

Exploration of DDR5 with the Open-Source Simulator DRAMSys

M.Sc. Lukas Steiner¹, Dr.-Ing. Matthias Jung², Prof. Dr.-Ing. Norbert Wehn¹

1: Technische Universität Kaiserslautern, Germany {lsteiner, wehn}@eit.uni-kl.de

2: Fraunhofer IESE, Kaiserslautern, Germany, matthias.jung@iese.fraunhofer.de

Abstract

Over the last five decades, we have seen a continuous evolution in DRAM technology, always targeting lower cost per bit, higher device capacity, higher bandwidth, and lower power consumption. The most recent DRAM standard released by JEDEC in mid 2020 is DDR5. It exhibits several new features, for example two channels on a single DIMM, same-bank refresh, and data rates up to 8400 MT/s. As a result, DDR5 greatly enlarges the DRAM device options, while the selection of a suitable device heavily depends on the application. In this paper, we investigate the performance of the new DDR5 standard in depth, compare it to its predecessor DDR4, and derive key observations that help selecting a suitable DRAM configuration. We present a new DDR5 simulation model based on the open-source simulator DRAMSys. This is to the best of our knowledge the first DDR5 simulation model. The model is the base for all of our investigations.

1 Introduction

Currently, we see a strong shift to memory-dominated applications. Thus, *Dynamic Random Access Memories* (DRAMs) play a large role in compute platforms. Over the last years, the number of DRAM standards specified by the *JEDEC Solid State Technology Association* has been growing rapidly. The most recent DRAM standard is DDR5, which was released in mid 2020. Because of the large amount of new features, system designers are either challenged to adopt the new standard or they can move on with well-established standards like DDR4. If DDR5 is a potential candidate for a specific application, a further challenge is the configuration of the DDR5 subsystem, which features a lot of parameter choices. Fast and accurate simulation models are mandatory to explore the new features and compare different configurations.

For our investigations we use the design space exploration framework DRAMSys, which relies on a *Domain Specific Language* (DSL) to specify the DRAM's architecture, states, and timing behavior in a compressed and comprehensive fashion. This DSL, called *DRAMml* [1], is based on a Petri Net semantic [2] and allows correct-by-construction source code generation for DRAMSys. To achieve a high simulation speed and high accuracy at the same time, the framework makes use of SystemC *Transaction Level Modeling* (TLM) and only simulates relevant state changes instead of individual signal changes in each clock cycle. In this way, the overall number of events is reduced drastically and performance results can be generated several orders of magnitude faster compared to an RTL simulation while maintaining the same accuracy.

By using DRAMml for the source code generation, we were able to create a full-featured DDR5 simulation model from scratch within two weeks. This shows the applicability of the DRAMml methodology within the DRAMSys framework for the adoption of new DRAM standards.

In this paper, we compare the DDR5 standard to its pre-

decessor DDR4 and investigate advantages and disadvantages for specific applications. Furthermore, we perform an in-depth analysis of different DDR5 configurations. In summary, the paper presents three new contributions:

- We present the first DDR5 simulation model, which is integrated in the DRAMSys framework.
- We provide an in-depth comparison between DDR5 and DDR4.
- We investigate the performance of different DDR5 configurations.

The paper is structured as follows: Section 2 gives some background on DRAM technology and the new main features of the DDR5 standard. In Section 3 we describe the DRAM design space exploration framework DRAMSys. The integration of the DDR5 model is described in Section 4. Experiments and key observations are presented in Section 5. In Section 6 the related work is reviewed and the paper is concluded by Section 7.

2 DRAM Background and DDR5

In this section we introduce the basic terminology of DRAM devices and their controllers and give an overview on the new features of the DDR5 standard.

2.1 DRAM Basics

As shown in Figure 1, DRAM can be organized in a multi-hierarchical fashion of *DIMMs*, *channels*, *physical ranks*, *devices*, *logical ranks*, *bank groups*, *banks*, *memory arrays*, *sub arrays*, *rows*, and *columns*. Several DRAM *channels* can be connected to the *Multi-Processor System on Chip* (MPSoC). These channels are completely independent of each other and have separate command/address and data buses. A channel can be composed of one or multiple *physical ranks*, which are sharing the data

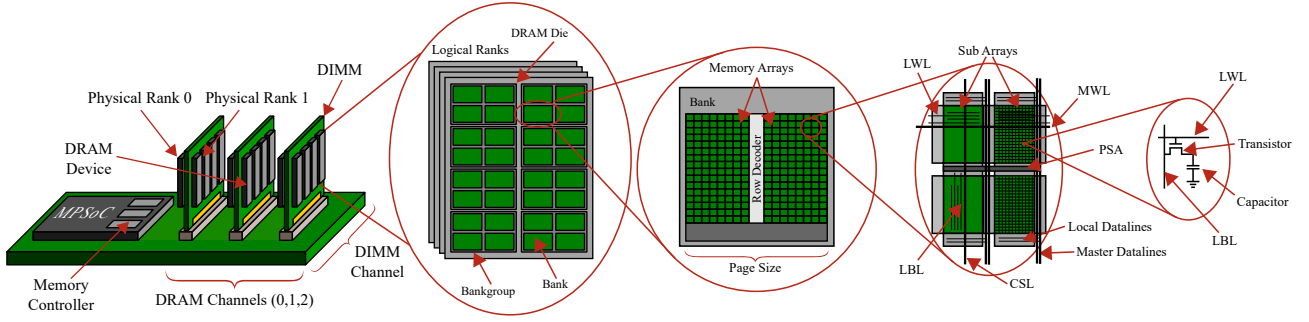


Figure 1 DRAM Architecture

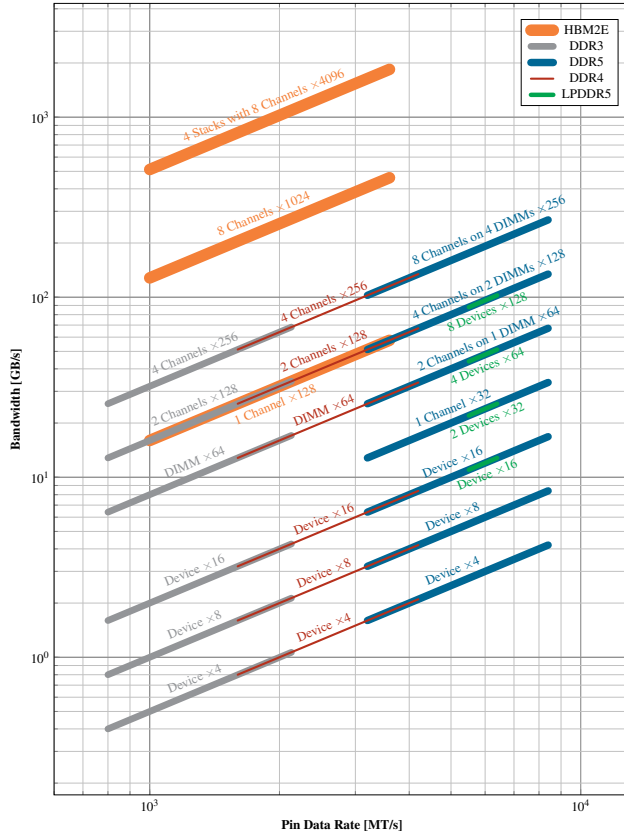


Figure 2 Bandwidth Evolution of DRAM Standards

and command/address bus. A *Dual Inline Memory Module* (DIMM) is a small PCB that accommodates several DRAM devices, which work completely synchronously. One single device is called $\times 16^1$ if it has an I/O data width of 16 bit. A DIMM is assembled for instance out of four $\times 16$ devices in order to have a total I/O data width $n = 64$ bit (called $\times 64$). While the I/O data width is usually very limited, inside the DRAM a lot of data can be fetched or stored in parallel. However, the time between consecutive internal data accesses is very long due to the optimization for storage density, while the interface can be operated on much higher frequencies. To align this mismatch DRAM uses a so-called *prefetching* technique. For a read large chunks of data are concurrently fetched to the interface and then transferred in one burst to the re-

