




# DRAMSys4.0: An Open-Source Simulation Framework for In-depth DRAM Analyses

Lukas Steiner<sup>1</sup>  · Matthias Jung<sup>2</sup> · Felipe S. Prado<sup>1</sup> · Kirill Bykov<sup>1</sup> · Norbert Wehn<sup>1</sup>

Received: 15 April 2021 / Accepted: 14 February 2022 / Published online: 12 March 2022  
© The Author(s) 2022

## Abstract

The simulation of Dynamic Random Access Memories (DRAMs) on system level requires highly accurate models due to their complex timing and power behavior. However, conventional cycle-accurate DRAM subsystem models often become a bottleneck for the overall simulation speed. A promising alternative are simulators 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 and most extensive open-source cycle-accurate DRAM simulation framework. DRAMSys4.0 includes a novel software architecture that enables a fast adaption to different hardware controller implementations and new JEDEC standards. In addition, it already supports the latest standards DDR5 and LPDDR5. We explain how to apply optimization techniques for an increased simulation speed while maintaining full temporal accuracy. Furthermore, we demonstrate the simulator's accuracy and analysis tools with two application examples. Finally, we provide a detailed investigation 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 · DDR5

---

✉ Lukas Steiner  
lsteiner@eit.uni-kl.de

Matthias Jung  
matthias.jung@iese.fraunhofer.de

Norbert Wehn  
wehn@eit.uni-kl.de

<sup>1</sup> Technische Universität Kaiserslautern, Kaiserslautern, Germany

<sup>2</sup> Fraunhofer IESE, Kaiserslautern, Germany

## 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 fit 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.

A DRAM subsystem is composed of a memory controller and memory devices. Although the JEDEC standards define a framework of rules that apply to the order and minimum timing between DRAM commands, the controller still has to take many scheduling decisions. Different controller implementations exploit this freedom in order to optimize for different metrics (e.g., latency, bandwidth, power). Therefore, a DRAM subsystem simulation not only represents one specific JEDEC standard, but also only one specific controller implementation. This simulation can be performed on several levels of abstraction, each offering a certain trade-off between speed and accuracy. Figure 1 provides an overview of common simulation models.

Non-cycle-accurate models (right side) allow high simulation speeds but lack in accuracy. The simplest, pure functional model of a DRAM subsystem is a fixed-latency model [1]. This approach processes all request with the same constant latency and all subsystem internals are omitted. However in reality, the latencies of DRAM accesses reach from a dozen to several hundred cycles due to the complex device architecture. Therefore, a pure functional model is ineligible for any performance estimations. Cycle-approximate models try to mimic the latency

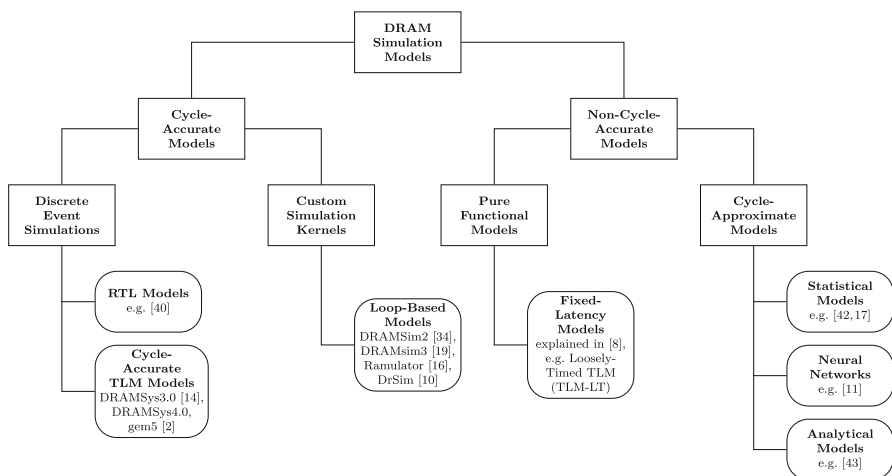


Fig. 1 Different DRAM simulation models

behavior of real DRAM subsystems by utilizing e.g. statistical methods [2, 3] or neural networks [4]. Unfortunately, their accuracy can vary greatly 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 reliable investigations, but usually take a lot more time to execute. RTL descriptions of real DRAM controllers [5] can be simulated using a *Discrete Event Simulation* (DES) kernel. State-of-the-art cycle-accurate DRAM simulators, namely DRAMSim2 [6], DRAMsim3 [7], Ramulator [8] and DrSim [9], 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, all these simulation models evaluate every single clock cycle, whether or not there are any state changes in the system. This leads to a direct proportionality of consumed wall-clock time and simulated time, while the memory access density only has a partial impact on wall-clock time (see Sect. 5). Thus, when coupled with modern CPU simulators, the DRAM simulation can become a bottleneck of the overall simulation speed [10].

To reach higher simulation speeds, many commercial simulation tools use (TLM). This approach is also based on a DES kernel, however, neither individual signals nor individual clock cycles have to be modelled. Processes are only evaluated when state changes occur (see Sect. 2.1) and, as a result, the simulation speed is only dependent on the memory access density. That way, TLM models can surpass the speed of all abovementioned simulators significantly. Nevertheless, they can also be developed to preserve full accuracy. The design space exploration framework DRAMSys3.0 [11] provides a cycle-accurate DRAM subsystem model based on the SystemC/TLM2.0 IEEE 1666 Standard [12]. Despite its frequent use in research, DRAMSys3.0 has never been open sourced and does not support the latest DRAM standards. gem5 [13], one of the most popular open-source full-system simulators, uses a similar TLM concept to increase simulation speed. Major drawback of its detailed DRAM model [14] is a close link to the DDR3 and DDR4 standards, which leads to a reduced accuracy for simulations with other JEDEC standards as shown in [10].

To the best of our knowledge, there exists no DRAM simulator that is open source, models the latest JEDEC standards with cycle accuracy, and performs simulations at a speed sufficient for fast design space exploration.

Therefore, we introduce DRAMSys4.0, a completely revised version of DRAMSys3.0 [11]. Besides its supports for the latest JEDEC standards (e.g., DDR5 and LPDDR5), it is optimized to achieve between  $10\times$  to  $20\times$  higher simulation speeds compared to its predecessor and offers a largely extended toolbox for analysis and validation. The framework is now also open sourced on GitHub.<sup>1</sup>

This paper is an extension of [15]. In addition to the original work, we show two application examples that demonstrate the flexibility of DRAMSys4.0. The first example is also used to evaluate the simulator's accuracy against real hardware, while the second one serves as a setup to compare the full-system performance

<sup>1</sup> <https://github.com/tuki-msd/DRAMSys>

impact of DDR5 to its predecessor DDR4. In both application examples the new features of the Trace Analyzer are required for result interpretation. In summary, this paper makes the following contributions:

- We present the novel software architecture of DRAMSys4.0, which enables high simulation speeds as well as high flexibility for a fast adaption to different DRAM controller implementations and fast expansion by new JEDEC standards.
- We demonstrate how RTL descriptions of DRAM controllers can be embedded into the framework.
- We showcase the simulator’s flexibility by adapting its behavior to a real hardware controller and expanding its models by the latest JEDEC standard DDR5.
- We demonstrate how the graphical Trace Analyzer tool can be used to explain and verify simulation results.
- We provide a detailed and fair comparison of the most prominent cycle-accurate open-source DRAM simulators with regard to their features and simulation speed.

The remaining paper is structured as follows: In Sect. 2 an overview of the simulator and its features is given. The adaption and expansion examples are presented in Sects. 3 and 4, respectively. Section 5 discusses related cycle-accurate simulators, provides a detailed comparison among them, and presents cycle-approximate approaches for DRAM simulation. Section 6 concludes the work.

## 2 Simulator Overview

In this section we introduce DRAMSys4.0, which is faster, more flexible, and more comprehensive compared to its predecessor. More precisely, we present the architecture and functionality, discuss optimizations to increase simulation speed, explain the Trace Analyzer’s analysis features, and showcase the embedding of RTL controllers.

