CSE 502:
Computer Architecture
Memory / DRAM
SRAM vs. DRAM

• SRAM = Static RAM
  – As long as power is present, data is retained

• DRAM = Dynamic RAM
  – If you don’t do anything, you lose the data

• SRAM: 6T per bit
  – built with normal high-speed CMOS technology

• DRAM: 1T per bit (+1 capacitor)
  – built with special DRAM process optimized for density
Hardware Structures

SRAM

DRAM

wordline

b

wordline

b

b
Implementing the Capacitor (1/2)

- You can use a “dead” transistor gate:

  But this wastes area because we now have two transistors

  And the “dummy” transistor may need to be bigger to hold enough charge
Implementing the Capacitor (2/2)

- There are other advanced structures

“Trench Cell”

- Cell Plate Si
- Cap Insulator
- Storage Node Poly
- Field Oxide
- Refilling Poly
- Si Substrate
DRAM Chip Organization (1/2)

DRAM is much denser than SRAM
DRAM Chip Organization (2/2)

• Low-Level organization is very similar to SRAM

• Cells are only single-ended
  – Reads **destructive**: contents are erased by reading

• **Row buffer** holds read data
  – Data in row buffer is called a **DRAM row**
    • Often called “page” - not necessarily same as OS page
  – Read gets entire row into the buffer
  – Block reads always performed out of the row buffer
    • Reading a whole row, but accessing one block
    • Similar to reading a cache line, but accessing one word
Destructive Read

After read of 0 or 1, cell contents close to $\frac{1}{2}$
DRAM Read

• After a read, the contents of the DRAM cell are gone
  – But still “safe” in the row buffer
• Write bits back before doing another read
• Reading into buffer is slow, but reading buffer is fast
  – Try reading multiple lines from buffer (*row-buffer hit*)

Process is called *opening* or *closing* a row
DRAM Refresh (1/2)

- Gradually, DRAM cell loses contents
  - Even if it’s not accessed
  - This is why it’s called “dynamic”

- DRAM must be regularly read and re-written
  - What to do if no read/write to row for long time?

Must periodically refresh all contents
DRAM Refresh (2/2)

• Burst Refresh
  – Stop the world, refresh all memory

• Distributed refresh
  – Space out refresh one row at a time
  – Avoids blocking memory for a long time

• Self-refresh (low-power mode)
  – Tell DRAM to refresh itself
  – Turn off memory controller
  – Takes some time to exit self-refresh
DRAM Organization

- All banks within the rank share all address and control pins.
- All banks are independent, but can only talk to one bank at a time.
- x8 means each DRAM outputs 8 bits, need 8 chips for DDRx (64-bit).
- Why 9 chips per rank? 64 bits data, 8 bits ECC.

Dual-rank x8 (2Rx8) DIMM
DRAM Read Timing

Original DRAM specified Row & Column every time
FPM enables multiple reads from page without RAS
SDRAM Read Timing

SDRAM uses clock, supports bursts

Double-Data Rate (DDR) DRAM transfers data on **both** rising and falling edge of the clock
Actual DRAM Signals

DDR3-1066 Write Data Eye (min SSO)

DDR3-1066 Write Data Eye (max SSO)
Distance matters, even at the speed of light
North Bridge can be Integrated onto CPU chip to reduce latency.
CPU-to-Memory Interconnect (2/3)

Discrete North and South Bridge chips
CPU-to-Memory Interconnect (3/3)
Memory Channels

- One controller, One 64-bit channel
- One controller, Two 64-bit channels
- Two controllers, Two 64-bit channels

Use multiple channels for more bandwidth
Memory-Level Parallelism (MLP)

• What if memory latency is 10000 cycles?
  – Runtime dominated by waiting for memory
  – What matters is overlapping memory accesses

• Memory-Level Parallelism (MLP):
  – “Average number of outstanding memory accesses when at least one memory access is outstanding.”

• MLP is a metric
  – *Not* a fundamental property of workload
  – Dependent on the microarchitecture
AMAT with MLP

• If ...
cache hit is 10 cycles (core to L1 and back)
memory access is 100 cycles (core to mem and back)

• Then ...
at 50% miss ratio, avg. access: $0.5 \times 10 + 0.5 \times 100 = 55$

• Unless MLP is > 1.0, then...
at 50% mr, 1.5 MLP, avg. access: $(0.5 \times 10 + 0.5 \times 100)/1.5 = 37$
at 50% mr, 4.0 MLP, avg. access: $(0.5 \times 10 + 0.5 \times 100)/4.0 = 14$

In many cases, MLP dictates performance
Memory Controller (1/2)

