DRAMsim: A Memory System Simulator

David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob

Dept. of Electrical & Computer Engineering
University of Maryland, College Park
{davewang,brinda,ohm,ktbaynes,ajaleel,blj}@eng.umd.edu
http://www.ece.umd.edu/dramsim/

As memory accesses become slower with respect to the processor and consume more power with increasing memory size, the focus of memory performance and power consumption has become increasingly important. With the trend to develop multi-threaded, multi-core processors, the demands on the memory system will continue to scale. However, determining the optimal memory system configuration is non-trivial. The memory system performance is sensitive to a large number of parameters. Each of these parameters take on a number of values and interact in fashions that make overall trends difficult to discern. A comparison of the memory system architectures becomes even harder when we add the dimensions of power consumption and manufacturing cost. Unfortunately, there is a lack of tools in the public-domain that support such studies. Therefore, we introduce DRAMsim, a detailed and highly-configurable C-based memory system simulator to fill this gap. DRAMsim implements detailed timing models for a variety of existing memories, including SDRAM, DDR, DDR2, DRDRAM and FB-DIMM,with the capability to easily vary their parameters. It also models the power consumption of SDRAM and its derivatives. It can be used as a stand-alone simulator or as part of a more comprehensive system-level model. We have successfully integrated DRAMsim into a variety of simulators including MASE[15], Sim-alpha[14], BOCHS[2] and GEMS[13].The simulator can be downloaded from www.ece.umd.edu/dramsim.

1 Introduction

CPU speed has doubled roughly every 18 months, while DRAM core speed has increased at a more modest rate of roughly 7%. This difference in speed contributes to the growing latency of a memory request, in terms of CPU cycles. This resulting latency increase has been termed the “memory wall” [12]. DRAM chip density has increased by roughly 4X-2X every three years and is said to effectively track Moore’s law for DRAM. These increases in capacity and increasing software complexity have fuelled the increases in installed DRAM capacity in a server and desk-top systems.

As memory accesses become slower with respect to the processor and consume more power with increasing memory size, the focus of memory performance and power consumption have become increasingly important. Many studies show that memory power and access time dominate over 50% of the total power and performance for computations with large storage requirements[10,11,1]. With the trend to develop multi-threaded, multi-core processors, the demands on the memory system will continue to scale.

Memory system performance is sensitive to a large number of parameters including DRAM timing parameters, memory controller policies, and memory system topologies. Each of these parameters take on a number of values and interact in fashions that make overall trends difficult to discern [9]. Each type of DRAM i.e., SDRAM, DDR SDRAM etc., has different behaviors and architectures. The DRAM type limits the memory system architecture in terms of supportable bandwidths and topologies, i.e. number of DIMMs, number of channels, which in turn effects performance. The choice of address mapping policy is sensitive to both the memory controller row buffer management policy (open-page vs closed page) and the system topology. The efficacy of a memory access reordering policy (FIFO, priority based, read centric) is impacted by all the address mapping policy, row buffer management policy, and the system topology. A comparison of the memory system architectures becomes even harder when we add the dimensions of power consumption and manufacturing cost.

Although prior research demonstrates the performance tuning possible by varying memory system parameters, there is a lack of tools in the public-domain that support such studies. Some tools implement the memory as a constant time and constant energy per access. Others implement the memory as banks, but simplify the interactions between the memory commands and elements.

