Administrivia

• Mini project is graded
  – 1\textsuperscript{st} place: Justin (75.45)
  – 2\textsuperscript{nd} place: Liia (74.67)
  – 3\textsuperscript{rd} place: Michael (74.49)
Administrivia

• Project proposal due: 2/27
  – Original research
    • Related to real-time embedded systems/CPS
  – Building a cyber-physical system (robot)
    • Must include real-time performance evaluation on a selected hardware platform
  – Repeating the evaluation of a chosen paper
    • Any one of the suggested papers.
Administrivia

• Addition presentation schedule

  – 2 papers/day on Week 15 (a week before final)
    • eliminate individual meeting

Or

  – 2 papers/day on Week 11,12,13
    • Keep individual meeting
Real-Time DRAM Controller

Heechul Yun
Memory Performance Isolation

Part 1
- Core1
- LLC

Part 2
- Core2
- LLC

Part 3
- Core3
- LLC

Part 4
- Core4
- LLC

- Memory Controller
- DRAM

Q. How to guarantee predictable memory performance?
How Page Works

REQUEST #1 ARRIVES

*close the previous page and load new one*

REQUEST #1 COMPLETES, REQUEST #2 ARRIVES

*Latency of Request #1*

REQUEST #2 COMPLETES (with open page)

*page is already open, just issue read command*

<table>
<thead>
<tr>
<th>*</th>
<th>Latency – First Access</th>
<th>Latency – Further Accesses</th>
<th>Data Cycles for each core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Core</td>
<td>35</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

- in clock cycles on a JEDEC-compliant DDR3 module
Effects of Contention

ALL REQUESTS ARRIVE AT THE SAME TIME, TARGETED AT SAME BANK AND RANK

<table>
<thead>
<tr>
<th></th>
<th>Latency – First Access</th>
<th>Latency – Further Accesses</th>
<th>Data Cycles for each core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Core</td>
<td>35</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Multiple Cores – same bank/rank</td>
<td>35*N</td>
<td>35*N</td>
<td>4</td>
</tr>
</tbody>
</table>
# Effects of Contention

All requests arrive at the same time, targeted at different ranks.

<table>
<thead>
<tr>
<th>*</th>
<th>Latency – First Access</th>
<th>Latency – Further Accesses</th>
<th>Data Cycles used by each access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Core</td>
<td>35</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>N Cores – same bank/rank</td>
<td>35 + 35*(N-1)</td>
<td>35 + 35*(N-1)</td>
<td>4</td>
</tr>
<tr>
<td>N Cores – different ranks</td>
<td>35 + 4*(N-1)</td>
<td>9 + 4*(N-1)</td>
<td>4</td>
</tr>
</tbody>
</table>
Real-Time Memory Controllers

• Provided guaranteed performance in accessing DRAM.
Real-Time Memory Controllers

- Common techniques
  - Command grouping
    - Force to use ALL banks for each memory access
  - Private banking
    - Assign private DRAM banks to cores
  - Scheduling
    - Use analysis friendly scheduling (e.g., round-robin) over difficult ones (e.g., FR-FCFS)
Worst-case

- 1 bank b/w
  - Less than peak b/w
  - How much?
Worst-Case For Single-Bank: Horrible

Worst-case efficiency [%]

Burst size 8
Burst size 4

Memory device

- DDR2-400B
- DDR2-533B
- DDR2-667C
- DDR2-800C
Bank Interleaving and Groups

Figure 3: Read group (above) and write group (below) with burst length 8 for DDR2-400.
Arbitration: CCSP

\[
\hat{\delta}_p = \frac{\max_{r \in R} \hat{S}_r + \sum_{j=0}^{p} \sigma_j}{1 - \sum_{j=0}^{p-1} \rho_j}
\]
Controller Architecture
Real-Time Memory Controllers (RTMC)

• Predator
  – Command grouping, CCSP arbitration
• AMC
  – Command grouping, round-robin arbitration
• PRET-MC
  – Private bank, TDMA arbitration
• DcMc, MEDUSA
  – RR + FR-FCFS hybrid, bank partitioning
• Read/Write Bundling
  – Reduce bus turn-around overhead.
RTMC References

• An analyzable memory controller for hard real-time CMPs, IEEE Embedded Systems Letters, 2009
• PRET DRAM controller: Bank privatization for predictability and temporal isolation, CODES+ISSS, 2011
• A dual-criticality memory controller (dcmc): Proposal and evaluation of a space case study, RTAS, 2015
• Improved DRAM Timing Bounds for Real-Time DRAM Controllers with Read/Write Bundling, 2016
Real-Time Multi/Many-Core Architecture

• Why is it difficult to analyze WCET?

