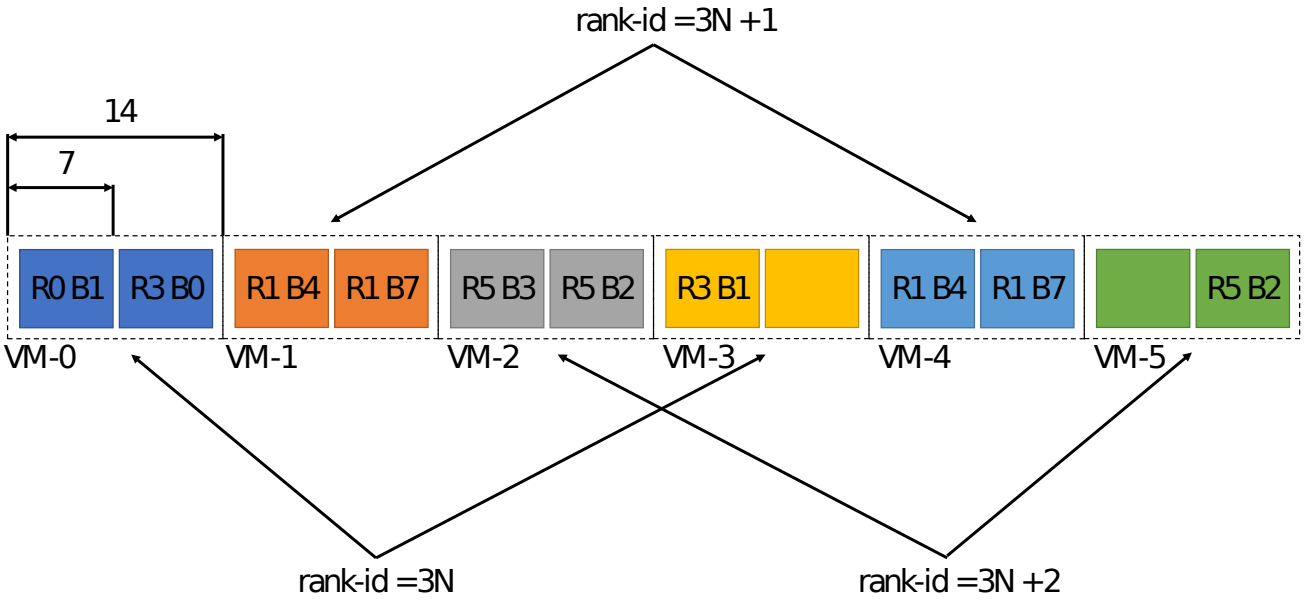

Figure 4: RTA example schedule.

restrictions for the leakage-free scheduler. With 2-way rank subsetting, accesses to two different ranks must now be separated by 10 cycles (previously 7 cycles), to two different banks by 25 cycles (previously 21 cycles), and to the same bank by 47 cycles (previously 43 cycles). While these gaps have grown, we show later that this penalty is offset by the higher parallelism.

Given these parameters, the equivalent of 3R/2B now requires a turn length of 60 cycles. We adapt RTA to these new timing parameters. We create a scheduler, *Ranked Penta Alternation (RPA)*, that offers each VM a 10-cycle turn, where that VM is allowed to access ranks such that rankid modulo 5 = turnid modulo 5. This guarantees that two accesses to the same bank are separated by at least 50 cycles. Again, RPA is pursuing RTA's philosophy to have short turns that cater to the low MLP in applications.

4 METHODOLOGY

For our simulations, we use Windriver Simics [3] interfaced with the USIMM memory simulator [2] to perform cycle-accurate simulations. USIMM is configured to model a DDR3 memory system. Simics and USIMM parameters are summarized in Table 1. Our baseline scheduler uses FR-FCFS, while the leakage-free schedulers add the constraints described earlier. We use an XOR address mapping for all simulations [18].

As benchmark suites, we use a collection of SPEC2k6 workloads. They are run in rate mode where eight copies of the same program are used. The programs are fast-forwarded 50-billion instructions and the simulations are completed after 1 million reads have occurred. The Mix1 workload has four copies each of libquantum and bwaves. Mix2 has two copies of mcf, astar, bwaves, and sjeng.