### 2.1 Architecture and Functionality

DRAMSys<sup>2</sup> uses the concept of TLM based on the SystemC/TLM2.0 IEEE 1666 Standard [12] for a fast and fully cycle-accurate, i.e., JEDEC compliant, simulation. All components are designed as SystemC modules (`sc_module`) and connected by TLM sockets. The simulator relies on the *Approximately Timed* (AT) coding style, which defines a non-blocking four-phase handshake protocol.<sup>3</sup> Four phases are required to model the subsystem’s pipelined behavior and out-of-order responses to the initiators. However, since a single memory access can translate into multiple DRAM commands depending on the current device state (e.g., precharge (PRE) -

<sup>2</sup> DRAMSys without a version number refers both to DRAMSys3.0 and DRAMSys4.0.

<sup>3</sup> The TLM-AT base protocol consists of the phases `BEGIN_REQ`, `END_REQ`, `BEGIN_RESP` and `END_RESP`.

activate (ACT) - read (RD)/write (WR) for a row miss), four phases are not sufficient to model the communication between controller and device with full temporal accuracy. Therefore, a custom TLM protocol called *DRAM-AT* [16] with application-specific phases is used to represent individual DRAM commands. 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 executed control flow overhead. Therefore, DRAMSys only evaluates clock cycles in which state changes occur. Figure 2 shows an example for an ACT command and its timing dependency<sup>4</sup>  $t_{RCD}$  to a following RD command. While a loop-based simulator would evaluate ten clock cycles to issue both commands, DRAMSys only evaluates the first one, notifies an event after  $t_{RCD}$ , and directly evaluates the tenth clock cycle to issue the RD command. All clock cycles in between are skipped and the simulation time is advanced. Especially in 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 Sect. 5) without losing accuracy.

From an architectural point of view, DRAMSys4.0 consists of an arbitration and mapping unit (short arbiter) as well as independent channel controllers and DRAM devices for each memory channel, shown in Fig. 3. The arbiter cross-couples multiple initiators and DRAM channels on the basis of a specified address mapping. In addition, it can buffer several requests and responses to increase throughput and to restore the original request order on the return path. The arbiter is followed by independent channel controllers for each DRAM channel. They process incoming requests by translating them into special commands depending on the devices' current states and the simulated JEDEC standard. The connected DRAM devices store transferred data and establish a coupling to power estimation tools (DRAMPower [17]), thermal models (3D-ICE [18]) and retention time error models.

The main 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 various advanced RTL DRAM controller implementations (e.g., [19, 20, 5]). As shown in Fig. 4, it is composed of a scheduler, bank-granular bank machines, rank-granular refresh managers and 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 implemented. This is used to simulate different JEDEC standards and channel controller versions without requiring a recompilation of the tool or creating additional control flow (see Sect. 2.2). In addition, the predefined interfaces simplify and speed up the integration process of new features. An overview of all supported policies and JEDEC standards is provided in Table 4 in Sect. 5.

The scheduler buffers incoming requests in a configurable queue architecture and reorders them based on the implemented scheduling policy. After selecting one

<sup>4</sup> Timing dependencies are temporal constraints that must be satisfied between issued DRAM commands.

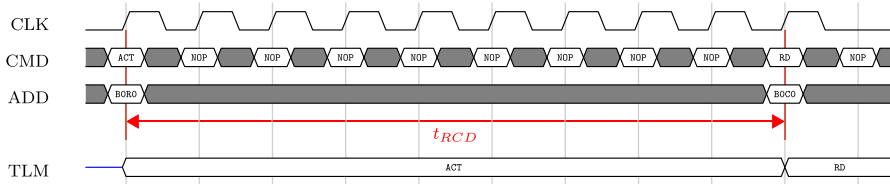


Fig. 2 TLM implementation of the ACT command [16]

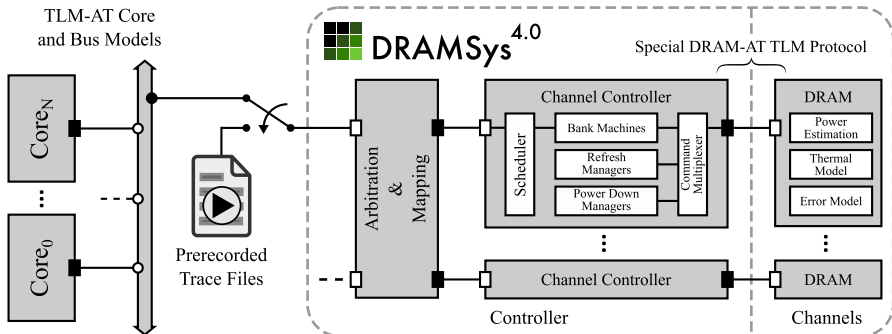
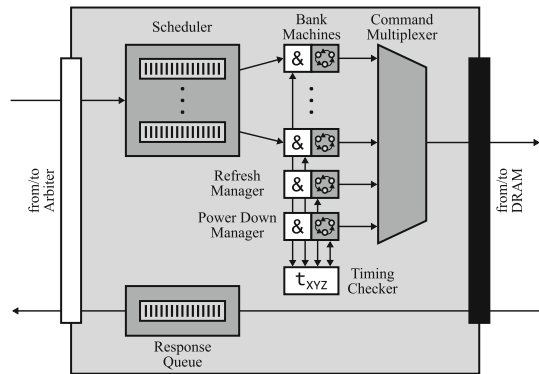


Fig. 3 Architecture of DRAMSys4.0

request, it is forwarded to the target bank machine, which keeps track of the associated bank's state and issues a sequence of commands to serve this request. Similar to the scheduler, the bank machines support various page policies [14] to improve the bandwidth, latency and power consumption of different workloads by automatically precharging the accessed 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 regularly issues refresh commands and triggers power-down operation during idle phases. These tasks are taken over by 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 information extracted from the simulated JEDEC standard, the timing checker determines this point in time. Since the timing dependencies 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 want 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 of them (e.g., earliest request first or round robin among ranks/banks). As last component the channel

**Fig. 4** Channel controller architecture



controller includes a response queue to buffer responses for a transport back to the arbiter.

## 2.2 Optimization of Simulation Speed

To further increase the simulation speed of DRAMSys while maintaining its 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 evaluated clock cycles, multiple events may still be fired in the same cycle that trigger separate processes or the same process several times. This mechanism is usually needed to model hardware 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 all internals. Table 1 shows the resulting event reduction for exemplary memory traces of the MediaBench benchmarks [21] simulated with a DDR3 DRAM (1 GB DDR3-1600, single channel, single rank, row-bank-column address mapping, FR-FCFS scheduling, open-page policy, run on an Intel Core i9 with 5 GHz). The simulations were performed with a disabled refresh mechanism to highlight the correlation between number of requests and events. The table also shows the large difference between total number of clock cycles and simulated events.

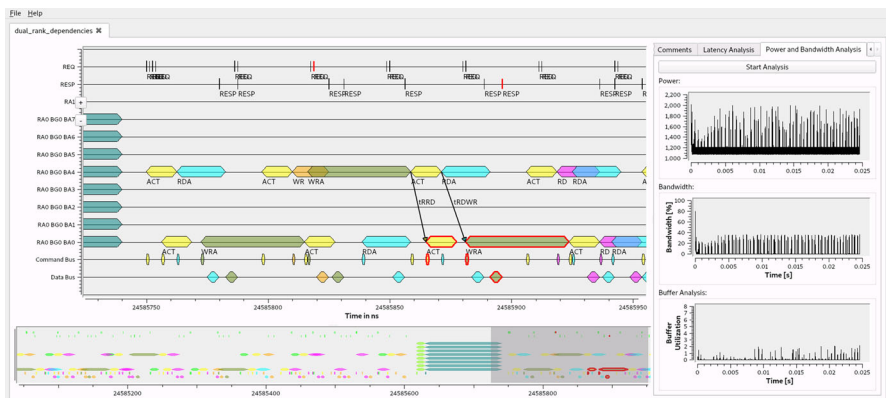
By means of the polymorphic software architecture, DRAMSys4.0 is capable of modeling different JEDEC standards and channel controller implementations without introducing any additional control flow that is executed frequently during the simulation and, thus, slows it down. Moreover, processing-intensive 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 10 in Sect. 5.1.