¹pronunciation: by-sixteen

quester, for a write the process is reversed. In addition, data is transferred at the doubled interface frequency (*double data rate*, short DDR). Current devices such as DDR4 use an $8n$ prefetch architecture, where n is the I/O data width, 8 the *Burst Length* (BL) and $8n$ the number of bits for an internal data transfer. That means with each DRAM access the total amount of data received or delivered is $BL \cdot n = 8 \cdot 64 \text{ bit} = 512 \text{ bit} = 64 \text{ B}$, which is the usual cache line size in today's computing systems. In combination with interface frequencies up to 1600 MHz or pin transfer rates up to 3200 MT/s (megatransfers per second) DDR4 reaches a maximum bandwidth of 25.6 GB/s per channel. Each device itself can consist of several 3D-stacked *logical ranks*, which can form several *bank groups* that include several *banks*. The concept of bank groups was introduced with GDDR5 and DDR4 in order to reduce the bank switching times to support a seamless burst behavior at high data rates and therefore a high bandwidth. All banks in a whole channel can be used concurrently (so-called *bank parallelism*). However, there are some constraints due to the shared buses. Each bank usually consists of 2^{12} to 2^{18} rows, and each row can usually store 512 B to 2 KB of data in its *columns*.

A memory controller is composed of a *front end* and a *back end*. The front end performs arbitration and scheduling of incoming read and write requests, whereas the task of the back end is to translate these incoming requests into a sequence of DRAM commands, which have to be orchestrated with respect to the current state of the device. To access data in a row of a certain bank, an *activate* (ACT) command must be issued by the controller before any column access, i.e., *read* (RD) or *write* (WR) commands, can be executed. The ACT command opens an entire row of the memory array, which is transferred into the bank's *row buffer*². It acts like a small cache that stores the most recently accessed row of the bank. The latency of a memory access to a bank largely varies depending on the state of this row buffer. If a memory access targets the same row as the currently cached row in the buffer (called *row hit*), it results in a low latency and low energy memory access.

²The row buffer is a model that abstracts the real physical DRAM architecture. It is basically a combination of *Primary Sense Amplifiers* (PSAs) and *Secondary Sense Amplifiers* (SSAs) of the memory arrays in one bank. This model is useful to describe the functionality of a memory controller and its scheduling algorithms. Unfortunately, it often leads to a misunderstanding of the real DRAM architecture. For further details on the internal DRAM architecture we refer to [3].

Whereas, if a memory access targets a different row as the current row in the buffer (called *row miss*), it results in a higher latency and energy consumption. If a certain row in a bank is active it must first be *precharged* (PRE) before another row can be activated. In addition to the normal RD and WR commands, there exist read and write commands with an integrated *auto-precharge* (RDA, WRA). If auto-precharge is selected, the row being accessed will be precharged automatically at the end of the read or write access. Because a DRAM cell uses a capacitor with leakage effects for data storage, it usually has to be refreshed every 64ms to retain the data stored in it. Modern DRAMs are equipped with an *all-bank refresh* (REFab) command to perform this operation automatically on all banks of a rank in parallel. However, a prerequisite is that all banks are in a precharged state. This can be achieved by issuing a special *all-bank precharge* (PREab) command in advance.

In addition to the commands, each DRAM standard defines a set of timing dependencies, which are temporal constraints that must be satisfied between issued commands. For example, between two ACT commands to the same bank the timing dependency t_{RC} (*row cycle time*) must be satisfied. Timing dependencies can also exist on other hierarchies of the DRAM, e.g., between commands to the same bank group, to the same logical/physical rank or to different logical/physical ranks. The selection of a DRAM subsystem usually has three main dimensions: bandwidth, latency, and capacity. Bandwidth is the amount of data that can be transferred between DRAM and a computational unit within a given time. As shown in Figure 2, the maximum theoretical DRAM bandwidth is limited to the number of data pins times the interface pin data rate (number of accesses per time per pin). Latency is the time that it takes to complete an access. In fact, latency helps bandwidth, but not vice versa [4]. For instance, lower DRAM latency results in more accesses per time, and therefore higher bandwidth, whereas increasing the number of data pins increases the bandwidth without decreasing latency. In realistic scenarios, the full theoretical bandwidth is never reached due to many timing dependencies, interference between different requests, and refresh. The actual achieved bandwidth for a specific application is called *sustainable bandwidth*.

2.2 DDR5 Standard

With the development of a new DRAM standard generation there are always several key parameters that should be enhanced, e.g., bandwidth, power consumption, and device capacity. Table 1 shows a comparison between key parameters of the new DDR5 standard and its predecessor DDR4. In the following we will also describe the most important differences in more detail.

For a higher bandwidth DDR5 raises the maximum pin data rate to 8400 MT/s compared to 3200 MT/s for DDR4. Because the frequency of internal data accesses stays more or less the same as a result of the capacity- and cost-optimized architecture, the prefetch was incremented from 8n to 16n. When using the same 64-bit-wide data bus for one channel as all previous DDR generations, this would

Table 1 Comparison of DDR4 and DDR5 Key Parameters

	DDR4	DDR5
Data Rates [MT/s]	1600 - 3200	3200 - 8400
Max. Channel BW [GB/s]	25.6	33.6
Max. DIMM BW [GB/s]	25.6	67.2
Error Correction	-	On-Die
Device Capacity [Gb]	2 - 16	8 - 64
Max. Stacked Devices	8	16
Channel Width [bit]	64	32
Channels per DIMM	1	2
Prefetch	8n	16n
Max. Banks	16	32
Max. Bank Groups	4	8
Refresh Modes	All-Bank	All-Bank, Same-Bank
Burst Length	8	16, 32
Supply Voltage [V]	1.2	1.1

result in 128 B of transferred data per access. However, since the usual cache line size of modern processors is only 64 B, the data bus of each DDR5 DIMM is split up into two independent channels of 32 bit width. That way only 64 B of data are transferred per access. Theoretical transfer rates then reach a maximum of 33.6 GB/s per channel and 67.2 GB/s per DIMM compared to 25.6 GB/s per channel/DIMM for DDR4, as shown in Figure 2. At the same time supply voltages are reduced from 1.2 V to 1.1 V for an improved power consumption.