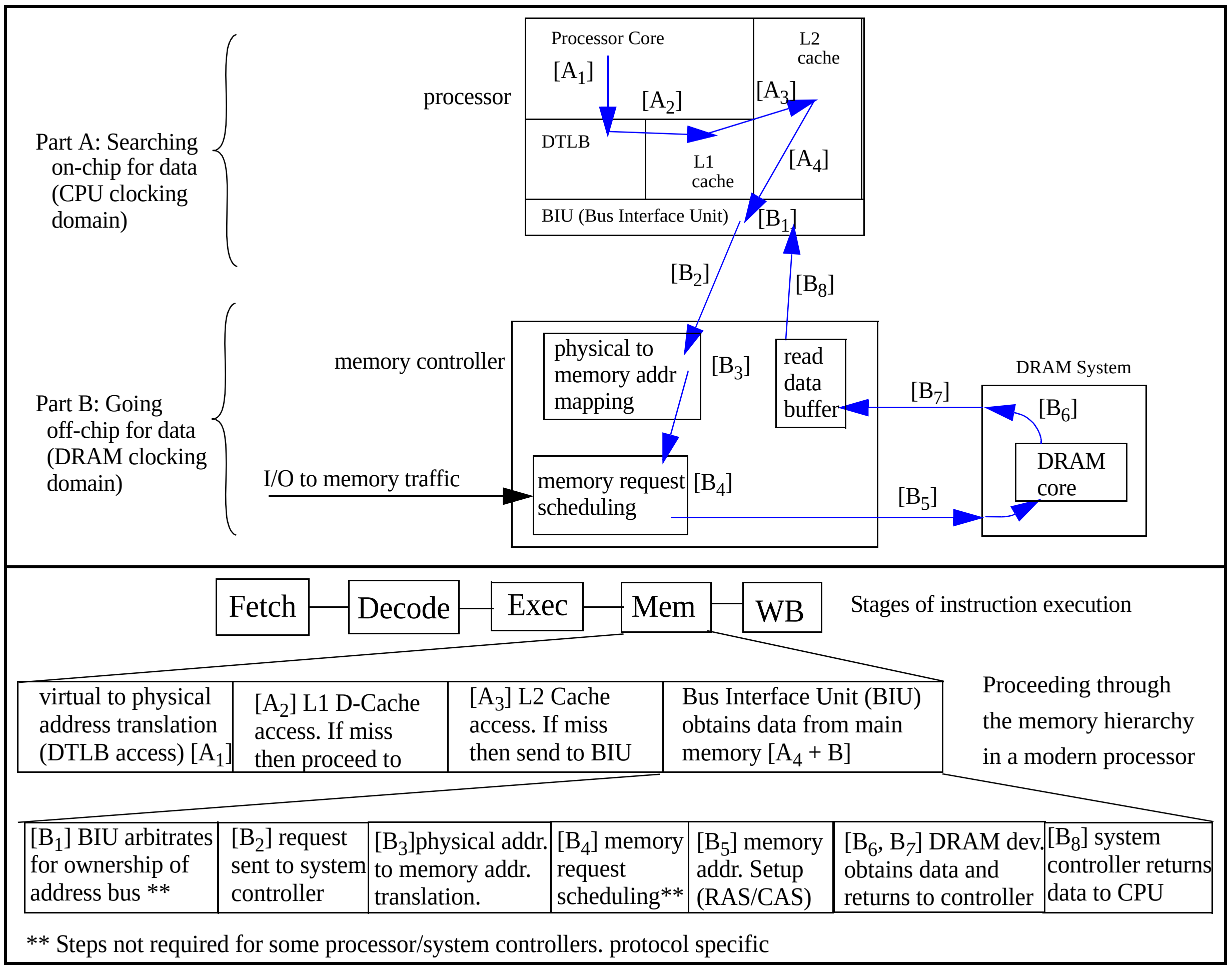
Part A: Searching on-chip for data (CPU clocking domain)
processor
Processor Core
[A1]
[A2]
L2 cache
[A3]
[A4]
DTLB
L1 cache
BIU (Bus Interface Unit)
[B1]
[B2]
[B8]
Part B: Going off-chip for data (DRAM clocking domain)
memory controller
physical to memory addr mapping
[B3]
read data buffer
DRAM System
[B7]
[B6]
DRAM core
I/O to memory traffic
memory request scheduling
[B4]
[B5]
Fetch
Decode
Exec
Mem
WB
Stages of instruction execution
virtual to physical address translation (DTLB access) [A1]
[A2] L1 D-Cache access. If miss then proceed to
[A3] L2 Cache access. If miss then send to BIU
Bus Interface Unit (BIU) obtains data from main memory [A4 + B]
Proceeding through the memory hierarchy in a modern processor
[B1] BIU arbitrates for ownership of address bus **
[B2] request sent to system controller
[B3]physical addr. to memory addr. translation.
[B4] memory request scheduling**
[B5] memory addr. Setup (RAS/CAS)
[B6, B7] DRAM dev. obtains data and returns to controller
[B8] system controller returns data to CPU
** Steps not required for some processor/system controllers. protocol specific

Fig 1: : Abstract Illustration of a Load Instruction in a Processor-Memory System.

