DRAMSim2

Elliott Cooper-Balis
Paul Rosenfeld
Bruce Jacob
University of Maryland
dramninjas *[at]* gmail *[dot]* com

Contents

1 Why Do We Need to Simulate DRAM Systems This Accurately?

Modern computer system performance is increasingly limited by the performance of DRAM-based memory systems. As a result, there is great interest in providing accurate simulations of DRAM based memory systems as part of architectural research. Unfortunately, there is great difficulty associated with the study of modern DRAM memory systems, arising from the fact that DRAM-system performance depends on many independent variables such as workload characteristics of memory access rate and request sequence, memory-system architecture, memory-system configuration, DRAM access protocol, and DRAM device timing parameters. As a result, system architects and design engineers often disagree on the usefulness of a given performance-enhancing feature, since the performance impact of that feature typically depends on the characteristics of specific workloads, memory-system architecture, memory-system configuration, DRAM access protocol and DRAM device timing parameters.

1.1 DRAM Scheduling Complexity is Growing

Figure 1: Timing diagram showing complexity of DRAM scheduling

Figure 1 shows the pipelined scheduling of a DDR2 SDRAM device. Despite the fact that the simulated memory system uses a closed-page policy and rotates through available banks on the DRAM device, which should simplify scheduling considerably, the scheduling of this system is actually more complex than earlier DRAM systems: for instance, new timing parameters such as $t_{RRD} and t_{FAW} are contributing to the growing set of timing constraints placed one a$

1.2 DRAM performance characteristics changes every generation

DRAM based memory systems are impacted primarily by two attributes: row cycle time and device datarate. Presently, DRAM row cycle times are decreasing at a rate of approximately 7% per year, and DRAM device datarates are increasing with each new generation of DRAM devices at the rate of 100% every three years,

Figure 2: DRAM row cycle time trends

Figure 3: DRAM device data rate trends

The difference in the scaling trends of the DRAM device means that fundamental DRAM device performance characteristics are changing every single generation, and the changing performance characteristics cannot be accurately predicted by linear extrapolations. The result is that no computer architect can rest easy knowing that he or she has obtained X% of performance improvement with a set of microarchitectural techniques on the current generation memory system, because the same set of microarchitectural techniques may not to be as effective in a future memory system due to the differences in the scaling attributes of DRAM devices,

1.3 The Sales Pitch

Our DRAM-system simulation work enables system architects not only to explore the impact of a set of microarchitectual techniques on a given memory system but also to examine the effectiveness of those microarchitectural techniques on a future generation memory system with future generations of DRAM devices.

2 About DRAMSim2

DRAMSim2 is a cycle accurate model of a DRAM memory controller, the DRAM modules which comprise system storage, and the buses by which they communicate.

The overarching goal is to have a simulator that is small, portable, and accurate. The simulator core has a simple interface which allows it to be CPU simulator agnostic and should to work with any simulator (see section 5.2). This core has no external run time or build time dependencies and has been tested with g++ on Linux as well as g++ on Cygwin on Windows.

3 Getting DRAMSim2

DRAMSim2 is available on github. If you have git installed you can clone our repository by typing:

```
$ git clone git://github.com/dramninjasUMD/DRAMSim2.git
```

4 Building DRAMSim2

To build an optimized standalone trace-based simulator called `DRAMSim` simply type:

```
$ make
```

For a debug build which contains debugging symbols and verbose output, run:

```
$ make DEBUG=1
```

To build the DRAMSim2 library, type:

```
$ make libdramsim.so
```

5 Running DRAMSim2

5.1 Trace-Based Simulation

In standalone mode, DRAMSim2 can simulate memory system traces. While traces are not as accurate as a real CPU model driving the memory model, they are convenient since they can be generated in a number of different ways (instrumentation, hardware traces, CPU simulation, etc.) and reused.
We've provided a few small sample traces in the `traces/` directory. These gzipped traces should first be preprocessed before running through the simulator. To run the preprocessor (the preprocessor requires python):

```
cd traces/
./traceParse.py k6_aoe_02_short.trc.gz
```

