Fast and Accurate DRAM Simulation: Can we Further Accelerate it?

Johannes Feldmann, Kira Kraft, Lukas Steiner, Norbert Wehn
Microelectronic Systems Design Research Group
Technische Universität Kaiserslautern
Kaiserslautern, Germany
{feldmann,kraft,wehn}@eit.uni-kl.de

Matthias Jung
Embedded Systems Division
Fraunhofer IESE
Kaiserslautern, Germany
matthias.jung@iese.fraunhofer.de

***Abstract*—The simulation of Dynamic Random Access Memories (DRAMs) in a system context requires highly accurate models due to the complex timing and power behavior of DRAMs. However, cycle accurate DRAM models often become the bottleneck regarding the overall simulation time. Therefore, fast but accurate DRAM simulation models are mandatory. This paper proposes two new performance optimized DRAM models that further accelerate the simulation speed with only a negligible degradation in accuracy. The first model is an enhanced Transaction Level Model (TLM), which uses a look-up table to accelerate parts of the simulation that feature a high memory access density for online scenarios. The second model is a neural network based simulator for offline trace analysis. We show a mathematical methodology to generate the inputs for the Look-Up Table (LUT) and an optimized artificial training set for the neural network. The enhanced TLM model is up to 5 times faster compared to a state-of-the-art TLM DRAM simulator. The neural network is able to speed up the simulation up to a factor of $10\times$, while inferring on a GPU. Both solutions provide only a slight decrease in accuracy of approximately 5%.**

***Index Terms*—DRAM, Simulation Acceleration, Neural Networks**

I. Introduction

Today's applications are more and more data driven, which puts the memories more into the focus. Therefore, *Dynamic Random Access Memories* (DRAMs) are key components, which have a large impact on the whole system's timing and power behavior. Thus, both fast and accurate DRAM simulation models are needed to perform exhaustive and truthful investigations. It is well known that there exists a challenging trade-off between the simulation speed and accuracy. On the one hand, there are cycle- and pin-accurate RTL models based on a *Discrete Event Simulation* (DES). They provide the highest temporal accuracy, but are inflexible with respect to modifications and require very long simulation times since all signals, processes and events have to be simulated.

On the other hand, many functional loosely-timed simulators that are utilized for pre-silicon software development use a simplistic DRAM model called *Fixed Latency Model* [1]. This model is the fastest realizable DRAM model where all memory requests experience the same constant amount of latency. However, the properties of an access, like the addressed bank and row, are not considered[1]. For design-space explorations or performance estimations, this approach is unrealistic since the latency of a DRAM access varies between a dozen and several hundred cycles [2].

[1]The fixed latency model of Figure 1 at least considers the additional latency caused by refreshes

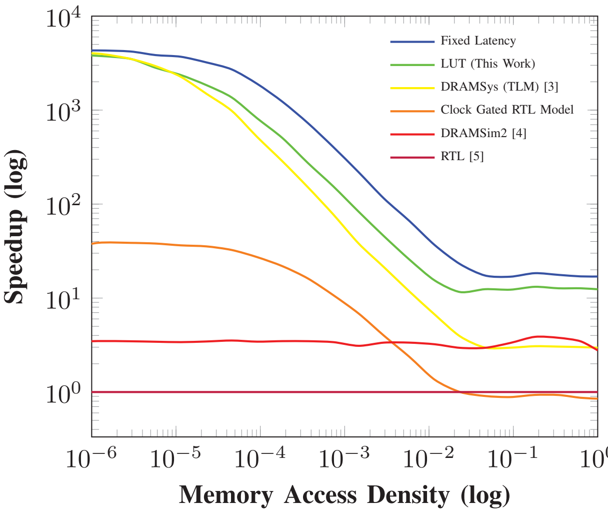


Fig. 1. Speedups Normalized to RTL Model

In the literature, there are different ways to speed up the DRAM simulations without losing accuracy. The standalone trace-based, cycle- but not pin-accurate simulator DRAMSim2 [4] is slightly faster than RTL models (c.f. Figure 1) because it does not rely on a DES kernel and therefore avoids the overhead of event sorting, delta cycles and signals. For DES-based simulation models, the key for the transition from slow RTL to fast system-level models is the reduction of *unnecessary simulation events*. For example, an RTL simulation can be accelerated by suppressing unnecessary events [6]. Similar to the idea of clock-gating in real circuits for power saving, turning off the clock signal during an RTL simulation saves a lot of simulation events, since clock signals have high event generation rates. Thus, clock-gating can tremendously speed up the simulation without decreasing the accuracy.