Therefore, we introduce DRAMsim, a detailed and highly-configurable C-based memory system simulator to fill this gap. DRAMsim implements detailed timing models for a variety of existing memories including SDRAM, DDR, DDR2, DRDRAM and FB-DIMM. It also models the power consumption of SDRAM and its derivatives. It can be used as a stand-alone simulator or as a part of a more comprehensive system-level model. We have successfully integrated DRAMsim into a variety of simulators including MASE[15], Sim-alpha[14], BOCHS[2] and GEMS[13].

2 Design Goals

Our goal is to provide a detailed, realistic, highly-configurable simulator for evaluating the memory system, independent of the host platform. The simulator provide us the capability of simulating a variety of memory types (SDRAM, DDR etc.) and easily vary their parameters. It allows us to configure memory controller policies and memory controller topologies. Explicit moduralization makes it easy for first-time users to familiarize themselves with the simulator. The simulator comes with a number of default configurations for a number of DRAM types. Users can choose to use these defaults as is or over-ride the default parameters using either configuration files or command-line options. The simulator provides a variety of statistics including detailed latency stats, resource usage numbers, bank conflict and hit history. Besides performace statistics, the simulator can also generate detailed breakdown power consumption numbers for each rank. All statistics can be generated either periodically or at the end of the complete run. Finally, the simulator is versatile enough that it can easily be integrated into any complete system simulator, or be used as a stand-alone simulator.

3 DRAMsim Overview

Figure 1 shows the system topology of the simulated processor-memory system. Three distinct and separate entities that interact in the life of a memory transaction request are assumed in this framework: processor(s), memory controller(s), and DRAM memory system(s).

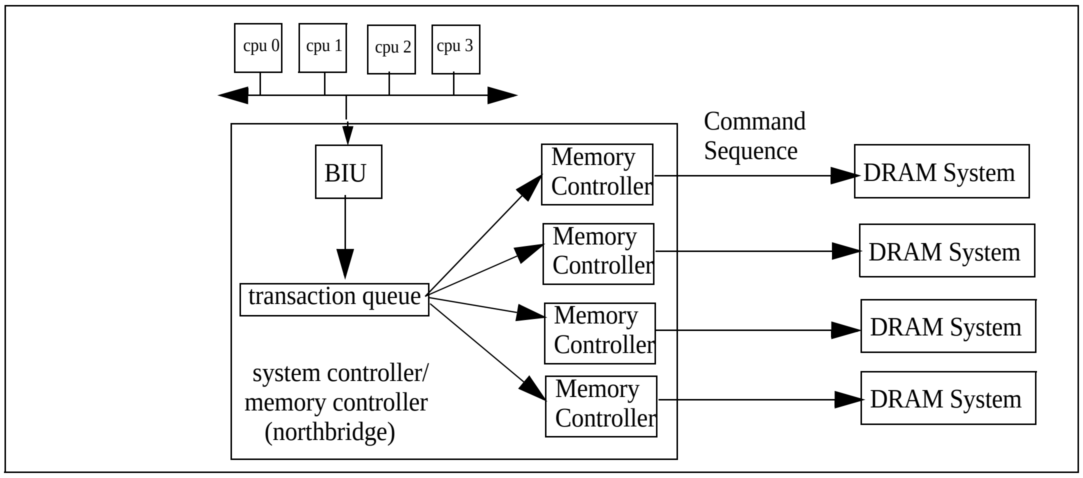

Fig 2: : Transaction Queue and Memory Controller(s) System Architecture.

Each of these entities is assumed to be an independently clocked synchronous state machine that operates in separate clocking domains. In the general implementation of the simulation framework, there are only two clocking domains: the CPU clock domain and the DRAM memory system clock domain, excluding FB-DIMM memory systems. Our simulator models the transaction's life once it leaves the processor core.

