# DRAMSys4.0: A Fast and Cycle-Accurate SystemC/TLM-Based DRAM Simulator

Lukas Steiner[1], Matthias Jung[2], Felipe S. Prado[1], Kirill Bykov[1], and Norbert Wehn[1]

[1] Technische Universität Kaiserslautern, Kaiserslautern, Germany
{lsteiner, wehn}@eit.uni-kl.de
[2] Fraunhofer IESE, Kaiserslautern, Germany
matthias.jung@iese.fraunhofer.de

**Abstract.** The simulation of DRAMs (Dynamic Random Access Memories) on system level requires highly accurate models due to their complex timing and power behavior. However, conventional cycle-accurate DRAM models often become the bottleneck for the overall simulation speed. A promising alternative are DRAM simulation models based on Transaction Level Modeling, which can be fast and accurate at the same time. In this paper we present DRAMSys4.0, which is, to the best of our knowledge, the fastest cycle-accurate open-source DRAM simulator and has a large range of functionalities. DRAMSys4.0 includes a novel simulator architecture that enables a fast adaptation to new DRAM standards using a Domain Specific Language. We present optimization techniques to achieve a high simulation speed while maintaining full temporal accuracy. Finally, we provide a detailed survey and comparison of the most prominent cycle-accurate open-source DRAM simulators with regard to their supported features, analysis capabilities and simulation speed.

**Keywords:** DRAM, Simulation, SystemC, TLM

## 1 Introduction

Since today's applications become more and more data-centric, the role of *Dynamic Random Access Memory* (DRAM) in compute platforms grows in importance due to its large impact on the whole system performance and power consumption. Over the last two decades, the number of DRAM standards specified by the *JEDEC Solid State Technology Association* has been growing rapidly. Because of the large variety of standards, system designers have to face the difficult task of choosing devices that match system requirements for performance, size, power consumption and costs best. A short time to market aggravates this choice and creates the need for DRAM simulation models that allow both fast and truthful design space exploration.

DRAM subsystems are composed of a DRAM controller and one or several DRAM devices. Although the DRAM standards define a framework of rules for the sequence and minimum time distance between DRAM commands, the controller still has some freedom on their placement and the scheduling of incoming requests. Different controller implementations exploit this freedom in order

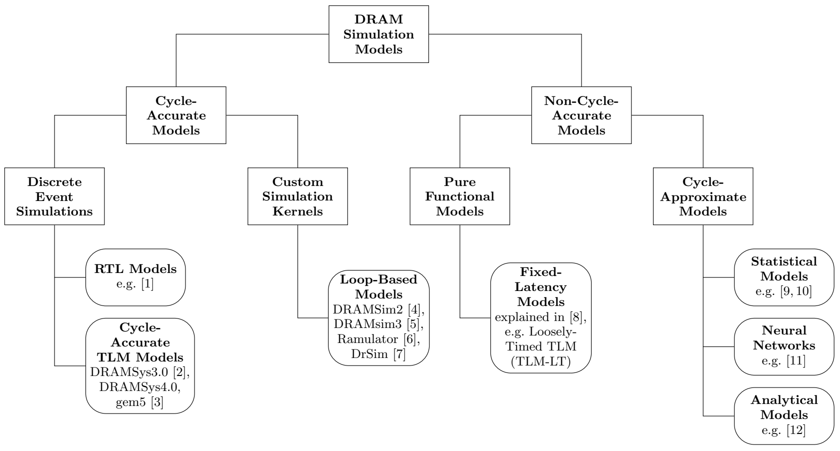

DRAM Simulation Models
Cycle-Accurate Models
Non-Cycle-Accurate Models
Discrete Event Simulations
Custom Simulation Kernels
Pure Functional Models
Cycle-Approximate Models
RTL Models e.g. [1]
Cycle-Accurate TLM Models DRAMSys3.0 [2], DRAMSys4.0, gem5 [3]
Loop-Based Models DRAMSim2 [4], DRAMsim3 [5], Ramulator [6], DrSim [7]
Fixed-Latency Models explained in [8], e.g. Loosely-Timed TLM (TLM-LT)
Statistical Models e.g. [9, 10]
Neural Networks e.g. [11]
Analytical Models e.g. [12]

**Fig. 1.** Different DRAM Simulation Models

to optimize for different metrics (e.g., latency, bandwidth, power). Therefore, a DRAM subsystem simulation represents (1) one specific DRAM controller implementation and (2) one specific DRAM standard. It can be performed on different levels of abstraction, each offering a certain trade-off between speed and accuracy. Figure 1 provides an overview of different models for DRAM subsystem simulations.

Non-cycle-accurate DRAM simulation models (right side) allow high simulation speeds but lack in accuracy. The simplest, pure functional model for a DRAM subsystem is a fixed-latency model [8]. Within this approach all request experience the same constant amount of latency and all subsystem internals are omitted. However in reality, the latencies of DRAM accesses vary between a dozen and several hundred cycles due to the complex device architecture. Therefore, a pure functional model is not useful for design space exploration and performance estimations. Cycle-approximate models try to mimic the latency behavior of real DRAM subsystems by utilizing e.g. statistical methods [9, 10] or neural networks [11]. Unfortunately, their accuracy can change from simulation to simulation, which makes them unsuitable for reliable performance estimations and design space exploration, too.

Cycle-accurate DRAM simulation models (left side) provide full temporal accuracy for truthful investigations, but usually take a lot more time to execute. RTL models of real DRAM controllers [1] can be simulated with the help of a *Discrete Event Simulation* (DES) kernel to represent a clock signal, trigger the execution of processes and model the hardware's concurrency. Most state-of-the-art cycle-accurate DRAM simulators, namely DRAMSim2 [4], DRAMsim3 [5], Ramulator [6] and DrSim [7], avoid the overhead of a DES kernel and use a simple loop to represent clock cycles. In addition, they do not model individual signals, which reduces the complexity and allows faster modifications. However, the processes of all these simulators as well as the RTL models are evaluated in each clock cycle, whether or not there are any state changes in the system. Consequently, the consumed wall clock time for a simulation grows linearly with the simulated time and is more or less independent of the memory access density (see Section 3). Thus, when coupled with modern CPU simulators, the cycle-accurate DRAM simulation even starts to become the bottleneck of the overall simulation speed for realistic workloads [13].

