Worst Case Delay Analysis of a DRAM Memory Request for COTS Multicore Architectures

Rafia Inam, Moris Behnam, Mikael Sjödin
Mälardalen Real-Time Research Centre,
Mälardalen University, Västerås, Sweden
Email: {rafia.inam, moris.behnam, mikael.sjodin}@mdh.se

ABSTRACT

Dynamic RAM (DRAM) is a source of memory contention and interference problems on commercial of the shelf (COTS) multicore architectures. Due to its variable access time, it can greatly influence the task's WCET and can lead to unpredictability. In this paper, we provide a worst case delay analysis for a DRAM memory request to safely bound memory contention on multicore architectures. We derive a worst-case service time for a single memory request and then combine it with the per-request memory interference that can be generated by the tasks executing on same or different cores in order to generate the delay bound.

1. INTRODUCTION

The real-time applications that are executed concurrently on COTS multicore platforms, face the new challenges due to sharing multiple different physical resources including CPU, shared caches, memory bandwidth and memory. Contention for the shared physical resources is a natural consequence of sharing [1]. It does not only reduce throughput but also affects the predictability of real-time applications.

Modern multicore architectures use a single-port Double Data Rate Dynamic RAM (DDR DRAM) as their main memory resource [2], which is shared among all cores. It is becoming a significant source of memory contention and interference problems that lead to unpredictability. It exhibits a highly variable DRAM access-time. Multiple studies provide bounds on memory interference delay by considering a constant access time [3, 4], and a variable access time [5, 6] for tasks executing concurrently on different cores and contending for memory accesses. Many hardware-based solutions have been proposed to eliminate these limitations at the level of DRAM controller [7, 8]. However, the real-time applications developed for COTS hardware cannot use this specialized hardware.

We provide a worst case delay analysis for a DRAM memory request to safely bound memory contention for multicore architectures. First, a worst-case service time for a single memory request is derived considering the worst case latency scenarios of DRAM commands. The service time is then combined with the per-request memory interference that can be generated by the tasks executing on same or different cores to generate the delay bound. Our analysis is similar to the work of [6], except that we have added additional constraints for shared memory banks (details in Section 4).

Section 2 provides the background on DRAM. Section 3 explains our system model. Memory interference delay analysis is presented in Section 4 and finally Section 5 concludes the paper.

2. DRAM BACKGROUND

A DRAM memory system consists of a DRAM controller and a memory device as shown in Figure 1. The controller serves the memory requests (i.e. schedules memory requests generated by CPU and sent to DRAM) and the memory device stores the actual data. The memory device consists of ranks and only one rank is accessed at a time. Each rank consists of multiple DRAM banks that can be accessed independently. The memory requests to different banks can be served concurrently, this is called *Bank-Level Parallelism (BLP)*. Only one of them can transfer data at a particular time on the data bus. [1]

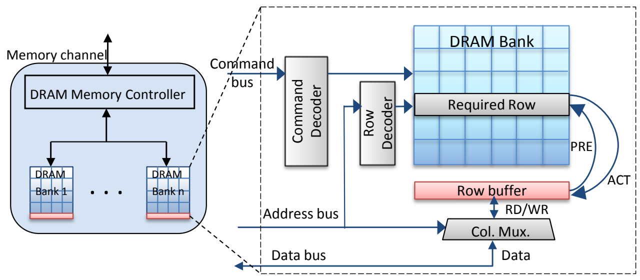

Figure 1: DDR DRAM banks : organization and functionality

Each bank consists of a *row-buffer*, and a two-dimensional structure of rows and columns of DRAM cells that actually stores data, as depicted in Figure 1. The data of a bank can only be accessed from the row-buffer of the bank. Thus to access the data from a bank, first the required row is cached into the row-buffer using the row decoder, and then the data is accessed from that row. The data of the particular column is accessed (read from and written to) from the row-buffer using the column multiplexer as shown in Figure 1. A column represents a small set of consecutive bits within a row. Thus the row-buffer serves as a buffer for the last-accessed request. All the subsequent requests to the columns of the same row do not require caching the row into the buffer, and are directly performed by accessing the required column of the row-buffer for faster access. The row that is cached is called an *open row*. A request to an open row is considered as a *row-hit*. If the currently opened row is different than the requested row, then first the opened row is saved and then the requested row is fetched into the row-buffer; it is called a *row-conflict*.