This should produce the file `traces/k6_aoe_02_short.trc`. Then, go back to the DRAMSim2 directory and run the trace based simulator:

```
cd .
./DRAMSim -t traces/k6_aoe_02_short.trc -s system.ini -d ini/DDR3_micron_64M_8B_x4_sg15.ini -c 1000
```

This will run a 1000 cycle simulation of the `k6_aoe_02_short` trace using the specified DDR3 part. The -s, -d, and -t flags are required to run a simulation.
A full list of the command line arguments can be obtained by typing:

```
$ ./DRAMSim --help
DRAMSim2 Usage:
DRAMSim -t tracefile -s system.ini -d ini/device.ini [-c #] [-p pwd] -q
  -t, --tracefile=FILENAME  specify a tracefile to run
  -s, --systemini=FILENAME  specify an ini file that describes the memory system parameters
  -d, --deviceini=FILENAME  specify an ini file that describes the device-level parameters
  -c, --numcycles=#     specify number of cycles to run the simulation for [default=30]
  -q, --quiet       flag to suppress simulation output (except final stats) [default=no]
  -o, --option=OPTION_A=234     overwrite any ini file option from the command line
  -p, --pwd=DIRECTORY   Set the working directory
```

Some traces include timing information, which can be used by the simulator or ignored. The benefit of ignoring timing information is that requests will stream as fast as possible into the memory system and can serve as a good stress test. To toggle the use of clock cycles, please change the `useClockCycle` flag in `TraceBasedSim.cpp`. If you have a custom trace format you'd like to use, you can modify the `parseTraceFileLine()` function ton add support for your own trace formats.
The prefix of the filename determines which type of trace this function will use (ex: k6_foo.trc) will use the k6 format in `parseTraceFileLine()`.

5.2 Library Interface

In addition to simulating memory traces, DRAMSim2 can also be built as a dynamic shared library which is convenient for connecting it to CPU simulators or other custom front ends. A `MemorySystem` object encapsulates the functionality of the memory system (i.e., the memory controller and DIMMs). The classes that comprise DRAMSim2 can be seen in figure 4. A simple example application is provided in the `example_app/` directory. At this time we have plans to provide code to integrate DRAMSim2 into MARSSx86, SST, and (eventually) M5.

