Classic Memory System Revisited

Andreas Hansson
ARM Research

gem5 User Workshop 2015
What are we modeling?

Source: ARM
Key changes and additions

- DRAM controller refinements
  - New DRAM features, power modeling
- Crossbar extensions
  - Interleaving and hashing
- Snoop filter addition
  - Steering snoops, tracking evictions
- Correctness checking
  - Memory-model checker and soak tests
- Performance tuning
  - Transaction support, cache latencies
What are we modeling?

Source: ARM
DRAM evolution

Source: Samsung
Same same…but different

### LPDDR3 & LPDDR3E
- **Die Organization**: 1ch X 8 banks X 32 IO
- **Channel #**: 1
- **Bank #**: 8
- **Density**: 4Gb – 32Gb
- **Page Size**: 4KByte
- **Max BW per die**: 6.4GB/s, 8.5GB/s (overclocking)
- **Max IO Speed**: 2133Mbps
- **Signal Pin #**: 62 per die
- **Package**: POP, MCP

### LPDDR4
- **Die Organization**: 2ch X 8 banks X 16 IO
- **Channel #**: 2
- **Bank #**: 8 per channel (16 per die)
- **Density**: 4Gb – 32Gb
- **Page Size**: 2KByte
- **Max BW per die**: 12.8GB/s, 17GB/s (overclocking)
- **Max IO Speed**: 4266Mbps
- **Signal Pin #**: 66 per die
- **Package**: POP, MCP

### Wide IO2
- **Die Organization**: 4ch X 8 banks X 64 IO
- **Channel #**: 4 & 8
- **Bank #**: 32 per die
- **Density**: 8Gb – 32Gb
- **Page Size**: 4KByte (4ch die), 2KB (8ch die)
- **Max BW per die**: 25.6GB/s & 51.2GB/s
- **Max IO Speed**: 1066Mbps
- **Signal Pin #**: ~430 per die (4ch die), ~850 per die (8ch die)
- **Package**: KGD

**Source**: Qualcomm
Top-down controller model

- Don’t model the actual DRAM, only the timing constraints
  - DDR3/4, LPDDR2/3/4, WIO1/2, GDDR5, HBM, HMC, even PCM
  - See `src/mem/DRAMCtrl.py` and `src/mem/dram_ctrl.{hh, cc}`

Hansson et al, *Simulating DRAM controllers for future system architecture exploration*, ISPASS’14
Controller model correlation

- Comparing with a real memory controller
  - Synthetic traffic sweeping bytes per activate and number of banks
  - See `configs/dram/sweep.py` and `util/dram_sweep_plot.py`

```
<table>
<thead>
<tr>
<th>Bytes per Activate</th>
<th>0-20</th>
<th>20-40</th>
<th>40-60</th>
<th>60-80</th>
<th>80-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Banks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>0-20</td>
</tr>
</tbody>
</table>
```

π Numbers for the above diagram are placeholders.

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```

```
Number of Banks: 1 to 256
Bytes per Activate: 0-20, 20-40, 40-60, 60-80, 80-100
```
DRAM power modeling

- DRAM accounts for a large portion of system power
  - Need to capture power states, and system impact
- Integrated model opens up for developing more clever strategies
  - DRAMPower adapted and adopted for gem5 use-case

**Energy Saving due to Power-Down (%)**

- **bbench**
- **AndeBench**

**BBench DRAM Energy Analysis (LPDDR3 x32)**

- Active Energy: 36%
- Precharge Energy: 64%
- Read/Write Energy
- Background Energy
- Refresh Energy

Naji et al, *A High-Level DRAM Timing, Power and Area Exploration Tool*, SAMOS’15
What are we modeling?