To reach higher simulation speeds while still providing full temporal accuracy, the design space exploration framework DRAMSys3.0 [2] uses the concept of *Transaction Level Modeling* (TLM) based on the SystemC/TLM2.0 IEEE 1666 Standard [14]. This approach also relies on a DES kernel, however, new events are not generated by a clock signal but only by the processes themselves. In this way the processes are only evaluated in clock cycles where state changes occur (see Section 2.1). As a result the simulation speed is heavily dependent on the memory access density and can be significantly higher than the speed of the abovementioned simulators. Up until now DRAMSys3.0 is not open sourced and does not support the latest DRAM standards. gem5 [3], an open-source full-system simulator, uses a similar TLM concept to speed up simulations. Major drawback of its provided cycle-accurate DRAM model [15] is a close link to the DDR3/4 standards, which leads to a reduced accuracy for simulations with other DRAM standards as shown in [13].

To the best of our knowledge, there exists no cycle-accurate open-source DRAM simulator that fully supports the latest DRAM standards, performs simulations in a speed that enables fast design space exploration, allows a direct coupling to other system components based on the state-of-the-art system-level modeling language SystemC or to the full-system simulator gem5, and offers enough capabilities for a holistic performance analysis.

In this paper we present DRAMSys4.0, a completely revised version of DRAMSys3.0 [2]. It supports the latest JEDEC DRAM standards (e.g., DDR4, LPDDR4, GDDR6 and HBM2), is optimized to achieve between 10× to 20× higher simulation speeds compared to its predecessor, and offers a large toolbox for analysis and validation. The framework will be open sourced on GitHub.[3]
In summary, this paper makes the following contributions:

– We present DRAMSys4.0, which is, to the best of our knowledge, the fastest open-source DRAM simulator with cycle accuracy. We present a novel simulator architecture that enables a fast adaptation to new DRAM standards and different DRAM controller implementations.
– We present a sophisticated approach to automatically generate the source code of this simulator for new standards from formal descriptions based on a Petri Net model. The same approach is used to validate the functional behavior of state-of-the-art simulators.
– We demonstrate how RTL descriptions of real DRAM controllers can be validated with DRAMSys4.0 by embedding them into the framework and by exploiting our analysis tools. To speed up the RTL simulation of the DRAM controller, we suppress unnecessary events by disabling the clock signal during idle phases.
– We provide a detailed survey and a fair comparison of the most prominent cycle-accurate open-source DRAM simulators with regard to their supported features and analysis capabilities (see Table 2). We also compare their simulation speed.

[3] https://github.com/tukl-msd/DRAMSys

The remaining paper is structured as follows: In Section 2 DRAMSys4.0 is presented, including all its functionalities for fast adaptation, result analysis and validation. Section 3 discusses related cycle-accurate simulators, provides a detailed comparison among them, and presents cycle-approximate approaches for a fast and accurate DRAM simulation. Section 4 concludes the work.

## 2 DRAMSys4.0

In this chapter we present DRAMSys4.0, which supports the latest JEDEC DRAM standards and is optimized to achieve much higher simulation speeds than the predecessor. More precisely, we present its architecture and functionality, discuss our optimizations to increase the simulation speed, and give an overview of the framework's unique features. Among them are the *Trace Analyzer* for visual and metric-based result analysis, the possibility for the co-simulation with RTL controllers, and the Petri-Net-based code generation and validation.

### 2.1 Functionality and Architecture

As mentioned in the introduction, DRAMSys[4] uses the concept of TLM based on the SystemC/TLM2.0 IEEE 1666 Standard [14] for a fast and fully cycle-accurate simulation. In accordance with the standard, all components are designed as SystemC modules (`sc_module`) and connected by TLM sockets. The simulator utilizes the *Approximately Timed* (AT) coding style, which defines a non-blocking four-phase handshake protocol.[5] A four-phase handshake is required to model the DRAM subsystem's pipelined behavior and out-of-order responses to the initiators. However, since a single memory access can cause the issuance of multiple DRAM commands depending on the device's current state (e.g., precharge (`PRE`) - activate (`ACT`) - read (`RD`)/write (`WR`) for a row miss), four phases are still not sufficient to model the communication between controller and device with full temporal accuracy. To close this gap, a custom TLM protocol (called DRAM-AT) that defines application-specific phases for all DRAM commands was introduced in [16]. These phases allow a projection of the cycle-accurate DRAM protocol to TLM.

The rule of thumb for making cycle-accurate simulations fast is to reduce the number of simulated clock cycles or events, respectively, and the control flow overhead that is executed. Therefore, DRAMSys only simulates the clock cycles in which state changes occur. Figure 2 shows an example for an `ACT` command and its timing dependency[6] $t_{RCD}$ to a following `RD` command. While all loop-based simulators would simulate ten clock cycles to issue both commands, DRAMSys only simulates the first clock cycle, notifies an event after $t_{RCD}$, and directly simulates the tenth clock cycle to issue the `RD` command. All clock cycles in between are skipped and the simulation time is fast-forwarded. Especially in

[4] DRAMSys without a version number refers both to DRAMSys3.0 and DRAMSys4.0.

[5] The TLM-AT base protocol consists of the phases `BEGIN_REQ`, `END_REQ`, `BEGIN_RESP` and `END_RESP`.

[6] Timing dependencies are temporal constraints that must be satisfied between issued DRAM commands.