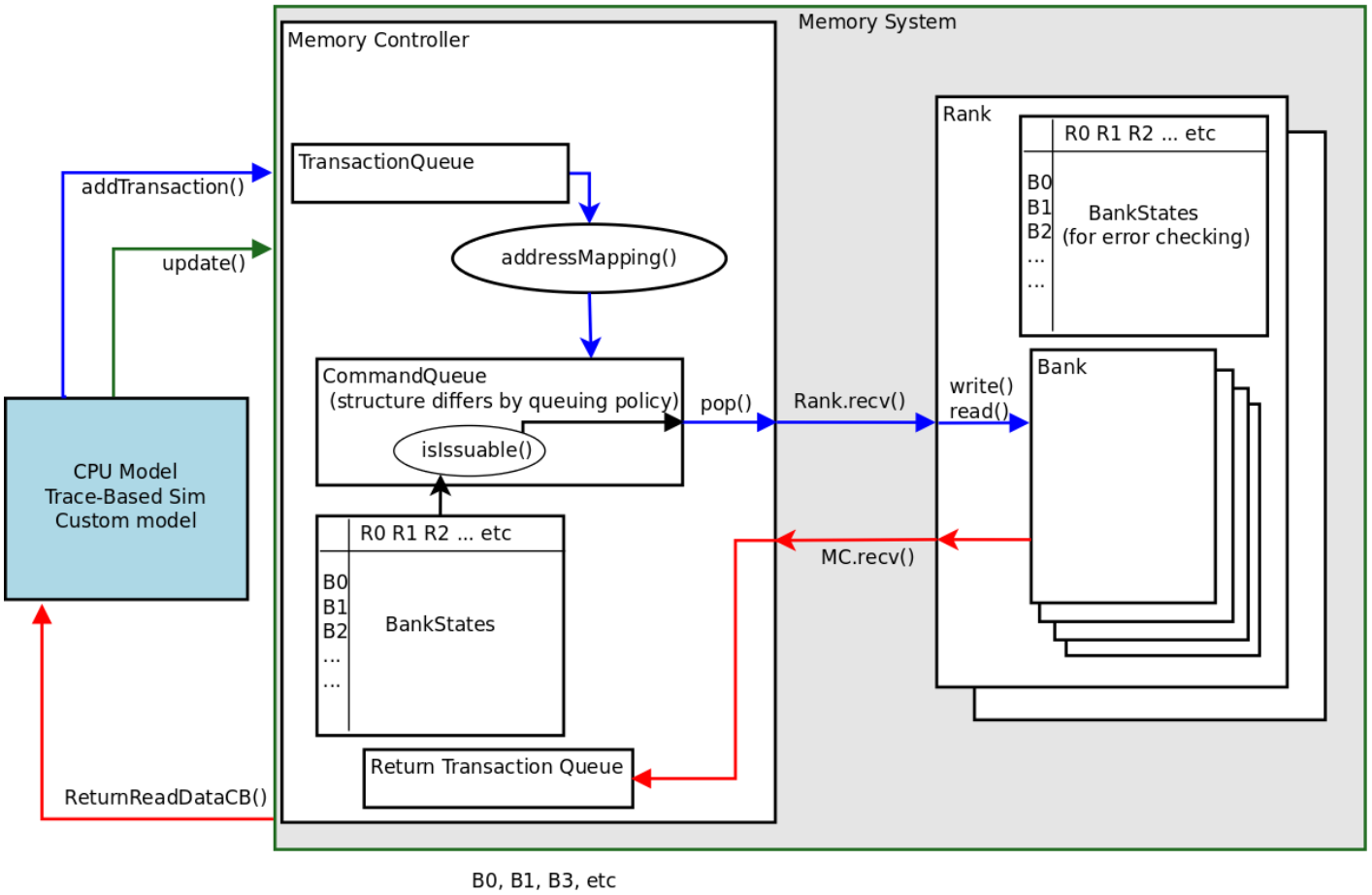

Figure 4: Block diagram of DRAMSim2. The `recv()` functions are actually called `receiveFromBus()` but were abbreviated to save sapce.

6 Example Output

The verbosity of the DRAMSim2 can be customized in the system.ini file by turning the various debug flags on or off.
Below, we have provided a detailed explanation of the simulator output. With all DEBUG flags enabled, the following output is displayed for each cycle executed.
NOTE : BP = Bus Packet, T = Transaction
MC = MemoryController, R# = Rank (index #)

```
 ----------------- Memory System Update ------------------
 ---------- Memory Controller Update Starting ------------ [8]
 -- R0 Receiving On Bus    : BP [ACT] pa[0x5dec7f0] r[0] b[3] row[1502] col[799]
 -- MC Issuing On Data Bus    : BP [DATA] pa[0x7edc7e0] r[0] b[2] row[2029] col[799] data[0]=
 ++ Adding Read energy to total energy
 -- MC Issuing On Command Bus : BP [READ_P] pa[0x5dec7f8] r[1] b[3] row[1502] col[799]
== New Transaction - Mapping Address [0x5dec800] (read)
  Rank : 0
  Bank : 0
  Row  : 1502
  Col  : 800
 ++ Adding IDD3N to total energy [from rank 0]
 ++ Adding IDD3N to total energy [from rank 1]
== Printing transaction queue
  8]T [Read] [0x45bbfa4]
  9]T [Write] [0x55fbfa0] [5439E]
  10]T [Write] [0x55fbfa8] [1111]
== Printing bank states (According to MC)
[idle] [idle] [2029] [1502]
[idle] [idle] [1502] [1502]

== Printing Per Rank, Per Bank Queue
 = Rank 0
    Bank 0   size : 2
       0]BP [ACT] pa[0x5dec800] r[0] b[0] row[1502] col[800]
       1]BP [READ_P] pa[0x5dec800] r[0] b[0] row[1502] col[800]
    Bank 1   size : 2
       0]BP [ACT] pa[0x5dec810] r[0] b[1] row[1502] col[800]
       1]BP [READ_P] pa[0x5dec810] r[0] b[1] row[1502] col[800]
    Bank 2   size : 2
       0]BP [ACT] pa[0x5dec7e0] r[0] b[2] row[1502] col[799]
       1]BP [READ_P] pa[0x5dec7e0] r[0] b[2] row[1502] col[799]
    Bank 3   size : 1
       0]BP [READ_P] pa[0x5dec7f0] r[0] b[3] row[1502] col[799]
 = Rank 1
    Bank 0   size : 2
       0]BP [ACT] pa[0x5dec808] r[1] b[0] row[1502] col[800]
       1]BP [READ_P] pa[0x5dec808] r[1] b[0] row[1502] col[800]
    Bank 1   size : 2
       0]BP [ACT] pa[0x5dec818] r[1] b[1] row[1502] col[800]
       1]BP [READ_P] pa[0x5dec818] r[1] b[1] row[1502] col[800]
    Bank 2   size : 1
       0]BP [READ_P] pa[0x5dec7e8] r[1] b[2] row[1502] col[799]
    Bank 3   size : 0
```

