Real-Time Architecture

Heechul Yun
Topics

- Introduction to Real-Time Systems, CPS
- CPS Applications
- Real-time architecture/OS
- Fault tolerance, safety, security
Topics

• Introduction to Real-Time Systems, CPS
• CPS Applications
• Real-time architecture/OS
  – Real-time cache, DRAM controller designs
  – Real-time microarchitecture/OS Support
  – Real-time support for GPU/FPGA
• Fault tolerance, safety, security
Real-Time Computing

• Performance vs. Determinism
  – Performance: *average* timing
  – Determinism: *variance* and *worst-case* timing

• Traditional real-time systems
  – Focused on determinism
  – So that we can analyze the system at *design time*
  – Many challenges exist in computer architecture
  – In general, performance demand was not high.

• High performance real-time systems
  – Such as self-driving cars and UAVs (intelligent robots)
  – Demand both performance and determinism
  – More difficult to satisfy both
Architecture for Intelligent Robots

- Time predictability
- High performance
Challenges for Time Predictability

• Software
  – Dynamic memory allocation, virtual memory

• Hardware
  – Interrupts
  – Frequency, voltage, temperature control
  – Pipeline, Out-of-order, Super-scalar
  – Caches
  – DMA devices and bus contention
  – Multicore, Accelerators (GPU, FPGA)
Cache

• Small but fast memory (SRAM)
• Hardware (cache controller) managed storage
  – Mapping: phy addr $\rightarrow$ mapping function $\rightarrow$ set index
  – Replacement: select victim line among the ways
• Improve **average** performance
• Transparent to software
  – It just works!
• But makes **timing analysis complicated**
  $\rightarrow$ Why?
Worst-Case Execution Time (WCET)

- Real-time scheduling theory is based on the assumption of known WCETs of real-time tasks.
WCET and Caches

• How to determine the WCET of a task?
• The longest execution path of the task?
  – Problem: the longest path can take less time to finish than shorter paths if your system has a cache(s)!

• Example
  – Path1: 1000 instructions, 0 cache misses
  – Path2: 500 instructions, 100 cache misses
  – Cache hit: 1 cycle, Cache miss: 100 cycles
  – Path 2 takes much longer
WCET and Caches

• Treat all memory accesses as cache-misses?
  – Problem: extremely pessimistic

• Example
  – 1000 instructions, 100 mem accesses, 10 misses
    • Cache hit: 1 cycle, cache miss: 100 cycles
  – Actual = $900 + 90 \times 1 + 10 \times 100 = 1990 = \sim 2000$ cycles
  – $WCET_{all\text{miss}} = 900 + 100 \times 100 = 10900 = \sim 11000$ cycles
    • >5X higher
WCET and Caches

• Take cache hits/misses into account?
  – To reduce pessimism in WCET estimation

• How to know cache hits/misses of a given job?
  – If we assume
    • the path (instruction stream) is given
    • the job is not interrupted.
    • A known “good” cache replacement policy is used
  – Then we can statically determine hits/misses
    • But less so when “bad” replacement policies are used
Review: Direct-Map Cache

- Cache-line size $= 2^L$
- # of cache-sets $= 2^S$
- Cache size $= 2^{L+S}$
Review: Set-Associative Cache

- Cache-line size = $2^L$
- # of cache-sets = $2^S$
- # of ways = $W$
- Cache size = $W \times 2^{L+S}$
Cache Replacement Policy

• Least Recently Used (LRU)
  – Evict least recently used cache-line
  – “Good” (analyzable) policy. Tight analysis exists.
  – Expensive to maintain order. Not used for large caches

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames
Cache Replacement Policy

• (Tree) Pseudo-LRU
  – Use a binary tree
  – Each node records which half is older
  – On a miss, follow the older path and flip the bits along the way
  – Approximate LRU, No need to sort, practical
  – But analysis is more pessimistic

Image credit: Prof. Mikko H. Lipasti
Cache Replacement Policy

• (Tree) Pseudo-LRU

Image credit: https://en.wikipedia.org/wiki/Pseudo-LRU
Cache Replacement Policy

• (Bit) PLRU or NRU (Not Recently Used)
  – One MRU bit per cache-line
  – Set 1 on access; when the last remaining 0 bit is set to 1, all other bits are reset to 0.
  – At cache misses, the line with lowest index whose MRU-bit is 0 is replaced.

