DRAM Memory Controllers

Reference: “Memory Systems: Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, & David Wang
Today's material & any uncredited diagram came from Chapter 13

Basic Requirements

• Manage data movement to/from DRAM
  • device level
    » electrical & timing restrictions
    » error correction
      » typical parity just means retry and flag
  • system level
    » arbitration fairness
      » will be necessary in multiple core/mem_ctlr configurations
    » maximize system performance
      » command scheduling
      » multiple conflicting performance metrics however
        » heat, power consumption, latency, bandwidth

• Lots of options increase complexity
  • variety of timing parameters & command sequences
    » specific to the target device
  • scheduling for some optimality target
Top-Level View

• 3 top-level policy/strategies
  • row buffer management policy
  • address mapping scheme
  • memory transaction and command ordering strategy

• Large body of research
  • partially due to huge timing differences
    » processors get faster & DRAM is fairly flat
  • seems to be reported primarily by the circuit community
    » according to recent look by Dave and Manu
      • ISPLED – Int. Symp. on Low Power Electronics and Design
    » and a bunch of reference cores put out by industry
    » main game played by northbridge chipset vendors

For Starters

• pubs
  » bank address mapping
    • Lin et al. “Reducing DRAM latencies ...” HPCA 2001
    • Zhang et al. “Breaking address mapping symmetry ...” JILP 2002
  » command ordering schemes
    • Hur & Lin “Adaptive history-based memory schedulers” MICRO04
    • Rixner “Memory controller optimizations for web servers” MICRO04
    • Rixner et al “Memory access scheduling ...” ISCA 2000.

• today
  » more general discussion of the issues
Basic MC Components

• Note
  • as memory access cost increases w.r.t. compute on CPU's
    » combining transaction and command scheduling is important
  • address translation targets rank and bank
    » transaction turned into a series of DRAM commands
      • optimization options occur with interleaved transactions
        – while still respecting device timing restrictions

Row Buffer Management

• Open-Page
  • good
    » both temporal and spatial locality exist in access pattern
      • spatial: amortizes large row activate energy cost
      • temporal: energy to keep row open results in improved bandwidth
        - latency limited by $t_{CAS}$ only
  • bad
    » energy: delay to same row access is infrequent
    » time: precharge, activate, access if target row is inactive
      • better to perform a col-rd-precharge command when new row is known
  • scheduling issues
    » similar to dynamic instruction issue
      • performance increases with a larger window
        - except when window is always slightly filled
        - multi-core/MC changes the probability
      • dependent and anti-dependent issues must be tracked
        - note write buffer in XDR (sound familiar?)
Closed Page

• Favors random access patterns
  • more likely
    » large processor count & large main memory capacity
      • e.g. database in DRAM datacenter
      • many-core devices with multiple MC's
      • highly threaded workloads break the temporal locality target
    » embedded systems
      • DRAM access is rare
      • energy cost of keeping row open breaks the energy/thermal threshold
  • less likely
    » if large number of banks are kept open
      • e.g. Direct RDRAM – 32 ranks x 2 banks/rank per channel
        • hence choice for the EV7
        • which didn't make it commercially for different reasons
      • each thread/core tends to hit the same bank
      • AND energy/thermal limits aren't surpassed

Hybrid Row Buffer Management

• Reality – closed vs. open choice isn't static
  • best choice depends
    » access pattern and rate
      • ratio of \( t_{RP} - t_{RC} - t_{RP} \)
        • row precharge interval, row cmd to data ready at sense amps delay
        • move to close page if falls below some (possibly dynamic) threshold
      • enter history tables and timers
        • timer can control the sense amps keeping the page "open"
        • wait too long and precharge since temporal locality has failed
  • one choice doesn't fit all
    » rank, bank, and channel patterns may vary
    » typical balance point argument
      • increased MC complexity & cost for how much gain
        • gain is metric specific: power, effective bandwidth, latency, ...
      • plus complicated decision process may slow DRAM command issue
        • since DRAM's are slow this has been less of a constraint
Rbuff Mgmt: Performance Impact

• Proper Approach
  • Includes
    » in depth analysis of queuing delays
    » simulation of the memory controller
      • using a variety of real and synthetic workloads
    » various scheduling approaches
      • might also include thread phase prediction
    » incorporation of thermal management issues
      • not directly a performance thing but can’t be ignored
    » refresh & associated resource availability issues

