Simulating DRAM controllers for future system architecture exploration

Andreas Hansson
Research & Development
ARM Ltd.
Cambridge, United Kingdom
andreas.hansson@arm.com

Neha Agarwal, Aasheesh Kolli, Thomas Wenisch
Advanced Computer Architecture Lab
The University of Michigan
Ann Arbor, United States
{nehaag,akolli,twenisch}@umich.edu

Aniruddha N. Udipi
Research & Development
ARM Inc.
Austin, United States
ani.udipi@arm.com

***Abstract*—Compute requirements are increasing rapidly in systems ranging from mobile devices to servers. These, often massively parallel architectures, put increasing requirements on memory bandwidth and latency. The memory system greatly impacts both system performance and power, and it is key to capture the complex behaviour of the DRAM controller when evaluating CPU and GPU performance. By using full-system simulation, the interactions between the system components is captured. However, traditional DRAM controller models focus on modelling interactions between the controller and the DRAM rather than the interactions with the system. Moreover, the DRAM interactions are modelled on a cycle-by-cycle basis, leading to inflexibility and poor simulation performance.**

In this work, we present a high-level memory controller model, specifically designed for full-system exploration of future system architectures. Our event-based model is tailored to match a contemporary controller architecture, and captures the most important DRAM timing constraints for current and emerging DRAM interfaces, e.g. DDR3, LPDDR3 and WideIO. We show how our controller leverages the open-source gem5 simulation framework, and compare it to a state-of-the-art DRAM controller simulator. Our results show that our model is 7x faster on average, while maintaining the fidelity of the simulation. To highlight the capabilities of our model, we show that it can be used to evaluate a multi-processor memory system.

I. Introduction

More complex algorithms, coupled with larger data sets, lead to a sharp growth in computational requirements in domains ranging from embedded systems to servers. To meet the computational demands, increasingly parallel architectures are being used, e.g. many-core CPUs and GPUs. The power and performance of these emerging systems is tightly coupled to the characteristics of the memory system [1]–[5]. Simulation studies using overly simplistic memory models, e.g. based on fixed latency or throughput, may mislead the design space exploration and architectural trade-offs [6]–[8]. Thus, having a representative memory-controller performance model is crucial [9]. However, building an accurate DRAM controller model is non-trivial, due to a wide range of timing-related constraints and optimisation goals [10].

The behaviour of a contemporary DRAM controller is heavily dependent on the interplay between the memory access pattern, various DRAM timing parameters and the underlying DRAM architecture [6], [7], [10]. The access pattern, in turn, is determined by complex CPU and GPU architectures, running elaborate software stacks, with feedback loops between the hardware and software [11]. Traces fail to capture these inter-dependencies [8]. By using full-system simulation, the complex interactions between the CPUs, GPUs, I/O devices, and the DRAM controllers are captured [2]. For future system exploration, the model must also offer enough flexibility to capture emerging memory technologies, e.g. stacked WideIO DRAM [1], [5], and non-conventional DRAM organisations such as Hybrid Memory Cube (HMC) [12]. The simulation performance is also critical as workloads routinely involve simulating millions, or even billions of cycles [13]. These issues are only partially addressed by current cycle-based DRAM models [9], [14], [15] that sacrifice speed and are tailored for a narrow set of memory technologies, thus restricting their applicability to future system architecture exploration.

As the major contribution of this work we present *a fast, accurate and modular DRAM controller model, readily available* as part of the open-source full-system simulator gem5 [16]. First, we show how focusing on the controller rather than the memory allows us to build a high-performance event-based model. We demonstrate how DRAM behaviour is captured with high accuracy by only modelling the state transitions of the banks and the busses, thus enabling a fast, yet accurate model. Second, we compare the simulated behaviour and performance with a state-of-the-art DRAM simulator. For a large number of benchmarks, we show that our model correlates well in terms of bandwidth and latency trends, and does so with a much-improved simulation performance and scalability. Lastly, we demonstrate how the proposed controller model is used as a valuable tool to study the impact of various future DRAMs on system performance.

The rest of this work is organised as follows. First we discuss the rationale behind our model and its design in Section II. Then, in Section III, we validate the behaviour of the model and its impact on system performance. Next, we exemplify how the model is used for system-level exploration in Section IV. Related work is reviewed in Section V, and we conclude and outline directions for future work in Section VI.

II. Rationale and design

DRAMs have a complex architecture to allow increasing bandwidths with a fairly constant internal clock speed. As elaborated on in [10], and depicted in Figure 1, they are organised in cell arrays, where rows (or pages) are read or written through a series of data movements. The DRAM provides parallelism through a number of banks that operate concurrently. The banks all share the same data, address and command bus (with the latter two being combined in mobile DRAMs). In addition, a number of DRAM devices can be connected to the same busses in ranks, offering additional parallelism. The complex internal organisation of DRAMs leads to a wide array of timing constraints governing the DRAM behaviour.

To study DRAM's impact on system power and performance, we need a controller model that is representative with respect to the controller architecture (Section II-A) and how the controller balances the many timing constraints of the memory (Section II-B) and the optimisation goals and constraints of the transactions (Section II-C). The performance of the model itself is also important as we need to evaluate billions of cycles, and our event-based modelling technique gives us high simulation performance with good accuracy (Section II-D). In addition to the controller itself, the overall system performance is affected by the transactions issued by the CPUs, GPUs and I/O devices in the system (Section II-E), and how transactions are distributed across the memory controllers and routed through the interconnect (Section II-F).

Next we describe how our model solves the aforementioned challenges, and how it captures the performance and power impact (Section II-G) of the DRAM controller without modelling the DRAM itself.

A. Controller architecture

Our model captures a generic modern DRAM controller architecture, with split read and write queues, and a shared response queue. We perform all buffering per controller instance, rather than per rank or bank. Our choices reflect what we believe is most representative for a contemporary DRAM controller [17], [18]. The queue sizes are parameters of each controller instance, as shown in Table I.

To express the memory organisation, the controller model has parameters determining the bus width, burst length, row-buffer size, as well as the number of devices per rank, ranks and banks. Address decoding into rank, bank, row and column is done by each controller independently, based on one of the address decoding schemes enumerated in Table I. Channel interleaving, on the other hand, takes place outside the controller in a separate crossbar (Section II-E). Thus, when instantiating a multi-channel DRAM controller, as illustrated in Figure 1, gem5 takes care of configuring the crossbars with the appropriate interleaving granularity.