DRAM controller performs *internal scheduling algorithms* to reorder memory requests in order to improve the row-hit ratio and to maximize the overall throughput [9]. Since the row-hit latency is much less than the row-conflict latency, the DRAM controller prefers the row-hit request over the row-conflict requests, thus it unfairly prioritizes threads with high row-buffer locality. It schedules memory requests using *First-Ready First Come First Served (FR-FCFS)* algorithm [10] that prioritizes the ready DRAM commands (row-hit memory request) over others and for ties it prior-

[1]This is similar for rank level parallelism. We consider one rank and one channel in this work.

itizes older requests. It means that the memory requests arriving earlier may be serviced later than ones arriving later in the memory system. FR-FCFS works better with the *open row policy* that keeps the row-buffer open rather than *close row policy* that closes the row-buffer after serving each request.

The following commands are used to access the data from a bank (see Figure 1): Activate command (ACT) loads the requested row into the row-buffer using the row decoder; Precharge (PRE) writes back the currently opened row; (RD) reads the required data from the row-buffer using the column multiplexer; and (WR) writes the data into the row-buffer using the column muxtiplexer. RD/WR commands are also called CAS. Additionally, a Refresh command is issued regularly to refresh DRAM capacitors.

The memory controller must satisfy different timing constraints that occur between various DRAM commands. The timing constraints are taken from the JEDEC standard [2] and are listed in Table 1 with values for DDR3-1333MHz device. We consider 1333 MHz as this speed is approximately in the middle of DDR3.

Parameters	Description	DDR3	Unit
t_{CK}	DRAM clock cycle	1.5	nsec
BL	Burst length of data bus	8	cols
CL	Read latency	9	cycles
WL	Write latency	7	cycles
t_{RCD}	ACT to Read/Write delay	9	cycles
t_{RP}	PRE to ACT delay	9	cycles
t_{WR}	Data end of Write to PRE delay	10	cycles
t_{WTR}	Write to Read delay	5	cycles
t_{RC}	ACT to ACT delay (same bank)	33	cycles
t_{RRD}	ACT to ACT delay (diff. bank)	4	cycles
t_{FAW}	Four ACT window	20	cycles
t_{RTP}	Read to PRE delay	5	cycles
t_{RAS}	ACT to PRE delay	24	cycles
t_{RTW}	Read to Write delay	7	cycles
t_{RFC}	Time to refresh a row	160	nsec
t_{REFI}	Average refresh interval	7.8	usec

Table 1: DRAM timing constraints [2].

Here we briefly mention different characteristics of the DRAM-system that influence its memory access time. The details can be found in [11]. (1) *Row-buffer locality*. Since a row-hit requires fewer steps than a row-conflict, the latency of a row-hit is less than a row-conflict and this is called *row-buffer locality*. (2) *Bank-level conflicts* occur when multiple requests access the same bank. It results in a higher number of row-conflicts to the same bank, consequently the requests are serviced completely serially, and in this case, the latency is increased significantly. (3) *The direction of the data bus* should be changed upon the requests' sequence read-to-write or write-to-read and results in read-to-write latency and write-to-read latency respectively. During this time the data bus cannot be utilized. This latency exists whether the requests are made to the same bank or different banks. (4) *Scheduling algorithm* FR-FCFS, that unfairly prioritizes threads with high row-buffer locality.

Bank-level conflicts can be reduced by using *private banks* (supported by few hardware architectures like Freescale p4080 or by OS-based bank partitioning [12]). Other three characteristics are usually taken care in the memory-interference delay analysis for using private and/or interleaved banks [5, 6].

3. SYSTEM MODEL

We assume a single-chip multicore processor with a set of identical cores that have uniform access to the main-memory. Each core has a set of local resources (primarily a set of caches for instructions and/or data and busses to access these from the cores) and a set of resources that are shared amongst all cores (typically a Last-Level Cache (LLC), a main-memory, a memory bus, and the LLC and DRAM are connected by a command bus and a data bus). The architecture like Intel i5 3550, etc. complies with these assumptions. For simplicity, we consider one rank and a single channel. More than one channel can be considered independently since each channel has a separate command and data bus.