3.1 Bus Interface Unit

The memory system can support requests from an in-order core or an out-of-order execution core, where different portions of the processor can all generate memory requests. Each request is tagged with a request id (rid) to enable the callback function to uniquely identify the request generator. The simulator assumes that each functional unit can sustain more than one memory transaction miss at a given instance in time, and the memory transaction may be returned by the memory system either out-of-order or in-order. The life of a memory transaction request begins when a requesting functional unit generates a DRAM memory request. The requesting unit begins this process by attempting to place the request into a slot in the *bus interface unit* (BIU), a data structure with multiple unordered entries/slots. The BIU has the functional equivalence to MSHR's in this simulator If there is a free slot available, then the request will be successfully placed into the bus interface unit, and the status MEM_UNKNOWN will be returned to the requesting functional unit, and the memory system will return the latency of the request at a later time. If all of the slots have been filled, and no free slot is available, then MEM_RETRY will be returned to the requesting functional unit, and the functional unit must retry the request at a later time to see if a memory slot has become available at the later time.

3.2 System Controller

In figure 2, we show a generalized system controller that supports multiple processors. The simulation of the system controller begins with the selection of a memory transaction from the BIU to the transaction queue. The transaction queue then takes the memory transaction and maps the physical address of the transaction to the memory address in terms of channel ID, rank ID, bank ID, row ID and column ID via an address mapping scheme. Then, depending on the row-buffer management policy used by the system, a sequence of DRAM commands are generated for each memory transaction. The simulated memory system supports multiple memory controllers, each of which can independently control a logical channel of memory. Each logical channel may contain multiple physical channels of memory.

3.3 Transaction Queue and Transaction Ordering Policy

After the appropriate BIU entry (slot) has been selected, the status of the BIU entry is marked as SCHEDULED, then a memory transaction is created in the memory transaction queue. The selection of the memory request from the BIU into the transaction queue is referred to as the ***transaction ordering policy***. Since the transaction queue is an in-order queue, where DRAM commands of an earlier memory transaction are given higher priority than DRAM commands from later transactions, the ***transaction ordering policy*** is of great importance to determine the bandwidth and latency characteristics of DRAM memory systems. In this simulation framework, four transaction ordering policies are supported: *First Come First Serve (FCFS), Read or Instruction Fetch First (RIFF), Bank Round Robin (BRR),* and *Command Pair Rank Hopping (CPRH).*

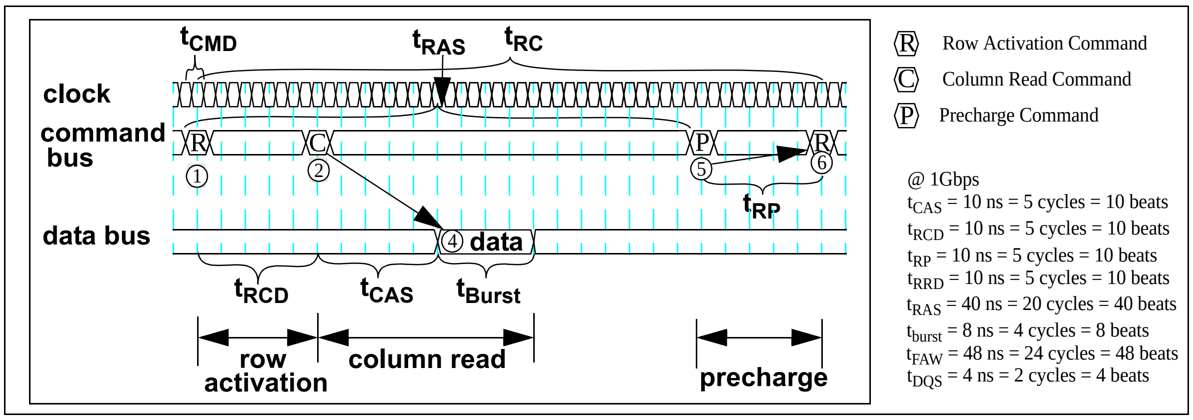
t_{CMD}
t_{RAS}
t_{RC}
clock
command bus
R
C
P
R
1
2
5
6
t_{RP}
data bus
4 data
t_{RCD}
t_{CAS}
t_{Burst}
row activation
column read
precharge
R Row Activation Command
C Column Read Command
P Precharge Command
@ 1Gbps
t_{CAS} = 10 ns = 5 cycles = 10 beats
t_{RCD} = 10 ns = 5 cycles = 10 beats
t_{RP} = 10 ns = 5 cycles = 10 beats
t_{RRD} = 10 ns = 5 cycles = 10 beats
t_{RAS} = 40 ns = 20 cycles = 40 beats
t_{burst} = 8 ns = 4 cycles = 8 beats
t_{FAW} = 48 ns = 24 cycles = 48 beats
t_{DQS} = 4 ns = 2 cycles = 4 beats

Fig 3: : A Complete "Read Cycle" in DDRx SDRAM Memory System (@ 1 Gbit).