To enable modelling of mobile DRAMs like LPDDR3, our model does not place any assumptions on the relation between requests made by the CPU and I/O devices, and the burst length (and thus burst size) of the DRAM. Thus, a cache line may be chopped into a number of DRAM bursts, depending on the interface width and the burst length of the DRAM. In contrast to [9], these sub-cache-line accesses are properly merged and dealt with by our controller, leaving the rest of the memory system oblivious to the DRAM burst size. This functionality removes the need to add complexity in other parts of the memory system, and, as shown in Section IV, also allows the controller to benefit from sequential sub-accesses.

TABLE I. DRAM CONTROLLER PARAMETERS

Parameter	**Description (unit)**
Write buffer size	Number of write queue entries
Read buffer size	Number of read queue entries
Write high/low threshold	High/low watermark for write queue
Scheduling policy	FCFS or FR-FCFS
Address mapping	RoRaBaCoCh, RoRaBaChCo, RoCoRaBaCh[1]
Page policy	Open or closed (adaptive or not)
Frontend latency	Static frontend latency (ns)
Backend latency	Static backend latency (ns)
Device bus width	Data bus with per DRAM device (bits)
Burst length	DRAM burst length (beats)
Row-buffer size	Device row buffer size (bytes)
Devices per rank	-
Ranks per channel	-
Banks per rank	-
Channels	Channel count for the address decoding
tRCD	Row to column delay (ns)
tRAS	Row access strobe (ns)
tRP	Row precharge time (ns)
tCL	Column access latency (ns)
tBURST	Burst duration (ns)
tRFC	Refresh cycle time (ns)
tREFI	Refresh command interval (ns)
tWTR	Write to read switching time (ns)
tRRD	Row to row activation delay (ns)
tXAW	Activation window (ns)
Activation limit	Number of activates in window

Our controller model responds to writes as soon as the request is placed in the write queue. The early write response gives a lower latency than waiting for the DRAM to complete the transaction. Incoming reads snoop in, and are potentially satisfied by the write queue. Furthermore, write transactions smaller than the DRAM burst size are merged if possible. These techniques are also used in [17]–[19]. The early response enables us to buffer (and delay) writes without incurring any performance impact on the system level.

Note that we do not model any separate command queues, in contrast to DRAMSim2 [9]. Thus, both our page policy and arbitration scheme (Section II-C) operate directly on the read and write request queue. Capturing the split request and command queues adds unnecessary cost and detail to the model, and, as we shall see in Section III, does not affect the system-level performance. Next we look at how we capture the DRAM state and timings in more detail.

B. DRAM timings

Similar to an actual controller implementation (and existing models [9]), our model tracks the state of all the banks, potentially distributed over a number of ranks. A simplified *DRAM state machine* [20] is thus implicitly encoded in the controller code. As we shall see, this information is needed to make scheduling decisions, and to adhere to the DRAM timings. We use the same timing information to determine the state transitions of our event-based model (Section II-D).

A contemporary DRAM is governed by a few dozens of timing constraints, and exhaustively modelling them is merely adding unnecessary detail, and does not contribute (significantly) to the system-level performance. Instead, we choose to model the most important timings, summarised in Table I. As discussed in [6], [7], these include the time required to open and close a row (tRCD, tRAS and tRP), and the

[1]Channel (Ch), Rank (Ra), Bank (Ba), Row (Ro), Column (Co)

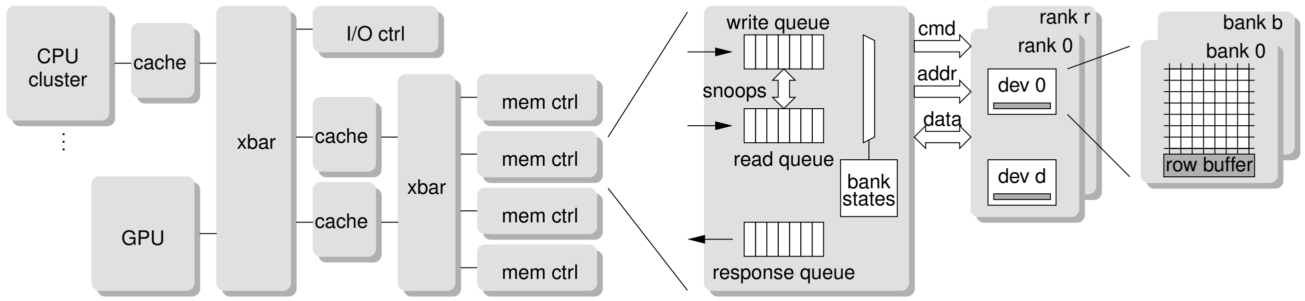


Fig. 1. Architecture overview depicting a system with a multi-channel DRAM controller.

time spent accessing the specific column (tCL and tBURST, where the former captures tWR and the latter implicitly models tCCD). Choi et al [6] do not model inter-bank interference, but highlight its importance. Indeed, we also model tTAW/tFAW (as a more generic tXAW) and tRRD. Moreover, we model the refresh timings, tREFI and tRFC, as they cause big latency spikes [9], and also capture the timing impact of read write switching (tWTR). Notable timings that are not currently modelled are the rank-to-rank switching constraints, and the differences caused by the introduction of bank groups.

Most DRAM timings do not scale with the interface frequency increase, and this makes the bus busy time a dominant factor in the DRAM performance [6]. We therefore model the data bus contention by tracking its availability in the controller. We do not model the command and address bus, as they are typically designed to not constitute a bottleneck (offering features like posted commands with additive latency [20]). Thus, the difference between LPDDR and DDR is only distinguished by their timings and DRAM organisations. Similarly, we do not explicitly distinguish single and double data rate (SDR and DDR) memories, and instead rely on the tBURST parameter to reflect the duration of the data transfer. We choose not to support Rambus-like memories with shared data and address bus as no emerging DRAMs are organised that way.

In addition to the DRAM timings, the model also has two timing parameters to capture the static *frontend* and *backend* latency. The frontend latency captures the pipeline stages of the controller, allowing us to reflect the controller design complexity. The backend latency, in turn, captures the PHY and IO, thus allowing us to study the impact of different interconnection options, e.g. Package-on-Package, Dual Inline Memory Module and Through Silicon Vias [1]. Next, we elaborate on how the state and timing information is used by the scheduler.

C. Scheduling

The memory controller schedules requests based on the Quality-of-Service requirements of the requesting CPUs and I/O devices (if any), and the aforementioned timing constraints, with the goal to maximise the efficiency, and thus the available bandwidth of the DRAM [3], [11]. The scheduling is also closely coupled to the page policy. Next, we discuss the page policy, read/write switching policy and scheduling in turn.

