Managing Shared L2 Caches on Multicore Systems in Software

David Tam, Reza Azimi, Livio Soares, Michael Stumm

University of Toronto

{tamda, azimi, livio, stumm}@eecg.toronto.edu

WIOSCA 2007
Multicores Today

Shared L2 Issues:

- Uncontrolled sharing \(\rightarrow\) threads compete for space \(\rightarrow\) LRU eviction policy \(\rightarrow\) interference
Uncontrolled Sharing is Bad

- Hardware partitioning mechanisms:
  - [Qureshi06],[Rafique06],[Chandra05],[Iyer04],[Suh04]
  - Not available today
Software Partitioning

- Implemented on real system
- Apply classic page-coloring technique
  - [Cho06], [Bershad94], [Lynch92], [Kessler92]
- Guided phys pages allocation → controlled L2 cache line usage
Software-Based Properties

- Deployable right now
  - OS, VMM

- Flexible
  - Can allocate for specific: users, applications, processes, threads
  - Unaffected by: # threads, location

25% Apache (4 colors) 75% MySql (12 colors)

MySQL
Apache
Linux Implementation

- Color-aware physical page allocation
  - Minimal changes to Linux
- List of local free physical pages:

Old: {Single list (per logical CPU)}

- To find target color quickly:

New: {Multiple lists (per logical CPU)}

- Looks like single list to rest of Linux
Experimental Setup

- 16 partitions (colors):
  - 120 KB each (L2), 2.25 MB each (L3)
- Linux 2.6.15
- SPECcpu2000, SPECjbb2000
Overview of Results

- Show ability to control cache usage
  - Single-programmed
- Show benefits of partitioning
  - Multi-programmed
- Metric selection for performance analysis
Overview of Results

- **Show ability to control cache usage**
  - Single-programmed

- **Show benefits of partitioning**
  - Multi-programmed

- **Metric selection for performance analysis**
Controlling Cache Usage

- Mechanism can control cache usage
- Single-programmed
Overview of Results

- Show ability to control cache usage
  - Single-programmed

- **Show benefits of partitioning**
  - Multi-programmed

- Metric selection for performance analysis
Benefits of Partitioning

- Multiprogrammed (1 application per core)
  - **Base**: multi-prg, no partitioning
Benefits of Partitioning

L2 Cache Size of gzip (# of partitions)

L2 Cache Size of equake (# of partitions)

L2 Cache Size of twolf (# of partitions)

L2 Cache Size of SPECjbb (# of partitions)

Normalized IPC (%)

gzip + twolf

SPECjbb + equake
Mechanism Issues

Inherent:

- L3 victim cache partitioned correspondingly
- Super-pages reduce # of partitions
- Time granularity
  - Allocating more/less partitions

Current Implementation:

- Static partitioning
  - Future work: dynamic partitioning
Overview of Results

- Show ability to control cache usage
  - Single-programmed
- Show benefits of partitioning
  - Multi-programmed
- Metric selection for performance analysis
Experiences with Metrics

- Explaining & predicting performance

**Obvious Metric:** L2 Miss Rate Curve (MRC)
  - Rate = per billion cycles
  - **Miss** rate ignores cost variations [Moreto07],[Qureshi06]
    - Miss latencies
    - Memory level parallelism [Qureshi06]

**Alternative Metric:** Stall Rate Curve (SRC)
  - **Stall** rate more inclusive [McGregor05]
    - Instruction retirement stall
    - Caused by memory latencies
    - Directly measures pipeline degradation
  - Measurable by hardware performance counters
L2 MRC Failure in Art

- Single-programmed *art*
  - L2 MRC does *not* show trend
  - SRC shows trend
  - (per billion cycles)

- L1 data cache miss SRC is adequate
L2 MRC Failure in Art

- Single-programmed *art*
- L3 and main memory hit rates explain trend
- (per billion cycles)
Conclusions

- Software partitioning is effective
  - Up to 17% improvement
  - Deployable today
- Stall metric is better than miss metric
  - More accurate for performance analysis
  - Measurable on real system

Work In Progress

- More workloads
- Dynamically allocate more/less partitions
- Dynamically determine optimal # of partitions
  - Using hardware performance counters