Source: ARM
Key changes and additions

- DRAM controller refinements
  - New DRAM features, power modeling
- Crossbar extensions
  - Interleaving and hashing
- Snoop filter addition
  - Steering snoops, tracking evictions
- Correctness checking
  - Memory-model checker and soak tests
- Performance tuning
  - Transaction support, cache latencies
Multi-channel memory support is essential
- Emerging DRAM standards are multi-channel by nature (LPDDR4, WIO1/2, HBM1/2, HMC)

Interleaving support added to address range
- Understood by memory controller and interconnect
- See `src/base/addr_range.hh` for matching and `src/mem/xbar.{hh, cc}` for actual usage
- Interleaving not visible in checkpoints

XOR-based hashing to avoid imbalances
- Simple yet effective, and widely published
- See `configs/common/MemConfig.py` for system configuration
With a bit of creativity…

- Hybrid Memory Cube (HMC) vaults
  - 32 channels of DRAM
  - HMC DRAM configuration

- HMC base layer
  - 4 non-coherent crossbars
  - HMC interleaving configuration

- HMC links
  - Bridges or custom link classes
  - Link interleaving on the host side

- …only using what is already part of gem5
What are we modeling?

Source: ARM
Key changes and additions

- DRAM controller refinements
  - New DRAM features, power modeling
- Crossbar extensions
  - Interleaving and hashing
- Snoop filter addition
  - Tracking evictions, steering snoops
- Correctness checking
  - Memory-model checker and soak tests
- Performance tuning
  - Transaction support, cache latencies
Snoop (probe) filtering

- Broadcast-based coherence protocol
  - Incurs performance and power cost
  - Does not reflect realistic implementations

- Snoop filter goes one step towards directories
  - Track sharers, based on writeback and clean eviction
  - Direct snoops and benefit from locality

- Many possible implementations
  - Currently ideal (infinite), no back invalidations
  - Can be used with coherent crossbars on any level
  - See src/mem/SnoopFilter.py and src/mem/snoop_filter.{hh, cc}* 

* Clean eviction patches are still on reviewboard
Key changes and additions

- DRAM controller refinements
  - New DRAM features, power modeling
- Crossbar extensions
  - Interleaving and hashing
- Snoop filter addition
  - Steering snoops, tracking evictions
- Correctness checking
  - Memory-model checker and soak tests
- Performance tuning
  - Transaction support, cache latencies
Memory system verification

- Check adherence to consistency model
  - Notion of functional reference memory is too simplistic
  - Need to track valid values according to consistency model

- Memory checker and monitors
  - Tracking in `src/mem/MemChecker.py` and `src/mem/mem_checker.{hh, cc}`
  - Probing in `src/mem/mem_checker_monitor.{hh, cc}`

- Revamped testing
  - Complex cache (tree) hierarchies in `configs/examples/{memtest, memcheck}.py`
  - Randomly generated soak test in `util/memtest-soak.py`
  - For any changes to the memory system, please use these
Key changes and additions

- DRAM controller refinements
  - New DRAM features, power modeling
- Crossbar extensions
  - Interleaving and hashing
- Snoop filter addition
  - Steering snoops, tracking evictions
- Correctness checking
  - Memory-model checker and soak tests
- Performance tuning
  - Transaction support, cache latencies
What are we modeling?
A more complete picture

- More control in device and cache interactions
  - Aligned with AMBA terminology and SystemC TLM
  - See `src/mem/packet.{hh, cc}`

- Extended set of supported transactions
  - Whole line writes without need for read exclusive*
  - Reads for non-dirty data and non-cacheable reads*
  - Proper handling of uncachable transactions
  - See `src/mem/cache/cache.{hh, cc}`

*Transaction support patches are still on reviewboard
Performance tuning

- Cache and crossbar latencies refined
  - Enable more representative behaviour with split into request/response/snoop flows
  - Allow caches with longer and asymmetric read/write latencies
  - See src/mem/cache/cache.{hh, cc} and src/mem/xbar.{hh, cc}
Where to next?

Source: ARM
What about Ruby?

- Slow
  - No support for atomic, and a clear bottleneck in timing mode

- Unnecessarily complex
  - Many times there is no need to explore coherency protocols

- Meta programming
  - C++ as text, making development inconvenient

- Compatibility issues
  - Need more flexibility in terms of address ranges, I/O devices, etc
Questions?