Another way to reduce the number of unnecessary simulation events is to move up to a higher abstraction level by using the idea of *Transaction Level Modeling* (TLM) [7]. The authors of [8] present the SystemC/TLM2.0 compliant simulator DRAMSys, which features a customized TLM protocol for modeling DRAM and its controller. This protocol only consists of the minimal number of events that are required for full accuracy. In their paper they made an important observation: The speedup of the TLM model depends on the density of the memory accesses

$$d = \frac{a}{t}, \tag{1}$$

where a is the number of memory transactions and t the length of the simulation.

Figure 1 shows results of several simulations of random traces with different densities performed on an Intel Core i9 with 5GHz. This graph compares the previously discussed DRAM modeling techniques and one of the two models presented later depending on the memory access density d. The RTL model is the slowest model and is used as a baseline, whereas the fixed latency model defines an upper bound for the maximum achievable speedup. It can be observed that the speedup of the fixed latency model decreases with increasing memory access density, i.e., with an increasing number of simulation events.

As depicted in the figure, if the trace density is low, the TLM model achieves a high speedup compared to the RTL model, since it can fast-forward during idle periods and thus has to model much less events. For the same reason, the performance speedup of the clock-gated RTL model can be observed in the low-density region. On the other hand, if the trace density is high, the TLM model's performance converges to the RTL implementation because only a small number of unnecessary events can be skipped. Since clock-gating introduces an additional overhead, the performance of the clock-gated RTL model even becomes worse than the non-clock-gated RTL model for high densities. In total, the TLM model is always the superior solution with respect to both speedup and accuracy.

As shown in the figure, the highest need for an acceleration of DRAM models exists for memory traces that exhibit a high density ($\gtrsim 10^{-2}$). Therefore, the focus of this paper is to further increase the speedup of the memory models especially for this scenario. However, to the best of our knowledge, further speedups without sacrificing accuracy are not possible. This paper proposes two new performance optimized DRAM models that further accelerate the simulation speed with only a slight degradation in accuracy. The first model is an enhanced TLM model which uses a *Look-Up Table* (LUT) to accelerate simulation regions with high bandwidth usage for online scenarios. The second is a neural network based simulator for offline trace analysis. Since DES are dominated by control flow, they cannot profit from to heterogeneous hardware architectures and accelerators like GPUs or TPUs [9], as they are especially designed for the fast execution of data flow applications. Neural networks are highly data flow driven. Therefore, we will use a neural network that is trained on the timing behavior of DRAM to convert the control flow of the DES into data flow and thus to be able to use GPUs and TPUs for a fast and highly parallel execution.

In summary, this paper makes the following new contributions:

- We presented a detailed analysis of different DRAM simulation approaches as a function of the memory access density. Based on this we identified the potential of further accelerations.
- We present a hybrid TLM-based simulator, shown in Figure 3, which switches from the classical TLM implementation to a LUT, if the density during the simulation increases, in order to speed up the simulation. If the density decreases it switches back to the standard TLM model.
- For the first time, we show how a trace based DRAM simulation can be accelerated by a neural network, which is inferred on a GPU, as shown in Figure 6.
- We present a mathematical methodology to efficiently create the keys and values for a minimized LUT and an optimized artificial training set for the neural network.

The paper is structured as follows: Section II discusses related work, whereas Section III introduces the DRAM background. In Section IV, two new accelerations of DRAM simulations are presented. Section V shows the experimental results, and Section VI concludes the paper.

II. Related Work

In literature, several DRAM simulation models exist, among them *DRAMSim2* [4], *Ramulator* [10], *DrSim* [11] and *USIMM* [12]. Each of them is focusing on different aspects like, e.g., scheduling, subsystem architecture, etc., and each has individual advantages and drawbacks. All of them are cycle-accurate simulators that slow down event-driven full-system simulations because every cycle has to be simulated.