Processor	
ISA	UltraSPARC III ISA
CMP size and Core Freq.	8-core, 3.2 GHz
ROB size per core	128 entry
Fetch, Dispatch, Execute, and Retire	Maximum 4 per cycle
Cache Hierarchy	
L1 I-cache	32KB/2-way, 1-cycle
L1 D-cache	32KB/2-way, 1-cycle
L2 Cache	4MB/8-way, shared,10-cyc
DRAM Parameters	
DRAM Frequency	1600 Mbps
Channels, ranks, banks	1 ch, 8 ranks/ch, 8 banks/rank
DRAM chips	4 Gb capacity
DRAM Timing Parameters (DRAM cycles) t_{RC} = 39, t_{RCD} = 11,t_{RAS} = 28, t_{FAW} = 24 t_{WR} = 12,t_{RP} = 11,t_{RTRS} = 2, t_{CAS} = 11 t_{RTP} = 6, t_{BURST} = 4,t_{CCD} = 4, t_{WTR} = 6 t_{RRD} = 5, t_{REFI} = 7.8μs, t_{RFC} = 260ns	

Table 1: Simulator and DRAM [7] parameters.

Mix3 has two copies of xalancbmk, tonto, sjeng, and hmmer. Table 2 shows the MPKI for the SPEC2k6 workloads used – these values are similar to those observed by other works [6, 11].

MPKI	
mcf	30.505
astar	2.584
omnetpp	15.222
libquantum	16.203
xalancbmk	14.003
dealII	0.236
bwaves	0.254
gromacs	2.584
gobmk	0.599
sjeng	0.529
hmmer	0.902

Table 2: MPKI for SPEC2k6 workloads.

5 RESULTS

5.1 Performance of RTA

In Figure 5, we show the performance of the baseline 3R/2B scheduler and the proposed RTA scheduler, normalized against a baseline non-secure scheduler. Recall that RTA uses 14-cycle turns and allows 2 requests in a turn such that rank-id modulo 3 = turn-id modulo 3. We also show results for a ranked alternation (RA) scheduler that uses 7-cycle turns and allows a request in a turn such that rank-id modulo 7 = turn-id modulo 7.

We observe that RA offers a 5% advantage over 3R/2B, while RTA offers a 11% improvement over 3R/2B. Much of this improvement is because of the shorter turns and the shorter wait times on average for each memory request. The wait times for RTA and 3R/2B are quantified in Figure 6. RTA is also able to issue 2 reads to the same rank or 2 writes to the same rank in a turn; this is beneficial in a few memory-intensive workloads, contributing an average 1% improvement. The RTA scheduler out-performs 3R/2B in every benchmark except hmmer. As reported by Wang et al. [15], hmmer is known to suffer from many bank conflicts. Such bank conflicts are more problematic in RTA given that the scheduler already imposes more constraints, e.g., only allowing requests to ranks that are multiples of 3 in a given turn.

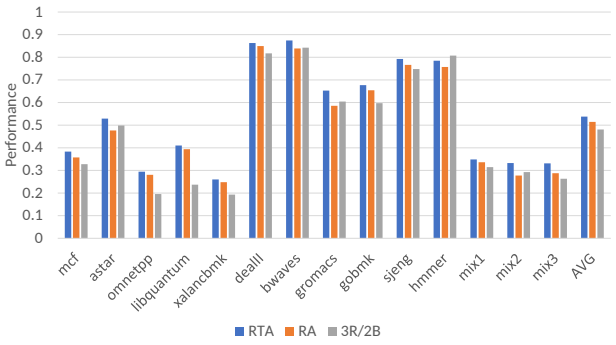

Figure 5: Performance comparision between baseline 3R/2B, RA and RTA. Performance is normalized to a baseline non-secure scheduler.

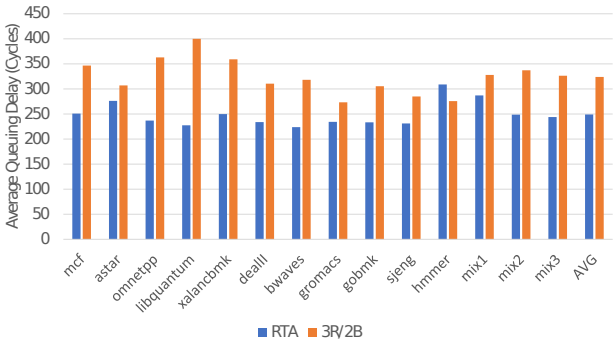

Figure 6: Average queueing delay of requests for RTA and 3R/2B.