The controller model offers two basic row buffer policies (and two variations of each), similar to [9], [19]. First, the *closed page* policy, opens the row for every column access and then closes it by means of an auto precharge. The improved *adaptive closed page* keeps the row open if there are already queued accesses to the open row, as suggested in [19]. Second, the *open page* policy, leaves a row open until a bank conflict occurs, in which case the previous row is closed and the new row opened. The *open adaptive page* policy does not wait until the conflict occurs, but instead closes the page in advance if there are accesses to a different row in the same bank (bank conflict), and no queued accesses to the open row [9], [19]. The schemes provided serve as a starting point, and more elaborate schemes can easily be added thanks to the model extensibility.

The first level of scheduling that takes place in our model is the choice between reads and writes. We choose to implement a write drain mode similar to what is proposed in [19] to minimise the cost of read/write switching (tWTR). A high water mark is used to determine when to forcefully switch to writes, and a minimum number of writes are allowed to be issued before switching back (unless the write queue is empty). If no reads are present, a low water mark determines when to start draining writes. The low water mark allows us to control how much write data is kept on chip.

The second level of scheduling involves selecting a request either from the read or write queue, depending on the current direction and page policy. We implement two basic (and very popular) scheduling algorithms, FCFS and FR-FCFS [21]. The former is merely included for comparison, whereas the latter provides a good baseline scheduler, achieving good performance, even compared to very advanced schedulers [3], [11], [22]. The FR-FCFS scheduler picks the access that is first ready, which in the case of an open-page policy prioritises row hits, and if no row-hits are available (or when using a closed-page policy) targets the first available bank.

Our goal is not to provide an exhaustive set of schedulers, but rather a representative baseline, and a framework in which more elaborate schedulers [3], [11], [22], can be evaluated.

D. Modelling technique

Cycle-based models are easily portable as they do not depend on any particular event semantics or simulation framework. However, as they need to be updated on a cycle-by-cycle basis they reduce simulation speed, sometimes by several orders of magnitude [13]. An event-based model, as proposed in this work, only executes when something changes, and thus skips ahead to the next event, as illustrated in Figure 2. Event-based models are, by necessity, tied to particular simulation

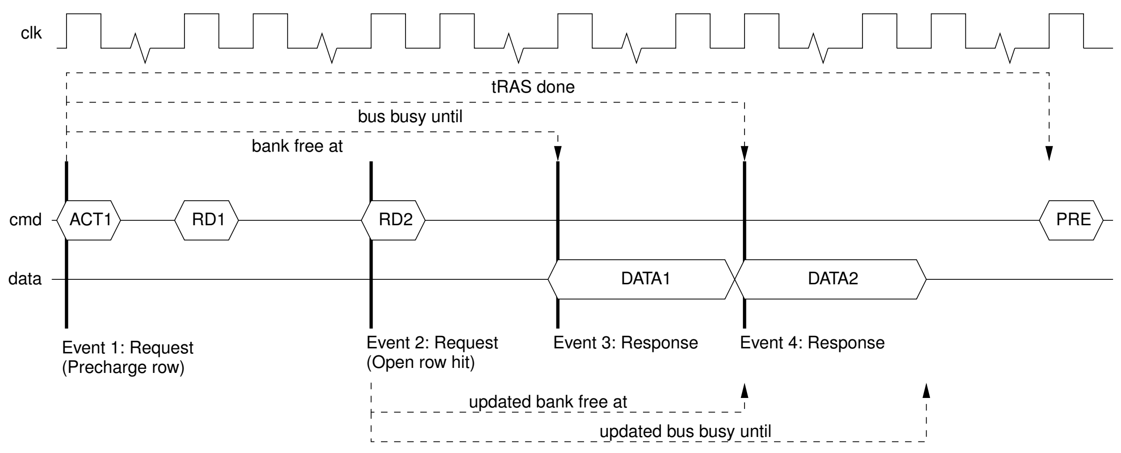


Fig. 2. Timing diagram illustrating the technique employed by our model.

frameworks, in this case gem5. However, our proposed technique is not unique to gem5, and can be applied to any discrete-event simulator with minimal effort. The speed benefit of an event-based model is impressive, as we shall see in Section III. The key challenge is to identify and capture only the relevant events.

Our model captures a (carefully chosen) subset of the DRAM bank state changes, along with the data bus occupancy, and the return of DRAM read data. The events are illustrated on a time line in Figure 2, along with the arcs that embody the timing constraints of the memory. First, we track when a bank is ready to execute a new command (possibly with auto precharge). Second, we record when a bank is able to precharge, i.e. when tRAS has been satisfied. Third, we track the earliest possible time a bank is allowed to be activated. With this information, we can approximate the DRAM state machine and respect the timings presented in Section II-B. In addition to the bank state, the model tracks the busy periods of the data bus, and knows when the bus becomes available. The model also tracks when a refresh is due. With the aforementioned information, we determine the earliest time the next scheduling decision must be made to not introduce any idle cycles on the data bus. Subsequently, we determine what latency an access incurs. Upon scheduling an access, we determine the time when response data is returned, and update the bank and data bus availability. As illustrated in the figure, most simulated cycles do not cause any state change and can be skipped. In addition, compared to cycle-based *memory-centric* models, our event-based *controller-centric* model is easily tuned to model new DRAM architectures, simply by changing model parameters. No code changes are necessary.

Next, we describe how our event-based controller is integrated in full-system simulator to form a complete architectural-exploration framework.

E. Integration

Our DRAM controller model is available as part of the open-source simulator gem5 [16], that already has a wide range of models for CPUs and devices. As such, gem5 boots unmodified Linux or Android, and runs full-system workloads for a variety of ISAs. In addition, gem5 has a range of models for on-chip interconnects, caches (Section II-F) and storage devices, similar to the environment described in [2].

Configuration of the memory controller and structural assembly of the system configurations is done using gem5's Python framework. Thus, the controller architecture parameters in Table I are specified like for any other gem5 component. Through the integration with gem5 we also re-use the elaborate statistics framework, allowing us to initialise, reset and output a large selection of performance-related numbers at arbitrary points in time. Together with existing performance visualisation tools [23], this enable us to study CPU, cache and DRAM behaviour on a common time line, combined with information about the OS and the running threads.