**Table 1** Event reduction and total speedup for MediaBench benchmarks [21]

Benchmark	Memory Requests	Clock Cycles	Simulated events		Event Reduction (%)	Total Speedup
			v3.0	v4.0		
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

### 2.3 Trace Analyzer

To offer advanced analysis capabilities for DRAM subsystem design space exploration and not only performance-related outputs to the console or a text file, DRAMSys provides the Trace Analyzer. This component has also been completely revised and lots of new features have been added. During a simulation, all TLM transactions of the channel controller can be recorded in an SQLite trace database. Afterwards, this database can be visualized and evaluated with the Trace Analyzer. It illustrates a time window of requests, DRAM commands and the utilization of individual banks as shown in Fig. 5. Since the view is fully customizable like a

**Fig. 5** Program interface of the Trace Analyzer [11]



wave viewer, also large subsystems with dozens or hundreds of banks can be analyzed. Exploiting the power of SQL, both recording and requesting data happens quickly, enabling a user-friendly handling and a smooth navigation through the whole trace with millions of requests and associated DRAM commands.

Traces can also be evaluated with the provided Python interface of the Trace Analyzer. Different metrics are described as formulas consisting of SQL statements and can be customized or extended to the user's needs. Predefined metrics are for instance memory utilization (bandwidth), average response latency or exploited bank parallelism.

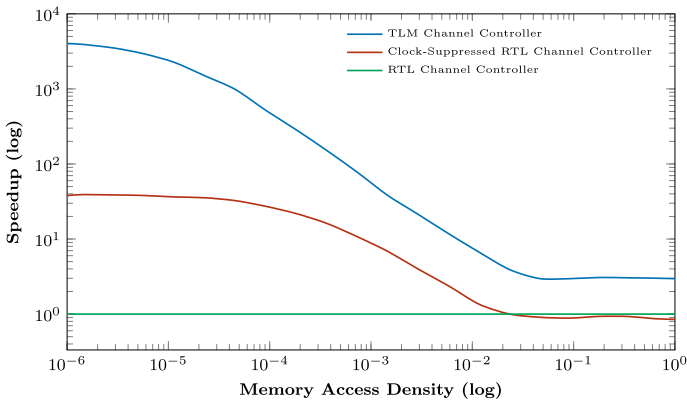
In addition to the request visualization and Python interface, memory bandwidth, scheduling buffer utilization and power are plotted over time. More insights into the channel controller can also be gained with the provided latency histogram. The histogram serves as an important metric to compare the behavior and performance of different controller implementations, containing much more information than simple numbers like average bandwidth or average response latency. One especially helpful feature of the Trace Analyzer is the timing dependency analysis. For each issued DRAM command the longest timing dependency and associated root command that caused the delay can be displayed. Statistics about the frequency of different timing dependencies are provided as well. This helps system designers to find and eliminate limiting factors of the subsystem.

## 2.4 Embedding of RTL Controllers

In addition to the simulation of DRAM subsystems based on the high-level TLM channel controller, DRAMSys4.0 offers the possibility to embed cycle-accurate RTL channel controller models into the framework. This allows the validation and analysis of hardware with the framework's provided tools (see Section 2.3) without any manual translation to a higher abstraction level. As an example, the Verilog description of the DDR3 channel controller presented in [5] was auto-translated into an equivalent SystemC RTL model by the Verilator tool<sup>5</sup>. To convert TLM transports from arbiter and DRAM devices into associated RTL signals (including a clock signal, which is not present in DRAMSys) and vice versa, a special transactor module is wrapped around the RTL design.

Just like a TLM simulation, an RTL simulation can be accelerated by skipping unnecessary events, see, e.g., [22]. 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 embedded RTL simulation. However, since the refresh counter of the RTL channel controller is not incremented by a suspended clock any more, its state has to be saved externally in the transactor module before suspending the clock and an event has to be notified at the time the next refresh command should be scheduled. When this event is fired or a new request arrives, the internal refresh counter is updated to

<sup>5</sup> <https://www.veripool.org/projects/verilator/>



**Fig. 6** Simulation speedups normalized to RTL channel controller

the proper value and the clock is resumed. Additional modifications to the RTL are not required.

Figure 6 shows the speedups achieved by TLM and clock-suppressed RTL for artificial traces with random access patterns and varying access densities (accesses per clock cycle) normalized to the plain RTL channel controller model (1 GB DDR3-1600, single channel, single rank, row-bank-column address mapping, FCFS scheduling, 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 and no cycles can be skipped. Instead, it creates a very small computational overhead. With decreasing densities the idle time increases and the speedup rises to a factor of 40. At this point it saturates because refresh commands have to be issued regularly also 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 access 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, thus saving lots of events.

### 3 TLM Controller Adaption to Real Hardware

For a design space exploration framework like DRAMSys4.0 a high configurability and flexibility is essential not only to find the optimal subsystem configuration, but also to adapt the software model to existing hardware implementations as fast and closely as possible. That way, the faster TLM controller can be used for simulation in place of the slower RTL to accelerate the system design process, enabling a shorter time to market and reduced costs. As an example, we demonstrate the adaption of our TLM controller to the hardware controller implementation that was embedded into DRAMSys4.0 in Sect. 2.4.

The reference RTL is an open-source DDR3 controller optimized for low power and low latencies. Internally, it is equipped with bank-granular request buffers and a

FIFO scheduler. The translation from incoming addresses to rows, banks and columns can be determined separately for each address bit. Refresh commands are neither postponed nor pulled in, and row precharges are always initiated with a separate precharge command (open-page policy). To reduce power and enable operation at high data rates also with slower logic nodes, the frequency ratio between controller and physical layer is 1:4, i.e., one controller clock cycle corresponds to four clock cycles on the command/address and data bus. New requests can only be accepted every fourth DRAM clock cycle, which, however, is still sufficient to utilize the full memory bandwidth because one DDR3 burst transfer also takes four clock cycles. Additional delays on the command bus are avoided by filling up four upcoming slots in each controller clock cycle. The minimum delay from an incoming request to an associated command being issued by the controller is three clock cycles. Similar to DRAMSys4.0, the RTL also uses a command multiplexer, but with a slightly different prioritization policy.

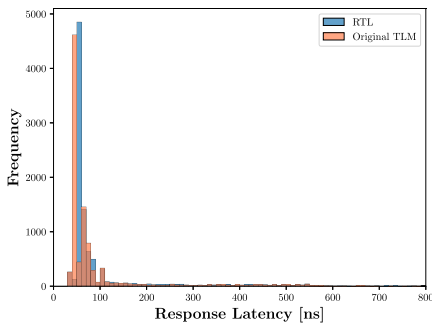
Nearly all specifications of the reference RTL can be directly configured in DRAMSys4.0. Major differences that remain are the 1:4 frequency ratio, which is fixed to 1:1 in DRAMSys4.0, and the exact command multiplexer implementation. By introducing small source code adjustments (i.e., less than 25 lines of code and one man-day of work), we are also able to reproduce these behaviors almost identically. To evaluate the accuracy, the Mediabench benchmarks [21] from Sect. 2.2 are simulated on the (1) RTL controller, (2) TLM controller without source code changes (called “unadjusted”) and (3) TLM controller with small source code changes (called “adjusted”). Table 2 contains the simulation results for average bandwidth without idle phases and average response latency evaluated with

**Table 2** Bandwidth and latency comparison of (1) RTL, (2) unadjusted TLM and (3) adjusted TLM controller for MediaBench benchmarks [21]