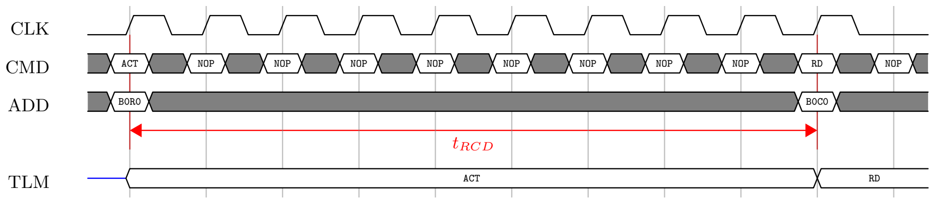


**Fig. 2.** TLM Implementation of the `ACT` Command [16]

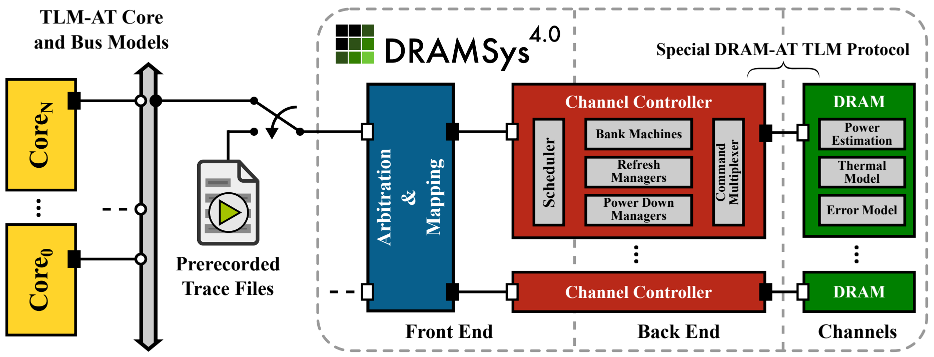


**Fig. 3.** Architecture of DRAMSys4.0

scenarios where the memory access density is low, this approach can lead to an enormous event reduction and a resulting simulation speedup of several orders of magnitude (see Table 1 and Section 3) while still yielding fully cycle-accurate results.

From an architectural point of view, DRAMSys4.0 consists like its predecessor of an arbitration & mapping unit (short arbiter) as well as independent channel controllers and DRAM devices for each memory channel, shown in Figure 3. The arbiter cross-couples multiple initiators and DRAM channels on the basis of a predefined address mapping. It is followed by independent channel controllers for each DRAM channel. Their task is to issue the requests to the DRAMs by sending required commands according to the devices' current states. The connected DRAM devices then manage the storage of transferred data and enable a coupling to power estimation tools (DRAMPower [17]), thermal models (3D-ICE [18]) and retention time error models.

The architectural difference between DRAMSys4.0 and its predecessor is in the simulator's core component, the channel controller. DRAMSys4.0's channel controller architecture is inspired by advanced hardware DRAM controllers (e.g., [1]). As shown in Figure 4, it is composed of a scheduler, $R \cdot B$ bank machines where $R$ is the number of ranks the channel is composed of and $B$ the number of banks per rank, $R$ refresh managers, $R$ power down managers, a command multiplexer, a response queue and a timing checker. Since SystemC is based on the object-oriented C++ programming language, all components can be designed polymorphically, which allows different policies to be selected during runtime. This is used to specify different DRAM standards and channel

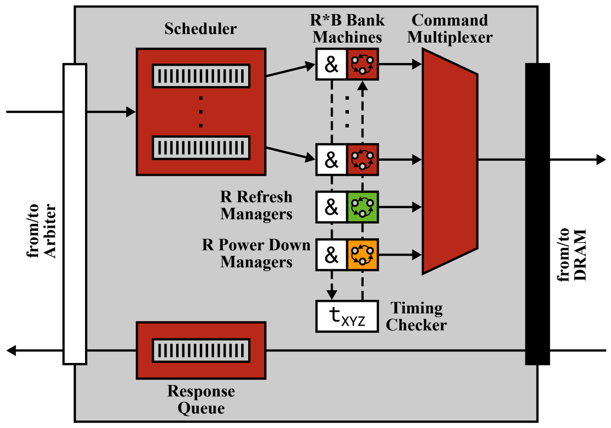


**Fig. 4.** Channel Controller Architecture

controller implementations without requiring a recompilation of the tool or creating additional control flow that results in a slowdown (see also Section 2.2). In addition, the predefined interfaces simplify and speed up the integration of new features. An overview of all supported policies and DRAM standards will be provided in Table 2 in Section 3.

The scheduler buffers incoming requests and reorders them with respect to bandwidth or latency improvements based on a scheduling policy. After selecting one request, it is forwarded to a bank machine, which keeps track of the associated bank's current state and issues a sequence of commands to serve this request. Similar to the scheduler, the bank machines support various page policies [15] to improve the bandwidth or latency of different workloads by automatically precharging the bank's opened row in some cases. Since DRAMSys4.0 models the timing, power, thermal and error behavior of DRAM devices in full detail, the channel controller also has to regularly issue refresh commands and has to trigger power down operation during idle phases. These tasks are taken over by the rank-wise refresh managers and power down managers. Both components are designed polymorphically as well to represent different refresh and power down policies. To find the earliest possible time for issuing a command to the DRAM while satisfying all timing dependencies, bank machines, refresh managers and power down managers invoke the timing checker. On the basis of the whole command history and required information extracted from the DRAM standards, the timing checker calculates this point in time. Since the timing dependencies slightly differ from standard to standard, DRAMSys4.0 uses a separate checker for each of them. If more than one bank machine, refresh manager or power down manager wants to issue a command in the same clock cycle, a conflict arises because of the shared command bus. The command multiplexer resolves this conflict by prioritizing one command (e.g., round robin among ranks/banks). As last component, the channel controller includes a response queue. It buffers the read responses for a transport back to the arbiter and can also reorder them.

**Table 1.** Event Reduction and Total Speedup for MediaBench Benchmarks [19]

| Benchmark | Number of Requests | Total Clock Cycles | Simulated Events v3.0 | Simulated Events v4.0 | Event Reduction | Total Speedup |
|---|---|---|---|---|---|---|
| h263decode | 9867 | 142185273 | 49904 | 36258 | 27.34 % | 9.22 |
| g721encode | 14655 | 152283166 | 65528 | 48900 | 25.38 % | 8.98 |
| g721decode | 19350 | 171781365 | 91828 | 70171 | 23.58 % | 9.73 |
| gsmdecode | 19734 | 42213726 | 93520 | 71158 | 23.91 % | 9.07 |
| c-ray-1.1 | 21627 | 132918262 | 119660 | 85124 | 28.86 % | 10.12 |
| fractal | 33895 | 64184959 | 228184 | 156697 | 31.33 % | 11.15 |
| jpegdecode | 43143 | 19675438 | 196408 | 148407 | 24.44 % | 9.23 |
| mpeg2decode | 72043 | 97603461 | 374848 | 272235 | 27.37 % | 10.01 |
| unepic | 129145 | 10557869 | 718716 | 536878 | 25.30 % | 10.02 |
| jpegencode | 173995 | 39209690 | 769872 | 580929 | 24.54 % | 9.35 |
| epic | 182957 | 55148722 | 940708 | 698595 | 25.74 % | 9.80 |
| mpeg2encode | 616935 | 798646158 | 3457084 | 2522754 | 27.03 % | 9.98 |
| h263encode | 858099 | 526757549 | 4312932 | 3148787 | 26.99 % | 9.35 |