The maximum number of banks per device increases from 16 to 32 distributed over 8 instead of 4 bank groups, the total capacity of a single device from 16 Gb to 64 Gb. In addition, up to 16 instead of 8 devices can now be stacked in a three-dimensional fashion (logical ranks)³. This enables stack capacities of up to 512 Gb (max. 16×32 Gb or 8×64 Gb because of limited address bits). One problem that always arises with higher device capacities is the increased refresh overhead, because each cell still has to be refreshed approximately every 64 ms. As a consequence, either the controller has to issue refresh commands more frequently or the individual refresh cycles take a longer time. Since all banks of a rank cannot be accessed during an all-bank refresh, it can lead to significant performance drops.

To overcome this problem, DDR5 introduces *same-bank refresh* (REFsb) and associated *same-bank precharge* (PREsb) commands as an alternative to *all-bank refresh* (REFab) and *all-bank precharge* (PREab) commands. When issuing them, only one bank in each bank group of the target rank is refreshed and inaccessible, while all other banks can still process incoming read and write requests. Most modern DRAM controllers use advanced re-ordering techniques for an improved performance so they can try to hide the same-bank refresh by sending requests to other banks in the meantime.

Finally, DDR5 devices implement an on-die error correction to improve the data integrity.

³The initial DDR4 standard did not specify any stacked devices. This feature was first introduced with an addendum in 2017.

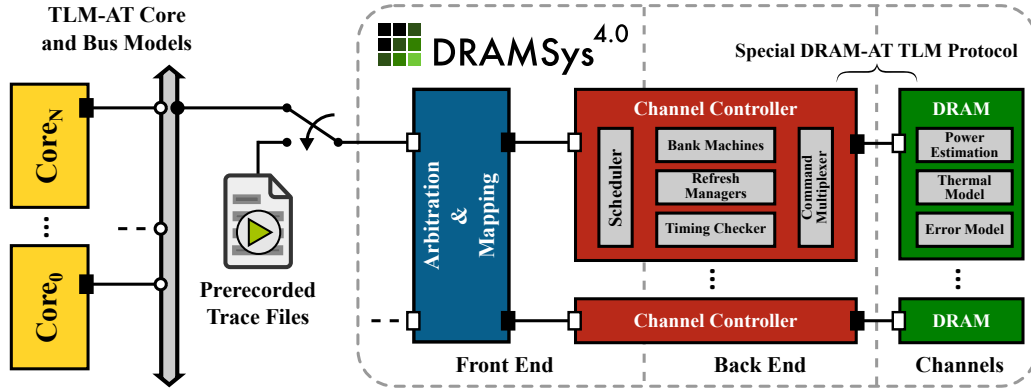


Figure 3 Architecture of DRAMSys

3 DRAMSys

The simulation of DRAMs on system level requires highly accurate models due to their complex timing and latency behavior. However, conventional cycle-accurate DRAM controller models often become the bottleneck for the overall simulation speed [5]. A promising alternative are DRAM simulation models based on TLM, which can be fast and accurate at the same time. In this paper we use the open-source simulator DRAMSys [6], which is, to the best of our knowledge, the fastest cycle-accurate open-source DRAM simulator and has a large range of functionalities. It features a very high simulation speed while maintaining full temporal accuracy. DRAMSys has a simulator architecture that enables a fast adaptation to new DRAM standards using a DSL. In the following we describe the DRAMSys simulation model, the internal architecture, the DSL-based code generation, and the integration of DDR5.

3.1 The DRAMSys Simulation Model

DRAMSys uses the concept of the SystemC/TLM2.0 IEEE 1666 Standard for a fast and fully cycle-accurate simulation. In accordance with the standard, all components are designed as *SystemC modules* and connected by *TLM sockets*. Each memory request is modelled with a special TLM transaction object, the so-called *generic payload*. It stores all relevant information of the request, e.g., the address, the transfer direction (read or write) or corresponding data, and is passed by reference between the modules. The simulator utilizes the *Approximately Timed (AT)* coding style, which defines a *non-blocking four-phase handshake protocol*. This protocol 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., PRE, ACT, RD/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 [7]. 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 events and the executed control flow overhead. Therefore, DRAMSys only simulates relevant state changes instead of individual signal changes in each clock cycle. 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 while still yielding fully cycle-accurate results [6].

3.2 Architecture

As shown in Figure 3, DRAMSys consists of three main components: a shared *arbitration & mapping unit* (short arbiter), a *channel controller* for each memory channel and a *DRAM device* for each memory channel. The arbiter cross-couples multiple initiators and DRAM channels and translates the address of each request into a corresponding channel, logical/physical rank, bank group, bank, row, and column. This translation is done on the basis of a predefined *address mapping*. Different address mappings can have a large impact on the overall system performance because of the timing dependencies between commands, which not only depend on the commands themselves, but also on the target location they are issued to (see Section 2.1). Thus, the address mapping should always be chosen with consideration of the application’s memory access pattern and the DRAM configuration.

Following the arbiter each memory channel has a channel controller. It is composed of five components that will be explained shortly. The *scheduler* enqueues incoming requests and reorders them on the basis of a specific *scheduling policy* to improve, e.g., bandwidth and latency by avoiding row misses. For each bank in the whole channel there is a separate *bank machine*. It keeps track of the current bank state and makes sure that commands are issued in a valid sequence, e.g., PRE, ACT, RD/WR for a row miss. The issuance of refresh commands to each DRAM rank at the right time is managed by *refresh managers*. To ensure that all timing dependencies between commands are satisfied and DRAMSys behaves compliant to the corresponding JEDEC standard, both bank machines and refresh managers ask the *timing checker* for the earliest point in time a command can be issued. The timing checker stores the relevant command history and all timing dependencies to calculate the right time. For each JEDEC standard there

