# Rethinking Cycle Accurate DRAM Simulation

Shang Li
shangli@umd.edu
University of Maryland, College Park

Rommel Sánchez Verdejo
rommel.sanchez@bsc.es
Barcelona Supercomputing Center (BSC)
Universitat Politécnica de Catalunya (UPC)
Spain

Petar Radojković
Barcelona Supercomputing Center (BSC)
Barcelona, Spain

Bruce Jacob
blj@umd.edu
University of Maryland, College Park

## ABSTRACT

Cycle accurate DRAM simulations have been the dominating architecture simulation model for DRAM for a long time. Although accurate, its poor simulation speed has not improved for years while a lot of other architecture simulators such as CPU and cache simulators have moved away from cycle-accurate models for better performance. In this paper, we discuss limitations of cycle-accurate DRAM models, through simulation experiments, we show that cycle-accurate DRAM simulator is becoming a dominant part of overall simulation time when paired with modern CPU simulators. We also demonstrate the inherent inflexibility of cycle-accurate models becomes the roadblock for faster simulation speed and integration with other non-cycle-accurate simulation frameworks. Finally, we discuss alternative modeling techniques for DRAM simulation and point out potential pathways to further DRAM simulation technique.

## CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; *Massively parallel and high-performance simulations*; *Simulation evaluation.*

## KEYWORDS

DRAM Modeling, Cycle Accurate Simulation, Architecture Simulation

**ACM Reference Format:**
Shang Li, Rommel Sánchez Verdejo, Petar Radojković, and Bruce Jacob. 2019. Rethinking Cycle Accurate DRAM Simulation. In *Proceedings of the International Symposium on Memory Systems (MEMSYS '19), September 30-October 3, 2019, Washington, DC, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3357526.3357539



## 1 INTRODUCTION

Architecture simulation is an essential method for researching and developing new architectures and systems. Cycle-accurate simulation has been seen as the necessity for simulation accuracy. In recent years, however, the proliferation of many-core systems changed the landscape of simulations: On one hand, the simulation time to simulate a multi-core CPU grows linearly, or even superlinearly. On the other hand, many-core systems can potentially speed up simulations substantially if simulators are designed to be running in parallel. With the drive of both forces, CPU simulators have moved away from cycle-accurate simulation models in pursuit of simulation speed and scalability. We will talk more about these non-cycle accurate techniques in Section 2.

Long been the prevalent main memory media, the accuracy of DRAM simulation is crucial to the overall accuracy of the simulated system. Like CPU simulators used to be, DRAM simulators are dominantly cycle-accurate models. Often times cycle-accurate DRAM simulators are integrated with CPU simulators to provide accurate memory timings. With the CPU simulators moving away from cycle-accurate models so that they can run much faster, DRAM simulation speed starts to bottleneck the overall simulation speed. To demonstrate how much time is spent in the DRAM simulators, we run a set of benchmarks with two types of CPU models using the same DRAM simulator, and breakdown the simulation time based on the wall timers we planted in our code. Detailed simulation configuration will be described in Section 3. As shown in Figure 1, with cycle-accurate out-of-order (O3) CPU model, the DRAM simulator only accounts for 10% to 30% of the overall simulation time. But as we switch to a faster CPU model, the DRAM simulation time bloats to 70% to 80% of overall simulation time. Note that the DRAM simulator we use here is already the fastest cycle-accurate DRAM simulator available, which signifies this is a fundamental issue of the cycle-accurate model rather than an implementation issue. Also, these results are not limited to specific CPU simulator implementations, because CPU simulator running at a similar speed will produce similar amount of memory requests in the same time frame, and therefore the DRAM simulator will be under the same amount of workload and cost the same amount of time to run. Performance aside, some non-cycle-accurate CPU simulators still manage a way to work with cycle-accurate DRAM simulators. But the incompatibility causes accuracy issues, which we will further discuss in Section 3.2.

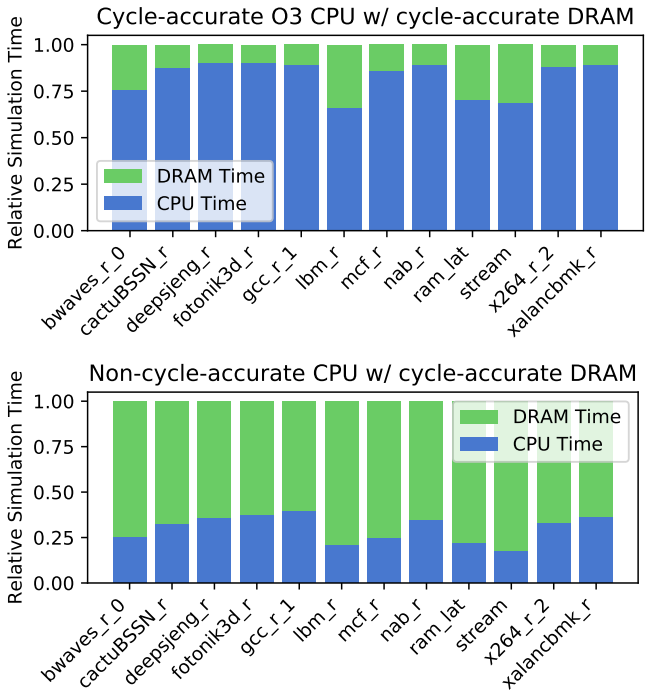



**Figure 1: Simulation time breakdown, CPU vs DRAM. Upper graph represents cycle-accurate out-of-order CPU model with cycle-accurate DRAM model. Lower graph represents modern non cycle-accurate CPU model. The DRAM simulators are the same in both graph.**

So, we believe it is time to review cycle accurate DRAM simulation, discuss its limitatiosn, and explore the alternative modeling techniques.