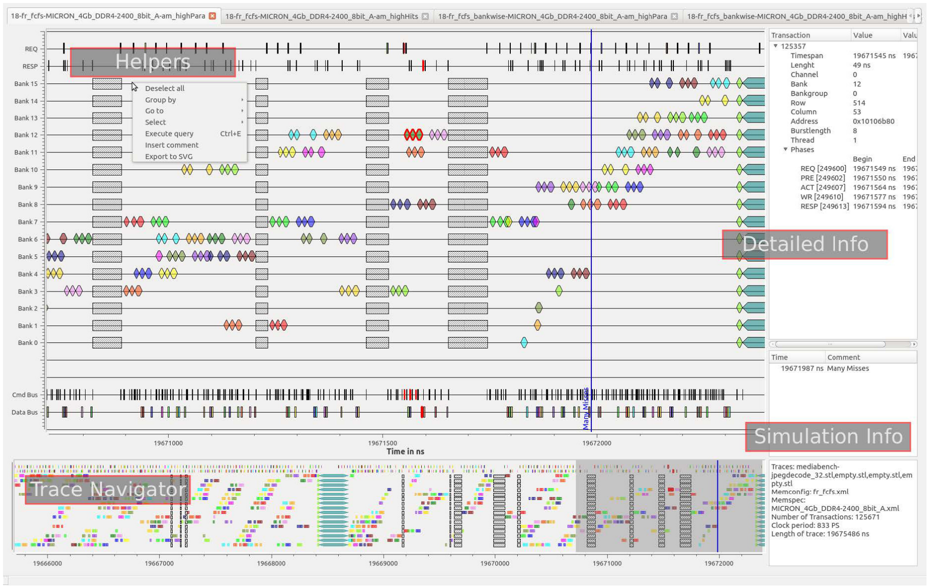
### 2.2 Optimizations for Simulation Speed

To further increase the simulation speed of DRAMSys while maintaining its cycle accuracy, several optimizations have been performed during the revision. As stated earlier, simulations can be sped up by reducing the number of simulated clock cycles or events, respectively, and the executed control flow overhead. Although the used TLM concept ensures a minimum of simulated clock cycles, multiple events may still be fired in the same clock cycle that trigger separate processes or the same process several times. This mechanism is usually needed to model the hardware's concurrency. While DRAMSys3.0's channel controller internally used three event-triggered processes, the new channel controller only needs a single event-triggered process to represent all functionality. It manages the communication and transfer of data between the internals. If, however, multiple events the process is sensitive to are still notified for the same clock cycle, the SystemC simulation kernel performs only a single execution. That way the numbers of simulated events and simulated clock cycles in DRAMSys4.0's channel controller are identical.

Table 1 shows the event reduction in the channel controller for memory traces of the MediaBench benchmarks [19] simulated with a DDR3 DRAM (1 GB DDR3-1600, single channel, single rank, row-bank-column address mapping, FR-FCFS scheduler, open-page policy, run on an Intel Core i9 with 5 GHz). The simulations were performed with a disabled refresh mechanism to see the correlation between the number of requests and events. The table also shows the large difference between the total number of clock cycles and simulated events.

By means of the polymorphic software architecture, DRAMSys4.0 is capable of modeling different DRAM standards and channel controller implementations without introducing any additional control flow that has to be executed frequently during the simulation and, thus, slows it down. For illustration let us assume that the channel controller should be capable of modeling both the open- and closed-page policy.[7] Instead of checking the page policy each time a read or

[7] Controllers that implement the open-page policy keep the corresponding row open after a read or write access, while controllers that implement the closed-page policy automatically precharge the corresponding row after a read or write access.



| Transaction | Value | Valu |
|---|---|---|
| ▾ 125357 | | |
| Timespan | 19671545 ns | 1967 |
| Lenght | 49 ns | |
| Channel | 0 | |
| Bank | 12 | |
| Bankgroup | 0 | |
| Row | 514 | |
| Column | 53 | |
| Address | 0x10106b80 | |
| Burstlength | 8 | |
| Thread | 1 | |
| ▾ Phases | | |
| | Begin | End |
| REQ [249600] | 19671549 ns | 1967 |
| PRE [249602] | 19671550 ns | 1967 |
| ACT [249607] | 19671564 ns | 1967 |
| WR [249610] | 19671577 ns | 1967 |
| RESP [249613] | 19671594 ns | 1967 |

| Time | Comment |
|---|---|
| 19671987 ns | Many Misses |

```
Traces: mediabench-
jpegdecode_32.stl,empty.stl,empty.stl,em
pty.stl
Memconfig: fr_fcfs.xml
Memspec:
MICRON_4Gb_DDR4-2400_8bit_A.xml
Number of Transactions: 125671
Clock period: 833 PS
Length of trace: 19675486 ns
```

**Fig. 5.** Program Interface of the Trace Analyzer [2]

write command is issued, DRAMSys4.0 instantiates bank machines that implement the selected policy once at the start of the simulation and implicitly issues the right commands for the remaining simulation.

Time-consuming string manipulations for the creation of debug messages or log files can be completely removed in the revised version if they are not required. The gained simulation speedup with disabled refresh mechanism for the MediaBench benchmarks is also shown in Table 1, more speedup results will be presented in Figure 8 in Section 3.1.

## 2.3 Trace Analyzer

To provide better analysis capabilities for DRAM subsystem design space exploration than the usual performance-related outputs to the console or a text file, DRAMSys4.0 provides the Trace Analyzer just like its predecessor. After having recorded all TLM transactions of the channel controller in an output trace SQLite database during a simulation, the data can be evaluated using the Trace Analyzer. It illustrates a time window of requests, DRAM commands and the utilization of all banks as shown in Figure 5, which can help system designers to understand the subsystem's internal behavior and to find limiting issues. Exploiting the power of SQL, the data aggregation happens quickly and the tool provides a user-friendly handling that offers a quick navigation through the whole trace with millions of requests and associated DRAM commands. However, since an enabled output trace recording still requires a reasonable amount of time and decreases the overall simulation speed, the most important metrics are also provided on the command line.