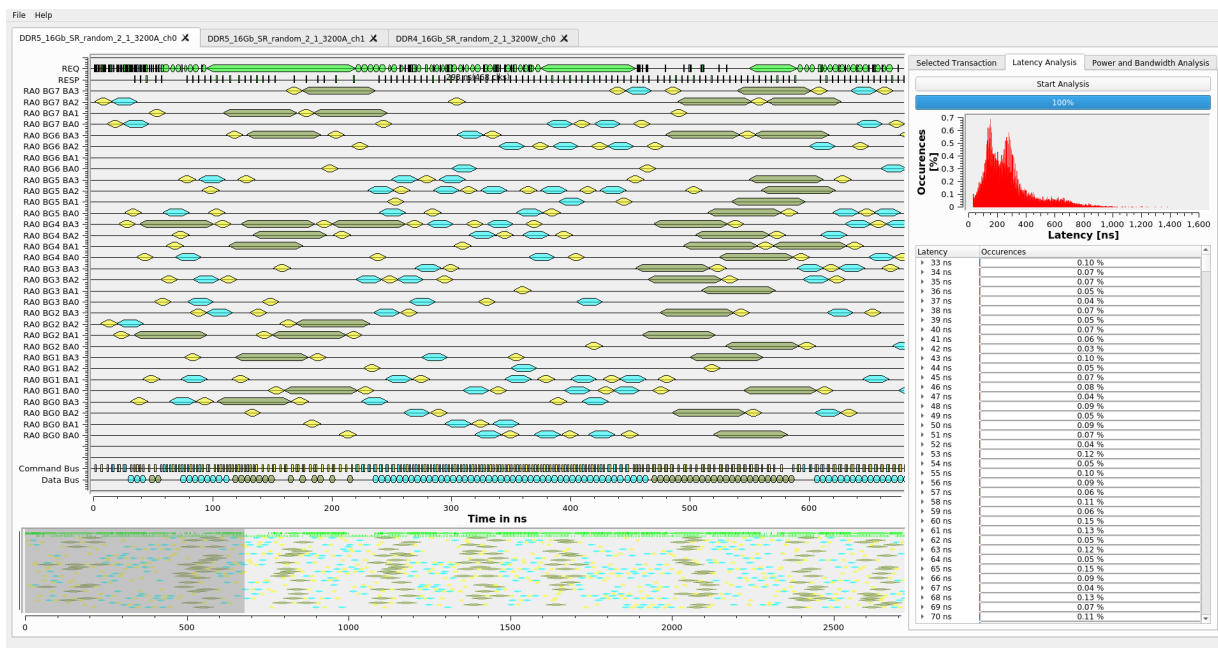


Figure 4 Graphical User Interface of the Trace Analyzer

is a separate timing checker that can be instantiated. Finally, each channel controller includes a *command multiplexer* to resolve conflicts between all bank machines and refresh managers, since only one request can be issued at a time due to the shared command/address bus.

As last component each channel controller is connected to a DRAM device. It manages the data storage and in addition can establish a connection to simulation tools for power estimation, thermal modeling or error modeling.

3.3 Trace Analyzer

To enable better analysis capabilities for the DRAM subsystem performance than the usual outputs to the console or a text file and to deeply investigate the channel controller’s scheduling decisions, DRAMSys provides the Trace Analyzer. During a simulation there is a functionality to record all TLM transactions of the channel controller in an SQLite trace database. Afterwards, this database can be evaluated graphically using the Trace Analyzer. The user interface is shown in Figure 4. It illustrates a time window of requests, DRAM commands and the utilization of all banks. This 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 very fast and the tool provides a user-friendly handling that offers a quick navigation through the whole trace database, sometimes with millions of requests and associated DRAM commands.

In addition, 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 number of accesses per activate (row hit rate).

3.4 DRAMml: A DSL for DRAMs

As stated in the introduction, an increasing number of different DRAM standards has been presented by JEDEC in recent years. Since each new standard comes with challenging 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 [1] we presented a comprehensive and formal DSL called DRAMml, which is based on a Petri Net [2] semantic. DRAMml can 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 can be generated automatically and correct by construction, as shown in Figure 5. One of the simulation models is also the channel controller’s standard-specific timing checker (see Section 3.2). In this way, the error-prone handwritten source code is replaced by source code generated from a high-level description.

3.5 Integration of DDR5

In order to expand DRAMSys by a DDR5 simulation model three steps had to be performed. All architectural parameters (e.g., number of channels/banks/bank groups, data bus width, burst length) as well as temporal parameters (frequency and timing values) of one specific DRAM subsystem configuration are defined in a JSON file and passed to the simulator as an argument. From the new DDR5 JEDEC standard JSON files for various speed grades and device sizes were put together first.

Second, a DRAMml description was created. Since DDR5 introduces same-bank refresh and specifies timing depen-

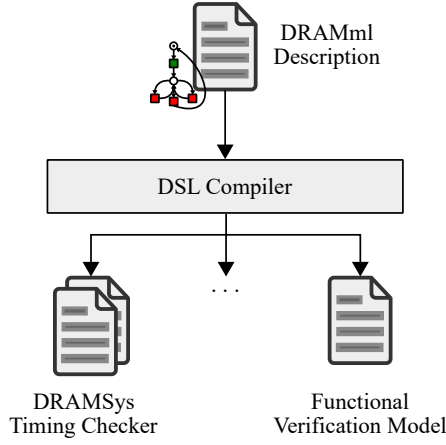


Figure 5 DRAMml Methodology

dependencies between commands to (1) the same bank, (2) different banks (same bank group), (3) different bank groups (same logical rank), (4) different logical ranks (same physical rank), (5) different physical ranks (same DIMM) and (6) different DIMMs⁴, the overall length of the description has increased by around 50 % compared to the predecessor DDR4. Nevertheless, it still fits on two pages and is very compact when considering the length of the actual JEDEC standard of almost 500 pages. In addition, the complexity has not increased. Because DRAMml fully relies on a Petri Net semantic, it is modular and could be adapted to all protocol changes of DDR5 without adding any language extensions. Listing 1 shows an excerpt of the initial DRAMml descriptions of DDR4 and DDR5 for the timing dependencies between two ACT commands. The increase in hierarchies (additional logical and physical ranks) could be directly applied to the code. From this description the new DDR5 timing checker source code for DRAMSys was generated automatically and correct by construction.

Listing 1 Timing Dependencies between ACT Commands

```

# DDR4:
ACT -<> ACT (tRC, tRRD_L, tRRD_S, 0);
# DDR5:
ACT -<> ACT (tRC, tRRD_L_slr, tRRD_S_slr,
            tRRD_dlr, 0, 0);
  
```

The third and final step was to derive a new same-bank refresh manager from the existing all-bank refresh manager. This was the only source code written by hand. Besides the change from PREab and REFab to PREsb and REFsb commands, the new component required some extra logic because REFsb commands have to be issued in a specific order to refresh all banks evenly. All in all, the whole integration process could be performed within two weeks. This time also included the study of the new JEDEC standard as well as testing and debugging. One major benefit of our DRAMml approach is that the major part of the source code can be generated correct by construction, so there are only very limited possibilities for introducing any errors, which keeps the debugging effort low.

⁴This increase results from the three-dimensional stacking of devices and the move of the power management integrated circuit (PMIC) from the main PCB onto the DIMM.

4 Experiments and Observations

In this section we analyze the bandwidth and latency behavior of different DDR5 device configurations with respect to two common application characteristics and compare it to DDR4. We will describe the experimental setup, the results and conclude with several key observations.

4.1 Experimental Setup

To evaluate the performance we use two characteristic benchmarks: (1) accesses with linearly-increasing addresses and (2) accesses with a random address distribution over the whole address space. These two benchmarks are the extreme borderline cases of all possible applications. Therefore, they are well suited to show the different limits, e.g., for server applications with many cores (random) and data-flow-oriented tasks (linear).

In both benchmarks the number of read requests is twice as high as the number of write requests. The total number of requests is set to 100,000 to omit short-time effects and to obtain results in a reasonable amount of time. For the linear benchmark a burst of 16 read requests is followed by a burst of 8 write requests, for the random benchmark read-write switches are also done randomly. The frequency in which requests are issued to the DRAM subsystem can be chosen arbitrarily and depends on the test case.