## 2 BACKGROUND & RELATED WORK

### 2.1 DRAM Modeling

Each DRAM cell operates like a capacitor: it can be charged, discharged and needs to be periodically refreshed to retain its value. DRAM cells form rows and columns in a bank that is a basic semi-independent operational unit, each bank has its sense amplifiers that amplify signals of DRAM cells for transmission on the external bus. Like capacitors, these operations on DRAM cells take time to finish, and thus imposing mandatory timing constraints on DRAM operations. For example, a *READ/WRITE* command needs to be at least *tRCD* cycles apart from previous row activation command *ACT*. Banks share a command and data bus, leading to another layer of constraints. The DRAM controller has the sole responsibility of bookkeeping these commands and the constraints to ensure timing correctness and no bus conflicts. On top of this, to maximize the performance and fairness, the controller also has the responsibility of scheduling the requests efficiently. Studies have shown properly designed scheduling algorithms can lead to huge performance gain[14]. Therefore, an accurate modeling of a DRAM controller should take into account of both correctness and scheduling performance.

Before cycle accurate simulators were adopted en masse, researchers used very simplistic models for DRAM simulations. For example, fixed-latency model assumes all DRAM requests take the same amount of time to finish, which completely ignores scheduling and queuing contentions that may cause significantly longer latency. There is also queued models that account for the queuing delay, but they fail to comply with various DRAM timing constraints and ignore the scheduling mechanisims that present in real controller designs. Previous study[18] has shown that such simplistic models suffer from low accuracy comparing to cycle-accurate DRAM models.

Then came along cycle accurate DRAM models, such as [3, 6, 8, 16, 21] and DRAMsim3[10]. These cycle-accurate DRAM simulators usually have validated DRAM timings, and provide non-trivial scheduling. But as we have shown, they start to negatively impact the simulation performance.

Other than cycle accurate models, there are also event based models such as[5, 7]. Event based model is more efficient than cycle-accurate model when the event or state updates are less frequent than a cycle-by-cycle basis. To get as much event sparsity as possible, event based models typically does not enforce all DRAM timing constraints, or simplify scheduling, which may result in accuracy or scheduling performance loss. Just as [7] point out, when memory workloads gets more intensive, the simulation performance of event based models will eventually come close to cycle-accurate models. In this study we also tested one event-based DRAM model and its accuracy is less than ideal comparing to cycle-accurate models.

Finally, there are analytic DRAM models such as [4, 22]. [4] presents a DRAM timing parameter analysis but does not provide a simulation model. The model in [22] provides predictions on DRAM efficiency instead of per-access timing information. These analytic models provide insights on the timing parameters and high-level interpretations, but have limited usage comparing to cycle-accurate models.

### 2.2 CPU Simulation Techniques

While this work primarily focuses on DRAM simulation and modeling, it is also very important to know about how CPU simulators have improved over time. Because CPU simulators are usually the "driver" for DRAM simulators, and DRAM simulators can certainly take lessons from CPU simulators on how to balance accuracy and simulation speed.

Traditionally, to achieve simulation fidelity, CPU simulators are designed to be cycle-accurate, meaning that just like real processors, the simulator states change cycle by cycle, and during each cycle, the microarchitecture of CPU (and cache) is faithfully simulated. Other simulation components such as DRAM simulators or storage simulators also synchronize with the CPU simulator every cycle. While simulating all the microarchitecture details achieves good accuracy, the The downside of this approach is the simulation speed is very slow, especially when CPUs are getting more and more cores and deeper cache hierarchy. Simulations can easily take days sometimes even weeks to finish.

A lot of techniques are explored to accelerate cycle-accurate simulations, for instance, checkpointing, which saves the simulator and program state at certain point to a file and allows the simulator to recover from that checkpoint later with the exact same state. This is mostly used to skip the warmup period and make sure simulations start at the same state. Similarly, some CPU simulators use a simpler, non-cycle-accurate model to fastforward the simulation to a warmed-up state and then switch to cycle-accurate model for further simulation.

Some researchers such as [13] take a statistic approach, which instruments and sample the simulated workload, uses statistic methods to identify distinctive program segments, and then extract these distinctive segments for future simulation. The extracted segments, which are typically called simulation points, can be then simulated with a cycle-accurate simulator. This way, the simulation time is cut short by simulating fewer instructions, instead of improving the simulator itself.