F. Interconnect

Our controller model uses the normal (transaction-level) gem5 port interface, with appropriate flow control to model blocking and back pressure. Thus, it can be connected to arbitrary locations in the memory system, e.g. to a crossbar or directly to a Last Level Cache (LLC). Moreover, the cache model in gem5 allows arbitrary configurations with a complex cache hierarchy, also offering a range of prefetchers. Together with a generic on-chip crossbar, this enables an extensive memory-system design space to be explored, much like [2].

We choose to use independent controllers for each channel, similar to actual controller implementations [17], and use gem5's crossbar model to do the address interleaving, as illustrated in Figure 1. The interleaving is done on arbitrary granularity, but by default uses either cache line or DRAM page granularity, depending on the address mapping scheme. The modularity and configurability makes it possible to model multi-channel UMA and NUMA configurations, or emerging heterogeneous memory systems. For example, a tiered memory is easily created by instantiating a WideIO and LPDDR3 DRAM, and a model of HMC [12] is only a matter of combining the crossbar model with 16 instances of our controller model (with the appropriate DRAM timing). Thanks to our fast event-based model, even a 16-channel memory system has limited impact of simulation performance.

G. Power modelling

To evaluate DRAM power, we use Micron's power model [24], in combination with simulation statistics (Section II-E). Our model collects information about the page hit rate, data bus utilisation for reads and writes, and tracks the time when all banks are precharged. This information is then used to calculate the DRAM power off-line. In combination with the Micron model, these statistics give us the power distribution for the specific DRAM.

Currently, we do not model the low-power states and associated timing constraints. Moreover, our model, similar to DRAMSim2 [9], does not model the DLL/PLL lock and wake up times which can be in the order of 500 cycles [1]. We plan to extend our framework with more detailed (and more generic) DRAM power models as power budget is an important design point for future memory architectures. Moreover, with the ongoing efforts to add power-modelling capabilities to gem5 [25], we hope to eventually add the ability to capture how the refresh rate varies with temperature.

III. Model validation

To validate our memory controller model, we compare its performance to DRAMSim2 [9] for a wide range of traffic scenarios. For a fair comparison we configure our model to match the timing parameters and scheduling policies of DRAMSim2. We use the same underlying DDR3 device (2 GBit, 8x8, 666 MHz) in a single rank and single channel organisation. As discussed in Section II-A, the read write queues in our architecture are distributed unlike the unified structure in DRAMSim2. To be fair on the queuing latencies, we match the queue sizes depending on the experiment, i.e. depending on the ratio between reads and writes. We set the frontend and backend latencies (see Table II-A) to zero to match DRAMSim2.

Our validation does not seek to ensure that our model captures every aspect of DRAM timing, or that it is exactly equivalent in behaviour to DRAMSim2. Rather, we seek to ensure that the system-level performance impacts of our (faster) model match those of the more detailed (and much slower) DRAMSim2 model. As already mentioned, the two controller models have intentional differences in architecture and policies, and differences in the results are therefore to be expected.

Next we describe how the controller is exercised, and explore the resulting timing behaviour and simulation speed.

A. Synthetic traffic generator

The gem5 simulation framework offers a number of traffic generators, either based on statistical behaviours or traces. Each synthetic traffic generator has a configurable read/write mix. In this study we use three types of generators: linear, random and *DRAM aware*. The linear generator produces bursts with a sequential address stream, while the random generator uniformly picks a random address for each burst, similar to [26]. The DRAM-aware traffic generator is created as part of this work, and is designed specifically for DRAM controller sensitivity studies.

The DRAM-aware traffic generator knows about the internal organisation of the DRAM, that is page size, number of banks and address mapping. As a result, it is possible to tailor the row-buffer hit rate to expose specific behaviours and timing constraints, e.g. tRCD, tCL, tRP, as well as targeting a particular number of banks enables to study bank utilisation and tRRD/tFAW timing implications. Additionally, by varying the read to write ratio, we see the impact of tWTR timing and various read/write switching schemes.

B. Test case formulation

To test the controller model, we generate different test cases by varying the access pattern, page policy, and read/write mix. For the (non-adaptive) open-page policy, we use the RoRaBaCoCh address mapping, as described in Table I. This mapping maximises page hits for sequential addresses. Correspondingly, for the (non-adaptive) closed-page policy, we use RoCoRaBaCh that maximises bank parallelism.

Next we show validation experiments done based on these test cases and the differences between DRAMSim2 and our memory controller.

C. Validation results

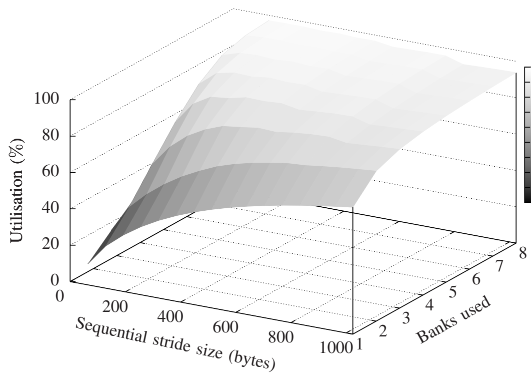
We compare bandwidth, latency, power and model performance for a range of test cases, as these are the key system-level metrics.

1) Bandwidth: We use our DRAM-aware traffic generator to sweep the sequential stride size from burst size to page size in steps of the burst size. The increment in stride size should increase the bus utilisation for the open-page policy, as it enables more row buffer hits. However, for the closed-page policy we expect a decrease in bus utilisation, as a longer stride inevitably leads to additional bank conflicts (accessing a row we just closed). In addition to the stride size, we also sweep the number of banks targeted by the generator. As the number of banks increase, the available parallelism should result in increased bus utilisation. We also expect the bank parallelism to be limited by tRRD and tFAW for smaller sequential strides.

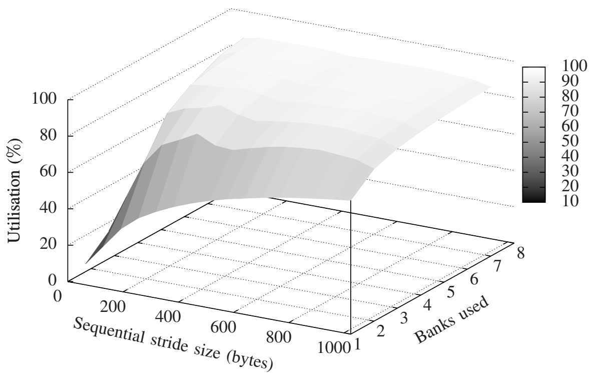
We expect the results to be similar for the two models, and as shown in Figures 3 through 5, they are to a first order the same. Looking in detail at the results, Figure 3 shows that with an open-page policy and only read traffic, both models achieve around 90% utilisation. The slope of the curve is close to identical for the two models, highlighting the impact of page hit rate and bank parallelism on DRAM utilisation. As expected, we observe similar trends for write traffic (left out for space reasons).

