Memory Controller Basics and Workload Characteristics

Credit where credit is due: Slides contain original artwork (© Jacob, Wang 2005)
Seen This Before

1. L1 Access Miss
2. L2 Request
3. L2 Tag
4. Addr to NB
5. Clock Boundary
6. SRQ
7. GART/Addr Decode
8. Crossbar
9. Coh./Order Check
10. Memory Controller
11. Req to DRAM Pins
12. ... DRAM Access
13. Data to MCT
14. NB Route
15. Clock Boundary
16. CPU route/mux/ECC
17. Write L1 D$ & FWD

NB = northbridge

SLIDE 2
Reads to Diff. Rows of Same Bank

Best Case: $t_{RAS}$ already satisfied

Worst Case: $t_{RAS}$ not yet satisfied

Smart controller can extract maximum performance (lowest latency + max BW)
Maybe Easy is Better?

Best Case: $t_{RAS}$ already satisfied

Worst Case: $t_{RAS}$ not yet satisfied

Always assume worst case.
Simple design = Small controller
Poor performance? Does it matter?
What is the goal?

- Low Power
- Bandwidth
- Latency
- Low die cost

Or, little of everything?
What’s Important?

- Row Buffer Management Policy
- Address Mapping Scheme
- System Configuration
- Workload Characteristics
- VA to PA translation?
- Command Re-ordering?
- Transaction Re-ordering?
Row Buffer Mgmt: Open Page

Open Page: Keep data at sense amps

Good: If request stream has good locality
Good: If request rate is fairly high, but not too high*

Bad: Harder to schedule. More corner cases to deal with
Bad: If access rate is too low, keeping data at sense amps eat power
Row Buffer Mgmt: Close Page

Close Page: One CAS per RAS

Good: If request stream has no locality
Good: If request rate is very high
Good: Easier to schedule (deep pipeline)

Bad: One CAS per RAS means high power consumption ($t_{RRD}$ & $t_{FAW}$)
Bad: Nightmare scenario if all accesses are to same bank
Row Buffer Mgmt: Dynamic Page

“Dynamic” Page: many CAS per RAS

Typically based on open page mechanism

Keep timer: If timer expires, close page

Keep history: If many conflicts occur, switch to close page policy.
Address Mapping

Directs requests to different locations in DRAM Memory System.
Address Mapping: Open Page

Keep cacheline $i$ and cacheline $i + 1$ to same bank

Using 4 ranks of 8 x8 1 Gbit DDR2 SDRAM Devices
Address Mapping: Close Page

32 bit physical address (byte addressable)

Cacheline i and cacheline i +1 are sent to different banks

Why is rank id mapped to high addr range?
Address Mapping: Expandability I

Different organizations, density, Variable number of ranks (use address range register)
Address Mapping: Expandability II

Expandable

Move rank id lower, get more parallelism, but requires matching ranks (Intel 875P - “Dynamic Addressing”)
Bank Address Aliasing

32 bit physical address (byte addressable)

C[i] = A[i] + B[i]

Suppose that all arrays are size $2^k$; $K > 16$

load C[0x0E1C0000];
load A[0x0E3C0000]; Bank Conflict
load B[0x0E5C0000];

Or is there?
TLB VA to PA Translation

32 bit physical address (byte addressable)

31 30 29 16 15 13 12 3 2 0

rank id row id bank id column id not used

4 KB Page boundary

If VA to PA translation is random

load C[0x0F1E2000];
load A[0x1428B000];
load B[0x2AC23000];

Random chance for bank conflict
**Software Solution (Offset insertion)**

```c
double A[SIZE + OFFSET];
double B[SIZE + OFFSET];
double C[SIZE + OFFSET];
```

All array elements offset by OFFSET

```c
load C[0x0E1C0000]; load A[0x0E3C2000]; load B[0x0E5C4000];
```

No bank conflict as array marches

<table>
<thead>
<tr>
<th></th>
<th>Copy</th>
<th>Scale</th>
<th>Add</th>
<th>Triad</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Offset (MB/s)</td>
<td>2331</td>
<td>2473</td>
<td>2584</td>
<td>2496</td>
</tr>
<tr>
<td>With Offset (MB/s)</td>
<td>2448</td>
<td>2474</td>
<td>3164</td>
<td>3157</td>
</tr>
<tr>
<td>Difference (MB/s)</td>
<td>117</td>
<td>1</td>
<td>580</td>
<td>661</td>
</tr>
</tbody>
</table>

Measured on Dell Power Edge 400SC
Hardware Solution

Permute physical addr before mapping to memory addr
Retain 1:1 mapping characteristic, reduce bank conflicts
during array marches


Command Re-ordering

No Re-ordering : Easy to do

With Re-ordering: Gets harder, look to schedule opportunistically
Transaction Re-ordering

How many requests are in flight? (memory level parallelism)
How deep are the queues?
Access rate? Pattern? Locality? Type?
Bus Trace Viewer (BTV)

Initialization

One execution loop

164.gzip

Written by yours truly to look at traces.
176.gcc

bursty, lots of IFetches (L2$ = 256$KB)
197.parser

zoomed in
Initialization

Asymmetric Execution Loops
Quake 3 (random segment 0)

one frame
SETI@HOME

System Context Switch: Every 10 ms.
Recall

- 3D gfx processor
- System Controller (North Bridge)
- AGP
- Graphics memory
- Z-buffer
- Texture
- Harddisc
- Keyboard
- Mouse
- Ethernet card
- Ethernet packet
- Multi megabyte texture
- Collision detection/geometry information
- Game AI
- OS/drivers/etc.
- RAM

Diagram showing the memory systems architecture and performance analysis.
What about MP/CMP/MT?

- 3D gfx processor
- AGP
- System Controller (North Bridge)
- CPU

- Graphics memory
- Z-buffer
- Texture

- Harddisc

- I/O Controller (Southbridge)

- Keyboard
- Mouse

- Ethernet card

- RAM

- OS/drivers/etc.
- Game AI
- Collision detection/geometry information
- Multi megabyte texture
- Ethernet packet
- Ethernet packet