More recently, CPU researchers are moving away from the strictly cycle-accurate model due to its scalability issues. For example, SST, Graphite[11, 15] and Gem5[1] Timing CPU Model employ One-IPC model, meaning that every instruction is one cycle in the pipeline. Sniper[2] and ZSim[17] use approximation models for IPC which allows them to simulate out-of-order pipeline with relatively faster speed. Another benefit of applying this approximation model is that CPU cores and caches can be efficiently simulated in parallel, which allows multi-core even many-core CPU simulation applicable with decent scaling efficiency.

## 3 EMPIRICAL STUDY

In this section we setup our simulation framework to quantitatively evaluate DRAM models on simulation speed and accuracy. Table 1 shows our simulation setup.

**Table 1: Simulation Setup**

| | |
|---|---|
| Core Models | Gem5 Timing CPU (IPC=1) 4GHz<br>Gem5 O3 CPU, 4GHz 8-issue |
| L1 I-Cache | private, 32KB, 4-way associative,<br>64 Byte cache line, LRU |
| L1 D-Cache | private, 64KB, 4-way associative,<br>64 Byte cache line, LRU |
| L2 Cache | private, 256KB, 8-way associative,<br>64 Byte cache line, LRU |
| L3 Cache | shared, MOESI protocol, 2MB,<br>16-way associative, 64 Byte cache line, LRU |
| Main Memory | DRAMsim3: DDR4-2400, HBM<br>Event based Model: DDR4-2400, HBM |
| Benchmarks | A representative subset of SPEC CPU2017<br>STREAM<br>LMBench-like latency benchmark (*ram_lat*) |

We choose Gem5 not only because of its reputation in accuracy, but also because it supports multiple CPU models and DRAM models and can be easily swapped. This allows us to directly compare two different models, whether they're CPU models or DRAM models, while keeping all other components of the simulation the same. And therefore we can fairly compare and evaluate different models.

We have two CPU model choices here. First is out-of-order (O3, or DerivO3) CPU, that faithfully simulate the details of the core architecture, but only simulate at the rate of tens of thousands instructions per seconds on the host machine. The other is Timing CPU model, this is an One-IPC core model, which does not offer core microarchitecture simulation, but runs more than 10 times faster than O3 CPU model. Note we only use this Timing CPU model for simulation speed experiments, in which case it represents other CPU simulators runs at similar rate. For all accuracy evaluations, we use O3 CPU as it is the most accurate and reliable choice we have.

For DRAM models, we use DRAMsim3 as the representative of cycle-accurate simulator, because it offers the best simulation speed, and it is also hardware validated. For event based model, we choose [5], because it is conveniently integrated into Gem5 and offers similar DRAM protocols to DRAMsim3 that allows us to directly compare against. The DDR4 configuration in both models are single channel, dual rank, and has the same timing parameters. The HBM configuration in both models are 8 channel and 128 bits wide each.

To test a wide range of memory characteristics, we use a subset of SPEC CPU2017 benchmarks that are most representative according to [12]. We also include *STREAM*, which is very bandwidth sensitive, and *ram_lat*, an LMBench-like memory benchmark that is latency sensitive. These benchmarks will show us the full spectrum of memory characteristics and behaviors.

### 3.1 Cycle-accurate DRAM Simulation Time

First we experiment how much simulation time is spent in DRAM simulator versus CPU simulator. The two subgraphs in Figure 1 was obtained by using O3 CPU model and Timing CPU model respectively and have the same DRAMsim3 HBM backend.

Note that because HBM has 8 channels, and each channel has an independent DRAM controller, and therefore it takes more time to simulate HBM than a regular 1 channel DRAM. To quantify how number of channels affects simulation time, we sweep 1, 2, 4 channels of DDR4 with Timing CPU and show the absolute simulation time in Figure 2.

It can be seen that even with only one channel of DDR4, the cycle-accurate DRAM simulator still accounts for an average 40% of overall simulation time with a minimum of 30% and a maximum of 56%. For two channel DDR4, DRAM simulation time ranges from 46% to 69% with an average of 53%. For 4 channels, the min, max and average number are 62%, 81% and 68% respectively. While these numbers are produced with a single simulated core, modern CPU simulators such as [2, 11, 15, 17] can utilize multiple host cores to simulate multiple simulated cores, making the core simulation time scalable, therefore we can still conclude that DRAM cycle-accurate simulation does not scale with regards to number of channels, and it takes a significant proportion of simulation time even with only 1 DRAM channel.

### 3.2 ZSim: A Case Study

Besides the poor simulation speed, cycle accurate DRAM simulation also poses compatibility issues when integrated with modern CPU simulators or frameworks, especially those that rely on parallel

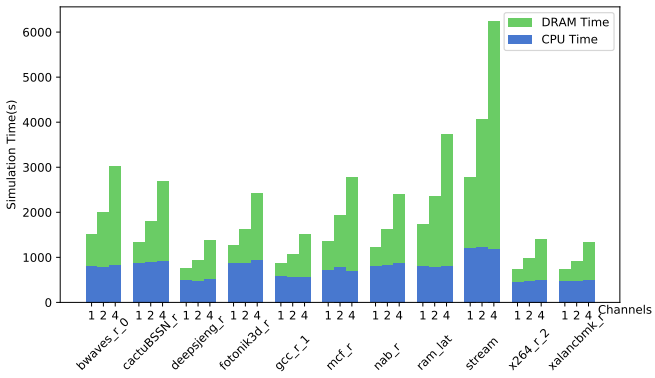



**Figure 2: Absolute simulation time breakdown of Timing CPU with 1, 2, and 4 channels of cycle-accurate DDR4. The bottom component of each bar represents the CPU simulation time and the top component is the DRAM simulation time.**

simulation for speed, as cycle accurate model requires synchronization every cycle, which will create huge overhead for parallel performance. For example, [2, 11] do not include a cycle accurate main memory backend at all. [17] supports cycle accurate memory backend, but as we will see soon, it has its issues when integrating a cycle accurate memory backend.