Udacity Lecture: https://www.youtube.com/watch?v=8CjifA2yw7s
## Cache Replacement Policies

- **How to know which policy is used?**
  - Manual (if you are lucky)
  - Reverse engineering

<table>
<thead>
<tr>
<th>Architecture</th>
<th>L1 Data</th>
<th>L1 Instruction</th>
<th>L2 Unified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Atom D525</td>
<td>24kB</td>
<td>6</td>
<td>ATOM</td>
</tr>
<tr>
<td>Intel Pentium 3 900</td>
<td>16kB</td>
<td>4</td>
<td>PLRU</td>
</tr>
<tr>
<td>Intel Core 2 Duo E6300</td>
<td>32kB</td>
<td>8</td>
<td>PLRU</td>
</tr>
<tr>
<td>Intel Core 2 Duo E6750</td>
<td>32kB</td>
<td>8</td>
<td>PLRU</td>
</tr>
<tr>
<td>Intel Core 2 Duo E8400</td>
<td>32kB</td>
<td>8</td>
<td>PLRU</td>
</tr>
<tr>
<td>Intel Core i5 460M</td>
<td>32kB</td>
<td>8</td>
<td>PLRU</td>
</tr>
<tr>
<td>Intel Xeon W3550</td>
<td>32kB</td>
<td>8</td>
<td>PLRU</td>
</tr>
<tr>
<td>AMD Athlon 64 X2 4850e</td>
<td>64kB</td>
<td>2</td>
<td>LRU</td>
</tr>
<tr>
<td>AMD Opteron 8360SE</td>
<td>64kB</td>
<td>2</td>
<td>LRU</td>
</tr>
</tbody>
</table>

Image source: [Abel and Reineke, RTAS 2013]
Problems of Static Timing Analysis

• A lot of assumptions
  – The path (instruction stream) is given
  – The job is not interrupted.
  – Processor architecture (incl. cache) is analyzable

• Reality
  – Worst-case path is difficult to know
  – OS jitters change cache state
  – Most processor architectures are NOT analyzable
Timing Anomalies

- Locally faster != globally faster

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

• Locally faster != globally faster

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

• What architecture does static analysis work?
  – Basically simple, in-order architecture, with 1-level LRU caches (I/D).
  – E.g.,) ARM7 [Axer et al., 2014]

• Most architectures
  – Non timing-compositional
  – Because: prefetchers, out-of-order, superscalar, speculative execution, ...
Measurement Based WCET Analysis

• Well, actually measure the execution times
• Tools support
  – automatically measure execution times w/ subset of all possible inputs & collect timing distribution

• Benefits
  – Can apply to ANY processors
  – Closer to exact WCET (no pessimism)
  – Widely used in practice (in industries)

• But,
  – No guarantees, because you cannot test all inputs
Summary

• Terminologies: WCET, ACET, BCET
• Cache-aware static timing analysis
  – Possible but hard
• Impact of cache replacement policies
  – LRU (good, analyzable), PLRU (not good)
• Timing compositional architecture
  – Analyzable processor architecture (e.g., ARM7)
• Timing anomalies
  – Locally fast ≠ globally fast on non-timing compositional architectures (i.e., most architectures)
References


• [Wilhelm et al., 2008] *The Worst-case Execution-time Problem---Overview of Methods and Survey of Tools*, TECS

• [Wilhelm et al., 2009] *Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems*, TCAD


• [Axer et al., 2014] *Building Timing Predictable Embedded Systems*, TECS