3.4 Row Buffer Management Policy

Modern memory controllers typically deploy one of two policies to manage operations of the sense amplifiers. Since a DRAM access is essentially a two step process, in cases where the memory access sequence has a high degree of spatial locality, it would be favorable to direct the memory access sequences to the same row of memory. The *Open Page* row buffer management policy is designed to favor the case by keeping sense amplifiers open and holding an entire row of data for ready access. In contrast, the *Close Page* row buffer management policy is designed to favor random accesses to different rows of memory.

3.5 Address Mapping

Before data can be read from or written to a memory location, the physical address given by the CPU is translated into memory addresses in the form of channel ID, rank ID, bank ID, row ID, and column ID. For memory controllers that use the open page row buffer management policy, the address mapping scheme should optimize the temporal and spatial locality of the address request stream and direct memory accesses to an open DRAM row (bank) and minimize DRAM bank conflicts. In a closed-page system, the goal of the address mapping scheme is to minimize temporal and spatial locality to any given bank and instead distribute memory accesses throughout different banks in the memory system.

Address mapping scheme depends not only on the row buffer management policy, but also on the configuration of the DRAM memory system as well as the expandability/non-expandability of the memory system. For example, depending on design, the channel ID or rank ID can be mapped to the low order address bit to obtain the most bank parallelism, but in memory systems that allow end users to flexibly configure the memory system by adding more ranks or changing channel configurations, the channel ID and rank ID's are typically mapped to the high order address bits. In the simulator, the users can choose the address mapping scheme from *burger_base_map, burger_alt_map, intel845g_map, sdram_base_map, sdram_hiperf_map,* and *sdram_close_page_map.*

3.6 DRAM Command Chain

Each memory transaction is translated into one or more DRAM commands, which are RAS, CAS and PRECHARGE. In this simulation framework, this sequence of DRAM commands is referred to as the DRAM command chain. The sequence of DRAM commands in the command chain depends on the row buffer management policy as well as on the state of the DRAM memory system. In an open page memory system, a memory transaction may be translated into: a single column access command if the row is already open, a precharge command, a row access command and a column access command if there is a bank conflict, or just a row access command and a column access command if the bank is currently idle.

In a close page memory system, all of the memory transactions translate to a sequence of three DRAM commands that completes a read cycle. Figure 3 illustrates a read cycle in a close-page DDRx SDRAM memory system. DRAMsim can be configured to print a detailed text-based view of the command and data buses.

3.7 Power

With continuing emphasis placed on memory system performance, DRAM manufacturers are expected to push for ever higher data transfer rates in each successive generation of DRAM devices. However, just as increasing operating frequencies lead to higher activity rates and higher power consumption in modern processors, increasing data rates for DRAM devices also increase the potential for higher activity rates and higher power consumptions on DRAM devices. In modern DRAM devices, each time a row is activated, thousands of bits are discharged, sensed, then recharged in parallel. As a result, the row activation command is a relatively

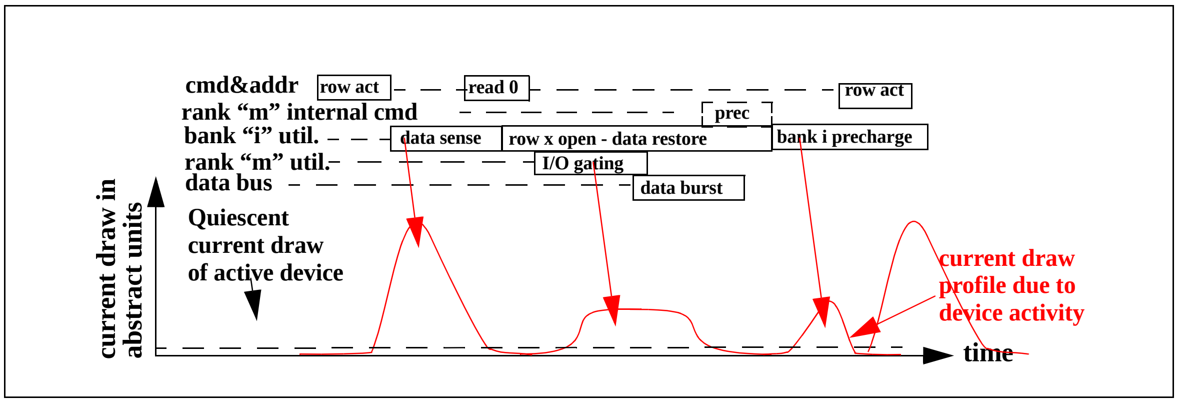

Fig 4: : Current Profile of a DRAM Read Cycle.