For DDR5 we use DIMMs with 8 Gb (1 Gb × 8) devices containing 16 banks and with 16 Gb (2 Gb × 8) devices containing 32 banks to find out the impact of the bank increase. Each channel is then composed of 4 devices and the whole DIMM of 8 devices. In addition to single-rank DIMMs (SR) we also use dual-rank DIMMs (DR) with 16 devices. Speed grades are varied from the slowest one (DDR5-3200) up to the fastest one (DDR5-6400) finally specified by JEDEC⁵. For comparison we use a DDR4 DIMM with 16 Gb (2 Gb × 8) devices containing 16 banks. The channel/DIMM is then composed of 8 devices. As for DDR5 we also investigate the behavior of a dual-rank DIMM with 16 devices. Speed grades are again varied from the slowest one (DDR4-1600) up to the fastest one specified in the standard (DDR4-3200). All-bank refresh is selected by default for both standards. In the case of DDR5 we always show the bandwidth sum of both channels, however, the number of data pins, which usually limits the amount of channels, is identical for DDR5 and DDR4.

All channel controllers implement separate queues for reads and writes with a queue depth of 32 each. Smaller and larger queue depths (16 and 64) were also investigated for completeness but only caused small shifts in all results, the general observations were the same. Thus, they are not presented in this work. As scheduling policy for incoming requests *First-Ready First Come First Served* (FR-FCFS) [8] is applied. The address mapping is chosen to yield the highest performance for the linear benchmark using techniques like channel and bank group interleaving. This means subsequent requests are mapped to different channels or bank groups and row or bank switching penal-

⁵Speed grades up to DDR5-8400 are planned but not yet finally specified.

ties are minimized. For the random benchmark the address mapping does not influence the performance at all because toggling rates for all address bits are uniformly distributed. Finally, for the linear benchmark a row is kept open after a read or write access because row hits are likely to happen (*open-page policy*), while for the random benchmark a row is automatically precharged after each read or write access because row hits practically never happen (*closed-page policy*).

4.2 Experimental Results

In the following we present the experimental results for different DDR5 device configurations, for the comparison of DDR5 and DDR4 and for the different DDR5 refresh modes. All results have been analyzed and verified with the Trace Analyzer.

4.2.1 Different DDR5 Configurations

First, we investigate the bandwidth of all DDR5 configurations both for the linear and the random benchmark. The requests are issued as fast as possible such that either the read or the write queue is always filled up completely. The corresponding results are presented in Figure 6a and 6b. What stands out most obviously is that for the linear benchmark all four configurations come much closer to the maximum bandwidth as for the random benchmark. This is because the address mapping is optimized for the linear case to reduce switching penalties. Only the refresh and read-write switches cannot be omitted and constantly decrease the bandwidth. Hence, also the bank increase from 16 (8 Gb device) to 32 (16 Gb device) and the move from single rank to dual rank yield rather small improvements (on average 2.7 % and 11.4 %), the 8 Gb dual rank configuration even outperforms the 16 Gb dual rank configuration due to shorter refresh cycles. For increasing data rates the slope slightly drops because refresh cycles take a fixed amount of time and thus more clock cycles. The waviness of some graphs is caused by timings that limit the maximum bandwidth and do not take a fixed amount of time, but a fixed amount of clock cycles. This behavior can also be observed in Figure 6c and 6d and in Figure 8.

However, the random benchmark results in Figure 6b look a lot different. While at low data rates all configurations are still in a close range, they largely drift apart when the data rates become higher. Then the gap between 16 and 32 banks and the move from single to dual rank is enormous (up to 41.8 % and up to 63.4 % bandwidth increase, on average 26.4 % and 42.4 %). When we consider that an 8 Gb dual-rank configuration also provides in total 32 banks per channel it clearly shows that for high data rates and random accesses DDR5 requires at least the parallelism of 32 banks to keep the bandwidth up. To find a simple explanation for this behavior we have to recall several things into our minds. For the chosen configuration random requests always translate into a full row cycle consisting of an ACT and a RDA/WRA. These row cycles require the fixed minimal amount of time t_{RC} (minimum time between ACT commands to the same bank, see Section 2.1). With increasing data rates the absolute time t_{RC} stays more or less

constant and therefore the corresponding number of clock cycles increases (around 74 for DDR5-3200 and 149 for DDR5-6400). But at the same time the number of clock cycles each request occupies the data bus stays constant (for DDR5 8 clock cycles at burst length 16). To successfully align this mismatch, hide the delays, and achieve a high data bus utilization and high sustainable bandwidth, a certain amount of bank parallelism is required to do row cycles in parallel on all banks. Simply put, with 149 clock cycles for one row cycle and 16 banks we can issue a new request every $\frac{149}{16} = 9.3$ clock cycles, but each request only takes 8 on the bus. This means even for an ideal command placement, a perfectly uniform distribution of requests over all banks and without refresh we will never reach the theoretical bandwidth limit. However, with 32 banks we can issue a new request every $\frac{149}{32} = 4.7$ clock cycles, which is fast enough even for the highest data rates of DDR5.

4.2.2 DDR4 versus DDR5

Next up, we compare the bandwidth of DDR5 to its predecessor DDR4 for single- and dual-rank configurations with 16 Gb devices (16 banks per rank for DDR4, 32 banks per rank for DDR5). The corresponding results are provided in Figure 6c and 6d. Each figure starts on the left side with DDR4-1600, the results for DDR5 start at a data rate of 3200 MT/s. DDR4 finishes at this data rate but is additionally extrapolated to DDR4-4400. This extrapolation is done because by this time there are lots of over-clocked DDR4 devices on the market with data rates up to 4400 MT/s, so in reality the data rate limits specified in the JEDEC standard are not valid any more. For the linear benchmark the bandwidths of DDR4 and DDR5 come very close to each other at data rates where they overlap (average increase of 7.5 % with SR and 8.2 % with DR from DDR4 to DDR5). The slightly better performance of DDR5 can be explained when we look on the number of read-write switches for each standard. While the total number of requests issued to each subsystem is identical, the DDR5 DIMM is composed of two channels, both of which receive only half of the requests. Considering that a read-write switch on each channel is always done after a fixed amount of requests (usually when the write buffer is full and the read buffer empty or the other way round), the number of switches on each of them is divided in half. However, the total runtime is not divided in half but almost identical for DDR4 and DDR5 because of the doubled burst length of DDR5. To sum it up, when choosing the memory for a platform where transfer rates lower than 4400 MT/s are targeted and the application's memory access scheme is rather linear, an upgrade from DDR4 to DDR5 is not worth the effort especially because prices for DDR5 at release are expected to be higher.

