Lecture Notes: Memory Systems

Rajeev Balasubramonian

March 29, 2012

1 DRAM vs. SRAM

On a processor chip, data is typically stored in SRAM caches. However, an SRAM cell is large enough that a single processor chip can only accommodate a few mega bytes of data. Applications typically have memory requirements that run into many giga bytes. To fulfil this memory requirement, a memory system is constructed on a motherboard using several high-density and low-cost DRAM chips. Since we're going off the processor chip to access this memory, high latency is inevitable. Often, the initial focus in constructing the memory system is high capacity because that has one of the biggest impacts on overall performance; note that if data is not found in memory, a disk access is required, which is far more expensive. This is why the memory system uses chips with DRAM cells, which have high density.

An SRAM cell uses two back-to-back inverters to store a bit. Because of this feedback loop, data is retained in the cell in spite of continuous leakage current within the cell. Because of this data retention, the cell is called *Static* Random Access Memory. The use of two inverters per cell also leads to low density per bit.

A DRAM cell, on the other hand, is optimized for high density. A cell is basically comprised of a single capacitor. When charge is stored in the capacitor, it represents a '1', else a '0'. Over time, the charge in the capacitor leaks away and the data bit is lost. Hence, to retain the value, it must be read and re-written before the charge is completely lost. Because of this refresh process, the cell is referred to as *Dynamic* Random Access Memory.

2 Memory Channel

A processor has a memory controller that is responsible for issuing commands to the memory system. The memory chips are largely "dumb"; they are almost entirely controlled by commands emanating from the memory controller. The memory controller is connected to pins on the processor chip, which in turn are connected to a memory channel (wires on a motherboard).

The memory channel is made up of a data bus and an address/command bus. The address/command bus is a unidirectional bus that carries about 17 bits of address and 6 bits of command from the memory controller to memory chips. The data bus is bidirectional and carries 64 bits of data from the memory chips to the memory controller for a read request, and vice versa for a write.

The memory channel typically operates at a frequency slower than the processor – modern channels are typically 800 MHz or 1066 MHz. The memory system also follows the DDR (Double Data Rate) standard, which uses both the rising and falling clock edges to carry signals, i.e., a channel wire carries two bits per cycle.

In order to support a high channel frequency, the channel must be relatively short and drive a small load. Hence, only a few memory modules can be connected to the channel.

Modern processors can have as many as four DDR3 memory channels, with each having an independent memory controller.

3 DIMMs, Ranks, Banks, Arrays

The memory modules that plug into the memory channel are referred to as *Dual Inline Memory Modules (DIMMs)*. Each DIMM is a printed circuit board (PCB) with memory chips on the front and back. Since memory components follow the JEDEC standard, they are typically interchangeable, i.e., if you wanted to upgrade the memory in your system, you could pull out a DIMM and replace it with another that has higher capacity.

A DIMM is organized into 1, 2, or 4 ranks. A *rank* is a collection of DRAM chips that all work together to keep the 64-bit data bus busy in a cycle. If a single DRAM chip has a data input/output width of 8 bits, it is referred to as a x8 (“by 8”) chip. We would require 8 such x8 chips to feed a 64-bit data bus; this collection of 8 x8 chips is referred to as a rank. You could construct a rank with 64 x1 chips, 32 x2 chips, 16 x4 chips, 8 x8 chips, 4 x16 chips, 2 x32 chips, and 1 x64 chip (the last two are not commercially popular).

A single data wire on the memory channel is only connected to one DRAM chip pin on a rank. If a memory channel supports four ranks, a data wire must drive four different pins. The number of ranks on a channel is kept small to reduce this load on the bus. On the other hand, an address/command wire on the memory channel is connected to every DRAM chip on every rank on the channel. Because of this higher load, the address/command wires do not use DDR. To reduce this load, sometimes special DIMMs, called buffered DIMMs, are used. A buffered DIMM has a buffer chip that receives the address/command signals and then broadcasts them to all the DRAM chips on the DIMM, i.e., an address/command wire on the memory channel only drives one buffer chip per DIMM, not every DRAM chip on the channel.