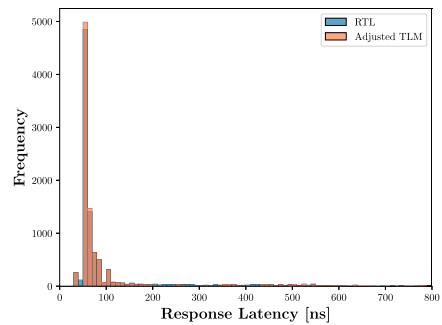
Benchmark	Avg. bandwidth w/o idle [Gib/s]			Avg. response latency [ns]		
	(1)	(2)	(3)	(1)	(2)	(3)
h263decode	12.43	+ 6.89%	+ 2.17%	117.8	− 1.02%	+ 1.36%
g721encode	9.42	+ 4.86%	+ 1.23%	63.6	− 5.03%	− 1.10%
g721decode	11.88	+ 4.03%	+ 0.93%	66.9	− 3.89%	− 0.60%
gsmdecode	15.69	+ 1.94%	+ 0.83%	229.7	− 0.74%	+ 0.22%
c-ray-1.1	11.78	+ 5.34%	+ 1.65%	97.2	− 1.54%	+ 0.62%
fractal	10.38	+ 5.32%	+ 1.01%	90.3	− 5.43%	− 2.55%
jpegdecode	16.03	+ 1.06%	+ 0.71%	305.5	+ 0.13%	+ 0.69%
mpeg2decode	11.70	+ 6.38%	+ 1.09%	81.3	− 6.03%	− 2.09%
unepic	13.08	+ 4.05%	+ 0.66%	197.1	− 0.96%	+ 0.20%
jpegencode	10.53	+ 8.13%	+ 1.68%	83.8	− 6.44%	− 2.03%
epic	11.78	+ 6.36%	+ 1.16%	116.3	− 3.27%	− 0.86%
mpeg2encode	13.07	+ 4.77%	+ 0.83%	118.9	− 2.52%	− 0.42%
h263encode	14.02	+ 4.28%	+ 1.12%	120.4	− 1.91%	+ 0.17%

the Trace Analyzer’s Python interface. The numbers of the RTL controller are presented absolute while the numbers of the TLM controllers are shown as relative deviations from the reference model.

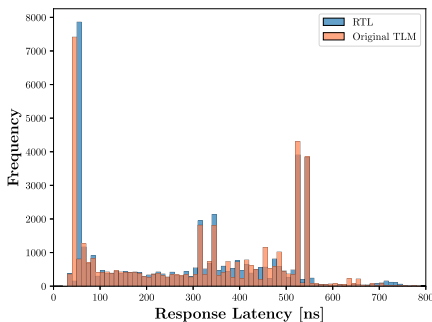
In terms of average bandwidth the adjusted TLM always achieves a higher accuracy compared to the unadjusted TLM with an average absolute deviation of 1.2% compared to 4.9%. In terms of average response latency the results are similar (1.0% compared to 3.0%) except for *h263decode* and *jpegdecode* where the unadjusted TLM controller actually outperforms the adjusted one. It should be noted, however, that averaged metrics have to be treated with caution. Negative and positive deviations cancel each other out, and even a fixed-latency model (see Sect. 1) can be tuned to the reference RTL behavior in a way that it shows the same average bandwidth and response latency. For that reason, the accuracy is also investigated using the new response latency histograms of the Trace Analyzer (see Sect. 2.3). Due to space limitations, we only present histograms of the two aforementioned benchmarks in Fig. 7, noting that all other benchmarks show even smaller deviations.



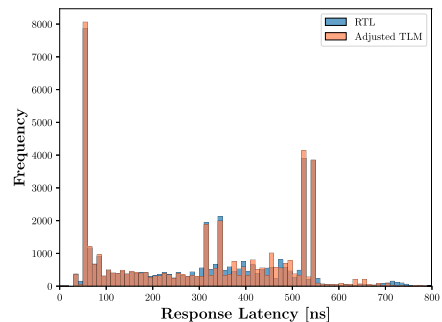
(a) *h263decode*, RTL vs. Unadjusted TLM



(b) *h263decode*, RTL vs. Adjusted TLM



(c) *jpegdecode*, RTL vs. Unadjusted TLM



(d) *jpegdecode*, RTL vs. Adjusted TLM

**Fig. 7** Exemplary response latency histograms of RTL vs. unadjusted TLM and RTL vs. adjusted TLM controller for MediaBench benchmarks [21]

In general, both the unadjusted and the adjusted TLM come close to the reference RTL from a response latency perspective, which reinforces the results in Table 2. The unadjusted TLM model has one large mismatch for the bin between 50 ns and 60 ns, which is caused by the 1:1 clocking that allows new requests to be acknowledged in each clock cycle and not only in every fourth clock cycle. Apart from that, we also see slight deviations of both TLM models for latencies above 300 ns. They result from the exact placement of refresh commands, which is handled less strictly in the reference RTL and has not been adjusted for the sake of simplicity. This leads to a transaction-wise mean relative error in response latency across all benchmarks of 9.75% and 6.12% for the unadjusted and adjusted TLM controller, respectively. When disabling refresh, the adjusted version even achieves an error of less than 1%. In summary, the section proves that DRAMSys4.0 can be closely adapted to reference RTL controller implementations with minimal effort, enabling both fast and accurate simulations at the same time.

## 4 Simulator Expansion by DDR5

Since DRAMSys4.0 is a framework especially used in early design phases for the exploration of new JEDEC standards, it is crucial to enable a fast expansion. As JEDEC has published the latest DRAM standard DDR5 just recently, we demonstrate the simulator's expandability by integrating the new standard. The framework is then also used for a full-system performance comparison to the predecessor standard DDR4.

### 4.1 The DDR5 Standard

Before the new standard is integrated, the most significant differences to its predecessor DDR4 are outlined. DDR5 raises the maximum interface data rate from 3200 MT/s to 6400 MT/s, effectively doubling the maximum theoretical memory bandwidth per data pin. A further increase to 8400 MT/s is planned in the future.

DIMMs are now composed of two independent 32-bits-wide channels instead of one 64-bits-wide channel. This way the internal prefetch can be doubled to support the increased interface speeds without doubling the amount of transferred data per burst access from 64 bytes to 128 bytes. Higher interface speeds also require a higher bank parallelism to keep the sustainable bandwidth up. Therefore, the number of banks per device rises from 16 to 32, while the maximum capacity of a device rises from 16 Gb to 64 Gb. In addition, up to 16 dies can be stacked to form a three-dimensional device with so-called logical ranks (to distinguish them from physical ranks, which are distributed over a DIMM).

One problem that always comes with higher storage capacities is the increased refresh overhead, which can lead to significant performance drops. To overcome this problem, DDR5 introduces a new refresh mode called *same-bank refresh* (REFsb) as an alternative to the conventional all-bank refresh (REFab). When using same-bank refresh commands, only a quarter of all banks is blocked at a time, while all other banks can still process requests. Finally, DDR5 offers special refresh

management commands (RFM<sub>a,b</sub>/RFM<sub>s,b</sub>) to prevent row hammer attacks. They have to be issued by the controller when the number of row activations between refreshes exceeds a certain bank-granular threshold.

## 4.2 Expansion Steps

The expansion of DRAMSys4.0 by a new JEDEC standard starts with the creation of a new timing checker. In comparison to DDR4 the overall number of timing dependencies increases significantly because of new commands (REF<sub>s,b</sub>/RFM<sub>a,b</sub>/RFM<sub>s,b</sub>) and the distinction between logical and physical ranks. Nevertheless, the creation does not take significantly more time because the source code can be generated automatically and error free from a high level domain-specific language description called *DRAMml* [23, 15]. Two channels per DIMM can be modeled with two separate channel controllers and do not require any adjustments. Specifications of different speed grades, device sizes and rank combinations translate into configuration files that are loaded at the start of a simulation.