Anything sent on the bus is encapsulated in an BusPacket (BP) object. When printing, they display the following information:

```
BP [ACT] pa[0x5dec818] r[1] b[1] row[1502] col[800]
```

The information displayed is (in order): command type, physical address, rank #, bank #, row #, and column #.

Lines beginning with " – " indicate bus traffic, ie,

```
    -- R0 Receiving On Bus       : BP [ACT] pa[0x5dec7f0] r[0] b[3] row[1502] col[799]
    -- MC Issuing On Data Bus    : BP [DATA] pa[0x7edc7e0] r[0] b[2] row[2029] col[799] data[0]=
    -- MC Issuing On Command Bus : BP [READ_P] pa[0x5dec7f8] r[1] b[3] row[1502] col[799]
```

Sender and receiver are indicated and the packet being sent is detailed.

Lines beginning with " ++ " indicate power calculations, ie,

```
    ++ Adding Read energy to total energy
    ++ Adding IDD3N to total energy [from rank 0]
    ++ Adding IDD3N to total energy [from rank 1]
```

The state of the system and the actions taken determine which current draw is used. For further detail about each current value, see Micron datasheet.

If a pending transaction is in the transaction queue, it will be printed, as seen below:

```
  == Printing transaction queue
   1]T [Read] [0x45bbfa4]
   2]T [Write] [0x55fbfa0] [5439E]
   3]T [Write] [0x55fbfa8] [1111]
```

Currently, at the start of every cycle, the head of the transaction queue is removed, broken up into DRAM commands and placed in the appropriate command queues. To do this, an address mapping scheme is applied to the transaction's physical address, the output of which is seen below:

```
 == New Transaction - Mapping Address [0x5dec800] (read)
  Rank : 0
    Bank : 0
    Row  : 1502
    Col  : 800
```

If there are pending commands in the command queue, they will be printed. The output is dependent on the designated structure for the command queue. For example, per-rank/per-bank queues are shown below:

```
  = Rank 1
   Bank 0   size : 2
      0]BP [ACT] pa[0x5dec808] r[1] b[0] row[1502] col[800]
      1]BP [READ_P] pa[0x5dec808] r[1] b[0] row[1502] col[800]
   Bank 1   size : 2
      0]BP [ACT] pa[0x5dec818] r[1] b[1] row[1502] col[800]
      1]BP [READ_P] pa[0x5dec818] r[1] b[1] row[1502] col[800]
   Bank 2   size : 1
      0]BP [READ_P] pa[0x5dec7e8] r[1] b[2] row[1502] col[799]
   Bank 3   size : 0
```

The state of each bank in the system is also displayed:

```
    == Printing bank states (According to MC)
    [idle] [idle] [2029] [1502]
    [idle] [idle] [1502] [1502]
```

Banks can be in many states, including idle, row active (shown with the row that is active), refreshing, or precharging. These states will update based on the commands being sent by the controller.

7 Results Output

In addition to printing memory statistics and debug information to standard out, DRAMSim2 also produces a 'vis' file in the `results/` directory. A vis file is essentially a summary of relevant statistics that is generated per

epoch (the number of cycles per epoch can be set by changing the `EPOCH_COUNT` parameter in the `system.ini` file).

We are currently working on DRAMVis, which is a cross-platform viewer which parses the vis file and generates graphs that can be used to analyze and compare results.