There exist TLM-based DRAM simulators like *DRAMSys* [3] that show an highly increased simulation speed while providing the same accuracy of a cycle-accurate RTL model by using a custom TLM protocol, called DRAM-AT. *gem5* [13], a full-system simulator, has integrated a realistic DRAM controller model [14] which is very similar to the one implemented in DRAMSys as it uses events similar to DRAM-AT to trigger the simulation submodules and to execute the active tasks. In contrast to the previous work, our approach can achieve a further speedup of $5\times$ to $10\times$ (cf. Figures 1 and 10).

Further approaches for simulation and modeling of DRAM devices exist. In [15] the authors propose an analytical DRAM performance model that uses traces to predict the efficiency of the DRAM sub-system. Todorov et al. [16] 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.

III. DRAM Background

In this paper, we focus on the estimation of the DRAM latency, since many other features like bandwidth and power consumption can be estimated from it. The latency of a DRAM access is defined as the time difference between the transaction

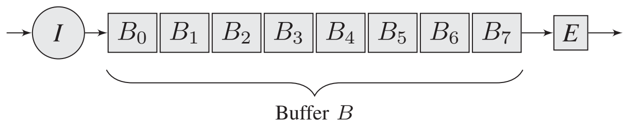


Fig. 2. Memory Controller's Input Buffer with a Size of 8

to the memory controller and its corresponding response to the initiator. The DRAM latency depends on the memory controller, the DRAM protocol and the previously issued DRAM commands. As can be seen in Figure 3, a DRAM access needs to pass several functional units in the memory controller, which is composed of a frontend and a backend. The frontend performs arbitration and scheduling of incoming read and write requests, whereas the task of the backend is to translate these incoming requests to a sequence of DRAM commands which have to be orchestrated with respect to the current state of the device and given timing dependencies. To access data in a row of a DRAM bank, an activate command (ACT) must be issued by the controller before any column access, i.e., read (RD) or write commands (WR), can be executed. The ACT command opens an entire row of a bank, which is transferred into the banks row buffer. 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 *hit*), it results in a low latency and low energy memory access. Whereas, if a memory access targets a different row as the current row in the buffer (called *miss*), it results in higher latency and energy consumption. If a certain row in a bank is active it must be precharged (PRE) before another row can be activated. Furthermore, the switching between reads and writes requires a turnaround time, which also increases the latency of the memory access and following accesses. Additionally, there exist a lot more timing dependencies between the DRAM commands that also influence the latency.

IV. Acceleration of DRAM Simulations

In this section we will present two new accelerations of DRAM simulation that go beyond the state of the art (cf. Figure 1). Figure 2 shows the DRAM controller's input buffer with a size of 8 elements (as the RTL model [5]), on which the scheduling is performed. An incoming access I is placed into this buffer, and the scheduler decides which access from $B_0 - B_7$ is executed next (E).

As shown in Figure 1, the speed of the TLM model is decreasing and converges towards the cycle accurate implementations when the memory access density is increasing. For densities $\gtrsim 10^{-2}$ this input buffer is always full and incoming requests are refused by a back-pressure mechanism until the currently executed DRAM access is completed. Therefore, in this scenario, there are always eight accesses buffered and one currently executed. Since the behavior of the memory controller is deterministic, the latency time of a new access I only depends on the 9 previous accesses and itself.

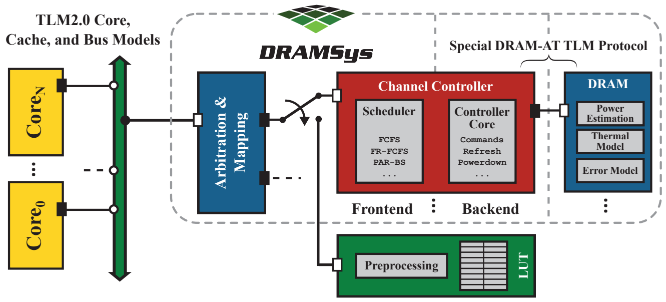


Fig. 3. Hybrid TLM Simulator with LUT

In the following we make two assumptions: (1) for the usage of the LUT (cf Section IV-A) and the usage of the neural network (cf. Section IV-B) the input buffer is always full since there exists the highest potential for acceleration, (2) for the sake of reducing complexity, we focus on the modeling of a single DRAM bank. The modeling of multiple banks is considered as future work.