The most challenging innovation DDR5 comes with is same-bank refresh and refresh management. Same-bank refresh requires a new refresh manager implementation besides the existing all-bank refresh manager, which is much more complex. To minimize any negative impact on bandwidth and latency, refresh commands should be issued preferably to idle banks. While all-bank refresh targets all banks at once and has no room for optimizations, the target banks of a same-bank refresh command are selectable to some extent. However, the flexibility is limited because all banks have to be refreshed once before one bank can be refreshed for a second time. During phases of high DRAM activity, the controller is also allowed to postpone a number of refresh commands to a later point in time. Refresh management brings an additional level of complexity. The manager has to track the number of activate commands to each bank and issue refresh management commands between regular refreshes if a certain threshold is exceeded. Although this behavior has to be modeled as a large state machine, it requires only 300 lines of code and is much less complex compared to a real hardware implementation.

All in all, the whole expansion was performed in about two man-weeks, allowing system designers to generate first simulation results almost immediately after the release of the new standard and long before any hardware is available.

## 4.3 Evaluation

As a last step, the new simulation model is used to compare the performance of DDR5 and DDR4. In [24] it was already shown that in terms of maximum sustainable bandwidth a memory subsystem based on the new standard always outperforms a memory subsystem based on the predecessor. However, the performance of a full system is a far more important metric, since a high sustainable memory bandwidth does not necessarily lead to a high system performance and vice versa. Parameters like average and maximum memory latency play an essential role, too. Therefore, this evaluation focuses on the full-system performance in terms of processor *Instructions Per Cycle* (IPC) that can be

achieved with DDR5 and DDR4 memory subsystems. The following sections describe the experimental setup and present the simulation results, which are summarized into several key observations afterwards.

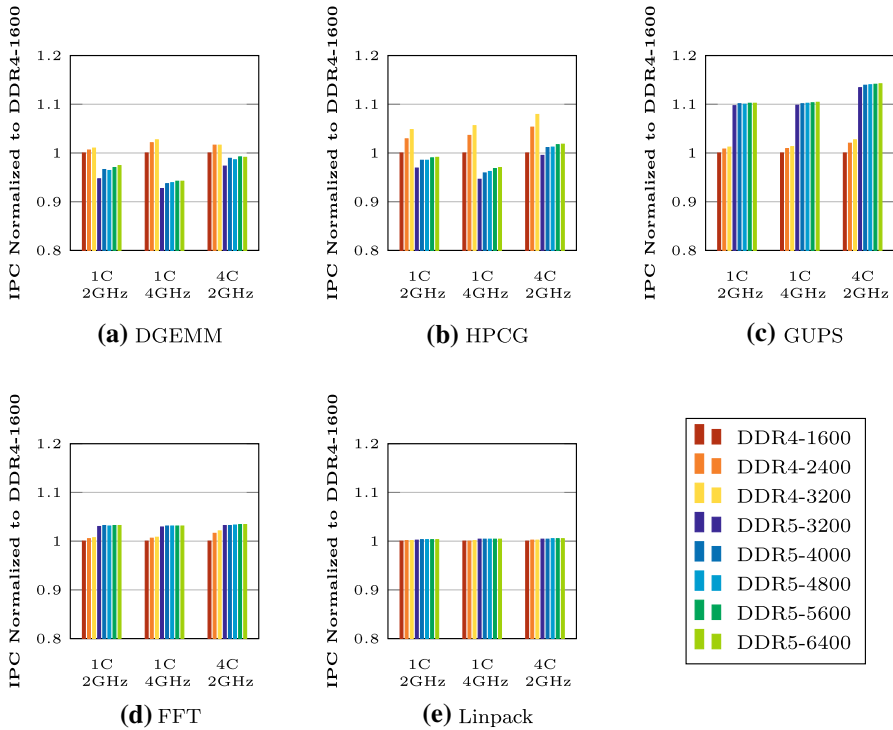
### 4.3.1 Experimental Setup

To generate system performance results we couple DRAMSys4.0 to the latest version of gem5 [25], which is one of the most popular open-source computer architecture simulators and a state-of-the-art tool for performance evaluations. Although gem5 is not based on SystemC/TLM2.0, it can be coupled to TLM-based memory subsystems using special translation modules [26]. That way the gem5 main memory model can be replaced with DRAMSys4.0 for DDR5 simulations, which are not yet supported by the tool natively.

Unfortunately, gem5's detailed processor models have one major drawback: simulation speed. Booting a Linux kernel can already take hours, and quite often benchmarks run for several days or weeks [27]. To avoid such long simulation times with only losing minimal accuracy we make use of prerecorded *elastic traces* [28]. Unlike fixed memory traces they capture data and load/store order dependencies by instrumenting gem5's detailed out-of-order processor model for recording. Afterwards, they can be played back on a special *TraceCPU* model with different cache and main memory subsystems, achieving both high simulation speeds and a high simulation accuracy at the same time. This approach perfectly fits our needs for the evaluation. We select elastic traces of five example applications from the *High-Performance Computing* (HPC) domain: DGEMM [29], HPCG [30], GUPS [29], FFT [31] and Linpack [32]. The performance is evaluated based on the single-core IPC of an ARM processor at 2 GHz and 4 GHz and the multi-core IPC at 2 GHz. The processor is coupled to DDR4 and DDR5 memory subsystems with different speed grades. Table 3 contains detailed information about the system setup.

**Table 3** System setup for evaluation

Processor	ARM ISA, 16/16-entry load/store buffers, 40-entry ROB, 1 core @ 2 GHz/4 GHz, 4 cores @ 2 GHz
Caches	per-core L1-I cache: 48 kB, 3-way set associative, per-core L1-D cache: 32 kB, 2-way set associative, shared L2 cache: 2 MB for each core, 16-way set associative
Channel controller	32/32-entry read/write request queues, FR-FCFS scheduling [33], open-page policy
DDR4 memory	32 GB, distributed over 1 channel, 2 ranks per channel, 16 banks per rank, all-bank refresh, speed grades 1600, 2400, 3200
DDR5 memory	32 GB, distributed over 2 channels, 2 ranks per channel, 32 banks per rank, same-bank refresh, speed grades 3200, 4000, 4800, 5600, 6400



**Fig. 8** Processor IPC for HPC benchmarks

### 4.3.2 Simulation Results

The simulation results for all benchmarks are presented in Figure 8. To show the relative performance improvement that is achieved with faster DDR4 speed grades and different DDR5 speed grades the IPC is normalized to the slowest memory (DDR4-1600). At first sight the results can be grouped into three categories. (1) For DGEMM and HPCG (Figure 8a and 8b) DDR4 slightly outperforms DDR5 in single-core IPC at 2 GHz (up to 8% difference). (2) For GUPS and FFT (Figures 8c and 8d) the results are the other way round, i.e., DDR5 slightly outperforms DDR4 in single-core IPC at 2 GHz (up to 10% difference). (3) For Linpack (Figure 8e) the IPC is almost independent of the selected memory subsystem. Furthermore, the results of DDR5 are particularly bad at a higher CPU frequency (4 GHz), where it loses up to 11% in IPC over DDR4 for HPCG, while it performs especially well in the multi-core case, gaining up to 14% in IPC over DDR4 for GUPS. In order to understand these results we have to take a look at the average memory bandwidth of the benchmarks using the Trace Analyzer's metrics.