The problem was first discovered by [20], who observed a memory latency error of about 20ns when they tested a memory latency benchmark. But [20] did not answer where this 20ns missing latency comes from as was suspecting the error came from the cycle accurate DRAM simulator or the NoC latency that was not modeled. We will analyze this problem and provide a conclusive answer to this question. We will also illustrate other "side effects" we discovered along the way, such as the model incompatibility issue.

To replicate the issue independently, we developed a simplified version of LMBench(*ram_lat* we referred in Table 1) that randomly traverse a huge array, and measure the average latency of each access. When the array is too large to fit in the cache and most accesses go to DRAM, the average access latency will include the DRAM latency. The benchmark inserts timestamps before and after the memory traversal, and uses them to determine the overall latency of a certain number of memory requests, and divides the number of requests to obtain average memory latency. This average memory latency consists of cache latency and DRAM latency, and thus we use the term ***overall latency*** in the following discussion.

Like [20], we ran this benchmark natively on our machine to obtain "hardware measured" latency(72ns), then ran it in ZSim along with DRAMSim2 as DRAM backend, and we were able to reproduce similar results as [20]. That is, the *overall latency* (43ns) is 29ns lower than hardware measurement (72ns). To determine whether this is a ZSim specific issue or DRAM simulator issue, we ran the same benchmark in Gem5 with the same cache and DRAM parameters, and this time, the *overall latency* is 78ns, much closer to our hardware measurement. So we conclude this is a ZSim specific issue not a DRAM simulator issue. We then further looked into the simulator statistics, and found that the DRAM latency reported by the DRAM simulator in Gem5 is 55ns, which makes sense as the *overall latency* (78ns) should be a combination of DRAM latency (55ns) and cache latency (23ns). However, in ZSim, the DRAM latency reported by the DRAM simulator is 73ns, much higher than *overall latency*, which makes no sense. Figure 3a visualizes these results. This again confirms the issue lies within the ZSim memory model.

The way ZSim memory model works is, it has two phases of memory models, the first phase is an fixed latency model that assumes a fixed "minimum latency" for all memory events. The purpose is to simulate instructions as fast as possible, and generates a trace of memory events. After the memory event trace is generated, the second phase kicks in and that is when the cycle accurate DRAM simulator actually works, the cycle accurate simulation uses the event trace as input and update latency timings associated with these events.

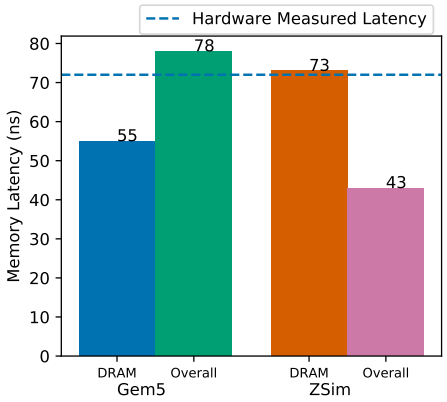
For instance, Figure 3b demonstrates how ZSim memory model handles memory requests differently from hardware/cycle accurate models. Suppose there are 3 back-to-back memory requests(each relies on the completion of previous one). In real hardware or a cycle accurate model, each memory request's latency may vary and next request cannot be issued until the previous request is returned. In ZSim Phase 1, all requests are assumed to be finished with "minimum latency", and therefore finish earlier than they should. Then in ZSim Phase 2, cycle accurate simulation is performed, more accurate latency timing is produced by cycle accurate simulator and all 3 requests update their timings. But even if all memory requests obtain correct timings in Phase 2, unfortunately, when the simulated program, like our benchmark, has instrumenting instructions such as reading system clock, it will obtain the timing numbers during Phase 1, which can be substantially smaller. This is why the *overall latency* is much smaller than DRAM latency.

So in other words, the "minimum latency" ZSim parameter will dictate the latency observed by the simulated program. To verify this claim, we run the same simulation with different "minimum latency" parameters, and plot them against the benchmark reported latency and DRAM simulator reported latency altogether, as in Figure 4.