Rbuff Mgmt: 1st order approximation

• Less sim based & more analytical
  • using timing parameters (see last lecture)
    » normally idle ⇒ close page approach
      • read latency is \( t_{RCD} + t_{CAS} \)
    » open page read latency
      • min: \( t_{CAS} \) for access to an active row
      • max: \( t_{RP} + t_{RCD} + t_{CAS} \) for bank conflict
      • if \( x \% \) of accesses hit an open row
        - average read latency = \( x \cdot t_{CAS} + (1-x) \cdot (t_{RP} + t_{RCD} + t_{CAS}) \)
        - crossover for open vs. close page
          - \( t_{RCD} + t_{CAS} = x \cdot t_{CAS} + (1-x) \cdot (t_{RP} + t_{RCD} + t_{CAS}) \)
          - \( x = t_{RP} / (t_{RP} + t_{RCD}) \)
        - SURPRISE!!
          - for Micron DDR2 UDIMMs (U for Unbuffered)
          - \( t_{RP} = t_{RCD} \) = see next slide for values
          - therefore break even point is \( x = 50\% \)
    » argues against trying too hard to hot row schedule
**Micron Data Sheet Excerpts**

512MB, 1GB, 2GB, 4GB (x64, DR) 240-Pin DDR2 SDRAM UDIMM

**Features**

**DDR2 SDRAM UDIMM**

MT16HTF6464A – 512MB
MT16HTF12864A – 1GB
MT16HTF25664A – 2GB
MT16HTF51264A – 4GB

For component data sheets, refer to Micron’s Web site: www.micron.com

---

**Table 1: Key Timing Parameters**

<table>
<thead>
<tr>
<th>Speed Grade</th>
<th>Industry Nomenclature</th>
<th>Data Rate (MT/s)</th>
<th>tCCD (ns)</th>
<th>tRP (ns)</th>
<th>tRC (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10A</td>
<td>PC2-2500</td>
<td>CL = 7</td>
<td>1066</td>
<td>800</td>
<td>537</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 6</td>
<td></td>
<td>800</td>
<td>532</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CL = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-80E</td>
<td>PC2-6400</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-600</td>
<td>PC2-6400</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-40E</td>
<td>PC2-6400</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

---

**Rbuff Mgmt: Power Impact**

- **Performance isn’t everything**
  - power is topping the charts these days
  - consider a RDRAM system
    - 16 x 256 Mbit Direct RDRAM devices
      - 3 modes
        - active (all banks active)
        - standby (active but takes longer to bring back to active and then read)
        - NAP (inactive banks so row access must be redone)

<table>
<thead>
<tr>
<th>Condition</th>
<th>Current mA</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 device read active, 15 in NAP</td>
<td>1195</td>
<td>1</td>
</tr>
<tr>
<td>1 device read active, 15 in standby</td>
<td>2548</td>
<td>2.1</td>
</tr>
<tr>
<td>1 device read active, 15 also active</td>
<td>3206</td>
<td>2.7</td>
</tr>
</tbody>
</table>

Missing: cost to reactivate a row but close page appeal is clear
Address Mapping

- Main memory so all addresses are physical
  - but how do they map to channel, rank, bank, row, & col ID’s
- general goal: performance
  » map adjacent requests to maximize command parallelism
    - channels are parallel
    - ranks require $t_{	ext{BST}}$ switching time but are otherwise parallel
    - decent overlap to different active rows in different banks
  » key is to avoid bank conflict
    - which is the biggest sequential penalty: $t_{	ext{RP}} + t_{	ext{RCD}} + t_{	ext{CAS}}$
- unlike row buffer management
  » address mapping can’t be dynamically changed
  » physical address to dram channel, rank, bank, row, col is fixed
    - simple swizzle of the $P_{\text{adr}}$ bits
  » virtual to physical address still done by the TLB
    - but OS manages TLB
    - ?? any leverage to be had here – not clear ??
- power goal
  » different options