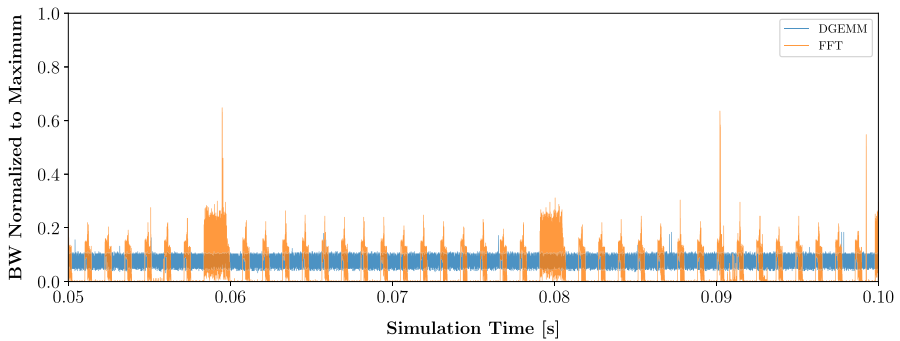
For Linpack the average memory bandwidth is less than 1% of the maximum theoretical bandwidth<sup>6</sup> in all test cases. This explains why the results are independent of the selected memory subsystem: the application accesses memory only very rarely, thus its performance is barely influenced by the access delays. For all other applications the average memory bandwidth varies between 5% for FFT with DDR4-1600 and up to 11.5% for HPCG with DDR4-1600 executed on a single core. With four cores active these values approximately multiply by four. Although in theory the fastest DDR5 device (DDR5-6400) can achieve double the bandwidth of the fastest DDR4 device (DDR4-3200), DDR5 outperforms DDR4 for the FFT benchmark with a lower average bandwidth and loses out against DDR4 for HPCG when more bandwidth is required. This shows us that apparently the average memory bandwidth is not the only parameter that influences the results. To find the actual reason, the memory access patterns and bandwidth distribution over time have to be investigated. At this point the Trace Analyzer's new plots pay off.

HPCG and DGEMM distribute all memory accesses evenly over time (see Fig. 9 blue curve). Even the slowest memory subsystem DDR4-1600 can handle the requests one after another, they neither overlap nor influence each other. FFT and GUPS show different access schemes, which are mixtures of high-bandwidth and idle segments (see Fig. 9 orange curve). These applications require a lot of data at once, but not during the whole runtime. Since DDR5 is optimized for higher bandwidths, it outperforms DDR4 in those cases. The reason why DDR4 actually performs better than DDR5 for HPCG and DGEMM can be attributed to the minimum memory access latency: besides the increase in bandwidth, DDR5 also comes with increased access latencies over DDR4, which are caused by the hardware shrinking, higher bank count and increased burst length [34, 35]. After calculating the average memory access latency for DGEMM and HPCG with the Trace Analyzer metrics, we observe that accesses to DDR4 are already completed after around 32 ns, while accesses to DDR5 take around 38 ns. This results in more waiting cycles for the processor and a lower IPC count. Higher speed grades slightly reduce the latency, but its major part is independent of the speed grade.<sup>7</sup> Similar observations have already been made when DDR4 was compared to its predecessor DDR3 [36].

Using this background information, we are now also able to explain the results for a higher CPU frequency and for the multi-core simulations. At a higher CPU frequency the memory requests move closer together. For DGEMM and HPCG all DDR4 memories can still handle the requests fast enough and keep the IPC up due to their low latency. In contrast, the higher latency of DDR5 has an even bigger negative impact on the IPC because with a higher frequency the number of waiting cycles in the processor increases. For the multi-core simulations the requests do not move closer together, but multiple memory requests arrive at the same time. As we

<sup>6</sup> The maximum theoretical memory bandwidth is calculated as the product of DRAM interface width and DRAM interface speed. For a single channel of DDR4-1600 the maximum theoretical bandwidth is  $64 \text{ bit} \cdot 1600 \text{ MT/s} = 12.8 \text{ GB/s}$ .

<sup>7</sup> A higher speed grade only reduces the time it takes to transfer the data over the interface. Delays for precharging and activating banks or for transferring data from the DRAM core to the interface are independent of the speed grade.



**Fig. 9** Bandwidth over time behavior of DGEMM and FFT for DDR4-1600

have already observed previously, DDR5 is optimized for high-bandwidth segments and achieves even more performance compared to DDR4.

One last subject to look into is the actual impact of higher speed grades on the system performance. For the latency-critical benchmarks (DGEMM and HPCG) we see an improvement with higher speed grades both for DDR4 (up to 8%) and DDR5 (up to 3%). It is achieved because the burst duration decreases, whereby the minimum memory latency becomes shorter. However, when we double the interface frequency from DDR4-1600 to DDR4-3200 or from DDR5-3200 to DDR5-6400, respectively, the latency only changes by 2.5 ns in both cases, which is small compared to the overall latency of 32 ns for DDR4 and 38 ns for DDR5 that we have measured before. For the bandwidth-critical benchmarks (GUPS and FFT) the result impact is a lot smaller, although the maximum theoretical bandwidth is directly proportional to the interface frequency. This highlights that DDR5 gains its performance over DDR4 not from faster interface speeds, but mainly from the new architecture with two separate memory channels, an increased amount of memory banks and the same-bank refresh mechanism, which allows the handling of more requests in parallel. Faster interface speeds only pay off when systems max out the upper bandwidth limits (e.g., data-flow dominated applications or high processor core counts). One important parameter that is not taken into consideration in this evaluation, but also has to be kept in mind, is the out-of-order capability of each application, i.e., if memory access latencies can be hidden by performing subsequent operations in the meantime. This parameter might also shift the results in both directions.

### 4.3.3 Key Observations

To conclude this section, we sum up the key observations of our evaluation:

- DDR4 outperforms DDR5 for system configurations with low memory bandwidth requirements due to its lower latency.
- DDR5 outperforms DDR4 for system configurations with high memory bandwidth requirements because of the DRAM architecture changes.

- Higher speed grades mostly pay off when system configurations exhaust the upper memory bandwidth limits.

## 5 Related Work

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

### 5.1 Cycle-Accurate Simulators

As stated in the introduction, there are several publicly-available cycle-accurate DRAM simulators. Table 4 compares all of them, including DRAMSys3.0 and DRAMSys4.0. For simplicity, we only focus on DRAM standards specified by JEDEC since they are the most relevant ones for system developers.

DRAMSys3.0, DRAMSim2 and DrSim were already developed several years ago but never updated over time, thus supporting only 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 currently offers a model for the latest JEDEC standard DDR5. For request initiation all simulators provide trace players and a coupling to gem5. In addition, DRAMsim3 supports a coupling to the simulation frameworks SST [40] and ZSim [41]. DRAMSys3.0 and DRAMSys4.0 can be coupled to TLM-AT-based core models. While all simulators seem to be JEDEC compliant at first view, we were able to find missing timing dependencies in DRAMsim3, Ramulator and in the gem5 DRAM model (e.g., missing command bus dependencies for multi-cycle commands. Besides the performance perspective, for most of today's system developments the power consumption and thermal behavior is of great interest, especially in the field of embedded systems. All simulators except DrSim allow power estimations. 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. DRAMSim2 additionally supplies DRAMVis [6], 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 Sect. 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 relation 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<sup>8</sup>), the memory

<sup>8</sup> 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.