5.2 Performance with Rank Subsetting

Next, we show the impact of rank subsetting on 3R/2B and on RPA, which is the alternation scheduler adapted for rank subsets. We will first consider 2-way rank subsets, i.e., each rank in our baseline is partitioned into two. We normalize all results against the non-secure baseline without rank subsets. In Figure 7, we see that rank subsets worsen performance for 3R/2B because they increase the turn length. However, the use of rank subsets with RPA is able to accommodate relatively short turns. The short turns and the fewer conflicts from rank subsets are able to offset the longer gaps between consecutive requests. Overall, we see that rank subsets and RPA are able to improve upon the performance of RTA by 3%. While this alone is not a large enough benefit to create custom non-DDR-compliant DIMMs, it adds to the arguments in favor of DIMMs that employ narrow ranks: lower energy, higher parallelism, and now better tolerance to the constraints imposed by leakage-free schedulers.

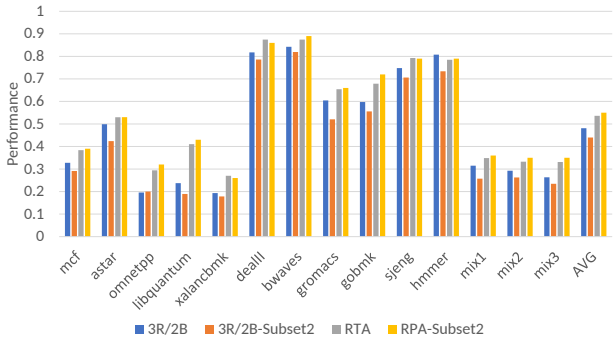

Figure 7: Performance comparison with and without rank subsetting for 3R/2B and for Ranked Alternation (RTA and RPA).

Figure 8 shows the performance of 4-way rank subsetting and 2-way rank subsetting with ranked alternation. With 4-way rank subsetting, the time between two accesses to the same rank is now 18 cycles and 55 for two accesses to the same bank. For 4-way

rank subsetting, we use RTA with 18 cycle turns, and for 2-way rank subsetting, we use RPA with 10 cycle turns. The longer turn length with 4-way rank subsetting and the resulting longer data transfer time cause a degradation relative to the RTA without rank subsetting. Therefore, the higher parallelism provided by 2-way rank subsetting hits a sweet spot and even higher parallelism is detrimental.

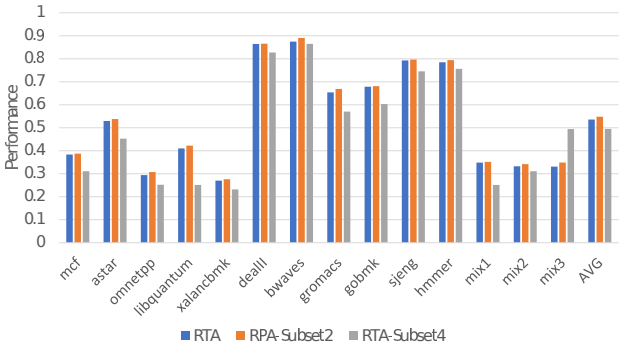

Figure 8: Performance comparison between 4-way and 2-way rank subsetting with ranked alternation.

5.3 Overall Performance

Finally, to set the performance of RTA in context, we show the primary design points in Figures 9 and 10. RTA improves upon 3R/2B by 11%; adding rank subsetting takes the performance improvement up to 14%. All of these techniques are hardware-based and require no additional support from the OS. These figures also show the performance benefit if there was additional OS support, as proposed by Shafiee et al. [12], to map each VM to its own banks (bank partitioning) or its own ranks (rank partitioning). We show that our hardware-based techniques are now approaching the performance of a technique that relies on the OS for bank partitioning.