Alternative Viewpoints

- Impulse (Utah)
  » use an extra level of indirection to support multiple strides
    » get the cache line you want
      - not just the contiguous block that you usually get
    » use shadow memory (not in the physical address map)
      - index of actual targets for user defined access patterns
      - this “map” changes based on strides in play
  » memory controller
    - controls map to minimize bank conflict
- FB-Dimm
  » on DIMM ASIC could be impulse like
    » each DIMM is a channel
    » Impulse like game could be played
      - albeit with a bit more control logic in the AMB chip
**Address Mapping Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Variable Dependence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Independent</td>
<td># of channels in system</td>
</tr>
<tr>
<td>L</td>
<td>Independent</td>
<td># of ranks per channel</td>
</tr>
<tr>
<td>B</td>
<td>Independent</td>
<td># of banks per rank</td>
</tr>
<tr>
<td>R</td>
<td>Independent</td>
<td># of rows per bank</td>
</tr>
<tr>
<td>C</td>
<td>Independent</td>
<td># of columns per row</td>
</tr>
<tr>
<td>V</td>
<td>Independent</td>
<td># of bytes per column</td>
</tr>
<tr>
<td>Z</td>
<td>Independent</td>
<td># of bytes per cache line</td>
</tr>
<tr>
<td>N</td>
<td>Dependent</td>
<td># of cache lines per row</td>
</tr>
</tbody>
</table>

Total Memory Capacity = K*L*B*R*C*V

\[ N = \frac{CV}{Z} \] & \[ CV = NZ \] (since we care about cache lines)

Since we're whacked on powers of 2 let:

- \[ L = 2^l \]
- \[ B = 2^b \]
- \[ etc. for simplicity \]

non powers of 2 could be used but it wastes address bits

---

**Swizzling \( k+l+b+r+c+v \) Address Bits**

- **Baseline mapping**
  - **open page** – performance goal assuming locality
    - stripe adjacent cache lines across different channels
      - then map to same row, bank, and rank
        - avoids \( t_{lat} \) & bank conflict for as long as possible
    - **address bits partitioned**
      - \( rlbmbkxz \) (high order addr. bits spec. row ID \( \rightarrow \) avoid bank conflict)
      - note \( z \) only needs to be used in critical word first return systems
        - initialized burst size feature removes need for \( z \)
  - **close page**
    - stripe adjacent cache lines across channels
      - same as open page BUT pipeline delays due to bank close
        - prefer to then stripe over banks, then ranks
    - **address partition**
      - \( rnlbikxz \) (\( n \) in 2\textsuperscript{nd} high order spot avoids delay with row precharge to next cache line)
Expansion Capability

• Many systems allow user to buy more memory
  • for most systems this means more ranks
    » e.g. box comes with 1 2-rank DIMM
    • add another 2-rank DIMM
  » hence l is mapped to the high order addresses
  • expansion comes at the cost of rank parallelism
    » when application uses a subset of the available ranks
    » problem self-mitigates as $t_{OP}$ goes up
• other cases – multiple channels can be independently configured
  » a.k.a. “asymmetric channels”
  » now channel (k) bits become high order as well
  • reduced channel parallelism results

• new baselines
  » expandable open page: $kbrnbnkz$ rather than $ribnknkz$
  » expandable close page: $kbrnbnkz$ rather than $nndbknkz$

Example: Intel 82955X MCH

• MCH = memory control hub
  • 2 memory controllers
    » each independently control 2 DDR2 channels
    • each channel supports up to 4 ranks
  • possible rank configurations

<table>
<thead>
<tr>
<th>Rank Cap. MB</th>
<th>Config banks, rows, cols, colsize</th>
<th>Rank dev.cap x dev.cnt</th>
<th>Rank config BxRxCxV</th>
<th>Bank Addr bits b</th>
<th>Row Addr bits r</th>
<th>Col Addr bits c</th>
<th>Col Addr Offset v</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>4x8192x512x2</td>
<td>256 Mb x 4</td>
<td>4x8192x51</td>
<td>2</td>
<td>13</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>256</td>
<td>4x8192x1024x2</td>
<td>512 Mb x 4</td>
<td>4x8192x1024x8</td>
<td>2</td>
<td>13</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>256</td>
<td>4x8192x1024x1</td>
<td>256 Mb x 8</td>
<td>4x8192x1024x8</td>
<td>2</td>
<td>13</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>512</td>
<td>8x8192x1024x2</td>
<td>1 Gb x 4</td>
<td>8x8192x1024x8</td>
<td>2</td>
<td>13</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>512</td>
<td>4x16384x1024x1</td>
<td>512 Mb x 4</td>
<td>4x16384x1024x8</td>
<td>2</td>
<td>14</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
MCH Options

