Real-Time Xen: Beyond CPU Scheduling

Meng Xu
University of Pennsylvania

Collaborator: Washington University in Saint Louis
Who I Am?

• PhD Student on real-time scheduling & analysis
• Working on RT-Xen research project since 2012
• Contributed RTDS scheduler in Xen 4.5
Background: RTDS Scheduler

- Real-time scheduler on multicore

Real-time = fast?
Background: RTDS Scheduler

- Real-time scheduler on multicore

Real-time = predictable performance

A guest VM wants \(<\text{budget}\)> in every \(<\text{period}\>\), we guarantee it once we accept the VM!

No matter who else is running in the future...
Background: RTDS Scheduler

- Real-time scheduler on multicore
- Introduced as experimental in Xen 4.5
- Plan: mark it as !experimental for 4.6
- New Plan: mark it as !experimental for 4.7, hopefully... :-P
  - More efficient implementation
  - Less schedule overhead
  - Better toolstack support: get/set per-VCPU params
How about other resources that affect real-time performance?
Such as Last Level Cache...
Shared Cache Cause Unpredictable Performance

- LLC improves average performance, but
- Prediction of RT app. performance become more challenging

Goal: Eliminate LLC interference among VMs in virtualization

- System services
- Real-time task
- Cache-bomb
- Dedicated to

\[ \text{dom1} \]
\[ \text{dom2} \]

\[ \text{VP}_0, \text{VP}_1 \]

\[ \text{C0}, \text{C1} \]

\[ \text{C2}, \text{C3} \]

\[ \text{LLC} \]
Partition Shared Cache

- Page coloring mechanism
  - Machine: 2MB 16-way associative cache, cache line 64B
  - PV guest page size: 4KB.
  - Address bits to index cache sets: [A16, A6]
  - Address bits to index memory page: [A47, A12]
  - 5 bits [A16, A12] overlapped
  - Cache color: cache sets with same bits [A16, A12]
  - Hypervisor controls allocated memory pages to control cache areas to use
Design and Implementation

• Design
  – Construct cache-aware memory pool that hold \( k \) free memory pages in the system (\( k \) is configurable)
  – Redirect Xen memory alloc/free functions to our cache-aware memory operation functions; redirect:
    • \texttt{alloc_domheap_pages()}
    • \texttt{free_domheap_pages()}

CA-Heap data structure:

- \texttt{heap}
- \texttt{ca_domheap}
- node
- zone
- color
Evaluation on Worst-Case Execution Time

• Benchmark
  – Run PARSEC benchmark in dom1 and polluting task (that accesses an array with LLC size) in dom2, measure execution time in Xen and CART-Xen

• Synthetic workload
  – Replace the benchmark with synthetic workload which we can control the access pattern; Compare the performance under different cache access pattern

• **Expected result**
  – WCET without cache partition should be much worse than the WCET with cache partition
  – Shared-cache interference from another domain is eliminated in CART-Xen by the cache partition mechanism
  – Execution time of workload depends on the size of cache area allocated to the workload
Evaluation: Benchmark

  - Focus on shared-memory programs on multiprocessors
  - We evaluate 11 benchmarks from PARSEC

- **Choose canneal benchmark**
  - Simulated cache-aware annealing to optimize routing cost of a chip design

![Diagram showing benchmarks and system services]
Canneal: Simulated cache-aware annealing to optimize routing cost of a chip design

Larger reserved cache area lead to better performance

\[ \text{slowdown} = \frac{\text{WCET}_{\text{env}}}{\text{WCET}_{\text{alone}}} \]
Yes! This is what we expect!
Really?
Is it always true?
Execution Time vs. Number of Cache Partitions

Fluidanimate: Fluid dynamics for animation purposes with Smoothed Particle Hydrodynamics (SPH) method

More cache not always means better performance
Example: Larger Cache Cause Larger WCET

Scenario 1: If cache has 4 cache sets, \( \# \text{ conflict miss} \leq 3 + 1 \)

<table>
<thead>
<tr>
<th>$set$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>

Scenario 2: If cache has 5 cache sets, \( \# \text{ conflict miss} \leq 10 \)

<table>
<thead>
<tr>
<th>$set$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>1</td>
</tr>
</tbody>
</table>
Performance of cache partition mechanism depends on cache-access pattern of tasks
## Evaluation: Synthetic Workload

**Replace PARSEC benchmark with synthetic workload**
Measure latency of accessing an array with size $s$ for 1024 times and pick 99% quantile;

**Different cache-access patterns (e.g., sequential or random) may affect latency of cache access (due to HW prefetch in cache, TLB miss)**

**NB**: Only show the result of seq-each-line access pattern in the slide now. We have data and figures for all four access patterns.