An evaluation of the traces can be performed with the powerful Python interface of the Trace Analyzer. Different metrics are described as SQL statements and formulas in Python and can be customized or extended without recompiling the tool. Typical metrics are for instance memory utilization (bandwidth), average response latency or the percentage of time spent in power down.

### 2.4 Co-Simulation with RTL Controller

In addition to the simulation of DRAM subsystems based on the previously introduced high-level channel controller (see Section 2.1), DRAMSys4.0 offers the possibility of embedding cycle-accurate RTL channel controller models into the framework. This allows the validation and analysis of the RTL with the tools provided by DRAMSys4.0 without any manual translation to a higher abstraction level. For example, the Verilog design of a memory controller can be auto-translated into an equivalent SystemC RTL model by the Verilator[8] tool.

To convert the TLM transports into associated RTL signals (including a clock signal, which is not present in DRAMSys) and vice versa, a special transactor module has to be wrapped around the RTL design as shown in Figure 6. Such a transactor was developed for the DDR3 channel controller presented in [1] and exhaustively tested with DRAMSys4.0.

An RTL simulation can also be accelerated by suppressing unnecessary events, see, e.g., [20]. Similar to the idea of clock gating in real circuits for power saving, turning off the clock signal during idle phases of an RTL simulation saves a lot of simulation events since clock signals have high event generation rates. Thus, the so-called clock suppression can tremendously speed up a simulation without changing its results. We adopted this technique for the RTL co-simulation. However, since the internal refresh counter of the RTL channel controller is not incremented by the suspended clock, the transactor saves the refresh counter state externally before suspending the clock and notifies an event at the time the next refresh command should be executed. When this event is fired or a new request arrives, the internal refresh counter is updated to the proper value and the clock is resumed.

Figure 7 shows the speedups of DRAMSys4.0 and the clock-suppressed RTL for artificial traces with random access patterns and varying access densities (accesses per clock cycle) normalized to the plain RTL model (1 GB DDR3-1600, single channel, single rank, row-bank-column address mapping, FCFS scheduler, open-page policy, run on an Intel Core i9 with 5 GHz). For high densities the clock suppression mechanism does not bring an advantage because the controller never turns idle. Instead, it creates a very small computational overhead. With decreasing densities the idle time increases and the speedup rises until it saturates to a factor of 40 because of the refresh commands that also have to be issued regularly during idle phases (self refresh operation is not supported by the RTL controller). The TLM model achieves an even higher speedup across the entire range, which is of factor 3 for high densities and rises to a factor of 4000 for low densities (self refresh operation of the TLM model was disabled for a fair comparison). This is mainly a result of the higher abstraction level that does not model individual signals and thus saves lots of events.

[8] https://www.veripool.org/projects/verilator/

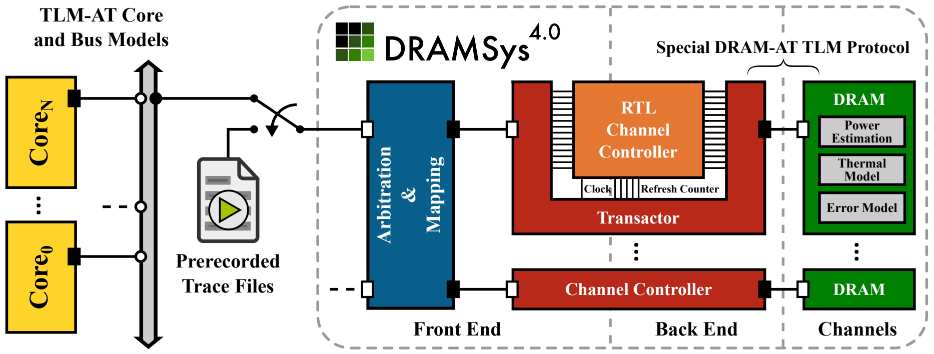


**Fig. 6.** Architecture of DRAMSys4.0 with Embedded RTL Channel Controller

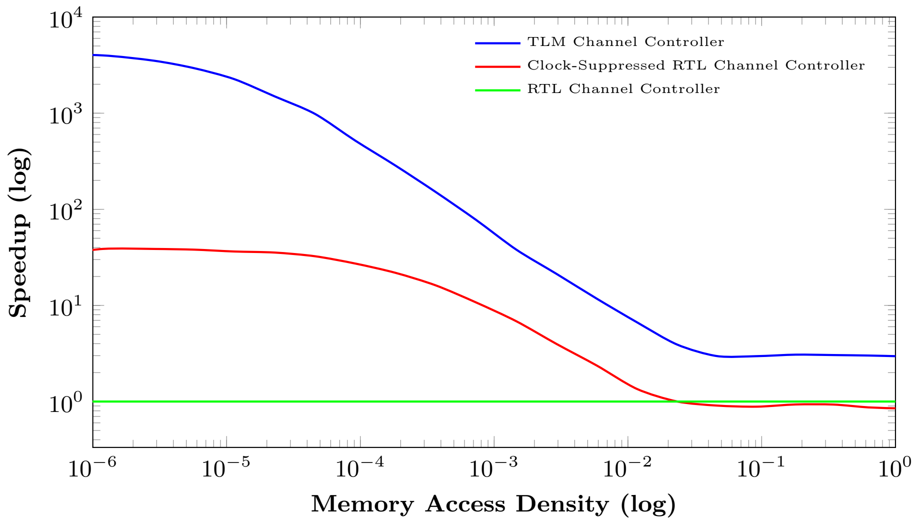


**Fig. 7.** Simulation Speedups Normalized to RTL Channel Controller

## 2.5 Code Generation and Validation

As stated in the introduction, an increasing number of different DRAM standards have been presented by the JEDEC in recent years. Since each new standard comes with slight changes in the DRAM protocol compared to previous ones, the memory simulation models as well as the RTL models must be modified and validated repeatedly. In order to keep pace with these frequent changes and the large variety of standards, a robust and error-free methodology for a fast adaption must be established.

In [21] we presented a comprehensive and formal *Domain Specific Language* (DSL) based on Petri Nets [22] to describe the entire memory functionality of a DRAM standard including all timing dependencies in just a few lines of code.