Figure 4 shows a mixed traffic pattern with 1:1 read to write ratio. Once again the difference in utilisation is very minor. For this case, the result is not so obvious, as DRAMSim2 intersperses the reads and writes to the same page, where as our model buffers the writes and drains them at a later point. The result highlights that the benefit from an increased row hit rate is removed by increased read/write switching.

Next we study the impact of closed-page policy on these traffic patterns. Figure 5 shows a write-only traffic pattern. The utilisation in our model decreases with increasing stride size, in agreement with DRAMSim2. This trend highlights

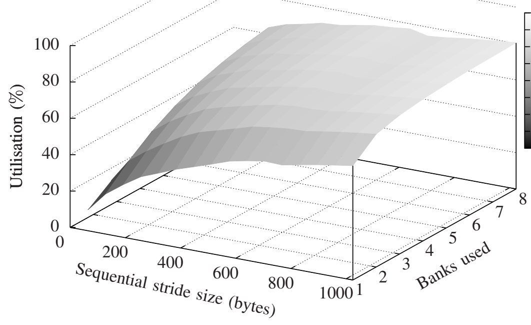


(a)

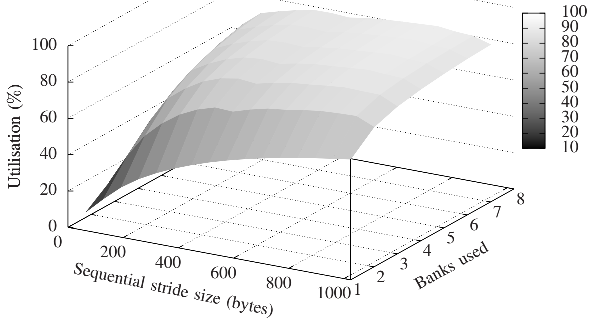


(b)

Fig. 3. Read traffic with open-page policy for (a) our model and (b) DRAMSim2.



(a)



(b)

Fig. 4. Mixed read/write traffic with open-page policy for (a) our model and (b) DRAMSim2.

how unsuitable the non-adaptive closed-page policy is for the sequential traffic pattern. Bank-level parallelism improves the utilisation for both the models. It is interesting to note that DRAMSim2 has around 15% lower utilisation than our model as the number of banks utilised increases. This low utilisation is attributed to the way we handle writes in our model. We wait for the number of outstanding writes to hit a threshold before servicing them, which gives us a wider window to reschedule the writes to leverage parallelism in the memory system.

Note that for read-only and mixed traffic patterns using a closed-page policy, bandwidth utilisation correlates well with DRAMSim2. We omit these results due to space constraints.

2) Latency: We use linear and random traffic generators to study the variation in latency that the two models offer. Note that latency plots are only shown for read latencies as in both the controller models, write requests are immediately acknowledged and are later handled internally by the controller. The latency is measured from the traffic generator, and includes the on-chip memory system and any queuing and serialisation.

Figure 6 shows the latency distribution for read-only linear traffic using an open-page policy. As expected, the latency distributions correlate well between the two models. We again omit results for the closed-page policy, which also correlate well. Figure 7 shows an interesting distribution for mixed read/write linear traffic pattern using a closed-page policy. The distribution for our model is bimodal in contrast to that of DRAMSim2. We observe this bimodal behaviour because of the difference in read/write scheduling policy we employ. In our model we reorder reads and writes with respect to each other. Writes are buffered and are only issued once the write queue fills beyond its high-water mark, after which a minimum number of writes are drained. This policy leads to the bimodal read latency distribution seen for our model; some reads are delayed while the write queue drains while others are serviced immediately. As DRAMSim2 does not reorder writes in this way, it does not exhibit this bimodal latency distribution.

Overall, for all synthetic workloads we observed the expected behaviour from our model. The distribution in latencies averages out and the difference in average latency we see is within 1% of that of DRAMSim2.

3) Power: For all the tests we expect power numbers from both the models to correlate well as both of them use the Micron power model. Indeed, we observe a maximum difference of 8% and an average of 3% while comparing the

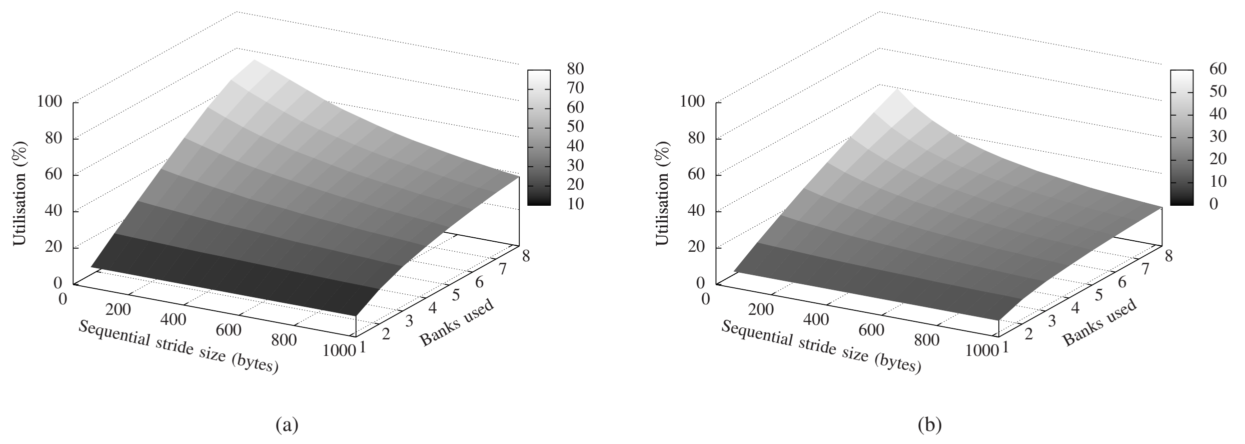


Fig. 5. Write traffic with closed-page policy for (a) our model and (b) DRAMSim2.