A. Acceleration with a Look-Up Table (LUT)

The purpose of the LUT is to further remove events and calculations from the simulation by using previously computed latency values. As mentioned before, if the input buffer is full, the newest DRAM request's latency depends only on itself and the previous requests in that buffer. Thus, the latency scenario can be described by $A = I + B + E = 10$ consecutive DRAM accesses. The LUT stores all possible scenarios of memory requests in the input buffer as its keys and the resulting latencies as values. In Figure 3, we show a hybrid simulation model, which consists of the normal DRAMSys and the LUT. When the buffer of the memory controller is full, the TLM model can switch from the complex latency calculation, which is performed in the channel controller, to this look-up table in order to speed up the simulation. If the density of memory accesses decreases, it switches back to the standard TLM model.

Although a LUT is a straightforward approach, there are two challenges: We first need to find suitable keys to reflect the last A DRAM accesses, and second an efficient method to create the corresponding latency values. The naïve way to generate a key is to take the DRAM address and the corresponding command (RD/WR) of each access which results for a 32 bit system in 33 bits per access. Since ten accesses determine the latency, a key size of 330 bits is needed which would result in a LUT with $2^{330} \approx 2.187 \cdot 10^{99}$ entries. It is impossible to store those values in a memory.

An analysis of the memory accesses showed that their resulting access latency is determined by two main features, namely whether the command is a read (R) or write (W) command, and whether it results in a row hit (H) or row miss (M). Thus, a single access can be described by one of the following symbols RH, RM, WH, WM. A simple preprocessing, as shown in Figure 3, is needed in order to extract those features from incoming requests.

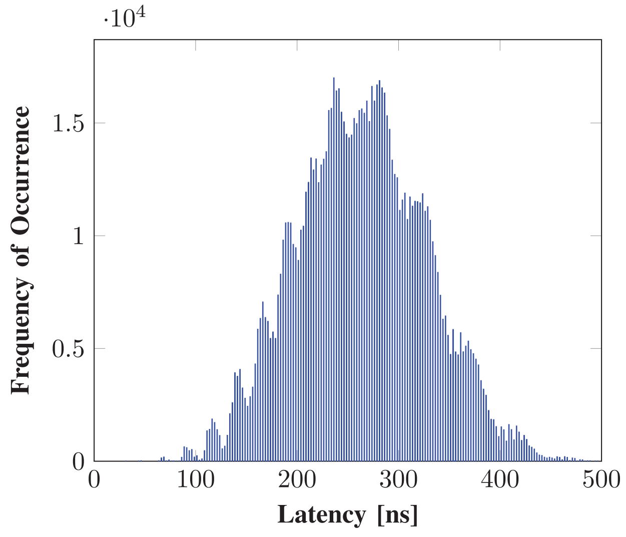


Fig. 4. DRAM Latency Histogram of De Bruijn Sequence

Each symbol can be represented as a two bit value which decreases the number of bits for each access from 33 to 2. Therefore, only 20 bits are required as key size, given that $A = 10$. The number of all possible combinations of the input buffer C can be calculated using Equation 2, where P is the number of features.

$$C = \left(2^P\right)^A \tag{2}$$

For $P = 2$ and $A = 10$ this leads to $\left(2^2\right)^{10} = 4^{10} = 1{,}048{,}576$ possible combinations. The latency values are stored in a 32 bit unsigned integer array as picoseconds, since all values are ranging from 0 ps to 500,000 ps. Therefore, the memory size of the LUT is 4 MiB which is in the range of the cache size in state-of-the-art desktop CPUs. Thus, the CPU will store frequently used parts of the LUT inside its cache which enables faster simulation speed.