For the random benchmark the results in Figure 6d tell a completely different story. DDR4 with a single rank cannot keep up the bandwidth at all for increasing data rates, the dual-rank configuration performs better but also flattens for higher data rates. Their gap to DDR5 is large in both cases (approximately up to 130 % and 57 %, on average 107 % and 42 %). The problem is the same as for the 8 Gb DDR5 device mentioned previously. In the case of DDR4 a sin-

gle data transfer occupies the data bus only for 4 clock cycles and the device only has 16 banks. With two ranks the number of banks increases to 32, however the burst length is still too small. As a result, the data bus turns idle because row cycle delays cannot be hidden completely. With 32 or even 64 banks and a burst length of 16, both DDR5 configurations perform a lot better over the whole range. This shows that DDR5 is especially optimized for random workloads and is clearly ahead of DDR4 even at low data rates.

To compare both standards also from a different point of view, we fix the data rate to 3200 MT/s and vary the frequency requests are issued to the DRAM subsystem. Starting at 25 MHz and increasing in steps of 25 MHz we then plot the average response latency of all requests (average latency from entry of requests until data has been transferred over the bus) over the bandwidth. These results are presented in Figure 7. For the linear case (Figures 7a and 7b) DDR4 and DDR5 show a very similar behavior. In the beginning the bandwidth increases directly proportional to the frequency increase while the latency stays almost constant (around 40 ns). Each request is immediately serviced by the DRAM in this region and the latency is mainly composed of the read or write latency (latency from issuing a RD or WR command until the data transfer starts) and the data transfer over the bus. At higher input frequencies multiple requests are in the DRAM subsystem in parallel and interfere with each other. The controller starts reordering requests to keep the bandwidth up but to the disadvantage of the response latency, which increases exponentially. At an input frequency around 350 MHz the maximum sustainable bandwidth (~ 22 GB/s) is reached and both bandwidth and response latency remain unchanged. These values come close to the maximum theoretical bandwidth of 25.6 GB/s at an input frequency of 400 MHz. DDR5 has a worse average response latency at this point mainly due to the doubled burst length (~ 270 ns vs. 150 ns), however, real systems would never be operated at this point anyway. Advanced memory controllers rather take a slight loss in bandwidth with the advantage of a much lower average response latency, which is a lot more beneficial for the overall system performance. The memory controller configuration we used for our experiments is only optimized for bandwidth and therefore also allows these operating points.

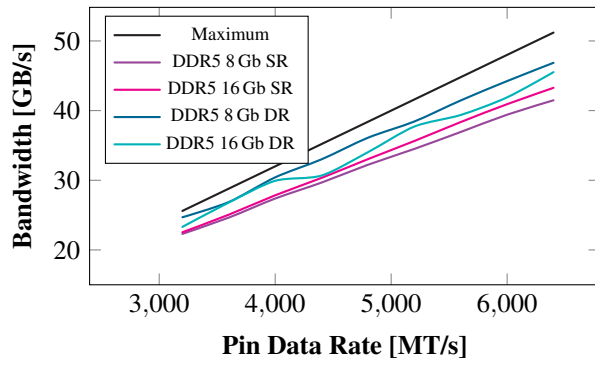
When both standards are compared for the random benchmark on the basis of these facts (Figures 7c and 7d), the benefits of DDR5 are large again. At the same speed grade DDR5 can handle random request streams with much higher frequencies and the average response latency rises much slower. For the single-rank configuration DDR4 exceeds an average response latency of 100 ns already at an input frequency of 125 MHz, DDR5 reaches this latency at input frequencies between 200 and 225 MHz. As a consequence, there is also a large gap between the maximum sustainable bandwidths of both standards (~ 11 GB/s for DDR4 vs. 21 GB/s for DDR5). In the case of two ranks the gap becomes smaller, but DDR5 still offers a big advantage over DDR4 both with respect to average response latency and bandwidth. These results confirm the optimization of DDR5 for random input patterns once more and show the

advantage of the architectural changes compared to DDR4 (higher bank count, higher burst length, two channels per DIMM).

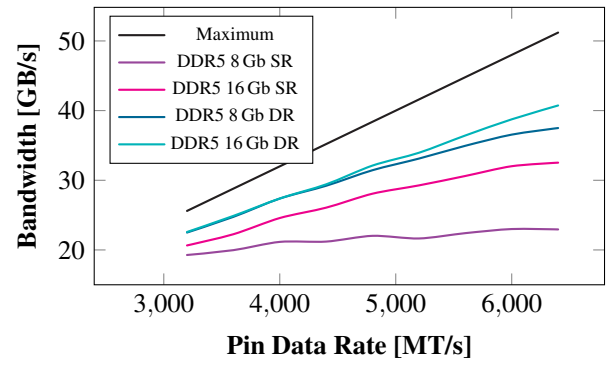
4.2.3 DDR5 Refresh Modes

Finally, we investigate the performance of the new same-bank refresh in comparison to all-bank refresh. Again, this is done with 16 Gb DDR5 devices for both benchmarks, for single- and dual-rank DIMMs and for all available DDR5 speed grades. The corresponding bandwidth results are shown in Figure 8. For reference, also the maximum theoretical bandwidth and the sustainable bandwidth without any refresh is shown. In the linear case (Figures 8a and 8b) same-bank refresh clearly leads to a higher bandwidth compared to all-bank refresh over the whole range of data rates and for both DIMM configurations (on average 7.9 % for single rank and 9.1 % for dual rank). Also, the gap to the bandwidth completely without refresh is very little (in some cases less than 1 %). In the case of a single rank, all-bank refresh causes a large drop in performance because the data bus turns idle during the whole refresh cycle. With two ranks, refresh commands are staggered, i.e., there is an offset between them and only one rank is refreshed at a time. The other rank can still process requests in the meanwhile, but only as long as requests for it are available in the queues. Since the selected address mapping makes sure that work is equally distributed over both ranks for an increased average performance, at some point the queues are completely filled with requests for the blocked rank and the data bus also turns idle. In addition, refresh commands have to be issued twice as often compared to a single rank. An increased queue depth could solve this issue, but at the same time would require more hardware resources and would make the scheduling more complex. In contrast, during a same-bank refresh cycle only every fourth (SR) or every eighth (DR) bank is inaccessible and each refresh cycle is also shorter. In that case the blocking on individual banks can easily be compensated by the remaining banks and the data bus does not turn idle.

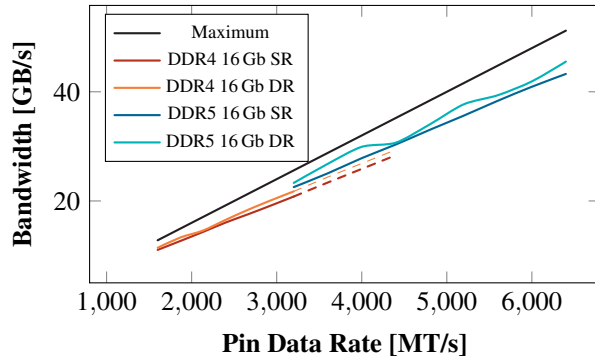
For the random benchmark the results with two ranks (Figure 8d) are more or less identical (on average 10.0 % bandwidth increase), but with only a single rank (Figure 8c) same-bank refresh does not bring an advantage (on average less than 1 % bandwidth increase). The reason for this behavior is already well-known and also occurred in the previous results: not enough bank parallelism. Although a single 16 Gb DDR5 device already offers 32 banks, the blocking times of refresh cycles still noticeably lower the performance. In this case the refresh mode makes no difference because the overall bank blocking times induced by both modes are nearly identical. With all-bank refresh there are less but longer idle phases on the data bus, with same-bank refresh more but shorter ones. Only with two ranks and 64 banks the bank parallelism is large enough to compensate the refresh penalty and the performance depends on the queue depth as for the linear case. As a conclusion, DDR5's new same-bank refresh strategy performs better or at least just as good as the all-bank refresh in all tested configurations and should therefore always be preferred.