We assume that a local cache miss is stalling, which means whenever there is a miss in a LLC, the core is stalling until the cache-line is fetched from memory. We assume that all memory requests from the LLC to the shared DRAM go through the same channel and the data and command busses can be used in parallel. DRAM controller actually queues requests, however, at any given time only one of these requests is being served by the channel. Since the data is transferred in the *burst-mode* (a burst read/write allows to read/write the whole cache line after specifying only its start address, for a DIMM in a COTS-system has a cache line size of 64 B, a burst is 8 consecutive 64-bit pieces of memory), therefore, a single memory request can cache the data of one cache miss, thus the number of LLC misses is equal to the number of generated memory requests. Similar to [13, 6] we assume that each task has its own private partitions in the cache [14] that is sufficient to store one row of a DRAM bank. Further, we assume that cache-related preemption delays (CRPD) [15] are zero due to the partitioned cache.

We assume that the multicore processor uses DDR DRAM as its main memory, and it is not put on low power state at any time. The memory controller uses *open row policy* and employs FR-FCFS policy similar to [6] and we assume that DRAM records the arrival times of memory requests when they arrive at the controller. DRAM bank partitioning is considered to divide banks into partitions where memory request can access one bank in DRAM. We assume both *private banks* and *interleaved banks* (where memory request can access all banks in DRAM) are available and only one can be used at a time.

4. MEMORY INTERFERENCE DELAY ANALYSIS

We present an analysis of *worst case delay for a DRAM memory request ($D\ell$)*. The analysis depends on the hardware architecture and on the number of cores in the system. It is a sum of (1) worst case service time for a single memory request and (2) worst case delay this request under analysis can be delayed by other simultaneous requests (generated by other tasks executing on other cores).

4.1 Worst-case service time for a single memory request ($Dl_{ser.time}$)

We compute the worst-case service time for both, private banks (denoted as $Dl^{p}_{ser.time}$) and interleaved (or shared) banks ($Dl^{s}_{ser.time}$). We consider the worst cases of all previously mentioned characteristics that influence the memory access time of DRAM, i.e., row-conflicts, a change in the data bus direction for each request, and rescheduling algorithm. Since a row-conflict consists of three DRAM commands: ACT, PRE and CAS (RD/WR) (see Figure 1), thus $Dl_{ser.time}$ is a sum of latencies for ACT, PRE and CAS commands plus the DRAM timing constraints (given in Table 1) to meet these three commands.

$$\begin{aligned} Dl_{ser.time} \quad = \quad & (Dl_{PRE} + Dl_{ACT} + \max(TC_{PRE}, TC_{ACT}) + \\ & Dl_{CAS} + TC_{CAS}) \times t_{CK} \end{aligned} \tag{1}$$

where $Dl_{PRE}, Dl_{ACT}, Dl_{CAS}$ present latencies for ACT, PRE and CAS commands respectively, while $TC_{PRE}, TC_{ACT}, TC_{CAS}$ present the timing constraints for these commands respectively.

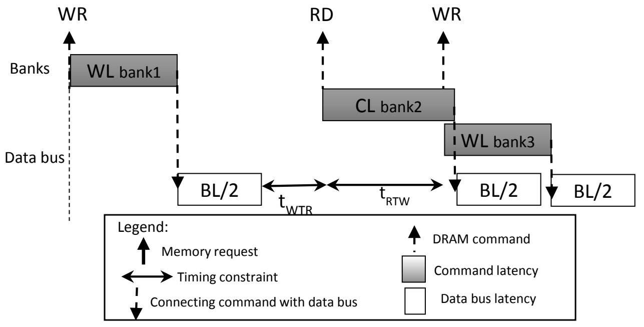

Figure 2: Timing constraints CAS commands for private banks (WR to bank1, RD to bank2, WR to bank3)

4.1.1 $Dl^p_{ser.time}$ for private banks

The worst-case service time is denoted by $Dl^p_{ser.time}$, and all other latencies are also presented with ^{p} symbol.

PRE command latency: $Dl^p_{PRE} = t_{CK}$ each command takes one clock cycle on address/command bus.