energy intensive operation. Figure 4 shows the current profile of a DRAM read cycle. Figure 4 shows that an active DRAM device draws a relatively constant current level. The DRAM device then draws additional current for each activity on the DRAM device. The total current draw of the DRAM device is simply the summation of the quiescent current draw and the current draw of each activity on the DRAM device.

DRAMsim is incorporated with a power model for DDR, DDR2 SDRAM, and fully-buffered DIMMS. The model calculates the average power in one activation-to-activation cycle, by calculating the power in each DRAM state, according to the current drawn, and multiplying it with the fraction of time the device spends in each state in one activation-to-activation cycle. DDR and DDR2 power models are based on the power model from Micron [7]. The power model for the fully-buffered DIMMS has an additional component, the Advanced Memory Buffer (AMB).

The power consumption statistics of the memory system are broken down into its various components, depending on the power states. These statistics can be obtained for individual ranks in the system. They can be logged periodically or at the end to obtain an over-time view or global view respectively.

4 Interfacing with other simulators

The DRAM simulator can be easily integrated with any processor simulator or memory simulation frame-work that supports the following features

- *Variable Memory Latency* The latency of a memory request is not a fixed value. It is a function of the state of the DRAM, temporally adjacent requests to memory and controller policies. For example, in a system that uses the open page policy, a request that accesses an already open row will have a lower latency than one that has a bank conflict.
- *Unknown Memory Latency* at the point of issue The dram simulator is an execution driven simulator. Hence the latency of a request is determined only after processing the request and not at the time of issue. Upon the completion of a request, the DRAM simulator calls back the processor simulator using a specified function with the timing information. An advantage of this approach is that it allows us to study memory controller scheduling policies that reorder requests.
- *Support for Retries* As described earlier, requests sent to memory are placed in a slot in the bus interface unit (BIU) queue. When all available slots are occupied, the simulator will refuse to process additional requests. The processor simulator should handle the inability of the DRAM simulator to process requests due to queues filling up and retry such requests later.

4.1 Functional Interface

DRAMsim can be used as part of a larger simulation framework. For the purpose of interfacing, several functions are provided. These functions are classified based on their actions and described in further detail below.

Initialization : is done using the function *init_dram*. The main purpose of this function is to create and initialize the simulators data structures in the default configuration. By default, the simulator models an SDRAM-based memory system. The memory controller uses a closed page policy and a FIFO memory access ordering policy. Statistics are enabled to be collected and written to stdout.

Configuration: DRAMsim supports the configuration of a wide variety of parameters including DRAM parameters like row activation time (t_ras), column access latencies (t_cas), etc., memory topology informations like number of ranks, channels, etc., and memory controller policies like row buffer management policies, memory access ordering policy, etc. The configuration details can be specified using a configuration file, also known as the *spd* file. Power consumption details can be specified using an additional file. In order to support a high-degree of configurability, DRAMsim also allows the user to over-ride the base configuration file parameters. This configuration capability is supported in the basic stand-alone version via a number of command-line flags. These flags are described in detail in the simulator manual [16]. A processor simulator can perform this configuration using the *configure_sim_dram* function, which takes as an argument a configuration structure (*dram_config_info*). Both the *dram_config_info* data structure and the configuration file specify identical parameters. This capability we found was very useful for the purpose of automating our simulations for design space explorations.

Request Dispatch: The processor sends a request to the memory sub-system using the function *dram_access_memory*. The processor passes on access specific information using a special data structure, *dram_access_t*. The essential elements of this structure include unique rid (request id), a memory address, and a callback function, which is to be called by the DRAM simulator upon the completion of a memory request. Besides these the structure also has optional elements which are used to simulate particular memory controller policies. These include priority i.e. processor-specified priority which can be used by priority-based memory access reordering schemes and a thread id which is useful in multi-threaded or multi-processor contexts.