Scheduler

Buffer

Read Queue

Write Queue

Response Queue

Commands

Data

To/From CPU

Channel 0

Channel 1
Memory Controller (2/2)

• Memory controller connects CPU and DRAM
• Receives requests after cache misses in LLC
  – Possibly originating from multiple cores
• Complicated piece of hardware, handles:
  – DRAM Refresh
  – Row-Buffer Management Policies
  – Address Mapping Schemes
  – Request Scheduling
Row-Buffer Management Policies

• **Open-page**
  – After access, keep page in DRAM row buffer
  – Next access to same page $\rightarrow$ lower latency
  – If access to different page, must close old one first
    • Good if lots of locality

• **Close-page**
  – After access, immediately close page in DRAM row buffer
  – Next access to different page $\rightarrow$ lower latency
  – If access to different page, old one already closed
    • Good if no locality (random access)
Address Mapping Schemes (1/3)

• Map consecutive addresses to improve performance

• Multiple independent channels $\rightarrow$ max parallelism
  – Map consecutive cache lines to different channels

• Multiple channels/ranks/banks $\rightarrow$ OK parallelism
  – Limited by shared address and/or data pins
  – Map close cache lines to banks within same rank
    • Reads from same rank are faster than from different ranks
  – Accessing rows from one bank is slowest
    • All requests serialized, regardless of row-buffer mgmt. policies
    • Rows mapped to same bank should avoid spatial locality
  – Column mapping depends on row-buffer mgmt. (Why?)
### Address Mapping Schemes (2/3)

#### bank column

<table>
<thead>
<tr>
<th>Address</th>
<th>Address</th>
<th>Address</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>0x00400</td>
<td>0x00800</td>
<td>0x00C00</td>
</tr>
<tr>
<td>0x00100</td>
<td>0x00500</td>
<td>0x00900</td>
<td>0x00D00</td>
</tr>
<tr>
<td>0x00200</td>
<td>0x00600</td>
<td>0x00A00</td>
<td>0x00E00</td>
</tr>
<tr>
<td>0x00300</td>
<td>0x00700</td>
<td>0x00B00</td>
<td>0x00F00</td>
</tr>
</tbody>
</table>

#### column bank

<table>
<thead>
<tr>
<th>Address</th>
<th>Address</th>
<th>Address</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000</td>
<td>0x00100</td>
<td>0x00200</td>
<td>0x00300</td>
</tr>
<tr>
<td>0x00400</td>
<td>0x00500</td>
<td>0x00600</td>
<td>0x00700</td>
</tr>
<tr>
<td>0x00800</td>
<td>0x00900</td>
<td>0x00A00</td>
<td>0x00B00</td>
</tr>
<tr>
<td>0x00C00</td>
<td>0x00D00</td>
<td>0x00E00</td>
<td>0x00F00</td>
</tr>
</tbody>
</table>
Address Mapping Schemes (3/3)

• Example Open-page Mapping Scheme:
  
  *High Parallelism*: [row rank bank column channel offset]
  
  *Easy Expandability*: [channel rank row bank column offset]

• Example Close-page Mapping Scheme:

  *High Parallelism*: [row column rank bank channel offset]
  
  *Easy Expandability*: [channel rank row column bank offset]
Request Scheduling (1/3)

• Write buffering
  – Writes can wait until reads are done

• Queue DRAM commands
  – Usually into per-bank queues
  – Allows easily reordering ops. meant for same bank

• Common policies:
  – First-Come-First-Served (FCFS)
  – First-Ready—First-Come-First-Served (FR-FCFS)
Request Scheduling (2/3)

• First-Come-First-Served
  – Oldest request first

• First-Ready—First-Come-First-Served
  – Prioritize column changes over row changes
  – Skip over older conflicting requests
  – Find row hits (on queued reqs., even if close-page policy)
  – Find oldest
    • If no conflicts with in-progress request → good
    • Otherwise (if conflicts), try next oldest
Request Scheduling (3/3)

• Why is it hard?

• Tons of timing constraints in DRAM
  – $t_{WTR}$: Min. cycles before read after a write
  – $t_{RC}$: Min. cycles between consecutive open in bank
  – ...

• Simultaneously track resources to prevent conflicts
  – Channels, banks, ranks, data bus, address bus, row buffers
  – Do it for many queued requests at the same time
    ... while not forgetting to do refresh
Overcoming Memory Latency

• Caching
  – Reduce average latency by avoiding DRAM altogether
  – Limitations
    • Capacity (programs keep increasing in size)
    • Compulsory misses

• Prefetching
  – Guess what will be accessed next
    • Put in into the cache