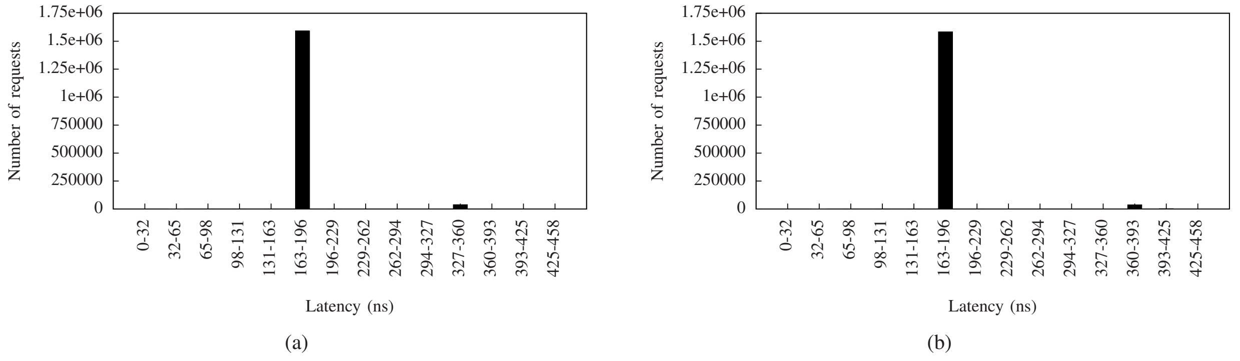


Fig. 6. Access latency distribution for linear read traffic with an open-page policy for (a) our model and (b) DRAMSim2.

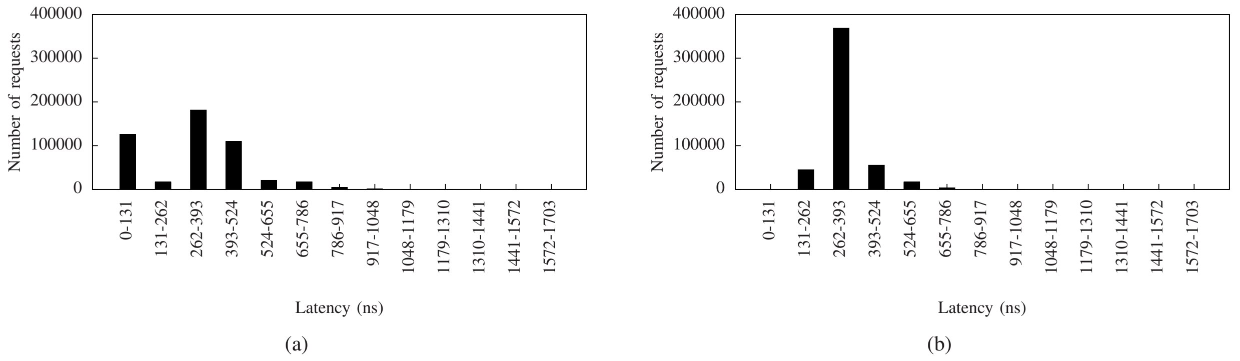


Fig. 7. Access latency distribution for linear read/write traffic with an closed-page policy for (a) our model and (b) DRAMSim2.

power numbers for all test cases. These minor differences are because of the differences in the controller architecture and scheduling policies as discussed in Section II-A and II-C respectively.

D. Model performance

State-of-the art DRAM controller models are based on cycle-by-cycle modelling of the DRAM. A key insight to our work is that by selecting only critical timing parameters, we build an accurate DRAM controller model, while eliminating most of the unnecessary state transitions. This technique results in a faster and simpler DRAM controller model. For the synthetic traffic studies, we see as much as 10x improvement in simulation speed with an average improvement of 7x across all synthetic workloads.

It is important to note that, even though our model is significantly faster than DRAMSim2, similar simulation speed ups cannot be expected for full-system simulations, as most of the time is spent simulating complex out-of-order cores. However, for a faster CPU simulator, e.g. a trace-based simulator, the difference is much bigger [13]. Similarly, for a modern 16-channel DRAM sub-system like HMC, we are seeing an

TABLE II. CPU CONFIGURATION

Core	2 GHz OoO 6-wide Dispatch, 8-wide Commit 40-entry ROB
I-Cache	32 kByte, 2-way, 64 Byte 1 ns cycle hit latency, 2 MSHRs
D-Cache	64 kByte, 2-way, 64 Byte 2 ns hit latency, 6 MSHRs
L2-Cache	512 kByte, 8-way, 64 Byte 12 ns hit latency, 16 MSHRs

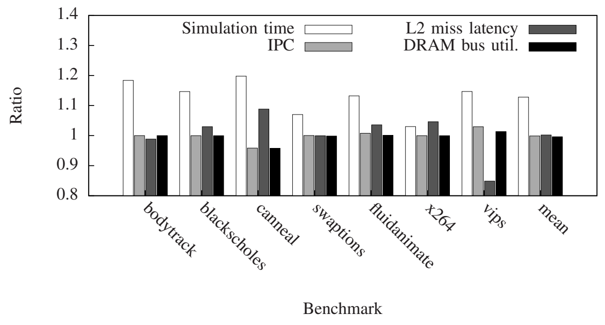


Fig. 8. Comparison between our model and DRAMSim2.

TABLE III. CONTROLLER CONFIGURATION

Write Buffer size	20
Read Buffer size	20
Write Buffer High/Low threshold	70%/50%
Scheduling Policy	FR-FCFS
Address Mapping	RoRaBaChCo
Page Policy	Open

TABLE IV. DRAM TIMING PARAMETERS

Timing Parameters	DDR3	LPDDR3	WideIO
Device Bus Width (bits)	64	32	128
Burst Length	8	8	4
Row Buffer Size (kByte)	1	1	4
Devices per Rank	8	1	1
Ranks per Channel	1	1	1
Banks per Rank	8	8	4
tRCD (ns)	13.75	15	18
tCL (ns)	13.75	15	18
tRP (ns)	13.75	15	18
tRAS (ns)	35	42	42
tBURST (ns)	5	5	20
tRFC (ns)	300	130	210
tREFI (us)	7.8	3.9	3.9
tWTR (ns)	7.5	7.5	15
tRRD (ns)	6.25	10	10
tXAW (ns)	40	50	50
Activation Limit	4	4	2

order of magnitude difference even with the detailed CPU core model. This clearly demonstrates the value of our approach, making it essential for future system exploration.

E. Summary

Our experimental results validate that the proposed model gives the expected behaviour under different traffic patterns and controller policies. The comparison with DRAMSim2 showcases the similarities and differences in the controller organisation. The first order correlation with DRAMSim2 validates our hypothesis of pruning the unnecessary details and concentrating on the important events in the controller model. We also show that our controller model works together with the Micron power model and can be further extended to plug in other models like DRAMPower [27].