ACT command latency: According to the JEDEC standard [2] (see Table 1), t_{RRD} is the minimum separation time between two ACT commands to different banks. And maximum four ACT commands can be issued during one t_{FAW} window. To consider the worst-case, we take the max of both as

$Dl^p_{ACT} = \max(t_{RRD}, t_{FAW} - 3.t_{RRD})$.

CAS command latency: CAS latency is the sum of RD/WR latency plus the time to transfer the data on the data bus. The read (RD) and write (WR) latencies are CL and WL respectively (see Table 1). The data is transferred in burst mode on both the rising and falling edges of the double data rate DDR bus, therefore, the time to transfer the data is $BL/2$. Thus the total time for RD command is $CL + BL/2$, and for WR command is $WL + BL/2$. For worst-case, we take the maximum of RD and WR latencies, i.e.,

$Dl^p_{CAS} = \max(CL + BL/2, WL + BL/2)$.

PRE, ACT and CAS commands' timing constraint latencies: For private banks, there is no timing constraint for PRE and ACT commands, thus $TC^p_{PRE} = 0$ and $TC^p_{ACT} = 0$.

The timing constraints for CAS commands are due to the change in the data flow direction of the data bus. It depends upon whether the direction of the data bus is switching from Write to Read, or from Read to Write. It is zero if there is no switching in the direction. These constraints are depicted in Figure 2. In figures, the values of commands are not drawn according to the scale. t_{WTR} starts after the data is transferred ($BL/2$), however, t_{RTW} starts at the start of the RD command.

$$TC^p_{CAS} = \begin{cases} t_{WTR} & \text{if switching from write} \\ t_{RTW} - (CL + BL/2) & \text{if switching from read} \\ 0 & \text{if not switching,} \end{cases} \tag{2}$$

Putting the values of all these latencies in equation 1 provides the service time of a memory request using private banks.

4.1.2 $Dl^s_{ser.time}$ for shared banks:

For shared banks, the worst-case service time is denoted by $Dl^s_{ser.time}$, and all latencies are presented with ^{s} symbol.

PRE command latency: When memory requests are accessing the same bank then $Dl^s_{PRE} = t_{RP}$ (see Table 1).

ACT command latency: is $Dl^s_{ACT} = t_{RCD}$ (see Table 1).

CAS command latency: RD and WR latencies for shared banks

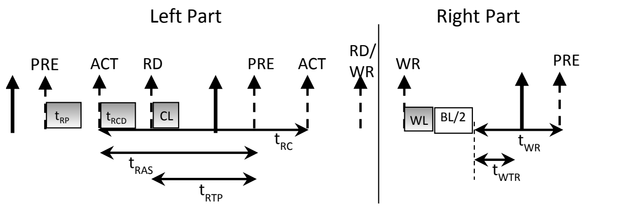

Figure 3: Timing constraints for shared banks

are the same as for private banks, i.e., CL and WL respectively. The time to transfer the data on the data bus is $BL/2$. Thus the total time for RD and WR is $CL+BL/2$, and $WL+BL/2$ respectively. Note that for RD command, data can be transferred in parallel (on the data bus) with the processing of data of the next command, therefore, $BL/2$ can be safely removed from all RD equations for simplicity. Thus RD latency becomes CL. For worst-case, we take the maximum of RD and WR latencies, i.e.,

$Dl^p_{CAS} = \max(CL, WL + BL/2)$.

PRE and ACT commands' timing constraint latencies: The timing constraints for PRE command depends on whether the previous command was RD or WR. It also depends on whether the row for the previous command was open or close.

Case 1: previous RD and open-row, (means only RD command was executed previously), thus t_{RTP} (RD to PRE delay) is considered (see Table 1). Since it includes the time to execute RD command (CL) (see left part of Figure 3), thus CL is subtracted from t_{RTP}. Thus the timing constraint is $\max(t_{RTP} - CL, 0)$.

Case 2: previous RD and close row, (means ACT and RD commands were executed for the previous request), so the timing constraint is taken from the ACT command of the previous request until the PRE command of the current request. t_{RAS} is the delay from ACT till PRE (see Table 1). It includes the time to execute previous ACT and RD commands within it (see left part of Figure 3). To calculate the timing constraint, the execution times of previous ACT and RD commands are subtracted from t_{RAS}; thus it becomes $t_{RAS} - T_{RCD} - CL$.