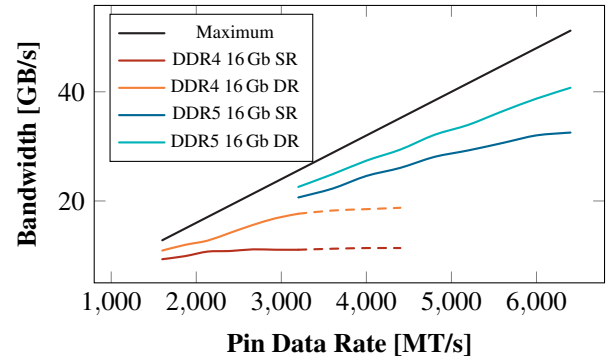
(a) DDR5, Linear



(b) DDR5, Random

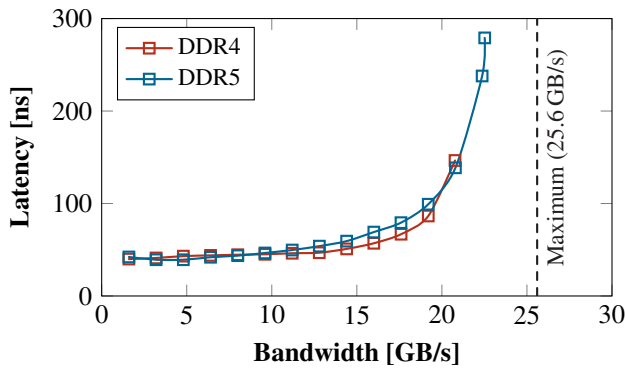


(c) DDR4 vs. DDR5, Linear

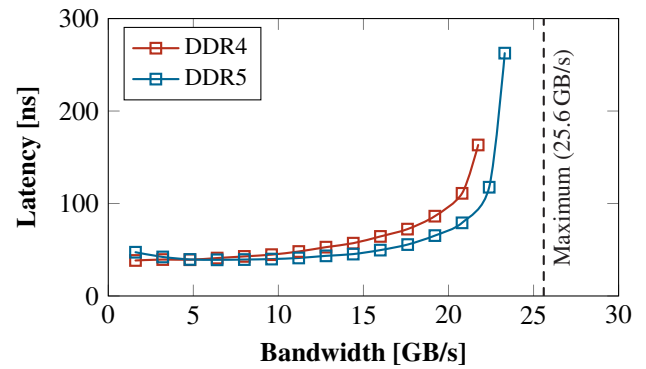


(d) DDR4 vs. DDR5, Random

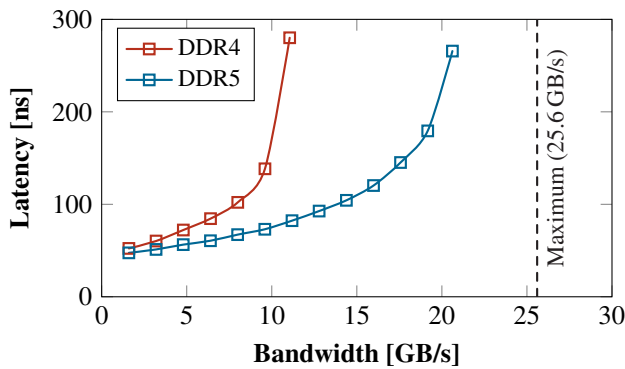
Figure 6 Sustainable Bandwidth for Different DDR5 Configurations and DDR4 vs. DDR5, SR: Single Rank, DR: Dual Rank



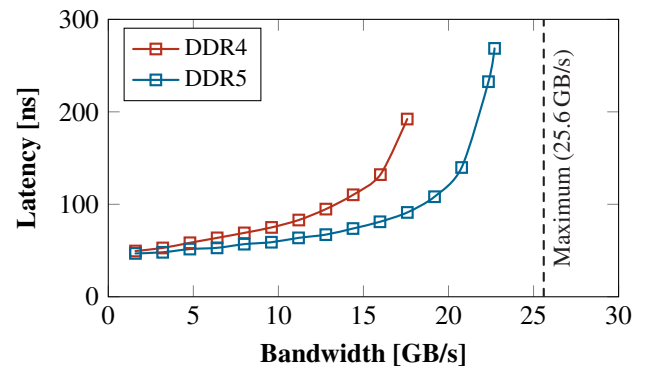
(a) 16 Gb Single Rank, Linear



(b) 16 Gb Dual Rank, Linear

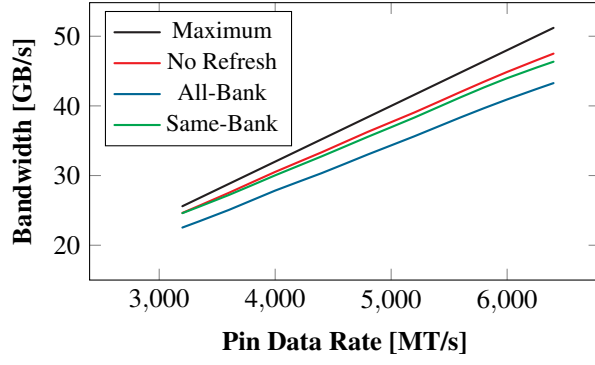


(c) 16 Gb Single Rank, Random

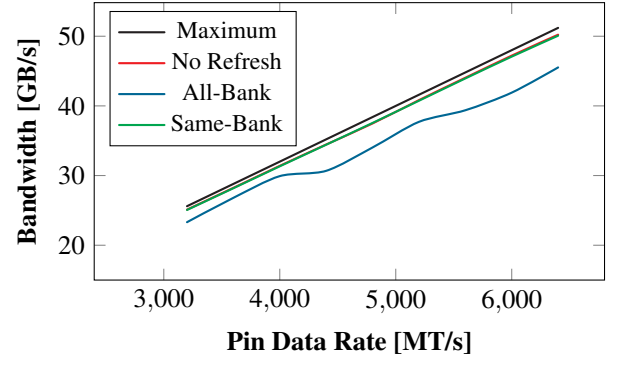


(d) 16 Gb Dual Rank, Random

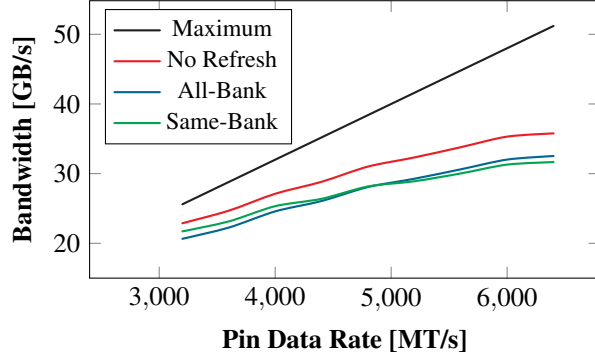
Figure 7 Average Response Latency over Bandwidth for DDR4-3200 vs. DDR5-3200