- 1 or 2 DIMMs per channel
  - If 2: ranks must be identically configured
- address mapping supports open page system
  - but with some flexibility
    » to account for configuration
    » support for symmetric or asymmetric channels
      - uses rank_config_registers
        - support rank by rank address mapping
- symmetric
  » consecutive $-lines map to alternating channels (k)
- asymmetric
  » channel capacities vary
    - phys addr maps 0:CHNL0cap and then to CHNL1cap
      - keeps access in single channel unless request size spans both

MCH Overview

Note: capacity the same even though rank config is different – still works for MCH symmetric mode

Note: channel cap and rank count differ so must go asymmetric
MCH Register Usage

- Set at system initialization time
  - Individual mapping by rank
    » addr. mapping regs
      » contain capacity and organization parameters of the DRAM devices
    » rank addr. boundary regs
      » resolve a physical address to a rank
    » rank architecture regs
      » org of the devices in each rank
      » disambiguates bank, row, and column addresses
  - note this does not include the channel address
    » mapped separately
      » depends on sym. vs. asym. mode
      » use channel boundary regs to get to proper controller

Per-rank Mapping

Per channel, per-rank address mapping scheme for single/asymmetric channel mode

<table>
<thead>
<tr>
<th>Rank</th>
<th>Config</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>8x32</td>
<td>00 00 00 00 00 00</td>
</tr>
<tr>
<td>256</td>
<td>8x32</td>
<td>80 80 00 00 00 00</td>
</tr>
<tr>
<td>512</td>
<td>8x32</td>
<td>80 80 00 00 00 00</td>
</tr>
<tr>
<td>1024</td>
<td>8x32</td>
<td>80 80 00 00 00 00</td>
</tr>
</tbody>
</table>

- single/asymmetric channel
- symmetric/dual channel
- row ID
- bank ID
- col ID
- byte offset per DIMM

Per-rank address mapping scheme for dual channel symmetric mode

- k:l:r:b:n:z
- a bit different
- l:r:b:n:k:z
- due to cache line interleaving
Bank Address Aliasing

- Problem
  - 2 large power of 2 arrays accessed concurrently
    - target is actually the same bank so bank conflict
    - matrix multiply of 2 $2^{17}$B arrays in MCH would conflict
      - since bank addr is paddr[14:16]

- Solutions (no works always solution however)
  - SW: use some hash function
  - HW: Lin 2001 & Zhang 2000 had a similar idea

```
<table>
<thead>
<tr>
<th>page index</th>
<th>bank index</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>p</td>
<td></td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>page index</th>
<th>new bank index</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Write Caching

- Usual benefit
  - writes are not typically critical
    - defer if it helps the schedule
  - still have to check the cache on a RAW access pattern
    - adds some complexity
    - also delays read if conservative
      - e.g. check cache and then go to DRAM
      - wise choice in power constrained environments
        - go eager otherwise

- DRAM specific benefit
  - high speed buses take time to turn around
    - bigger issue in DDRx where $x \geq 2$ land
    - hence RWRWRW... transactions are slow

- In use
  - common in XDR based RDRAM systems
  - Intel i8870 controller does it for JEDEC systems
Request Queues

• MC translates accesses to memory commands
  • tries for optimal schedule as well
• Priority is important but based on what?
  • request priority
  • current resource utilization
  • bank address
  • spread the load or heat
  • etc.
• Common to use request queues per bank
  • round robin over banks
  • reorder commands within the queue
    » more on this shortly
  • useful for high memory pressure systems
    » extra complexity for little gain in low pressure situations

Refresh Management

• Mentioned previously to some extent
• Simple MC takes on the job
  • refresh one rank at a time
    » all banks all rows
  • keep row-interval registers
    » ignore refresh when certain intervals don’t have valid data
• Self-refresh
  • capability exists in certain devices
    » each device self-refreshes based on a timer
    » MC can be put to sleep in low pressure scenarios
  • temperature compensated refresh counters exist
    » MobileRAM is one example
Agent Centric Q’ing

- Key schedule policy is fairness
  - usually interpreted as starvation free
  - agents have different priorities
    - I/O, CPU, GPU
    - read, write, refresh
    - latency vs. bandwidth needs

Feedback Directed Scheduling

- Similar to branch prediction idea
  - let history predict the future
  - approach hasn’t been as thoroughly explored for DRAM however
    - as DRAM becomes the bottleneck this will get some attention
      - see Hur & Lin Micro04
    - as memory controllers move onto the CPU
      - history state is cheaper to export to the MC
      - as is agent ID (thread, laddr, ...?)