Case 3: previous WR and open-row, the timing constraint is $t_{WR} - t_{WTR}$ (see right part of Figure 3).

Case 4: previous WR and close-row is similar to case 2, only RD is replaced by WR latency, i.e., $t_{RAS} - t_{RCD} - (WL + BL/2)$. Thus, for RD and WR commands, TC^s_{PRE} becomes

$TC^s_{PRE(RD)} = \max(t_{RTP} - CL, t_{RAS} - t_{RCD} - CL, 0)$.

$TC^s_{PRE(WR)} = \max(t_{WR} - t_{WTR}, t_{RAS} - t_{RCD} - (WL + BL/2))$.

For worst case we take the maximum of both, i.e.

$$TC^s_{PRE} = \max(TC^s_{PRE(RD)}, TC^s_{PRE(WR)}) \tag{3}$$

For timing constraint of ACT, t_{RC} is considered for the shared bank (see Table 1). It is the time starting from one ACT till the start of the next ACT to the same bank, therefore, it includes the delays of CAS and PRE commands within it. To compute the timing constraint of ACT, the latencies of ACT and CAS commands are subtracted from it (see left part of Figure 3).

$TC^s_{ACT} = t_{RC} - t_{RCD} - \min(CL, (WL + BL/2))$

TC^s_{ACT} includes the timing constraint for PRE command TC^s_{PRE} within it (see left part of Figure 3). Since we take maximum of TC_{PRE}, TC_{ACT} in equation 1, so in worst case, the TC^s_{ACT} will be chosen when row is closed. And if the row is open then ACT command will not be executed and TC^s_{ACT} is zero, thus TC^s_{PRE} would be chosen there. The analysis in [6] does not consider both these timing constraints (i.e. TC_{PRE} and TC_{ACT}) for shared banks. Timing constraints for the shared banks are higher than the private banks and a main source of increased latencies.

CAS command's timing constraint latency: depends upon the previous CAS command:

$$TC^s_{CAS} = \begin{cases} T_{WTR} & \text{if previous write} \\ 0 & \text{if previous read,} \end{cases} \quad (4)$$

4.2 Per-request interference delay

It is the interference delay to execute the number of memory requests present in the memory controller and to be served before the request under analysis. If M is number of cores, then $M-1$ requests from other cores will be there in worst-case. Because of our assumption that core is stalling until the cache-line is fetched from memory, the maximum number of requests does not increase M. Considering the worst-case service time for each request (as presented in the previous Section 4.1), the interference delay becomes $(M-1) \times Dl_{ser.time}$. Adding the service time of the request under analysis, the total time to serve the request including interference becomes $Dl = Dl_{ser.time} + (M-1) \times Dl_{ser.time}$ or simply

$$Dl = M \times Dl_{ser.time} \quad (5)$$

For the private banks, $Dl_{ser.time}$ is substituted by $Dl^p_{ser.time}$ in the above equation. However for the shared banks, the reordering effect should also be taken into account.

Consecutive row-hit requests: According to FR-FCFS policy, the row-hit requests are given priority over the row-conflict requests. Row-hit requests are reordered at the bank scheduler and served before row-conflict requests. For worst case for m consecutive row-hit requests, we consider alternate read and write requests (means a change in the direction of data bus at each request). The worst-case service time for m consecutive row-hits is $Dl_{conhit}(m) = \{\lceil m/2 \rceil \times (WL + BL/2) + \lfloor m/2 \rfloor \times CL + m \times \max(case1, case3) + TC^s_{PRE}\}$. Since ACT command is not issued for open-rows, timing constraints for ACT are not included in $Dl_{conhit}(m)$. Also case1 and case3 (from last section) are included for open-rows only. TC^s_{PRE} of eq 3 is added if a PRE command is issued after m consecutive hit requests.