Table 4 Overview of the most prominent open-source DRAM simulators

Feature	DRAMsys4.0 (this work)	DRAMsys3.0 [11]	DRAMSIM2 [6]	DRAMsim3 [7]	Ramulator [8]	DrSim [9]	gem5 DRAM Model [14]
DRAM standards	DDR3/4/5, LPDDR4/4X/5, 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, LPDDR2/3/5, Wide I/O 1, GDDR5, HBM1
Refresh modes	All-bank, per-bank, per-2-bank, same-bank	All-bank	All-bank	All-bank, per-bank	All-bank, per-bank	All-bank	All-bank
Power-down modes	Active and precharge power-down, self-refresh	Active and precharge power-down, self-refresh	Precharge power-down	Self-refresh	Active and precharge power-down, self-refresh	Active and precharge power-down	Active and precharge power-down, self-refresh
Address mappings	Any bijective boolean function [37]	Any bijective boolean function [37]	7 Different mappings	Mappings with hierarchy granularity	Mappings with hierarchy granularity	Fixed mapping with optional interleavings	3 different mappings
Scheduling policies	FCFS, FR-FCFS [33], FR-FCFS Grouping	FCFS, FR-FCFS [33], FR-FCFS Grouping, Par-BS [38], SMS [39]	Issue requests ASAP	Issue requests ASAP	FCFS, FR-FCFS [33], FR-FCFS Cap., FR-FCFS PriorHit	FCFS, FR-FCFS [33]	FCFS, FR-FCFS [33]
Page policies	Open, open adaptive, closed, closed adaptive [14]	Open, closed	Open, closed	Open, closed	Open, closed, closed with auto precharge, timeout	Open, closed	Open, open adaptive, closed, closed adaptive [14]
Configuration	Controller policies, address mapping, DRAM standard, organization, timings and currents (JSON file)	Controller policies, address mapping, DRAM standard, organization, timings and currents (XML file)	Controller policies, address mapping, DRAM organization, timings and currents	Controller policies, address mapping, DRAM standard, organization, timings and currents	Address mapping, DRAM standard, speed bin and size	Controller policies, address mapping, interleaving, DRAM organization and timings	Controller policies, address mapping, DRAM organization, timings and currents (Python file)

Table 4 continued

Feature	DRAMsys4.0 (this work)	DRAMsys3.0 [11]	DRAMsim2 [6]	DRAMsim3 [7]	Ramulator [8]	DrSim [9]	gem5 DRAM Model [14]
Request initiators	Trace players (fixed and elastic memory traces), SystemC-based core models, gem5 core models	Trace players (fixed and 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 [40], ZSim [41]	Trace player (fixed, untyped memory traces and timed CPU traces), gem5 core models	Trace player (fixed memory traces), gem5 core models	Trace players (fixed and elastic memory traces), gem5 core models
Power estimation	DRAMPower [17]	DRAMPower [17]	Micron power model	Micron power model and DRAMPower [17]	DRAMPower [17] and Vampire [42]	–	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 and result checking [23], 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 and power consumption, metrics for SQLite command trace	Average bandwidth and power consumption, metrics for SQLite command trace	Bandwidth, latency and 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 [6]	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 [12])	TLM-based (IEEE 1666 SystemC/TLM2.0 [12])	Loop-based	Loop-based	Loop-based	Loop-based	TLM-based (gem5 [13])

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 bandwidth, latency and total simulated time of all simulators are nearly identical (maximum deviations of 2 % because all simulators implement a different power-down operation).

The simulation speeds of all simulators are shown in Figure 10. 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, 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 lower densities (wall-clock time to evaluate pure idle cycles), the TLM-based simulators show their advantage by achieving a linear decrease, clearly outperforming all loop-based simulators. During long idle phases they initiate self-refresh operation of the DRAM devices. This way external refresh commands can be omitted and no clock cycles have to be evaluated at all. Since real applications often contain idle phases and resulting average memory access densities are located in 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 a 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 a result of the optimizations explained in Sect. 2.2. The simulation speeds of DRAMSys4.0 and the gem5 DRAM model are on the same level for high densities. At densities lower than  $10^{-3}$  the gem5 DRAM model starts to become slightly slower than DRAMSys4.0 because the switching to self-refresh operation takes more time.

As mentioned in the introduction, a DRAM subsystem simulation represents one specific memory controller implementation and one specific JEDEC standard. Although all simulators are more or less compliant with their supported JEDEC standards, each one implements a different controller behavior (power-down policy, refresh command placement, request buffer architecture, additional hardware delays, etc.). Similarly, there is no *golden* reference RTL controller to evaluate the accuracy against. Cycle accuracy can therefore only be assessed with respect to JEDEC compliance. To keep the comparison fair and to avoid making one simulator mistakenly look bad, no direct comparison of the accuracy is conducted. However, one alternative measure that can be used is the degree of configurability, because a high configurability is key to tune a simulator's behavior to a reference RTL controller. Table 4 shows that DRAMSys4.0 is at the top for almost all listed features. Furthermore, the ease of adapting our simulator to one specific RTL controller has been demonstrated in Sec. 3.

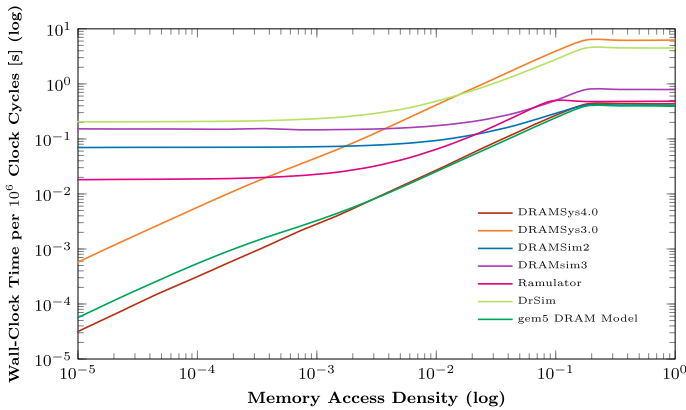


Fig. 10 Simulation speeds of state-of-the-art DRAM simulators

## 5.2 Cycle-Approximate DRAM Models

Beside the cycle-accurate DRAM simulators, further approaches exist that approximate the behavior (see Fig. 1). In [43] the authors propose an analytical DRAM performance model that uses traces to predict the efficiency of the DRAM subsystem. Todorov et al. [3] 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 [2] presents the modeling of DRAM behavior using decision trees. In [4] the authors introduce a performance-optimized DRAM model that is based on a neural network.

## 6 Conclusion

In this paper we presented DRAMSys4.0, an open-source cycle-accurate DRAM simulation framework. Due to the optimized architecture it reaches very high simulation speeds compared to state-of-the-art simulators while ensuring full JEDEC compliance. DRAMSys4.0 supports a large collection of controller features and standards, which allows system designers to adapt the tool to their needs with minimal effort. Moreover, it offers the unique Trace Analyzer tool for deep analyses and truthful design space exploration. For the future we plan to extend DRAMSys4.0 by further emerging JEDEC standards and analysis features.

**Acknowledgements** This work was supported within the Fraunhofer and DFG cooperation programme (Grant No. 248750294) and supported by the Fraunhofer High Performance Center for Simulation- and Software-based Innovation. Furthermore, we thank Rambus for their support.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Jacob, B.: *The Memory System: You Can'T Avoid It, You Can'T Ignore It*. Morgan and Claypool Publishers, You Can'T Fake It (2009)
2. Li, S., Jacob, B.: Statistical DRAM modeling. In: *Proceedings of the International Symposium on Memory Systems, MEMSYS '19*, p. 521–530. Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3357526.3357576>
3. Todorov, V., Mueller-Gritschneider, D., Reinig, H., Schlichtmann, U.: 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*, pp. 1066–1071. EDA Consortium, San Jose, CA, USA (2012). <http://dl.acm.org/citation.cfm?id=2492708.2492972>
4. Jung, M., Feldmann, J., Kraft, K., Steiner, L., Wehn, N.: Fast and accurate DRAM simulation: can we further accelerate it? In: *IEEE Conference on Design, Automation and Test in Europe (DATE)*, pp. 364–369. Grenoble, France (2020)
5. Sudarshan, C., Lappas, J., Weis, C., Mathew, D.M., Jung, M., Wehn, N.: A Lean, Low power, low latency DRAM memory controller for transprecision computing. In: Pnevmatikatos, D.N., Pelcat, M., Jung, M. (eds.) *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 429–441. Springer, Cham (2019)
6. Rosenfeld, P., Cooper-Balis, E., Jacob, B.: DRAMSim2: a cycle accurate memory system simulator. *Comput. Archit. Lett.* **10**(1), 16–19 (2011). <https://doi.org/10.1109/L-CA.2011.4>
7. Li, S., Yang, Z., Reddy, D., Srivastava, A., Jacob, B.: DRAMsim3: a cycle-accurate, thermal-capable DRAM simulator. *IEEE Comput. Archit. Lett.* pp. 1–1 (2020). <https://doi.org/10.1109/LCA.2020.2973991>
8. Kim, Y., Yang, W., Mutlu, O.: Ramulator: A fast and extensible DRAM simulator. *IEEE Comput. Archit. Lett.* **PP**(99), 1–1 (2015). <https://doi.org/10.1109/LCA.2015.2414456>
9. Jeong, M.K., Yoon, D.H., Erez, M.: DrSim: a platform for flexible DRAM system research. <http://lph.ece.utexas.edu/public/DrSim>. Accessed 15 Aug 2019
10. Li, S., Verdejo, R.S., Radojkoviundefined, P., Jacob, B.: Rethinking Cycle Accurate DRAM Simulation. In: *Proceedings of the International Symposium on Memory Systems, MEMSYS '19*, p. 184–191. Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3357526.3357539>
11. Jung, M., Weis, C., Wehn, N.: DRAMSys: a flexible DRAM subsystem design space exploration framework. In: *IPSS Transactions on System LSI Design Methodology (T-SLDM)* (2015). <https://doi.org/10.2197/ipsjtsldm.8.63>
12. IEEE Computer Society: *IEEE Standard for Standard SystemC Language Reference Manual (IEEE Std 1666-2011)* (2012)
13. Binkert, N., et al.: The gem5 simulator. *SIGARCH Comput. Archit. News* **39**(2), 1–7 (2011). <https://doi.org/10.1145/2024716.2024718>
14. Hansson, A., Agarwal, N., Kollu, A., Wenisch, T., Udipi, A.: Simulating DRAM controllers for future system architecture exploration. In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 201–210 (2014). <https://doi.org/10.1109/ISPASS.2014.6844484>
15. Steiner, L., Jung, M., Prado, F.S., Bykov, K., Wehn, N.: DRAMSys4.0: a fast and cycle-accurate systemC/TLM-based DRAM simulator. In: A. Orailoglu, M. Jung, M. Reichenbach (eds.) *Embedded*