<table>
<thead>
<tr>
<th>Access pattern</th>
<th>Definition</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-each-line</td>
<td>Sequentially access each cache line</td>
<td></td>
</tr>
<tr>
<td>seq-each-line-stride-page</td>
<td>First sequentially access the first cache line in each page of the array, and then access the second cache line in each page</td>
<td></td>
</tr>
<tr>
<td>random-in-page</td>
<td>Each page holds 64 cache lines; randomly access the 64 cache lines in each page; but sequentially access each page</td>
<td></td>
</tr>
<tr>
<td>random-full</td>
<td>An array has $(\text{size}/64)$ cache lines, randomly access all these cache lines</td>
<td></td>
</tr>
</tbody>
</table>

NB: Only show the result of seq-each-line access pattern in the slide now. We have data and figures for all four access patterns.
Evaluation: Environment

cache-bench: synthetic workload to measure latency of accessing an array
cache-bomb: keep polluting the LLC

RTXen: Xen with RTDS scheduler
CARTXen: Xen with RTDS scheduler and static cache partition

<table>
<thead>
<tr>
<th>Environment</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>native-alone</td>
<td>cache-bench alone in native Linux</td>
</tr>
<tr>
<td>native-pollute</td>
<td>cache-bench runs with cache-bomb on different cores in native Linux</td>
</tr>
<tr>
<td>RTXen-alone</td>
<td>cache-bench alone in dom1 with dedicated VCPU, nothing in dom2 with dedicated VCPU;</td>
</tr>
<tr>
<td>RTXen-pollute</td>
<td>cache-bench alone in dom1 with dedicated VCPU, cache-bomb in dom2 with dedicated VCPU</td>
</tr>
<tr>
<td>CARTXen-alone</td>
<td>cache-bench alone in dom1 with dedicated VCPU and cache colors [0-15]; nothing in dom2 with dedicated VCPU and cache colors [16-31]; (dom1 and dom2 each has half cache area)</td>
</tr>
<tr>
<td>CARTXen-pollute</td>
<td>cache-bench alone in dom1 with dedicated VCPU and cache colors [0-15]; cache-bomb in dom2 with dedicated VCPU and cache colors [16-31]; (dom1 and dom2 each has half cache area)</td>
</tr>
</tbody>
</table>
Execution Time vs. Working Set Size

99\% quantile

\[ \text{slowdown} = \frac{WCET_{env}}{WCET_{alone}} \]

LLC interference causes 6X slowdown to target domain!

Cache partition effectively eliminates LLC interference!
Hardware-Assisted Cache Partition

• Intel Cache Allocation Technology
  – Bits in capacity bitmask (CBM) indicate reserved cache partitions
  – Constraint: CBM only allows continuous 1 combination.[1]

<table>
<thead>
<tr>
<th></th>
<th>M7</th>
<th>M6</th>
<th>M5</th>
<th>M4</th>
<th>M3</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• ARM Cortex A9 PrimeCell Level 2 Cache Controller (PL310)
  – Xen cannot run on Cortex A9?
  – Cortex A15 do not have PL310 ☹

[1] Intel(R) Architecture Software Development Manuals, Chapter 17.16
## Hardware Approach vs. Software Approach

<table>
<thead>
<tr>
<th>Desired Property</th>
<th>Hardware Approach</th>
<th>Software Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not limited to 4KB page size</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Compatible with old machine</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No memory copy in cache partition migration</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>A task can be allocated a set of non-contiguous partitions</td>
<td>No (Intel CAT)</td>
<td>Yes (ARM PL310)</td>
</tr>
</tbody>
</table>
Conclusion & Challenge & WiP

• Conclusion
  – Cache partition effectively eliminates LLC interference and provide deterministic execution time of tasks

• Challenge
  – Hard to know task’s cache behavior (cache favorite?)
  – Intel CAT constraint

• WiP
  – Only support static partition currently
  – Dynamically increase/decrease/migrate cache partitions of a domU online
  – Incorporate cache management technique with real-time scheduling to improve the schedulability of whole system
Thank You! Questions?