Using the formal description of a corresponding Petri Net, different simulation and validation models are generated in this work:

- **TLM Model Generation:** The source code of the channel controller's standard-specific timing checkers (see Section 2.1) can be generated automatically and correct by construction from these DSL descriptions, replacing the error-prone handwritten SystemC implementation of a new memory standard by the fast generation of SystemC code from a high-level description.
- **RTL Controller Validation Model:** As shown in Section 2.4 and Section 2.3, DRAMSys4.0 offers functionalities for embedding an RTL model of a memory controller into the framework and for recording the executed DRAM commands in an output trace database. Using the formal DSL descriptions, a standard-specific executable C++ validation model can be created, which analyses a recorded DRAM command trace for standard compliance. This approach provides fast feedback to an RTL developer if a change in the RTL description led to a protocol violation.
- **DRAM Simulator Validation Model:** The same validation model is used to analyze recorded command traces of other state-of-the-art DRAM simulators. In Section 3.1 we reveal errors in DRAMsim3, Ramulator and in the gem5 DRAM model using our validation model.

## 3 Related Work and Results

This section provides a comparison between the most prominent open-source cycle-accurate DRAM simulators and introduces approaches for the cycle-approximate modeling of DRAM subsystems.

### 3.1 Cycle-Accurate Simulators

As stated in the introduction, there are several publicly-available cycle-accurate DRAM simulators. Table 2 provides a comprehensive comparison that also includes both DRAMSys3.0 and DRAMSys4.0. For simplicity, we only focus on DRAM standards and features specified by the JEDEC since they are best qualified for real system developments.

DRAMSys3.0, DRAMSim2 and DrSim were already developed several years ago but never updated over time, leading to an exclusive support of older standards and making them unsuitable for most current system developments. DRAMSys4.0, DRAMsim3, Ramulator and the gem5 DRAM model are all updated from time to time, however, only DRAMSys4.0 and DRAMsim3 currently support the latest standards like GDDR6 or HBM2. For request initiation all simulators provide trace players and a coupling to gem5. In addition, DRAMsim3 supports a coupling to the simulation frameworks SST [27] and ZSim [28]. DRAMSys3.0 and DRAMSys4.0 can be coupled to any TLM-AT-compliant core model. While all simulators seem to be cycle accurate at first view, some of them do not model the full set of timing dependencies for all standards they support. Using our Petri-Net-based validation model of Section 2.5, we were able to find missing timing dependencies in DRAMsim3, Ramulator and in the gem5 DRAM

**Table 2.** Overview of the Most Prominent Open-Source DRAM Simulators

| **Feature** | **DRAMSys 4.0** (this work) | **DRAMSys 3.0** [2] | **DRAMSim2** [4] | **DRAMsim3** [5] | **Ramulator** [6] | **DrSim** [7] | **gem5 DRAM Model** [15] |
|---|---|---|---|---|---|---|---|
| DRAM Standards | DDR3/4, LPDDR4, Wide I/O 1/2, GDDR5/5X/6, HBM1/2 | DDR3/4, Wide I/O 1 | DDR2/3 | DDR3/4, LPDDR3/4, GDDR5/5X/6, HBM1/2 | DDR3/4, LPDDR3/4, GDDR5, Wide I/O 1/2, HBM1 | DDR2/3, LPDDR2 | DDR3/4, LPDDR2/3, Wide I/O 1, GDDR5, HBM1 |
| Refresh Modes | all-bank refresh, per-bank refresh | all-bank refresh | all-bank refresh | all-bank refresh, per-bank refresh | all-bank refresh, per-bank refresh | all-bank refresh | all-bank refresh |
| Power Down Modes | active & precharge power down, self refresh | active & precharge power down, self refresh | precharge power down | self refresh | active & precharge power down, self refresh | active & precharge power down | active & precharge power down, self refresh |
| Address Mappings | any bijective boolean function [23] | any bijective boolean function [23] | 7 different mappings | mappings with hierarchy granularity | mappings with hierarchy granularity | fixed mapping with optional interleavings | 3 different mappings |
| Schedulers | FCFS, FR-FCFS [24], FR-FCFS Grouping | FCFS, FR-FCFS [24], FR-FCFS Grouping, Par-BS [25], SMS [26] | issue requests ASAP | issue requests ASAP | FCFS, FR-FCFS [24], FR-FCFS Cap, FR-FCFS PriorHit | FCFS, FR-FCFS [24] | FCFS, FR-FCFS [24] |
| Page Policies | open, open adaptive, closed, closed adaptive [15] | open, closed | open, closed | open, closed | open, closed, closed with auto precharge, timeout | open, closed | open, open adaptive, closed, closed adaptive [15] |
| Configuration | controller policies, address mapping, DRAM standard, organization, timings & currents (JSON file) | controller policies, address mapping, DRAM standard, organization, timings & currents (XML file) | controller policies, address mapping, DRAM organization, timings & currents | controller policies, address mapping, DRAM standard, organization, timings & currents | address mapping, DRAM standard, speed bin & size | controller policies, address interleaving, DRAM organization & timings | controller policies, address mapping, DRAM organization, timings & currents (Python file) |
| Request Initiators | trace players (fixed & elastic memory traces), SystemC-based core models, gem5 core models | trace players (fixed & elastic memory traces), SystemC-based core models, gem5 core models | trace player (fixed memory traces), gem5 core models | trace player (fixed memory traces), gem5 core models, SST [27], ZSim [28] | trace player (fixed, untimed memory traces & timed CPU traces), gem5 core models | trace player (fixed memory traces), gem5 core models | trace players (fixed & elastic memory traces), gem5 core models |
| Power Estimation | DRAMPower [17] | DRAMPower [17] | Micron power model | Micron power model & DRAMPower [17] | DRAMPower [17] & Vampire [29] | - | DRAMPower [17] |
| Thermal Modeling | 3D-ICE [18] (only Wide I/O 1) | 3D-ICE [18] (only Wide I/O 1) | - | custom model (all standards) | - | - | - |
| Error Modeling | custom model (only Wide I/O 1) | custom model (only Wide I/O 1) | - | - | - | - | - |
| Validation Method | Petri-Net-based code generation & result checking [21], result visualization | testing script, result visualization | DDR2/3 command traces fed into Micron Verilog model | DDR3/4 command traces fed into Micron Verilog model | DDR3 command trace fed into Micron Verilog model | comparison to DRAMSim2 | comparison to DRAMSim2 |
| Outputs/Metrics | average bandwidth & power consumption, metrics for SQLite command trace | average bandwidth & power consumption, metrics for SQLite command trace | bandwidth, latency & power per epoch | metrics in output file | metrics in output file, command trace | metrics in console | metrics in output file |
| Result Visualization | Trace Analyzer | Trace Analyzer | DRAMVis [4] | plot of metrics | - | - | - |
| Considered Timing Dependencies | all | no inter-rank dependencies | all | no multi-cycle-command dependencies | no multi-cycle-command dependencies | all | only DDR3/4 timing dependencies |
| Simulation Model | TLM-based (IEEE 1666 SystemC/TLM2.0 [14]) | TLM-based (IEEE 1666 SystemC/TLM2.0 [14]) | loop-based | loop-based | loop-based | loop-based | TLM-based (gem5 [3]) |

