Online Cache Modeling for Commodity Multicore Processors

Richard West, Puneet Zaroo, Carl A. Waldspurger and Xiao Zhang

Contact: richwest@cs.bu.edu
The “Big Picture”

Application threads

Socket

Shared LLC

PCPU
Cores/HTs

Socket

Shared LLC

PCPU
Cores/HTs
Chip Multiprocessors (CMPs) have multiple cores on same chip

CMP cores usually share last-level cache (LLC) and compete for memory bus bandwidth

Competition for microarchitectural resources by co-running workloads can lead to highly-variable performance

– Potential for poor performance isolation
The Software Challenge

- CMPs manage shared h/w resources (e.g., cache space, memory bandwidth) in opaque manner to s/w

- Software systems cannot easily optimize for efficient resource utilization or QoS without improved visibility and control over h/w resources
  - e.g., Cache conflict misses can incur several hundred clock cycle penalties for off-chip memory stalls
Hardware Solutions

• Provide performance isolation using cache partitioning
  – Optimal partition size?
  – Utility of cache space to a workload?

• Hardware-assisted miss-ratio (and miss-rate) curves (MRCs)
  – not applicable to commodity multicore processors
Improved Cache Management

• Expose state of shared caches (and other microarchitectural resources) to OS / hypervisor
  – Fairer / more efficient co-scheduling
  – Reduced resource contention

  – How do we do this on commodity CMPs?
Current Software Solutions

• Page coloring
  – Can reduce cache conflicts
  – Recoloring pages can be expensive for varying working set sizes and workloads

• S/W-generated MRCs
  – Existing solutions require special h/w support
    • e.g., RapidMRC uses SDAR on POWER5
  – Potentially high overhead
    • e.g., RapidMRC takes > 80ms on POWER5
Our Approach

• Online cache modeling for commodity CMPs

• Leverage commonly-available hardware performance counters
  – Construct cache occupancy estimators for individual workloads competing for cache
  – Construct cache performance curves (MRCs) using occupancy predictions
  – Low-cost and online
Leverage two performance events:
- local misses to thread $\tau_l$: $m_l$
- misses by every other thread $\tau_o$ sharing
  - cache: $m_o$
  - Misses drive cache line fills

Assume C cache lines accessed uniformly at random

\[ E' = E + (1 - E/C) \cdot m_l - (E/C) \cdot m_o \]

$E'$ = updated occupancy of $\tau_l$, $E$ = old value
Extended Occupancy Model

• Basic approach assumes uniform cache-line access

• Set associativity and LRU line replacement breaks this assumption

• Add support for likelihood of line reuse
  – Use cache hit information
Extended Occupancy Model

• Uses four performance events:
  – As for basic model plus
    • Local hits ($h_l$) and hits by all other threads ($h_o$)

• Now:
  \[ E' = E \cdot (1 - m_o p_l) + (C-E) \cdot m_l p_o \]  -- Equation 1

$p_l$ is probability miss falls on line for $\tau_l$

$P_o$ is probability miss falls on line for $\tau_o$
• Approximate LRU with LFU:
  – Model cacheline reuse by $\tau_1$ and $\tau_o$, respectively, as:

$$r_i = \frac{h_i + m_i}{E}$$
$$r_o = \frac{h_o + m_o}{C - E}$$
Approximating LRU Effects

- Model evictions due to misses inversely proportional to reuse frequencies:
  \[ \frac{p_o}{p_l} = \frac{r_l}{r_o} \]

- Given a miss must fall on some line:
  \[ p_l \cdot E + p_o \cdot (C-E) = 1 \]

Can calculate \( p_l \) and \( p_o \) and substitute into Equation 1
Occupancy Experiments

- Used Intel’s CMPSched$im
  - Binary execution of SPEC workloads
  - Modeled 2- and 4-core CMPs
    - 32KB 4-way per-core L1
    - 4MB 16-way shared L2
    - 64 byte cache line size
  - Sample perf counters every 1ms
  - Average occupancies over 100 ms intervals
Occupancy Results

Quadcore – 4 co-runners (3 shown)

- mcf
- art00
- wupwise00
Occupancy Results

Quadcore – 10 co-runners (3 shown)

- mcf
- art00
- wupwise00

Model tolerant of over-committed situations.
Cache Performance Curves

- Modeled performance (MPKI, MPKR, MPKC, CPKI, …) as function of cache occupancy

- Implemented CAFÉ scheduling framework in VMware ESX Server
  - 4-core 2.0 GHz Intel Xeon E5535 w/ 4GB RAM and 4MB L2 cache per 2-cores
  - Update workload occupancies every 2ms using basic model (2 perf ctrs)
    - 320 cycles overhead for occupancy update fn
Online Generation of Utility Curves

• Curve Types
  – Miss-ratio curve, y-axis being Misses-Per-Kilo-Instructions
  – Miss-rate curve, y-axis being Misses-Per-Kilo-Cycles
  – CPKI curve, y-axis being Cycles-Per-Kilo-Instructions

• Implementation issues
  – Monotonicity enforcement
  – Lack of updates across entire cache
  – Duty-cycle modulation enforcement
  – MPKC curves sensitive to memory bandwidth contention

mcf running under different amounts of memory read bandwidth
MRC Results

• Quantized into 8 occupancy buckets
• Configurable interval for curve generation frequency (here, several seconds)
• Expect monotonicity
  – Higher cache occupancy, fewer misses per instruction
  – Except on phase changes
• Monotonic enforcement algorithm updates MRC readings in order of bucket reference (highest to lowest)
Online MRC: Accuracy

- 6 apps on 2 cores sharing L2, each in a single-CPU VM
- Using page-coloring measurement as comparison baseline
Online MRC: Case Study

- Running mcf with different co-runners
Application of Utility Curves

• Guidance to improve fairness
  – CPU time compensation based on estimated performance degradation due to CMP resource contention

• Guidance to improve performance
  – Smart scheduling placement based on predicted cache space allocation among co-runners
Future Work

• Application of occupancy prediction to hardware-aided cache partitioning / enforcement

• Investigate techniques to improve coverage of cache space (0-100%) for utility curve generation
  – Co-runner interference control
  – MRCs at different tie granularities
    • Online phase change detection