IV. CASE STUDIES

In this section, we first validate the fidelity of our model against DRAMSim2 using full-system simulations of workloads from the PARSEC benchmark suite [28]. Next, we demonstrate the flexibility of the model by evaluating a range of memory systems for a server workload. The CPU configuration used for the simulations is shown in Table II.

A. Comparison with DRAMSim2

Similar to Section III, we use a DDR3 memory configuration that is matched between the two models, and both employ a closed-page policy. Figure 8 shows the comparison for a range of different metrics, including high-level metrics like simulation time, IPC and also DRAM-centric low-level metrics like average L2 miss latency and DRAM bus utilisation. Each bar in the graph shows the ratio of values observed with DRAMSim2 and our model for each of the four metrics. A ratio of 1 indicates that the two models correlate perfectly, while any deviations show that the models vary.

As evident from the figure, both the models correlate well with each other over the different metrics. However, the results show that our model reduces the overall simulation time by as much as 20% and by 13% on average. The few differences seen in Figure 8 are due to the different design choices made by the two models (write handling, split read-write queues, etc) explained in Section II. Overall, both the models correlate almost perfectly, highlighting the fact that our model significantly reduces simulation time without any loss in the fidelity of the results.

B. Future System Exploration

In this section, we present a case study of how our model can be used to analyse different memory technologies. One of the key contributions of our model is that by moving to a controller-centric approach, it allows for comparing different memory technologies without making changes to the model. As a case study, we compare three different memory technologies: DDR3 (1x 64-bit channel), LPDDR3 (2x 32-bit channels) and WideIO (4x 128-bit channels), all offering 12.8 Gbyte/s for "canneal" running on 16 cores (each core is configured in the same way as mentioned in Table II) with a shared LLC (8 MByte). This configuration is tailored to showcase the differences of the memory configurations. The architectural parameters of the controller is in Table III, and the various DRAM timing parameters for each of the memories are presented in Table IV.

Figure 9(a) shows the execution times observed when employing the different memory technologies. The difference in execution time is entirely due the varying access latencies of the three technologies. Figure 9(b) shows the breakdown of DRAM access latency for each. WideIO's high access latency is due to its relatively high bank access timing parameters (tRCD, tRP, etc from Table IV). Another interesting observation from the figure is the high queuing latency seen with the LPDDR3. This is a result of its low device bus width, which requires every DRAM request to be broken into two bursts.

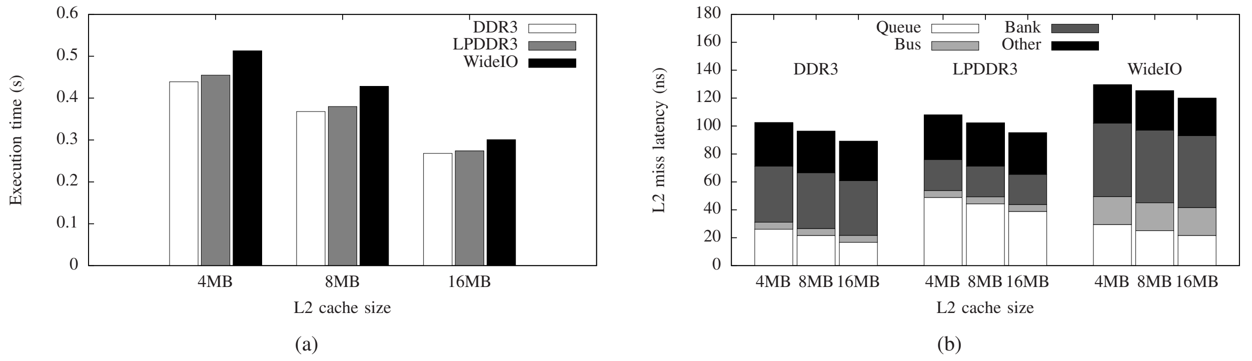


Fig. 9. Memory sensitivity and latency breakdown for a 16-core run of canneal.

Thus each request ends up seeing the queuing latency of two accesses. This also implies that the second burst always ends up causing a row buffer hit, reducing the average bank latency.

These case studies show that our model compares favourably to DRAMSim2 in modelling memory system behaviour while reducing simulation time and being flexible enough to model and reason about different memory technologies.

V. Related work

DRAMs have large impact on system power and performance [2], [4]. It is critical to study the entire memory system, including caches, DRAM and storage [2]. Studying various memory technologies and their differences is an important aspect in designing future system architectures [4], [1]. Previous works [6], [8], [9] show that overly simplistic memory models give misleading performance results, calling for more detailed models than fixed-latency or queuing models. We address all the aforementioned points with our accurate and flexible DRAM model, available as part of a full-system simulator.

A vast body of research improves the system performance and/or power through advanced DRAM scheduling [3], [11], [22]. These works span a diverse set of use-cases and system architectures, ranging from heterogeneous System on Chips to Chip Multi-Processors. Furthermore, some studies take DRAM optimisation one step further, and change either the physical organisation [4], [29] or the interface [30]. Our extensible and fast model serves as a starting point for future studies targeting critical system design points.

A number of DRAM controller models, summarised in Table V, span a large design space ranging from cycle-based models [9], [14], [15] to purely analytical models [6], [7]. DRAMSim2 [9] is the most prominent related work and we use it for our comparison studies. As already discussed, it is a cycle-based model that captures the timing constraints of a DDR2/DDR3 DRAM. DrSim [14] is similar in many aspects. However, DRAMSim2 stands out by offering the possibility to verify the timing constraints against a Verilog DRAM model. These cycle-based models inevitably add unnecessary detail that is not needed to accurately capture the impact on system-level performance. This trade-off negatively affects flexibility and simulation speed.

A vast body of research improves the system performance and/or power through advanced DRAM scheduling [3], [11], [22]. These works span a diverse set of use-cases and system architectures, ranging from heterogeneous System on Chips to Chip Multi-Processors. Furthermore, many works take DRAM optimisation one step further, and change either the physical organisation [4], [29] or the interface [30]. Our extensible and fast model serves as a starting point for future studies pursuing similar techniques.

TABLE V. Overview of DRAM controller models

Model	Speed	Full-system	Power	License
[9]	Cycle-based	gem5, MARSS	Micron	BSD
[14]	Cycle-based	gem5	-	BSD
[15]	Cycle-based	-	Micron	CRAPL
[6]	Analytical	gem5	-	-
[7]	Analytical	-	-	-
[5]	Event based	-	-	-
This work	Event based	gem5	Micron	BSD

The other extreme end is represented by the analytical DRAM models in [6], [7]. These models completely abstract from the controller architecture, and instead rely on timing equations and address distributions to determine the (approximate) performance impact for a given traffic stream. As we have seen in Section II-B, these works provide good insight into the relative importance of the DRAM timing constraints, something we use as part of our modelling technique to create a fast and accurate controller model. However, they fail to capture the importance of complex decisions made in the DRAM controller and their corresponding trade-offs.

