Main Memory and DRAM

Instructor: Nima Honarmand
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

Trench Capacitor
DRAM Chip Organization (1/2)

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

• Low-Level organization is very similar to SRAM

• Reads are **destructive**: contents are erased by reading

• **Row buffer** holds read data
  – Data in row buffer is called a **DRAM row**
    • Often called “page” – do not confuse with virtual memory 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 (or a few) row(s) 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
Typical DRAM Access Sequence (1/5)

[PRECHARGE and] ROW ACCESS

AKA: OPEN a DRAM Page/Row
or
ACT (Activate a DRAM Page/Row)
or
RAS (Row Address Strobe)
Typical DRAM Access Sequence (2/5)

COLUMN ACCESS

READ Command or CAS: Column Address Strobe
Typical DRAM Access Sequence (3/5)

DATA TRANSFER

CPU

MEMORY CONTROLLER

BUS

Data Out

Data In/Out Buffers

Column Decoder

Sense Amps

... Bit Lines...

Row Decoder

Memory Array

... with optional additional
CAS: Column Address Strobe
Typical DRAM Access Sequence (4/5)
Typical DRAM Access Sequence (5/5)

A: Transaction request may be delayed in Queue
B: Transaction request sent to Memory Controller
C: Transaction converted to Command Sequences
   (may be queued)
D: Command/s Sent to DRAM
E₁: Requires only a CAS or
E₂: Requires RAS + CAS or
E₃: Requires PRE + RAS + CAS
F: Transaction sent back to CPU

“DRAM Latency” = A + B + C + D + E + F
(Old) DRAM Read Timing

Original DRAM specified Row & Column every time
(Old) DRAM Read Timing w/ Fast-Page Mode

FPM enables multiple reads from page without RAS
(Newer) SDRAM Read Timing

SDRAM uses clock, supports bursts.
Banking to Improve BW

- DRAM access takes multiple cycles
- What is the miss penalty for a 4-word cache block
  - Consider these parameters:
    - 1 cycle to send address
    - 6 cycles to access each word
    - 1 cycle to send word back
  - $(1 + 6 + 1) \times 4 = 32$
- How can we speed this up?
- Make memory and bus wider
  - read out all words in parallel
- Miss penalty for 4-word block
  - $1 + 6 + 1 = 8$
- Cost
  - wider bus
  - larger expansion size
Simple Interleaved Main Memory

- Divide memory into \( n \) banks, “interleave” addresses across them so that word A is
  - in bank \((A \mod n)\)
  - at word \((A \div n)\)

<table>
<thead>
<tr>
<th>Bank 0</th>
<th>Bank 1</th>
<th>Bank 2</th>
<th>Bank n</th>
</tr>
</thead>
<tbody>
<tr>
<td>word 0</td>
<td>word 1</td>
<td>word 2</td>
<td>word n-1</td>
</tr>
<tr>
<td>word n</td>
<td>word n+1</td>
<td>word n+2</td>
<td>word 2n-1</td>
</tr>
<tr>
<td>word 2n</td>
<td>word 2n+1</td>
<td>word 2n+2</td>
<td>word 3n-1</td>
</tr>
</tbody>
</table>

- Can access one bank while another one is busy
  → Interleaving increases memory bandwidth w/o a wider bus

Use parallelism in memory banks to hide memory latency
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
SDRAM Topology

“Mesh Topology”
CPU-to-Memory Interconnect (1/3)

North Bridge can be Integrated onto CPU chip to reduce latency

Figure from ArsTechnica
CPU-to-Memory Interconnect (2/3)

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

Integrated North Bridge
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: $\text{AMAT} = 0.5 \times 10 + 0.5 \times 100 = 55$

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

In many cases, MLP dictates performance
Memory Controller (1/2)
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
Request Scheduling (1/3)

• Write buffering  
  – Writes can wait until reads are done

• Controller queues 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 requests)
    • 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
  – tWTR: Min. cycles before read after a write
  – tRC: 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
Row-Buffer Management Policies

• **Open-page Policy**
  – 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 Policy**
  – 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)

• Question: How to map a physical addr to channel ID, rank ID, bank ID, row ID, and column ID?
  – Goal: efficiently exploit channel/rank/bank level parallelism

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

• Single channel, Multiple 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 different 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>0x00000</th>
<th>0x00400</th>
<th>0x00800</th>
<th>0x00C00</th>
</tr>
</thead>
<tbody>
<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>0x00000</th>
<th>0x00100</th>
<th>0x00200</th>
<th>0x00300</th>
</tr>
</thead>
<tbody>
<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]
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