Fetching data from a rank can take a long time (about 50 ns) because of slow circuits on a DRAM chip. The memory system would be extremely slow if we had to wait for one request to complete before issuing the next. Therefore, multiple requests are serviced in parallel. For a read request, the “memory pipeline” is as follows: the request is issued on the address/command bus; the involved DRAM chips in a rank activate their circuits to pick out the requested data; the requested data is sent back on the data bus. To promote parallelism, once a request has moved on to the second stage of the pipeline, the first stage is available for use by other requests. So while one rank is busy picking out the requested data, we can use the address/command bus to fire requests to other ranks. Therefore, at a time, multiple ranks could be busy picking out the requested data; ultimately, all the data responses will happen sequentially on the single data bus that is shared by the ranks.

This 3-stage pipeline is not balanced; the first stage (address/command bus) takes about 1 ns, the second stage (data select) takes about 50 ns, and the third stage (data bus transfer) takes about 5 ns. If we want to keep the data bus fully utilized, several requests must be working on the second stage in parallel. The existence of up to four ranks on a channel helps with this parallelism, but is not enough. Therefore, a single rank is itself partitioned into multiple *banks*. If we assume that a channel has four ranks and each rank is broken into eight banks, a single channel now supports 32 banks. Each bank can work on its request independently, giving us a high degree of parallelism, and maximizing the chances that some data is ready to be sent on the data bus in every cycle.

If a rank is made up of eight DRAM chips, each bank in that rank also spans across eight DRAM chips. Each bank in this example is therefore partitioned into eight *sub-banks*, where each sub-bank resides entirely in one DRAM chip (sub-bank is a term I made up to reduce confusion). A sub-bank is itself partitioned into many data *arrays*. This is done to reduce the latency for data look-up and to reduce the interconnect overhead. You’ll find arrays being referred to with different names in the literature: mats, subarrays, tiles.

A single 64-byte cache line is fetched from a single bank. The cache line is scattered across all the DRAM chips that form the rank and bank. If we assume a rank that has 8 x8 chips, it means that each chip (and sub-bank) contributes 64 bits to the entire cache line transfer. Those 64 bits are themselves scattered across multiple arrays within that sub-bank. Once the data is selected, the data transfer on the memory channel will require eight 64-bit transfers. For each 64-bit transfer, each chip is contributing eight bits in this example. Keep in mind that the eight 64-bit transfers requires four data bus cycles since we're using double data rate.

4 Row Buffers

Ultimately, a data request will activate several arrays on several DRAM chips. Each array is simply a matrix of DRAM cells, with say, 1024 rows and 1024 columns (a 1 Mb array). The horizontal wire that spans a row is called a wordline. The data request will activate a single wordline in each involved array; all the DRAM cells in that row place their data on vertical wires called bitlines. At the bottom of the array are sense amplifiers that interpret the signals on the bitlines. It takes about 20 ns to bring a row of data into the sense amplifiers. Thus, a single cache line request may read out 1024-bit wide rows in (say) 64 different arrays. The sense-amplifiers therefore store as much as 64 Kb (8 KB) of data. This is referred to as the *row buffer*. The requested 64-byte cache line is then selected out of this 8 KB row buffer and sent back on the memory channel, with each array contributing eight bits in this example. A cache line request therefore leads to "overfetch" of data into the row buffer. The row buffer can serve as a cache inside the DRAM chips, storing 128 different cache lines in our example. If the next cache line request is already present in the row buffer, it can be serviced at a lower latency and energy cost. Each bank has a single row buffer that retains the last row that was read out of that bank.

Before a new row can be read out of a bank, the bitlines have to be precharged. This step, that readies the bank for the next access, also takes about 20 ns. As soon as the bitlines are precharged, the row buffer is lost. Note that the row buffer is simply the set of sense amplifiers connected to the bitlines.

This leads to three types of memory accesses. The most favorable case is a row buffer hit, where the requested cache line is already in the row buffer. Such an access has a latency of about 20 ns, which is the cost of shipping the cache line from the array to the DRAM chip output pins. The second case is an empty row access, where the bitlines have been precharged in advance and the row buffer stores nothing. Such an access takes 40 ns (20 ns for the row activation that populates the row buffer and 20 ns for the transfer to the output pins). The third case is a row buffer conflict, where a different row currently occupies the row buffer. This access takes about 60 ns – 20 ns to precharge the bitlines, 20 ns to read the new row, and 20 ns for data transfer to output pins. It is the responsibility of the memory controller to issue precharges at the right time so that row buffer hits and empty row accesses can be maximized.