The transaction-level model in [5] aims at a more abstract model of the DRAM and the controller, adopting the concept of an event-based model, similar to this work. However, the events are based on actual and complex DRAM state transitions which model low level details as compared to the simplified but critical states transitions modelled in our controller as explained in Section II. As a consequence, their work is significantly slower than what is possible with a more high-level abstract model. Furthermore, the model in [5] is limited by the backend of the controller being tightly coupled to the details of DRAM, not allowing for any exploration of the actual memory controller or system architecture.

Our DRAM controller is an abstraction model between the cycle-based detailed models and purely analytical models. In our controller architecure we have split read and write queues similar to [17], [18]. Our architecture is similar to the one proposed in [19], but different from the shared transaction queue in [9]. Our model combines an architectural model of the high-level controller blocks with carefully chosen events and delay calculations. As we have shown, this abstraction enables an accurate, fast model, well suited for system-level performance exploration. The integration with gem5 brings many benefits, including a full-system environment for memory system evaluation. However, the concepts are not tied to gem5, and can easily be implemented in any arbitrary discrete-event simulation framework, such as SystemC.

VI. Conclusions and future work

DRAM is playing an increasingly important role in determining system power and performance, whether used in a mobile phone or a server. Therefore, it is key to consider the impact of DRAM as part of future system architecture exploration. Overly simplistic memory models may mislead the design space exploration, whereas overly detailed models lead to inflexibility and poor simulation performance.

We show that it possible to create a fast, accurate and flexible model by capturing the key architectural building blocks of the controller, along with a subset of the state transitions of the DRAM. We compare our model to a state-of-the-art DRAM controller simulator for a selection of synthetic use-cases and a suit of full-system workloads. Our results show that the proposed model delivers the same system-level effects, while being 7x faster on average. We also demonstrate the flexibility of our model by evaluating three different memory systems for a multi-processor workload.

As part of our future work, we plan to add more detailed power modelling, and look at the thermal effects on DRAM.

References

[1] A. B. Kahng and V. Srinivas, “Mobile system considerations for SDRAM interface trends,” in *Proc. SLIP*, 2011.

[2] J. Stevens, P. Tschirhart, M.-T. Chang, I. Bhati, P. Enns, J. Greensky, Z. Chisti, S.-L. Lu, and B. Jacob, “An integrated simulation infrastructure for the entire memory hierarchy: Cache, DRAM, nonvolatile memory and disk,” *Intel Technology Journal*, vol. 17, no. 1, 2013.

[3] M. N. Bojnordi and E. Ipek, “PARDIS: A programmable memory controller for the DDRx interfacing standards,” in *Proc. ISCA*, 2012.

[4] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, “Towards energy-proportional datacenter memory with mobile dram,” in *Proc. ISCA*, 2012.

[5] M. Jung, C. Weis, N. Wehn, and K. Chandrasekar, “TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration,” in *Proc. RAPIDO*, 2013.

[6] H. Choi, J. Lee, and W. Sung, “Memory access pattern-aware DRAM performance model for multi-core systems,” in *Proc. ISPASS*, 2011.

[7] G. L. Yuan and T. M. Aamodt, “A hybrid analytical DRAM performance model,” in *Proc. PMBS*, 2009.

[8] S. Srinivasan, L. Zhao, B. Ganesh, B. Jacob, M. Espig, and R. Iyer, “CMP memory modeling: How much does accuracy matter?” in *Proc. MoBS*, 2009.

[9] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “DRAMSim2: A cycle accurate memory system simulator,” *IEEE Comput. Arch. Lett.*, vol. 10, no. 1, 2011.

[10] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.

[11] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, “MISE: Providing performance predictability and improving fairness in shared main memory systems,” in *Proc of HPCA*, 2013.

[12] J. T. Pawlowski, “Hybrid memory cube (HMC),” in *Proc. Hot Chips*, 2011.

[13] D. Sanchez and C. Kozyrakis, “ZSim: fast and accurate microarchitectural simulation of thousand-core systems,” in *Proc. ISCA*, 2013.

[14] M. K. Jeong, D. H. Yoon, and M. Erez, “DrSim: A platform for flexible DRAM system research,” http://lph.ece.utexas.edu/public/DrSim.

[15] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, “USIMM: the utah simulated memory module,” University of Utah, Tech. Rep. UUCS-12-002, 2012.

[16] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Arch. News*, vol. 39, no. 2, 2011.

[17] “Cadence design IP: Wide-I/O controller,” Cadence Design Systems, Inc., Tech. Rep., 2013.

[18] “AMBA LPDDR2 dynamic memory controller DMC-342 technical reference manual,” ARM, Ltd., Tech. Rep., 2009.

[19] N. Goran, “A preliminary exploration of memory controller policies on smartphone workloads,” Master’s thesis, University of Toronto, 2012.

[20] “DDR3 SDRAM,” JEDEC, Tech. Rep. JESD79-3F, 2010.

[21] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” in *ACM SIGARCH Computer Arch. News*, vol. 28, no. 2, 2000.

[22] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, “ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers,” in *Proc. HPCA*, 2010.

[23] D. Sunwoo, W. Wang, M. Ghosh, C. Sudanthi, G. Blake, C. D. Emmons, and N. Paver, “A structured approach to the simulation, analysis and characterization of smartphone applications,” in *Proc. IISWC*, 2013.

[24] “Calculating memory system power for DDR3,” Micron Technology, Inc., Tech. Rep., 2007.

[25] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras, “Introducing DVFS-management in a full-system simulator,” in *Proc. MASCOTS*, 2013.

[26] Y.-J. Kwon, “Parametrized DRAM model,” Master’s thesis, University of California, Berkely, 2010.

[27] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, “DRAMPower: Open-source DRAM power and energy estimation tool,” http://www.drampower.info.

[28] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, 2011.

[29] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, “Rethinking DRAM design and organization for energy-constrained multi-cores,” in *Proc. ISCA*, 2010.

[30] K. T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B. C. Lee, and M. Horowitz, “Rethinking DRAM power modes for energy proportionality,” in *Proc. MICRO*, 2012.