The maximum row-hits served by the system are N_{cols}/BL where N_{cols} is the number of columns in one row. In order to bound the reordering effect, a hardware threshold N_{cap} is also supported [10]. Thus in worst case the maximum number of row-hits prioritized over older row-conflicts is $N_{reorder} = \min(N_{cols}/BL, N_{cap})$ [6]. Substituting this number for m in Dl_{conhit}, i.e. $Dl_{conhit}(N_{reorder})$ equation gives the maximum number of row-hits served before older row-conflicts. $N_{reorder}$ can be greater than M. The assumption here is that each task can only have a single outstanding request, but once a hit from a task is served, it will unblock and can issue a new request that also results in a hit while the hits from other tasks are served. Considering worst-case, of $N_{reorder}$ hits and $M-2$ misses before the request under analysis, $Dl_{conhit}(N_{reorder})$ delay for hits and $Dl^s_{ser.time}$ for miss, the total delay becomes

$$\begin{aligned} Dl &= \max(Dl_{conhit}(N_{reorder}) + (M-2) \times Dl^s_{ser.time}, \\ &\quad M - 1 \times Dl^s_{ser.time}) \end{aligned}$$

In case of no hits $N_{reorder} = 0$, and Dl becomes $M \times Dl^s_{ser.time}$.

According to [5], the refresh effect is added as $k^{i+1} = \left\lceil \frac{(Total memory interference delay + k^i) \times t_{RFC}}{t_{REFI}} \right\rceil$ and $k^0 = 0$. Thus for DDR3-1333H, t_{RFC}/t_{REFI} is $160ns/7.8\mu s = 0.02$, thus will increase the total memory interference delay by 2%.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have safely bounded the memory contention for DDR DRAM memory controller that are commonly used in COTS multicore architectures. We have presented the worst case delay analysis of a memory request for private and shared memory banks. The analysis depends on the hardware architecture and on the number of cores. It is independent of the number of tasks executing in the system.

Previously, we have proposed a multi-resource server (MRS) [16, 17] approach to bound memory interference from other servers executing concurrently on other cores. The memory bandwidth has added as an additional server-resource to bound memory interference by considering a constant memory access time. In future, we intend to update the schedulability analysis of MRS by assuming a variable access time for the memory requests and combining our current analysis of ($D\ell$) for this purpose.

6. REFERENCES

[1] R. Rettberg and R. Thomas. Contention is no obstacle to shared-memory multiprocessing. *Commun. ACM*, 29(12):1202–1212, December 1986.

[2] Micron. 2Gb DDR3 SDRAM.

[3] S. Schliecker and M. Negrean and R. Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *(DATE)*, pages 759–764, 2010.

[4] R. Pellizzoni and A. Schranzhofer and J.-J.Chen and M. Caccamo and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *(DATE)*, 2010.

[5] Z-P. Wu, Y. Krish, and R. Pellizzoni. Worst case analysis of DRAM latency in multi-requestor systems. In *(RTSS)*, 2013.

[6] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. R. Rajkumar. Bounding memory interference delay in COTS-based multicore systems. In *(RTAS)*, Apr 2014.

[7] J. Reineke, I. Liu, H.D. Patel, S. Kim, and E.A. Lee. PRET DRAM controller: Bank privatization for predictability and temporal isolation. In *(CODES+ISSS)*, September 2011.

[8] L. Yonghui, B. Akesson, and K. Goossens. Dynamic command scheduling for real-time memory controllers. In *ECRTS*, 2014.

[9] K.J. Nesbit, N. Aggarwal, J. Laudon, and J.E. Smith. Fair queuing memory systems. In *In International Symposium on Microarchitecture (MICRO)*, 2006.

[10] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security Symposium*, 2007.

[11] R. Inam and M. Sjödin. Combating unpredictability in multicores through the multi-resource server. In *Workshop on Virtualization for Real-Time Embedded Systems*, 2014.

[12] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank level interference in multicore systems. In *PACT*, 2012.

[13] H. Yun and G. Yao and R. Pellizzoni and M. Caccamo and L. Sha. Memory- access control in multiprocessor for real-time systems with mixed criticality. In *(ECRTS)*, July 2012.

[14] J. Liedtke, H. Härtig, and M. Hohmuth. OS-controlled cache predictability for real-time systems. In *RTAS*, 1997.

[15] D. Hardy and I. Puaut. Estimation of cache related migration delays for multi-core processors with shared instruction caches. In *(RTNS 09)*, 2009.

[16] M. Behnam, R. Inam, T. Nolte, and M. Sjödin. Multicore composability in the face of memory bus contention. In *(CRTS)*, 2012.

[17] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin. The multi-resource server for predictable execution on multicore platforms. In *(RTAS)*, 2014.