Periodic Update Manager: The function *dram_update_system* serves as the heart-beat of the simulator. This function should be called every cycle that the dram sub-system is busy processing a request. In order to interface with simulators that use event managers it returns information whether the dram simulator has active requests or not.

Request Return: The periodic update manager, *dram_update_system,* checks for the completion of memory requests. While taking such requests off its queue, the dram simulator invokes the specified callback function. The actual callback is performed in the function, *invoke_callback_fn.*

Statistics: The simulator allows for the collection of a variety of statistics including latency distributions, BIU occupancy, bank hit and bank conflict information. The interface supports the initialization of this statistic gathering via the function *dram_stats_init* and its logging via the file, *dram_print_stats_common*, statistics are logged to a common file and *dram_print_stats_general*, for statistics to be logged onto individual files or stdout.

4.2 Existing Simulator Ports

SYSim - full system simulator . SYSim is a model of the entire memory hierarchy for a uni-processor system that includes performance and energy models for caches, DRAM, and disk. The main purpose of the project is to investigate the *systemic behaviors* of the entire memory hierarchy in virtual memory, with extremely detailed simulation. The SYSim project is incorporated with several simulators for each component of the system.

Bochs[2], a Pentium emulator, is used as the CPU model that generates the memory accesses and I/O interrupts. The cache model comes from Wattch[8] and Cacti [3]. DRAMsim is integrated into the system to give the timing behavior and also the power consumption of the memory system. Since the Disk model in Bochs takes the responsibility only to read and write data from/to the disk image, a modified DiskSim[4] simulator is integrated as described in the DRPM paper[5] into the system. The DRPM version of DiskSim is used for only timing and power consumption statistics collection.

The challenges of integration were to reflect the multiple memory and I/O requests on the fly while maintaining the correct interaction between the disk, caches, and DRAM communication via DMA. Memory management is another issue as careless allocation of the simulator memory, could cause the host system to crash.

GEMS . This is a multi-processor simulation framework developed at the University of Wisconsin Madison [13]. This simulator ships with several cache coherence protocols for both SMP and CMP machines. Our simulator currently does not support the concept of multiple BIUs and memory sub-systems, hence we ported our simulator to the CMP based protocols. This involved modifying the protocol definition files and insert calls to the memory system at the appropriate points. The challenges were in introducing the concept of retries due to resource shortages in the memory system. This was fairly trivial in the case of READ transactions. In the case of WRITE transactions the protocols by default evicted cachelines that were written back to memory. By postponing these evictions to only when the memory system indicated the completion of the write transaction, we discovered several state transitions that were not previously handled by the protocol.

Traditional uniprocessor simulators - MASE/ Sim-alpha . The DRAM simulator has been successfully integrated into sim-alpha[14], a detailed execution-drive simulator of the Alpha 21264, and into SIM-MASE[15], SimpleScalar 4.0.

5 Problems and Limitations

The development of sim-DRAM was started by David Wang while working on his Ph.D. research roughly three years ago[6]. The code since then has been worked on by a number of students and has expanded to support a wide variety of DRAM systems, provide different statistics and interface with several different processor simulators. The simulator comes with some limitations that we hope to fix in future releases.

- ***C-Based Implementation*** The simulator is entirely written in C. Many of the scalability issues that we face currently stem from the choice of programming language.
- ***Simulation of Distributed Memory*** The simulator data-structures by default assume the existence of a single memory system. Thus, there is no current support in the memory system for distributed physical memory.
- ***Lack of Event Queue*** There is no built-in event queue in the simulator. The absence of such a queue requires the simulator to be polled every cycle while a transaction is active.
- ***Clock Domains*** The simulation framework currently implements only two clocking domains: the CPU domain and the memory system domain (FBDIMM excepted). The DRAM memory system and the memory controller are operated in the DRAM memory system clock domain, and the CPU is operated in the CPU clock domain. This is true for legacy systems with separate memory controllers, while newer systems where the memory controllers is integrated into the CPU core the assumption may be reversed. A more generalized model would operate the CPU, memory controller and dram system in three independent clock domains. However, this implementation would be unnecessarily complex, and decrease simulation speed for minimal increase in the system simulation model flexibility and accuracy.