In order to obtain the latency values for the LUT, we have performed simulations using DRAMSys for all possible combinations. Creating a simulation trace by concatenating all combinations would result in a trace containing $10{,}485{,}760$ accesses. Alternatively, $1{,}048{,}576$ simulations of 10 accesses are needed. Both ways create a significant amount of simulation execution time.

To reduce the time needed to simulate all combinations, a series of memory accesses is needed which has the minimum length but still contains all combinations. The following definition formalizes this problem [17], [18].

Definition 1 (De Bruijn Sequence). *Let Σ be any alphabet of size $|\Sigma| = \sigma$ and $0 \neq n \in \mathbb{N}$ a natural number. A de Bruijn Sequence, also known as $P_n(\sigma)$-cycle, is a cyclic sequence of length σ^n, that contains every possible length-n string of elements of Σ exactly once.*

Note that by requiring that every possible length-n string is contained *exactly once*, a de Bruijn sequence is the minimum-length sequence with the requested properties.

As an example, let $\Sigma = \{0, 1\}$ be the binary alphabet with $\sigma = |\Sigma| = 2$ and $n = 3$. A possible de Bruijn sequence is given by $P_3(2) = 00010111$. It can be verified that all possible length-3 strings with elements from Σ, that is, every element from $\{000, 001, 010, 011, 100, 101, 110, 111\}$, are contained in the cyclic sequence. Figure 5 shows the corresponding *de*

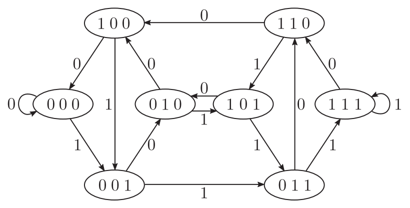


Fig. 5. De Bruijn Graph

Bruijn graph. Every de Bruijn sequence corresponds to a Hamiltonian cycle (i.e., a cycle that visits every node exactly once) inside this graph. In [17], the author proved that $P_n(\sigma)$-cycles exist for all possible values of n and σ and gave a construction algorithm for them. The problem of finding a minimum-length simulation trace containing all possible combinations of the required features can now be solved using these de Bruijn sequences. To obtain all combinations of the two required features (R/W and H/M) in memory accesses of length 10, a $P_{10}(4)$-cycle is constructed. This cyclic sequence has to be 'flattened' (i.e., by adding the first 9 digits to the end of the sequence), so the final trace length is $4^{10} + 9 = 1{,}048{,}585$. By construction, this is the minimum-length simulation trace with the required features. In comparison to the naïve approach of concatenating all 4^{10} possible combinations of length-10 access traces, the final trace length is reduced by a factor of $10\times$ using the de Bruijn sequences. After simulating the generated trace containing all combinations using DRAMSys, 180 different latency values appeared resulting in a distribution as it can be seen in Figure 4. These latency values are stored in the LUT with their corresponding keys.

B. Acceleration with a Neural Network

Since the LUT in Section IV-A maps $1{,}048{,}576$ keys to only 180 latency values, it is feasible to achieve a compressed approximation using a neural network. Since neural networks are data flow driven, they enable a fast execution on GPUs. Our neural network based approach, called DRAMnn, therefore focuses on the acceleration of offline trace analysis on GPUs.

As shown in Figure 6 the neural network is trained offline with the data generated by the de Bruijn sequence. However, the latency distribution of the de Bruijn sequence, as shown in Figure 4, does not reflect the distribution of realistic traces. If a neural network is trained using this trace, it focuses on latencies with high occurrence. Common cases like long sequences of read or write hit accesses are located in the tails of the distribution and are therefore not well represented. This leads to a high deviation in these cases resulting in an overall bad approximation. Thus, an equalization of all latency