model (e.g., missing command bus dependencies for multi-cycle commands of LPDDR4 or HBM1/2).[9] Besides the performance perspective, for most of today's system developments the power consumption and thermal behavior is of great importance, especially in the field of embedded systems. All simulators except DrSim offer a functionality for power estimation. DRAMSys3.0, DRAMSys4.0 and DRAMsim3 can also model the thermal behavior of devices. For performance evaluation, all simulators output bandwidth-, latency- and power-related statistics. Moreover, DRAMSim2 supplies DRAMVis [4], a tool that can visualize the bandwidth, latency and power over time. Similarly, DRAMSys3.0 and DRAMSys4.0 provide the Trace Analyzer for visual result analysis (see Section 2.3).

All simulators are also compared with regard to their simulation speed. As stated earlier, the wall clock time that a simulation requires does not only depend on the amount of simulated time, but also on the memory access density (accesses per clock cycle). This can especially be observed for the TLM-based simulators. For that reason, we investigate the simulation speed for a large range of densities using artificial traces. To minimize the impact of the simulators' different controller implementations (e.g., queuing mechanisms, scheduling policies, further bandwidth-improving techniques like read snooping[10]), the memory traces exclusively provoke read misses and utilize all banks uniformly. Different densities are created by increasing the gaps between accesses. Apart from that, all simulators are configured as similar as possible (1 GB DDR3-1600 since DDR3 is the only standard supported by all of them, single channel, single rank, row-bank-column address mapping, open-page policy, run on an Intel Core i9 with 5 GHz), built as release version, and run with a minimum of generated outputs. Using these traces, the achieved performance and total simulated time of all simulators except Ramulator are very similar (maximum deviations of 2 % because all simulators implement a different power down operation). Ramulator does not model the bank parallelism properly (commands for a new request are only issued if the previous request has been finished completely) and therefore achieves much lower bandwidths.

The simulation speeds of all simulators are shown in Figure 8. For high trace densities the speeds of the fastest loop-based and TLM-based simulators (DRAMSim2, Ramulator, DRAMSys4.0 and the gem5 DRAM model) do not differ much from each other because state changes occur in almost all clock cycles. At a density of around 0.2 (0.1 for Ramulator due to its lower bandwidth), the channel controllers start to turn idle and the consumed wall clock time decreases. While the graphs of all loop-based simulators converge to a fixed value for further decreasing densities (wall clock time to simulate pure idle cycles), the TLM-based simulators demonstrate their advantage in the form of a steady decrease, clearly outperforming all loop-based simulators. During long idle phases they initiate self refresh operation of the DRAM devices. In this way external refresh commands can be omitted and no clock cycles have to be simulated at all. Since

[9] We will report the missing timing dependencies to the developers of the other simulators.

[10] Using read snooping a read request can be served directly within the controller if an earlier write request to the same address is still pending.

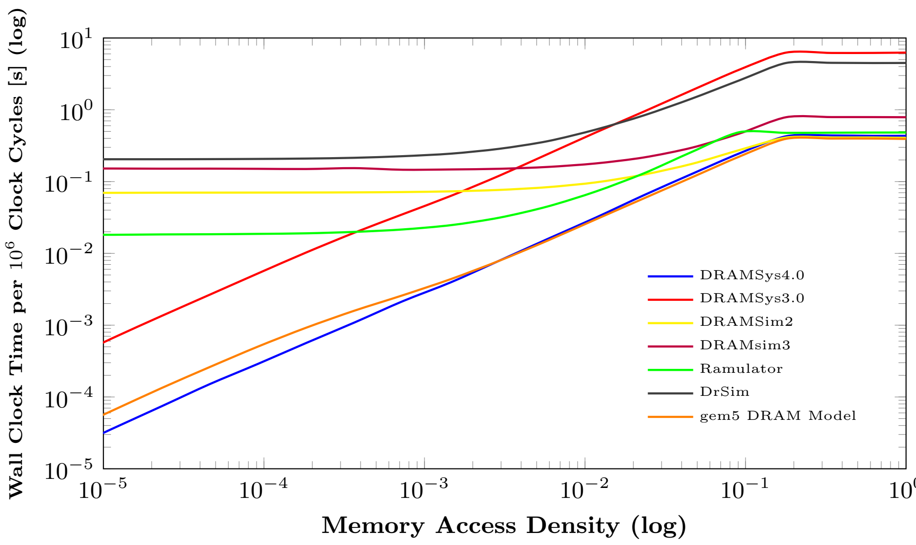


**Fig. 8.** Simulation Speeds of State-of-the-Art DRAM Simulators

the memory access density of real applications is often located in these lower ranges (e.g., $7 \cdot 10^{-5}$ - $1 \cdot 10^{-2}$ for the MediaBench benchmarks), TLM-based simulators can speed up the simulation by several orders of magnitude. Thus, the exact modeling of a DRAM subsystem in a system context is no longer the bottleneck from a simulation perspective.

For the TLM-based simulators, DRAMSys4.0 constantly outperforms its predecessor by a factor of 10 to 20, which is the result of our optimizations (see Section 2.2). The simulation speeds of DRAMSys4.0 and the gem5 DRAM model are more or less on the same level for high densities. At densities smaller than $10^{-3}$ the gem5 DRAM model starts to become slightly slower than DRAMSys4.0 because the switching to self refresh operation is implemented less efficiently.

As mentioned in Section 1, a DRAM subsystem simulation consists of a specific controller model and the model of a specific DRAM standard. Since each simulator represents a different controller implementation (scheduling policy, power down operation, request buffer etc.), a fair comparison of accuracy is not possible; all simulators might yield different results for the same inputs while still being cycle accurate and standard compliant.