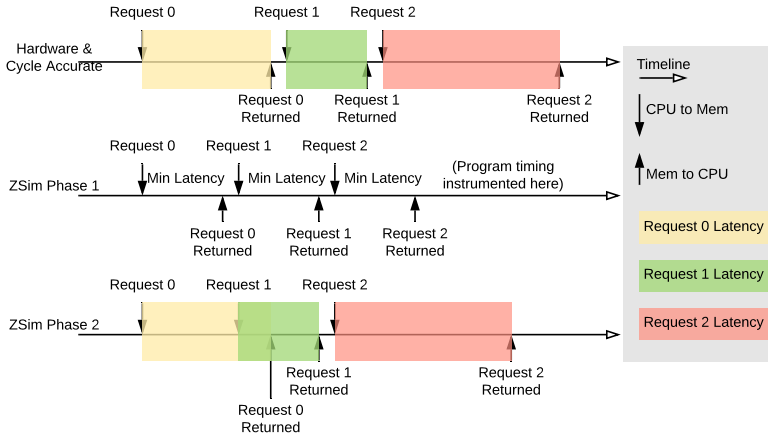
It can be seen in Figure 4 that, while we increase the "minimum latency" parameter, the overall latency pronounced by benchmark increases correspondingly, while the DRAM simulator reported latency keeps steady.

The reason that ZSim has to use a two-phase memory model is that it has to have a memory model that can give a latency upon first sight so that it can generate an event trace during an interval. Only model that is able to do so is fixed latency model but apparently it is not accurate enough and cannot handle dynamic contention and therefore ZSim requires a second, cycle accurate phase to correct the timings. In addition to this self-instrumenting errors, the broader issue is during the second phase, the memory requests received by the memory controller will have an inaccurate inter-arrival timing produced by Phase 1, which may alter the results of cycle accurate simulation results. In other words, the inaccuracy in Phase 1 can lead to further inaccuracy of Phase 2 memory simulation.

The root cause for the convoluted memory model of ZSim, and other fast simulator that do not support cycle accurate DRAM simulator is, cycle accurate DRAM simulator is no longer compatible




(a) DRAM latency and overall latency reported by Gem5 and ZSim.




(b) ZSim 2-phase memory model timeline diagram compared with real hardware/cycle accurate model. Three back-to-back memory requests (0, 1, 2) are issued to the memory model.

**Figure 3: Simulator memory latency analysis**

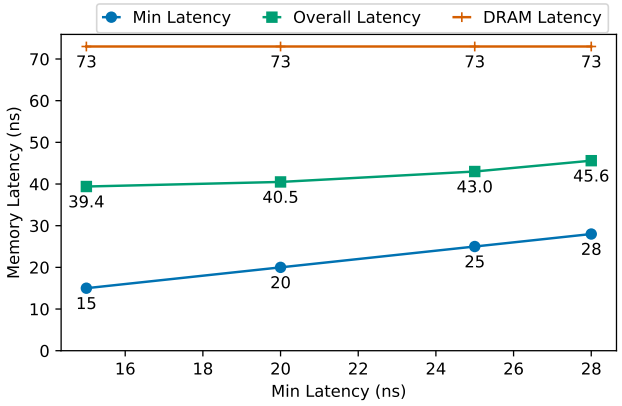



**Figure 4: Varying ZSim "minimum latency" parameter changes the benchmark reported latency, but has little to none effect on DRAM simulator.**

with these fast abstract simulation model, and there are yet no good alternatives that works with these abstract models.

## 3.3 Synchronization Overhead

With increasing channel-level parallelism of modern DRAM protocols and the logic independence of each channel, one would naturally think about using multi-threading to simulate these channels in parallel to speed up the simulation.

We optimize DRAMsim3 for parallel simulation so that there is no shared writable data structure among each channel simulated in parallel. We also use no more threads than number of channels simulated, and use low overhead thread scheduling to minimize the threading overhead. In our simulations, we use 8-channel HBM to hopefully have enough channel-level parallelism to start with. We run the simulation with single thread, 8 threads and 4 threads and then compare the overall simulation time. The results can be found in Figure 5.

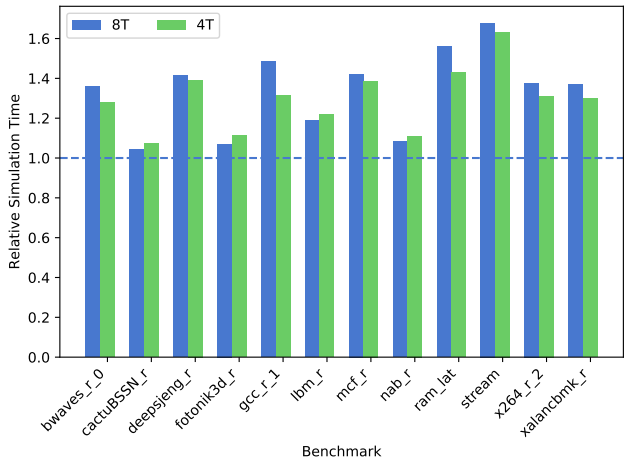



**Figure 5: Relative simulation time for 8-channel multi-threaded HBM simulation with 8 threads and 4 threads normalized to single thread simulation time.**

It can be seen that the multi-thread setups are, in all cases, slower than single thread version, in some cases it's even 1.6x slower. The reason is that the parallel region of the DRAM simulation only exists in each DRAM cycle, which has a such small granularity the overhead of doing thread synchronization weighs much more than the acceleration that can be brought by multi-threading.