- Computer Systems: Architectures, Modeling, and Simulation, pp. 110–126. Springer International Publishing, Cham (2020)
16. Jung, M., Weis, C., Wehn, N., Chandrasekar, K.: 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, RAPID0 '13, pp. 5:1–5:6. ACM, New York, NY, USA (2013). <https://doi.org/10.1145/2432516.2432521>
  17. Chandrasekar, K., et al.: DRAMPower: open-source DRAM power and energy estimation tool. <http://www.drampower.info>. Accessed 15 Aug 2019
  18. Sridhar, A., Vincenzi, A., Ruggiero, M., Brunswiler, T., Atienza, D.: 3D-ICE: fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling. In: Proceedings of ICCAD 2010 (2010)
  19. Cadence Inc.: CadenceR DenaliR DDR Memory IP. <http://ip.cadence.com/ipportfolio/ip-portfolio-overview/memory-ip/ddr-lpddr>. Accessed 15 Oct 2021
  20. Synopsys, Inc.: DesignWare DDR IP. <https://www.synopsys.com/designware-ip/interface-ip/ddr.html> (2021, Last Access: 12.10.2021)
  21. MediaBench Consortium: Mediabench. <http://euler.slu.edu/~fritts/mediabench/>. Accessed 28 Feb 2015
  22. Muhr, H., Holler, R.: Accelerating RTL simulation by several orders of magnitude using clock suppression. In: 2006 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pp. 123–128 (2006). <https://doi.org/10.1109/ICSAMOS.2006.300818>
  23. Jung, M., Kraft, K., Soliman, T., Sudarshan, C., Weis, C., Wehn, N.: Fast validation of DRAM protocols with timed Petri nets. In: Proceedings of the International Symposium on Memory Systems, MEMSYS '19, pp. 133–147. ACM, New York, NY, USA (2019). <https://doi.org/10.1145/3357526.3357556>
  24. Steiner, L., Jung, M., Wehn, N.: Exploration of DDR5 with the open-source simulator DRAMSys. In: MBMV 2021–24. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, pp. 31–41 (2021)
  25. Lowe-Power, J., et al.: The gem5 simulator: version 20.0+. ArXiv [abs/2007.03152](https://arxiv.org/abs/2007.03152) (2020)
  26. Menard, C., Jung, M., Castrillon, J., Wehn, N.: System simulation with gem5 and SystemC: the keystone for full interoperability. In: 2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 62–69 (2017). <https://doi.org/10.1109/SAMOS.2017.8344612>
  27. Sandberg, A., Nikoleris, N., Carlson, T.E., Hagersten, E., Kaxiras, S., Black-Schaffer, D.: Full speed ahead: detailed architectural simulation at near-native speed. In: 2015 IEEE International Symposium on Workload Characterization, pp. 183–192 (2015). <https://doi.org/10.1109/IISWC.2015.29>
  28. Jagtap, R., Diestelhorst, S., Hansson, A., Jung, M., Wehn, N.: Exploring system performance using elastic traces: fast, accurate and portable. In: IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS), July, 2016, Samos Island, Greece (2016)
  29. Luszczek, P.R., et al.: The HPC challenge (HPCC) benchmark suite. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06. ACM, New York, NY, USA (2006). <https://doi.org/10.1145/1188455.1188677>
  30. Dongarra, J., Heroux, M.A., Luszczek, P.: A new metric for ranking high-performance computing systems. *Natl. Sci. Rev.* **3**(1), 30 (2016). <https://doi.org/10.1093/nsr/nwv084>
  31. Takahashi, D.: FFTE: a fast Fourier transform package (2014). <http://www.ffte.jp/>
  32. Petitet, A., Whaley, R.C., Dongarra, J., Cleary, A.: HPL—a portable implementation of the high-performance linpack benchmark for distributed-memory computers (2016). <http://www.netlib.org/benchmark/hpl/>
  33. Rixner, S., Dally, W.J., Kapasi, U.J., Mattson, P., Owens, J.D.: Memory access scheduling. In: Proceedings of the 27th Annual International Symposium on Computer Architecture, ISCA '00, pp. 128–138. ACM, New York, NY, USA (2000). <https://doi.org/10.1145/339647.339668>
  34. JEDEC Solid State Technology Association: DDR4 SDRAM (JESD79-4C) (2020)
  35. JEDEC Solid State Technology Association: DDR5 SDRAM (JESD79-5) (2020)
  36. Ghose, S., Li, T., Hajinazar, N., Senol Cali, D., Mutlu, O.: Demystifying complex workload-DRAM interactions: an experimental study. pp. 93–93 (2019)
  37. Jung, M., Heinrich, I., Natale, M., Mathew, D.M., Weis, C., Krumke, S., Wehn, N.: ConGen: An application specific DRAM memory controller generator. In: Proceedings of the Second International Symposium on Memory Systems, MEMSYS '16, pp. 257–267. ACM, New York, NY, USA (2016). <https://doi.org/10.1145/2989081.2989131>

38. Mutlu, O., Moscibroda, T.: 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. (2008). <http://research.microsoft.com/apps/pubs/default.aspx?id=79626>
39. Ausavarungnirun, R., Chang, K.K.W., Subramanian, L., Loh, G.H., Mutlu, O.: Staged memory scheduling: achieving high performance and scalability in heterogeneous systems. In: Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12, pp. 416–427. IEEE Computer Society, Washington, DC, USA (2012). <http://dl.acm.org/citation.cfm?id=2337159.2337207>
40. Rodrigues, A.F., et al.: The structural simulation toolkit. SIGMETRICS Perform. Eval. Rev. **38**(4), 37–42 (2011). <https://doi.org/10.1145/1964218.1964225>
41. Sanchez, D., Kozyrakis, C.: ZSim: fast and accurate microarchitectural simulation of thousand-core systems. ACM SIGARCH Comput. Archit. News **41**, 475 (2013). <https://doi.org/10.1145/2508148.2485963>
42. Ghose, S., 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) (2018). <https://doi.org/10.1145/3224419>
43. Yuan, G.L., Aamodt, T.M.: A hybrid analytical DRAM performance model (2009)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.