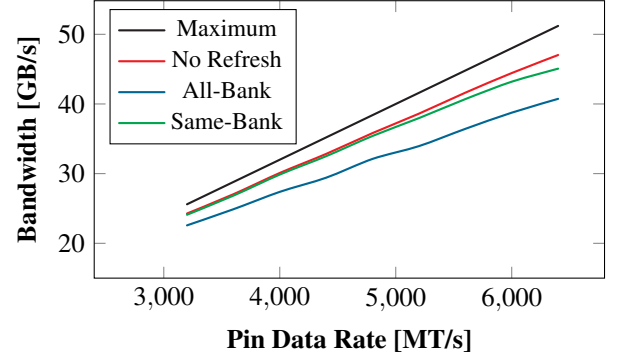
(a) 16 Gb Single Rank, Linear



(b) 16 Gb Dual Rank, Linear



(c) 16 Gb Single Rank, Random



(d) 16 Gb Dual Rank, Random

Figure 8 DDR5 All-Bank Refresh vs. Same-Bank Refresh

4.3 Key Observations

In this section we summarize the key observations for the performance of different DDR5 configurations and for the comparison to DDR4:

- *Applications with random memory access patterns:* Each DDR5 channel should be composed of at least 32 banks to achieve a high sustainable bandwidth.
- *Applications with linear memory access patterns:* DDR5 does not outperform DDR4 by much with respect to sustainable bandwidth and average response latency up to the highest DDR4 data rates.
- *Applications with random memory access patterns:* DDR5 should be preferred over DDR4 at all data rates with respect to sustainable bandwidth and average response latency.
- *Independent of the memory access patterns:* DDR5's same-bank refresh should always be preferred over all-bank refresh with respect to sustainable bandwidth.

5 Related Work

With respect to functional and cycle-accurate DRAM simulation there exist several open-source solutions. DrSim [9] was already developed several years ago but never updated over time, leading to an exclusive support of older JEDEC standards and making it unsuitable for most current system developments. DRAMSys [6],

DRAMsim3 [10], Ramulator [11] and the gem5 DRAM model [12] are all updated from time to time, however, only DRAMSys currently supports the DDR5 standard. Due to the Petri-Net-based source code generation [1] and the special TLM concept, DRAMSys can generate results a lot faster than the other cycle-accurate simulators. Furthermore, there are other approaches that only approximate the DRAM behavior. In [13] the authors propose an analytical DRAM performance model that uses traces to predict the efficiency of the DRAM subsystem. Todorov et al. [14] 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 [15] presents the modeling of DRAM behavior using decision trees. In [16] the authors present a performance-optimized DRAM model that is based on a neural network. However, to the best of our knowledge, there exists no analytical nor approximated DDR5 simulation model so far.

In terms of the simulation-based analysis and comparison of different DRAMs there is a comprehensive study by Ghose et al. [17]. It features a large variety of standards and workloads and also makes some key observations. However, it was already published in 2019 and thus does not include the new DDR5 standard.

6 Conclusion

With the release of DDR5 the total DRAM design space is greatly enlarged, making the selection of a suitable DRAM device even more complex. In this paper, we deeply investigated the performance of the new DDR5 standard for different configurations and compared it to its predecessor DDR4. In order to generate those results, we developed a DDR5 simulation model, which is based on the open-source DRAM simulator DRAMSys. We presented several insights and key observations for two characteristic benchmarks, which will guide system designers to make the right decisions. For the future we plan to run full-system simulations with our new DDR5 model to investigate the impact on the whole system performance more accurately.

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.

7 Bibliography

- [1] M. Jung, K. Kraft, T. Soliman, C. Sudarshan, C. Weis, and N. Wehn, “Fast Validation of DRAM Protocols with Timed Petri Nets,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: ACM, 2019, pp. 133–147. [Online]. Available: <http://doi.acm.org/10.1145/3357526.3357556>
- [2] C. A. Petri, “Kommunikation mit Automaten,” Ph.D. dissertation, Universität Hamburg, 1962.
- [3] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.
- [4] D. A. Patterson, “Latency Lags Bandwidth,” *Commun. ACM*, vol. 47, no. 10, pp. 71–75, Oct. 2004. [Online]. Available: <http://doi.acm.org/10.1145/1022594.1022596>
- [5] S. Li, R. S. Verdejo, P. Radojković, and B. Jacob, “Rethinking Cycle Accurate DRAM Simulation,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 184–191. [Online]. Available: <https://doi.org/10.1145/3357526.3357539>
- [6] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, “DRAMSys4.0: A Fast and Cycle-Accurate SystemC/TLM-Based DRAM Simulator,” in *International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS)*. Springer, July 2020.
- [7] M. Jung, C. Weis, N. Wehn, and K. Chandrasekar, “TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration,” in *Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:6. [Online]. Available: <http://doi.acm.org/10.1145/2432516.2432521>
- [8] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory Access Scheduling,” in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ser. ISCA '00. New York, NY, USA: ACM, 2000, pp. 128–138. [Online]. Available: <http://doi.acm.org/10.1145/339647.339668>
- [9] M. K. Jeong, D. H. Yoon, and M. Erez, “DrSim: A Platform for Flexible DRAM System Research,” <http://lph.ece.utexas.edu/public/DrSim>, (Last Access: 15.08.2019).
- [10] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, “DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator,” *IEEE Computer Architecture Letters*, pp. 1–1, 2020.
- [11] Y. Kim, W. Yang, and O. Mutlu, “Ramulator: A Fast and Extensible DRAM Simulator,” *IEEE Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2015.
- [12] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udiipi, “Simulating DRAM controllers for future system architecture exploration,” in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, March 2014, pp. 201–210.
- [13] G. L. Yuan and T. M. Aamodt, “A Hybrid Analytical DRAM Performance Model,” 2009.
- [14] V. Todorov, D. Mueller-Gritschneider, H. Reinig, and U. Schlichtmann, “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*, ser. DATE '12. San Jose, CA, USA: EDA Consortium, 2012, pp. 1066–1071. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2492708.2492972>
- [15] S. Li and B. Jacob, “Statistical DRAM Modeling,” in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 521–530. [Online]. Available: <https://doi.org/10.1145/3357526.3357576>
- [16] M. Jung, J. Feldmann, K. Kraft, L. Steiner, and N. Wehn, “Fast and Accurate DRAM Simulation: Can we Further Accelerate it?” in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 364–369.
- [17] S. Ghose, T. Li, N. Hajinazar, D. Senol Cali, and O. Mutlu, “Demystifying Complex Workload-DRAM Interactions: An Experimental Study,” 06 2019, pp. 93–93.