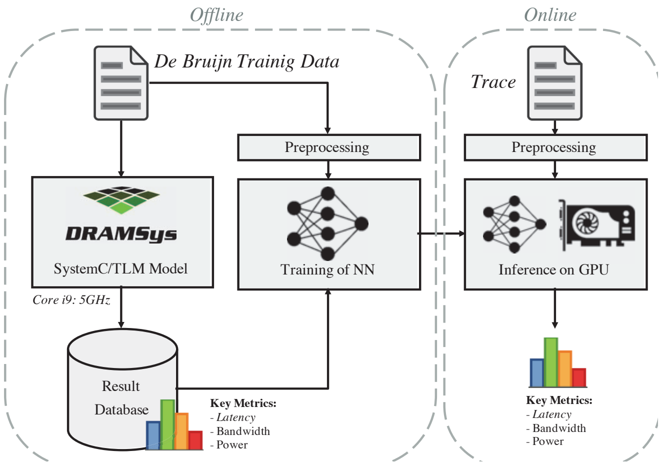


Fig. 6. Setup of NN Based Simulator

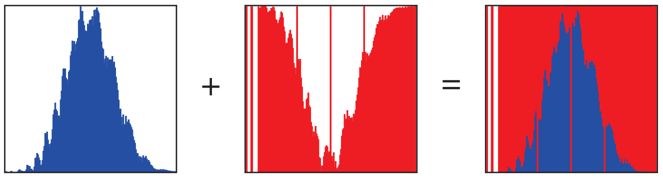


Fig. 7. Balancing of Training Data

occurrences needs to be performed as shown in Figure 7. This is done by duplicating access patterns of latencies such that an inverse distribution is formed. This inverse distribution is added to the original de Bruijn distribution in order to receive a training set which is uniformly distributed.

The inference of the neural network consists of two stages. The first stage, executed on CPU, reads the trace from a file and applies a preprocessing, similar to the LUT implementation, to extract the R/W feature and the H/M feature. Since the computational effort for getting both features is low, the computation time is neglectable compared to the time needed for reading the file. In the second stage the preprocessed data is transferred to the working memory of the GPU and inference is started. The used neural network topology is shown in Figure 8. It is a deep neural network (DNN) with four hidden layers and two sigmoid (σ) activation functions forming a funnel which ends in a regression layer.

In the next section, we will show the behavior of the two presented approaches with respect to accuracy and speedup.

V. Experimental Results

In Figure 1, the speedup of the hybrid DRAMSys with LUT is shown depending on the access density of the input trace. The LUT is able to accelerate DRAMSys by a factor of $5\times$ for traces with high density and therefore the speedup converges towards the fixed latency model closing the big performance gap for dense traces.

Even if the traces have a low density in average, there often exist local areas of high density, where the LUT can

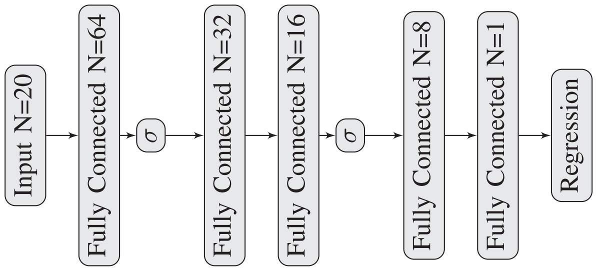


Fig. 8. Topology of the Neural Network for Latency Prediction.

achieve a speedup. However, this hybrid DRAMSys is not 100% accurate. It induces errors by switching from the LUT back to event-based simulation and vice versa. The simulator state is not updated during the usage of the LUT and therefore outdated.

For the quantification of the errors, the *Mean Absolute Percentage Error* (MAPE) is used. It is a measure of accuracy which expresses the deviation of a sequence A from a sequence B as a percentage, where A is the original sequence and B is the predicted sequence. It is defined by the following formula:

$$MAPE = \frac{100\%}{n} \cdot \sum_{t=1}^{n} \left| \frac{A_t - B_t}{A_t} \right| \tag{3}$$

Three benchmarks (JPEG Decoding, Linear Filter, FFT + 3D Rotation) are used to show the practical applicability of both acceleration approaches. As shown in Table I, the hybrid DRAMSys is able to accelerate the simulation by a factor of $4.2\times$ to $5.5\times$ exhibiting an MAPE error smaller than 1%.