Other than the limitation to multi-threading, another aspect of synchronization problem presents in cycle accurate model is the integration into other parallel simulation framework such as SST. As a simulation framework, SST can integrate individual component simulators (e.g. DRAMSim2) and provide an interface for each component to communicate with each other. Doing so allows SST to distribute simulated components to different cores or machines and simulate them in parallel. The implementation of the wrapper interface for DRAMSim2, for instance, treats each cycle of DRAM as an event. This means the simulation framework, when a cycle accurate DRAM simulator is present, has to synchronize with the DRAM simulator every single cycle, even if the synchronization event could be a costly MPI call over the wire. At this point it is hard to justify running the DRAM simulator in a separate thread or process in such simulation framework.

## 4 ALTERNATIVE MODELING TECHNIQUE

In Section 3, we quantitatively signified how cycle accurate models are holding back simulation performance, and becoming roadblocks to fit into modern simulation frameworks. In this section, we talk about alternative modeling techniques to cycle accurate DRAM simulation and how they may avoid these limitations.

### 4.1 Event Based DRAM Model

As we stated earlier, even based DRAM models typically offers better simulation performance than cycle-accurate models. But a general concern is the accuracy implication. To obtain a comprehension of event based model accuracy, we compare the event based DRAM model[5] included in Gem5 with DRAMsim3. Both simulators are integrated into the same Gem5 build so that we can conduct a fair comparison of same CPU, cache, and benchmark with only the DRAM model being different. For both DRAM models, we run all the benchmarks with a DDR4 profile and an HBM profile. The DDR4/HBM timing parameters are configured to the same in both DRAMsim3 and the event based model. The CPU model we use to evaluate accuracy is the Gem5 O3 CPU model, which provides deterministic, reproducible results. We use the CPI numbers obtained by DRAMsim3 backed simulations as baseline, and plot the relative CPI of event based simulations in percentage, shown in Figure 6.

The CPI difference ranges from 3% to almost 60% across all benchmarks. In general, less memory-intensive benchmarks tend to have lower CPI differences. The DDR4 event based model averages a 15% CPI difference and the HBM event based model averages a 28% CPI difference from their cycle accurate counterparts. While we cannot conclusively say the difference in CPI translates to inaccuracy as the event based model implements different scheduling policy for the controller, the CPI difference is way higher than those between cycle accurate models. So even though event based model can be several times faster than cycle accurate models, one has to make sure the accuracy is acceptable for the kind of workload he or she wants to simulate.

### 4.2 Separating Interface with Implementation

While cycle accurate model provides excellent accuracy, the interface of a cycle accurate simulator does not have to be cycle by

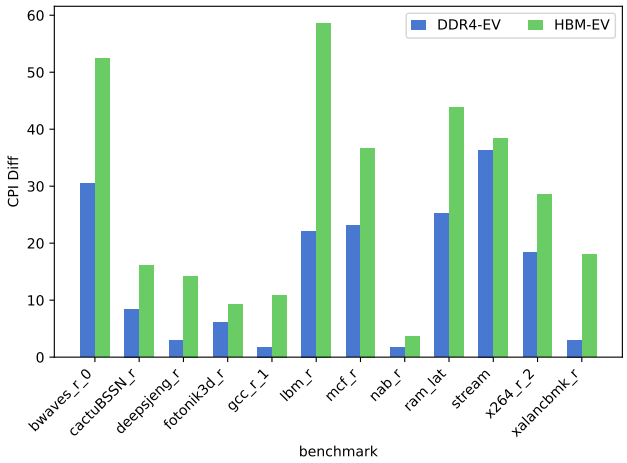



**Figure 6: CPI differences of an event based model in percentage comparing to its cycle-accurate counterpart. DDR4 and HBM protocols are evaluated.**

cycle based, and hence the concept of separating interface with implementation.

Separating the interface with the cycle accurate core can address a lot of the limitations discussed in Section 3. For example, when running the DRAM simulator along with another CPU simulator, we may be able to afford to run the DRAM simulator with a different thread on a different core because if the DRAM simulator does not have to synchronize with the CPU simulator so frequently, then running them concurrently can potentially be beneficial to the simulation performance.

The challenge here is how to implement the non-cycle-accurate interface. One simple but effective approach, as shown in [9], is to use a relax synchronization mechanism, which is to synchronize the CPU with DRAM simulator *every few cycles* instead of every cycle. Essentially this provides the CPU simulator a slower interface than the cycle accurate DRAM simulator. As shown in [9], synchronizing every 8 DRAM cycles will only have an average of less than 1% of accuracy loss in terms of CPI, but can speed up the simulation by 40% on average when the cycle accurate backend is running in parallel.

A more complicated approach would be an event based, or dynamic synchronization interface. That is, the DRAM and CPU simulator have to agree on the next synchronization point before they resume their own simulation. The challenge here is to develop the method that determines the next synchronization point: the next synchronization point, ideally, should be far enough so that it compensates the synchronization overhead, but not too far for either simulator to enter a irreversible state that causes unacceptable inaccuracy in simulations. For example, a irreversible state change could mean DRAM opening or closing a page, and say if there is a memory request from CPU that is supposed to be a page hit, but because the DRAM and the CPU did not synchronize before the DRAM controller decides to close that page, then that memory request will become a page miss, and thus producing more latency

than it should be. It also means the implementation of the DRAM simulator may need to be able to "fast forward" some requests to compensate the the delay these requests suffered when they were waiting for the synchronization on the CPU side.