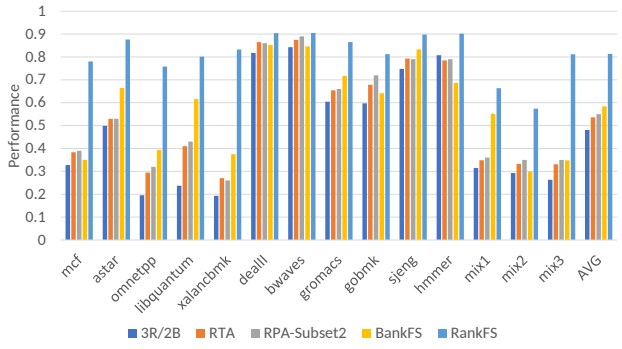

Figure 9: Overall performance of RTA, 3R/2B, RPA with 2-way rank subsetting, Fixed Service controller with bank partitioning, and Fixed Service controller with rank partitioning.

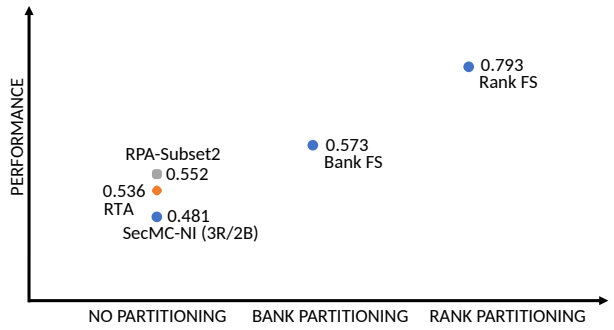

Figure 10: Overall performance of policies organized by OS support.

6 CONCLUSIONS

This paper introduces new scheduling strategies for the memory controller to improve performance while eliminating memory timing channels. The state-of-the-art scheduler 3R/2B uses relatively long turns that offer more issue slots per turn than can be exploited. Since most applications have low levels of MLP, we employ shorter turns and rely on scheduler constraints to prevent conflicts. The result is a scheduler, RTA, that improves performance by 11%. This improvement grows to 14% when the memory system is augmented with rank subsets and an adapted scheduler, RPA. The new schedulers take a step forward in reducing the overheads introduced by leakage-free memory controllers without relying on support from the OS.

ACKNOWLEDGMENT

We thank the anonymous reviewers for many helpful suggestions. This work was supported in parts by NSF grants CNS-1302663, CNS-1423583, and CNS-1718834.

REFERENCES

[1] J. Ahn, N. Jouppi, and R. S. Schreiber. 2009. Future Scaling of Processor-Memory Interfaces. In *Proceedings of SC*.
[2] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti. 2012. *USIMM: the Utah SImulated Memory Module.* Technical Report. University of Utah. UUCS-12-002.
[3] Wind Company. 2007. Wind River Simics Full System Simulator. http://www.windriver.com/products/simics/
[4] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh, and D. Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of ASPLOS.*
[5] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Sriram Vishwanath, Alex Dimakis, and Mohit Tiwari. 2015. Understanding Contention-driven Covert Channels and Using Them for Defense. In *Proceedings of HPCA.*
[6] A. Jaleel. 2018 (retrieved). SPEC CPU2006 Memory Characterization. http://www.jaleels.org/ajaleel/workload/.
[7] JEDEC. 2012. *JESD79-4: JEDEC Standard DDR4 SDRAM.*
[8] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. https://spectreattack.com/spectre.pdf.
[9] KT Nguyen. 2016. Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology.
[10] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM conference on Computer and Communications Security.* 199–212.

[11] D. Sanchez. 2018 (retrieved). ZSim Validation Data. http://zsim.csail.mit.edu/validation/time_series/.
[12] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari. 2015. Avoiding Information Leakage in the Memory Controller with Fixed Service Policies. In *Proceedings of MICRO*.
[13] J. Stuecheli. 2013. POWER8 Processor.
[14] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. 2014. Timing Channel Protection for a Shared Memory Controller. In *Proceedings of HPCA*.
[15] Y. Wang, B. Wu, and G.E. Suh. 2017. Secure Dynamic Memory Scheduling Against Timing Channel Attacks. In *Proceedings of HPCA*.
[16] Zhenghong Wang and Ruby B Lee. 2007. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. In *Proceedings of ISCA*.
[17] Zhenghong Wang and Ruby B Lee. 2008. A Novel Cache Architecture with Enhanced Performance and Security. In *Proceedings of MICRO*. 83–93.
[18] Z. Zhang, Z. Zhu, and X. Zhand. 2000. A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality. In *Proceedings of MICRO*.
[19] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. 2008. Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency. In *Proceedings of MICRO*.