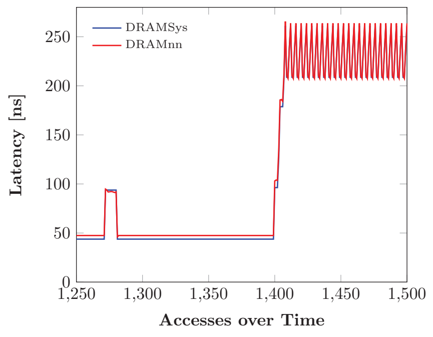
The speedup of DRAMnn compared to DRAMSys is shown in Figure 10. For randomly generated dense traces with over two million accesses, DRAMnn is able to speed up the simulation by factors of $10\times$ using a GPU and $4\times$ on a CPU. For the benchmarks in Table I, speedups from $5\times$ to $9\times$ can be achieved with GPU acceleration. In Figure 9, the latencies of DRAMnn are compared with DRAMSys. The induced error of the neural network inference is well distributed over all latency values. It shows that the network is able to approximate a DRAM in any way without producing large latency gaps. The error induced by the neural network is below 6.5%.

TABLE I
Speedup and Errors Compared to TLM

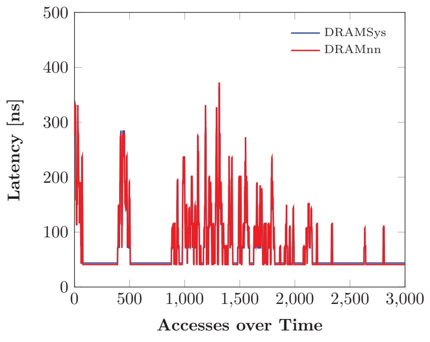
		DRAMnn (GPU)		**LUT**	
Benchmark	#Accesses	MAPE	Speedup	MAPE	Speedup
JPEG	$3 \cdot 10^3$	5.11%	$5\times$	0.21%	$4.2\times$
FFT + 3D Rot	$1.5 \cdot 10^6$	6.49%	$7\times$	0.0%	$4.6\times$
Filter	$2.4 \cdot 10^6$	1.02%	$9\times$	0.91%	$5.5\times$

VI. Conclusion

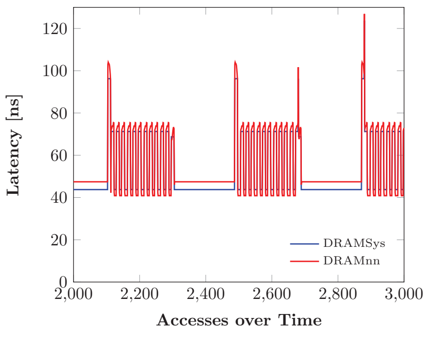
DRAM becomes more and more important in various fields as today's applications have an increased demand of memory. This fact makes fast and accurate DRAM simulation mandatory for truthful system investigations. In this paper we



(a) Filter



(b) JPEG



(c) FFT + 3D Rotation

Fig. 9. Accuracy of DRAMnn

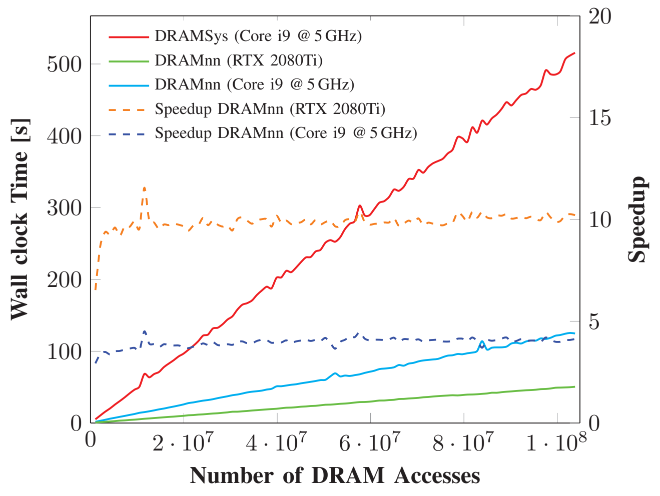


Fig. 10. Speedups for Random Traces with Different Length

showed that there exists a performance gap between fast and accurate DRAM simulations. To tackle this gap, we presented two new performance optimized DRAM models. First, we extended the state-of-the-art DRAM simulator DRAMSys by a LUT increasing the performance for dense traces with only a slight impact on the quality. Second, we generated a data flow model for offline DRAM latency estimation on GPUs based on a neural network. Furthermore, we showed a mathematical method for the generation of an artificial trace which includes all access patterns required for both the LUT and training of the neural network. In the future, we will extend our work to also support sparse traces and we will enable the support of multiple banks as well as power and bandwidth predictions.