### 4.3 Statistical Models

Different from analytic models that provide a high level analysis we discussed in Section 2.1 the statistical models here means to provide a on-the-fly DRAM timing per request based on a "trained" statistical or machine learning model.

The foundation of why such a statistical model would work on DRAM is that:

- DRAM banks only have a finite number of states.
- The timing of each DRAM request has already been largely dictated by the DRAM states when it arrives at the controller.
- Our observation shows most DRAM request latency fall into very few latency buckets, indicating it is likely the result of previous two points.

And we will explain/verify each of the claim one by one as following.

**DRAM banks only have a finite number of states:** a DRAM bank can be modeled as a state machine, it can be in idle, open, refreshing, or low power states. Although there are typically thousands of rows that can be opened or closed, what matters to a specific request to a bank is whether the row of that request is open or not, so it will reduce to 2 states in this regard. Similarly while there can be multiple banks in a rank and even multiple ranks in a channel, but for each request there is only a subset of these states that really matter to the timing of that request. Also, the queuing status when a new request arrives can also be accounted as states.

**The timing of each DRAM request has already been largely dictated by the DRAM states when it arrives at the controller:** intuitively speaking, when a request arrives at the DRAM controller, there are very limited actions for the controller can take. It either A) process this request, whether it's because it gets prioritized by the scheduler, or just because there is no other requests to be processed at the time, or B) hold the request whether it's because there are contention other events are happening such as the current rank/bank is refreshing. Most of the scenarios here can be represented as a "state" like we previously discussed.

**Our observation shows most DRAM request latency fall into very few latency buckets, meaning that they are likely to be predictable:** we plot the memory latency distribution of the 12 benchmarks we tested as Figure 7. We clip each histogram at the 99 percentile latency point for better visual. It can be seen that although every benchmark has a long tail latency that stretches to over 400 cycles (likely the results of having to wait for a refresh which is 420 cycles in this case), the 90-percentile line and the distribution itself indicates most of the memory latency are limited to quite a few latency buckets. Note that we are not claiming these few latency buckets translate to only a few unique latency values: in most cases, each bucket represents 10 cycles; there are also requests that are have low-count latency values, they are not obvious on the plots but are certainly there.

This distribution fits into a statistical or machine learning model very well: the majority of the cases are predictable while the corner cases are there to optimize. With a statistical or machine learning model, while we cannot handle 100% of the requests accurately like a cycle accurate simulator, but if we can accurately predict, say 90% of the requests at the cost of a fraction of simulation time, then the trade-off may be worth the accuracy loss, especially for CPU and cache researchers who only need a "accurate enough" but preferably much faster memory model.

There are early efforts such as [19] that tries to build memory controller models around the same idea. [19] treats the targeted memory controller as a "black box", and by observing and modeling the distribution of the memory latency, a statistical model can be built to simulate the targeted memory controller. However, [19] is not designed as an architecture simulator, and is not evaluated as such. We still need more concrete proof-of-concepts for the statistical idea.

## 5 CONCLUSION

In this study we empirically discussed the limitations of cycle accurate DRAM simulation models. We showed that while still being the most accurate model, cycle accurate DRAM models cannot keep up with the trend of architecture simulator development in terms of simulation performance and model compatibility. We further compared and explored other modeling techniques that are promising alternatives to cycle accurate models.

## REFERENCES


[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[2] Trevor E Carlson, Wim Heirmant, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[3] Niladrish Chatterjee, Rajeev Balasubramonian, Manjunath Shevgoor, Seth Pugsley, Aniruddha Udipi, Ali Shafiee, Kshitij Sudan, Manu Awasthi, and Zeshan Chishti. 2012. Usimm: the utah simulated memory module. *University of Utah, Tech. Rep* (2012).

[4] Hyojin Choi, Jongbok Lee, and Wonyong Sung. 2011. Memory access pattern-aware DRAM performance model for multi-core systems. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 66–75.

[5] Andreas Hansson, Neha Agarwal, Aasheesh Kolli, Thomas Wenisch, and Aniruddha N Udipi. 2014. Simulating DRAM controllers for future system architecture exploration. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 201–210.

[6] Min Kyu Jeong, Doe Hyun Yoon, and Mattan Erez. 2012. DrSim: A platform for flexible DRAM system research. *Accessed in: http://lph. ece. utexas. edu/public/DrSim* (2012).

[7] Matthias Jung, Christian Weis, Norbert Wehn, and Karthik Chandrasekar. 2013. TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration. In *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. ACM, 5.

[8] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2015. Ramulator: A fast and extensible DRAM simulator. *IEEE Computer architecture letters* 15, 1 (2015), 45–49.

[9] Shang Li. 2019. *Scalable and Accurate Memory System Simulations*. Ph.D. Dissertation. University of Maryland, College Park.

[10] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2019. DRAMsim3: A Cycle-accurate, thermal capable memory system simulator. *IEEE Computer Architecture Letters* (2019).

[11] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 1–12.