6 Conclusion

The focus of memory performance and power consumption have become increasingly important due to the memory wall, increase in memory size, and complexity of applications. The memory system performance is sensitive to a large number of parameters interacting in unexpected fashions. The memory system design decision becomes even harder when we add the dimensions of power consumption and manufacturing cost. Due to a lack of tools in the public-domain that support such detailed studies, we introduce DRAMsim, a detailed and highly-configurable C-based memory system simulator. DRAMsim implements detailed timing models for a variety of existing memories, including SDRAM, DDR, DDR2, DRDRAM and FB-DIMM, with the capability to easily vary their parameters. It also models the power consumption of SDRAM and its derivatives. It can be used as a stand-alone simulator or as a part of a more comprehensive system-level model. We have successfully integrated into a variety of simulators including MASE[15], Sim-alpha[14], BOCHS[2] and GEMS[13]. The simulator can be downloaded from www.ece.umd.edu/dramsim.

7 Acknowledgements

The work of Brinda Ganesh, Katie Baynes, and Aamer Jaleel was supported in part by NSF CAREER Award CCR-9983618, NSF grant EIA-9806645, and NSF grant EIA-0000439. The work of Dave Wang was supported in part by NSF CAREER Award CCR-9983618, NSF grant EIA-9806645, NSF grant EIA-0000439, and Cray Inc.The work of Nuengwong Tuaycharoen is supported by Dhurakij Pundit University, Bangkok, Thailand.. The work of Bruce Jacob was supported in part by NSF CAREER Award CCR-9983618, NSF grant EIA-9806645, NSF grant EIA-0000439, DOD MURI award AFOSR-F496200110374, the Laboratory of Physical Sciences in College Park MD, the National Institute of Standards and Technology, and Cray Inc.

References

[1] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, L.K. John, “Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach,” In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-8), Cambridge, MA, pages 141-150, February, 2002.

[2] The Bochs IA-32 Emulator Project. http://bochs.sourceforge.net

[3] S. Wilton and N. Jouppi, “An Enhanced Access and Cycle Time Model for On-chip Caches,” In WRL Research Report 93/5, DEC Western Research Laboratory, 1994.

[4] G. Ganger, B.Worthington, and Y. Patt, “The DiskSim Simulation Environment Version 2.0 Reference Manual,” http://www.ece.cmu.edu/ ganger/disksim/.

[5] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, “DRPM: Dynamic Speed Control for Power Management in Server Class Disks,” In the Proceedings of the International Symposium on Computer Architecture (ISCA), pages 169-179, June, 2003.

[6] David T. Wang, “Modern DRAM Memory systems: Performance Analysis and Scheduling Algorithm,” Ph.D. Dissertation, Electrical and Computer Engineering, University of Maryland at College Park, 2005.

[7] Jeff Janzen, The Micron System-Power Calculator. http://www.micron.com/products/dram/syscalc.html

[8] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” In Proceedings of the 27th Annual International Symposium on Computer Architecture, June 2000.

[9] Vinodh Cuppu and Bruce Jacob, “Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?,” In Proc. 28th International Symposium on Computer Architecture (ISCA 2001), pp. 62-71, Goteborg Sweden, June 2001.

[10] D. Lidsky and J. Rabaey, “Low-power design of memory intensive functions,” Proceedings of the IEEE Symposium on Low Power Electronics (Sept.), IEEE Computer Society Press, Los Alamitos, CA, 16-17, 1994.

[11] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle, “Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design,” Kluwer Academic, Dordrecht, Netherlands, 1998a.

[12] Wm. A. Wulf, Sally A. McKee, “Hitting the memory wall: implications of the obvious”, SACM SIGARCH Computer Architecture News, Volume 23,Issue 1 (March 1995),Pages:20 - 24.

[13] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS)Toolset," Computer Architecture News (CAN), TBA 2005.

[14] R. Desikan, D.C. Burger, S.W. Keckler, and Todd Austin. "Sim-alpha: a Validated, Execution-Driven Alpha 21264 Simulator.", The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-01-23.

[15] Eric Larson, Saugata Chatterjee, and Todd Austin, “MASE: A Novel Infrastructure for Detailed Micro architectural Modeling,” In the 2001 International Symposium on Performance Analysis of Systems and Software, Nov. 2001.

[16] David Wang, University of Maryland Memory System Simulator