Acknowledgement

We thank Marc Reichenbach and Sebastian Rachuj for the fruitful discussions. This work was supported within the Fraunhofer and DFG cooperation program (Grant no. WE2442/14-1), *Fraunhofer High Performance Center for Simulation- and Software-based Innovation*, as well as the *Virtual Engineering of Smart Embedded Systems* (ViSE) project.

References

[1] Bruce Jacob. *The Memory System: You Can'T Avoid It, You Can'T Ignore It, You Can'T Fake It*. Morgan and Claypool Publishers, 2009.

[2] Matthias Jung, et al. *Driving into the Memory Wall: The Role of Memory for Advanced Driver Assistance Systems and Autonomous Driving*. In Proceedings of the International Symposium on Memory Systems, MEMSYS '18, pages 377–386, New York, NY, USA, 2018. ACM.

[3] Matthias Jung, et al. *DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework*. IPSJ Transactions on System LSI Design Methodology (T-SLDM), August 2015.

[4] Paul Rosenfeld, et al. *DRAMSim2: A Cycle Accurate Memory System Simulator*. Computer Architecture Letters, 10(1):16–19, Jan 2011.

[5] Chirag Sudarshan, et al. *A Lean, Low Power, Low Latency DRAM Memory Controller for Transprecision Computing*. In Dionisios N. Pnevmatikatos, et al., editors, Embedded Computer Systems: Architectures, Modeling, and Simulation, pages 429–441, Cham, 2019. Springer International Publishing.

[6] H. Muhr et al. *Accelerating RTL Simulation by Several Orders of Magnitude Using Clock Suppression*. In 2006 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pages 123–128, July 2006.

[7] IEEE Computer Society. *IEEE Standard for Standard SystemC Language Reference Manual*. (IEEE Std 1666-2011), 2012.

[8] Matthias Jung, et al. *TLM modelling of 3D stacked wide I/O DRAM subsystems: a virtual platform for memory controller design space exploration*. In Proceedings of the 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[9] Norman P. Jouppi, et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit*. In Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[10] Yoongu Kim, et al. *Ramulator: A Fast and Extensible DRAM Simulator*. IEEE Computer Architecture Letters, PP(99):1–1, 2015.

[11] Min Kyu Jeong, et al. *DrSim: A Platform for Flexible DRAM System Research*. http://lph.ece.utexas.edu/public/DrSim, (Last Access: 15.08.2019).

[12] Niladrish Chatterjee, et al. *USIMM: the Utah SImulated Memory Module, A Simulation Infrastructure for the JWAC Memory Scheduling Championship*. Utah and Intel Corp., February 2012.

[13] Nathan Binkert, et al. *The gem5 simulator*. SIGARCH Comput. Archit. News, 39(2):1–7, August 2011.

[14] A. Hansson, et al. *Simulating DRAM controllers for future system architecture exploration*. In Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on, pages 201–210, March 2014.

[15] George L. Yuan et al. *A Hybrid Analytical DRAM Performance Model*, 2009.

[16] Vladimir Todorov, et al. *Automated Construction of a Cycle-approximate Transaction Level Model of a Memory Controller*. In Proceedings of the Conference on Design, Automation and Test in Europe, DATE '12, pages 1066–1071, San Jose, CA, USA, 2012. EDA Consortium.

[17] Monroe H Martin. *A problem in arrangements*. Bulletin of the American Mathematical Society, 40(12):859–864, 1934.

[18] T Aardenne-Ehrenfest et al. *Circuits and Trees in Oriented Linear Graphs*. Classic Papers in Combinatorics, pages 149–163, 1987.