[12] Reena Panda, Shuang Song, Joseph Dean, and Lizy K John. 2018. Wait of a decade: Did spec cpu 2017 broaden the performance horizon?. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 271–282.

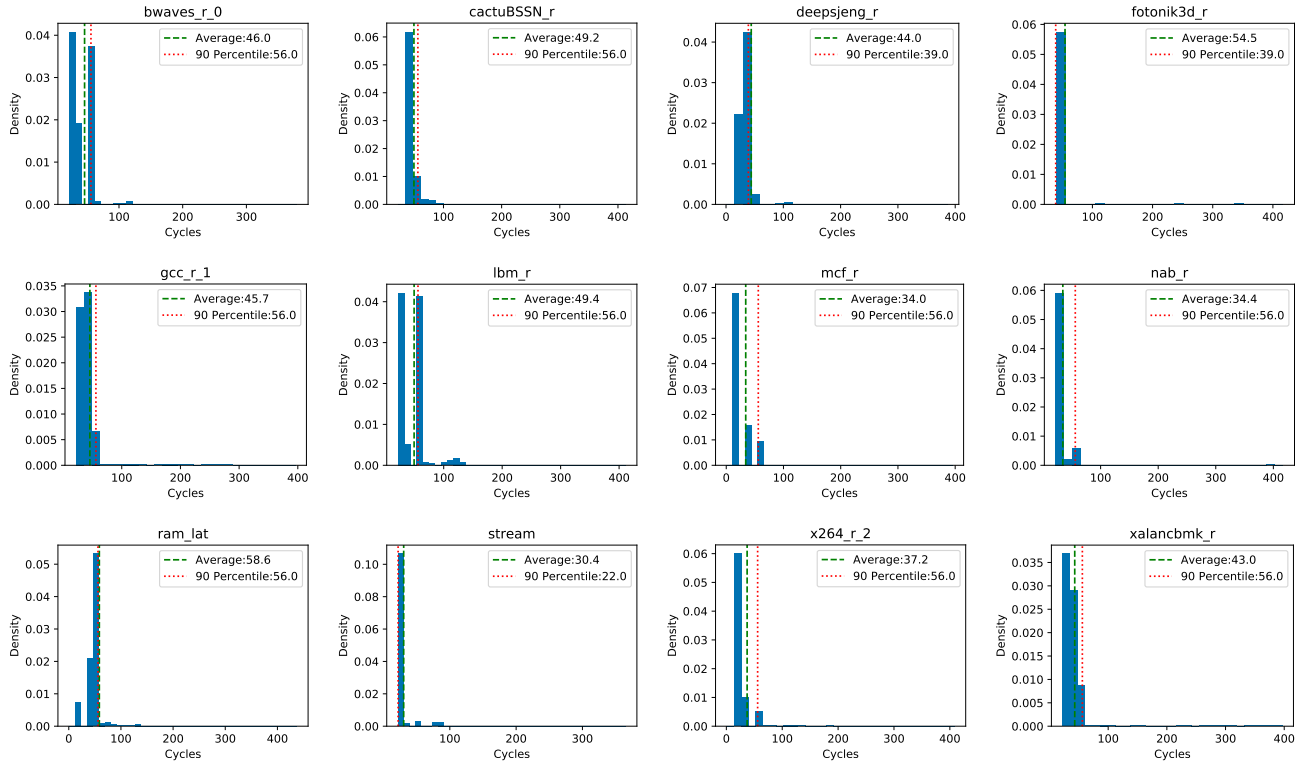



**Figure 7: Latency density histogram for each benchmark obtained by Gem5 O3 CPU and 1-channel DDR4 DRAM. X-axis of each graph is cut off at 99 percentile latency point, the average and 90-percentile point are marked in each graph for reference.**


[13] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. 2003. Using SimPoint for accurate and efficient simulation. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 31. ACM, 318–319.

[14] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. 2000. Memory access scheduling. In *ACM SIGARCH Computer Architecture News*, Vol. 28. ACM, 128–138.

[15] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. 2011. The structural simulation toolkit. *SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[16] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters* 10, 1 (2011), 16–19.

[17] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer architecture news*, Vol. 41. ACM, 475–486.

[18] Sadagopan Srinivasan, Li Zhao, Brinda Ganesh, Bruce Jacob, Mike Espig, and Ravi Iyer. 2009. CMP memory modeling: How much does accuracy matter? (2009).

[19] Vladimir Todorov, Daniel Mueller-Gritschneder, Helmut Reinig, and Ulf Schlichtmann. 2012. Automated construction of a cycle-approximate transaction level model of a memory controller. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 1066–1071.

[20] Rommel Sánchez Verdejo, Kazi Asifuzzaman, Milan Radulovic, Petar Radojković, Eduard Ayguadé, and Bruce Jacob. 2018. Main memory latency simulation: the missing link. In *Proceedings of the International Symposium on Memory Systems*. ACM, 107–116.

[21] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. 2005. DRAMsim: a memory system simulator. *ACM SIGARCH Computer Architecture News* 33, 4 (2005), 100–107.

[22] George L Yuan, Tor M Aamodt, et al. 2009. A hybrid analytical DRAM performance model. In *Proc. 5th Workshop on Modeling, Benchmarking and Simulation.*