## 3.2 Cycle-Approximate DRAM Models

Beside the cycle-accurate DRAM simulators, further approaches that approximate the behavior exist (see Figure 1). In [12] the authors propose an analytical DRAM performance model that uses traces to predict the efficiency of the DRAM subsystem. Todorov et al. [9] presented a statistical approach for the construction of a cycle-approximate TLM model of a DRAM controller based on a decision tree. However, these approaches suffer from a significant loss in accuracy. More promising approaches based on machine learning techniques have been presented recently. The paper [10] presents the modeling of DRAM behavior using decision trees. In [11] the authors present a performance-optimized DRAM model that is based on a neural network.

## 4 Conclusion

In this paper we presented DRAMSys4.0, a SystemC/TLM-based open-source DRAM simulation framework. Due to the optimized architecture it reaches very high simulation speeds compared to state-of-the-art simulators while ensuring full cycle accuracy and standard compliance. DRAMSys4.0 supports a large collection of controller features and DRAM standards and offers unique functionalities for adaptation and result analysis, making it perfectly suitable for both fast and truthful design space exploration. In addition, the framework can be used to validate RTL descriptions of real hardware memory controllers. In the future we plan to extend DRAMSys4.0 by further emerging DRAM standards (e.g., LPDDR5 and DDR5) and associated new features.

## Acknowledgements

This work was supported within the Fraunhofer and DFG cooperation programme (Grant no. WE2442/14-1) and supported by the *Fraunhofer High Performance Center* for *Simulation- and Software-based Innovation*. Furthermore, we thank Synopsys and the anonymous reviewers for their support.

## References

1. Chirag Sudarshan, et al. *A Lean, Low Power, Low Latency DRAM Memory Controller for Transprecision Computing.* In Dionisios N. Pnevmatikatos, et al., editors, Embedded Computer Systems: Architectures, Modeling, and Simulation, pages 429–441, Cham, 2019. Springer International Publishing.
2. Matthias Jung, et al. *DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework.* IPSJ Transactions on System LSI Design Methodology (T-SLDM), August 2015.
3. Nathan Binkert, et al. *The gem5 simulator.* SIGARCH Comput. Archit. News, 39(2):1–7, August 2011.
4. Paul Rosenfeld, et al. *DRAMSim2: A Cycle Accurate Memory System Simulator.* Computer Architecture Letters, 10(1):16–19, Jan 2011.
5. S. Li, et al. *DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator.* IEEE Computer Architecture Letters, pages 1–1, 2020.
6. Yoongu Kim, et al. *Ramulator: A Fast and Extensible DRAM Simulator.* IEEE Computer Architecture Letters, PP(99):1–1, 2015.
7. Min Kyu Jeong, et al. *DrSim: A Platform for Flexible DRAM System Research.* http://lph.ece.utexas.edu/public/DrSim, (Last Access: 15.08.2019).
8. Bruce Jacob. *The Memory System: You Can'T Avoid It, You Can'T Ignore It, You Can'T Fake It.* Morgan and Claypool Publishers, 2009.
9. Vladimir Todorov, et al. *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, DATE '12, pages 1066–1071, San Jose, CA, USA, 2012. EDA Consortium.
10. Shang Li et al. *Statistical DRAM Modeling.* In Proceedings of the International Symposium on Memory Systems, MEMSYS '19, page 521–530, New York, NY, USA, 2019. Association for Computing Machinery.

11. Matthias Jung, et al. *Fast and Accurate DRAM Simulation: Can we Further Accelerate it?* To be published in proceedings of the Conference on Design, Automation and Test in Europe, DATE2020, https://www.jung.ms/paper_2020_date.pdf, March 2020.
12. George L. Yuan et al. *A Hybrid Analytical DRAM Performance Model*, 2009.
13. Shang Li, et al. *Rethinking Cycle Accurate DRAM Simulation.* In Proceedings of the International Symposium on Memory Systems, MEMSYS '19, page 184–191, New York, NY, USA, 2019. Association for Computing Machinery.
14. IEEE Computer Society. *IEEE Standard for Standard SystemC Language Reference Manual.* (IEEE Std 1666-2011), 2012.
15. A. Hansson, et al. *Simulating DRAM controllers for future system architecture exploration.* In Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on, pages 201–210, March 2014.
16. Matthias Jung, et al. *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, RAPIDO '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.
17. Karthik Chandrasekar, et al. *DRAMPower: Open-source DRAM power & energy estimation tool.* http://www.drampower.info, Last Access 15.08.2019.
18. A. Sridhar, et al. *3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling.* In Proc. of ICCAD 2010, 2010.
19. MediaBench Consortium. *Mediabench.* http://euler.slu.edu/ fritts/mediabench/, 2015, last access 28.02.2015.
20. H. Muhr et al. *Accelerating RTL Simulation by Several Orders of Magnitude Using Clock Suppression.* In 2006 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pages 123–128, July 2006.
21. Matthias Jung, et al. *Fast Validation of DRAM Protocols with Timed Petri Nets.* In Proceedings of the International Symposium on Memory Systems, MEMSYS '19, pages 133–147, New York, NY, USA, 2019. ACM.
22. Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Universität Hamburg, 1962.
23. Matthias Jung, et al. *ConGen: An Application Specific DRAM Memory Controller Generator.* In Proceedings of the Second International Symposium on Memory Systems, MEMSYS '16, pages 257–267, New York, NY, USA, 2016. ACM.
24. Scott Rixner, et al. *Memory Access Scheduling.* In Proceedings of the 27th Annual International Symposium on Computer Architecture, ISCA '00, pages 128–138, New York, NY, USA, 2000. ACM.
25. Onur Mutlu et al. *Parallelism-Aware Batch-Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems.* In 35th International Symposium on Computer Architecture (ISCA). Association for Computing Machinery, Inc., June 2008.
26. Rachata Ausavarungnirun, et al. *Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems.* In Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12, pages 416–427, Washington, DC, USA, 2012. IEEE Computer Society.
27. A. F. Rodrigues, et al. *The Structural Simulation Toolkit.* SIGMETRICS Perform. Eval. Rev., 38(4):37–42, March 2011.
28. Daniel Sanchez et al. *ZSim: fast and accurate microarchitectural simulation of thousand-core systems.* ACM SIGARCH Computer Architecture News, 41:475, 07 2013.
29. Saugata Ghose, et al. *What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study.* Proc. ACM Meas. Anal. Comput. Syst., 2(3), December 2018.