• Projects on Real-Time CPU Architectures
Worst-Case Execution Time (WCET)

- Real-time scheduling theory is based on the assumption of known WCETs of real-time tasks

Image source: [Wilhelm et al., 2008]
Computing WCET

• Static analysis
  – Input: program code, architecture model
  – output: WCET
  – Problem: architecture model is hard and pessimistic (recall “Parallelism-aware...” paper)

• Measurement
  – No guarantee on true worst-case
  – But, widely used in practice
Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems
“Problematic” CPU Features

• Architectures are optimized to reduce average performance
• WCET estimation is hard because of
  – Pipelining
  – TLBs/Caches
  – Super-scalar
  – Out-of-order scheduling
  – Branch predictors
  – Hardware prefetchers
  – Basically anything that affect processor state
Static Timing Analysis
Control Flow Graph (CFG)

- Analyze code
- Split basic blocks
- Compute per-block WCET
  - use abstract CPU model
Timing Anomalies

• Locally faster != globally faster

Image source: [Wilhelm et al., 2008]
Timing Anomalies

- Locally faster != globally faster

Image source: [Wilhelm et al., 2008]
Real-Time CPU Architectures

• PRET
  – UC Berkeley.
• MERASA/parMERASA project
  – EU
• ACROSS
  – EU
• ARAMIS
  – Germany
• EMC2
  – EU
FlexPRET

Hard-Real-Time (HRT) Threads
Interleaved with Soft-Real-Time (SRT) Threads

HRT threads have **deterministic timing**. 
SRT threads share available cycles

SRAM scratchpad shared among threads

DRAM main memory provides **deterministic latency** for HRT threads. 
Conventional behavior for the rest.

Michael Zimmer
PRET Pipeline

Thread 1, Instruction 1
- THREAD #1: FETCH, DECODE, REGACC, MEM, EXECUTE, FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #2: FETCH, DECODE, REGACC, MEM, EXECUTE, FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #3: FETCH, DECODE, REGACC, MEM, EXECUTE, FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #4: FETCH, DECODE, REGACC, MEM, EXECUTE, FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #5: FETCH, DECODE, REGACC, MEM, EXECUTE, FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #6: FETCH, DECODE, REGACC, MEM, EXECUTE

Thread 1, Instruction 2
- THREAD #1: FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #2: FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #3: FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #4: FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #5: FETCH, DECODE, REGACC, MEM, EXECUTE
- THREAD #6: FETCH

1 clock
FlexPRET Pipeline

- 32-bit, 5-stage thread interleaved pipeline, RISC-V ISA
  - Hard real-time HW threads:
    scheduled at constant rate for isolation and predictability
  - Soft real-time HW threads:
    share all available cycles (e.g. HW thread sleeping) for efficiency
- Deployed on Xilinx FPGA (area comparable to Microblaze)
MERASA Multicore

Figure 1. The general MERASA multicore architecture is a simultaneous multithreaded (SMT) core architecture with a five-stage dual pipeline. It can execute both hard real-time (HRT) and non-HRT (NHRT) threads concurrently.
Figure 2. Block diagram showing the general MERSA multicore architecture. The main blocks are the cores, the real-time capable bus, real-time capable cache, real-time capable memory controller, and the synchronous DRAM (SDRAM) memory device.
Acknowledgement

• Some slides are from:
  – Prof. Rodolfo Pellizzoni, University of Waterloo
  – Prof. Edward A. Lee, University of Berkeley
Summary

• Timing anomalies
  – Locally fast != globally fast on non-timing compositional architectures (i.e., most architectures)

• Timing compositional architecture
  – Free of timing anomalies