An entire memory access can take about 100 ns (300 processor clock cycles at a 3 GHz frequency) – (say) 35 ns of queuing delay at the memory controller, 1 ns for address/command delays, 60 ns for a row buffer conflict access, and 4 ns for the transfer on the data bus.

5 Capacity vs. Energy

We've seen that a cache line request activates all the chips in the rank and many arrays within these chips. Each array is made large to reduce the overheads of the peripheral circuits for that array (decoders, sense amplifiers). This leads to wide rows and significant overfetch. This overfetch incurs a high energy penalty, but is done to maximize density and reduce cost-per-bit, which is a primary metric for DRAM chips.

If we use wide-output DRAM chips to construct a rank, say 4 x16 2 Gb chips, we can service a cache line by activating few chips, thus reducing overfetch and energy. But such a rank would only have a capacity of 8 Gb. On the other hand, if we used narrow-output DRAM chips, say 16 x4 2 Gb chips, we would increase activation energy and overfetch, but a single rank would now support 32 Gb capacity. These trade-offs must be considered when populating a memory system with the appropriate set of DIMMs.

6 Address Mapping

The memory controller can interpret addresses in different ways, each leading to a different placement of data in memory. The assumption is that applications will exhibit data locality and access consecutive cache lines within a short time window. To promote row buffer hits, consecutive cache lines can be placed in the same row. To promote parallelism, consecutive cache lines can be placed in different ranks and banks. For the first mapping policy, the data address can be interpreted as follows:

$$row : rank : bank : channel : column : blkoffset$$

The last few bits represent the data offset within the cache line. The next few bits represent different column bits within a row buffer. Consecutive cache lines will first differ in these column bits, thus representing different elements within a single fow buffer. The second mapping policy would interpre the data address as follows, where the last few fields represent different channels, banks, and ranks:

$$row : columnrank : bank : channel : blkoffset$$

7 Scheduling

The memory controller incorporates all the intelligence in the memory system. It is responsible for issuing commands in a manner that optimizes performance, energy, and fairness. The following considerations play a role in determining a good schedule.

First, as discussed above, the scheduler must try to keep a row open so it can service row buffer hits. This is referred to as an open-page policy and works well for applications that have high data locality. However, an open-page policy will eventually lead to an expensive row buffer conflict when it is time to access a new row. If an application has little locality, it is better to use a close-page policy that precharges the bitlines immediately after a cache line has been serviced. This eliminates row buffer hits and row buffer conflicts and leads to empty row accesses. Modern memory controllers use proprietary algorithms that implement a policy somewhere between an open- and close-page policy.

Second, the direction of the data bus must be switched every time we alternate between a read and a write. This is referred to as bus turnaround time (about 7.5 ns) and must be minimized. Hence, reads and writes are typically serviced in bursts. Reads are always prioritized and serviced first. Writes are not as critical (since a write to memory is generated when a dirty block is evicted from cache) and are placed in a write buffer. When the write buffer is close to filling up (reaches a high water mark), we switch the bus direction and start draining writes until it reaches a low water mark. At that point, the bus direction is switched again and we resume servicing reads.

Third, the scheduler must examine the queue of pending reads or pending writes every cycle to pick out an appropriate read or write that maximizes performance. Often, a FR-FCFS scheme is used that prioritizes potential row buffer hits over other requests, i.e., let's exploit the row buffer locality before the bank moves on to another row. Note that the memory controller is shared by several cores on a chip. Since FR-FCFS may end up prioritizing a thread with high row buffer locality, additional mechanisms may be required to enforce fairness.

Fourth, the scheduler must issue refresh operations at the right time. In the worst-case, a DRAM cell is expected to lose its charge within 64 ms. When a row is activated, the cells in that row are automatically refreshed. Since not all rows are activated within every 64 ms window, a background refresh process is required that sequentially steps through every row in the memory sytem and refreshes that row. The 64 ms window is partitioned into 8192 smaller windows, each lasting 7.8μs. In each 7.8μs window, the memory controller issues a refresh command that keeps the memory system busy for a few hundred cycles while it refreshes a few rows in every bank and every rank on the channel.