Dynamic Cache Partitioning for Simultaneous Multithreading Systems

G. Edward Suh

Larry Rudolph

Srinivas Devadas

LCS, MIT

Simultaneous Multithreading (SMT) Systems

- Combines superscalar architecture with multithreaded architectures
- Low IPC comes from two sources
  - Data dependencies
  - Data delay (memory bottleneck)
- SMT relieves the dependency problem
  - Helps to hide the memory latency
- SMT increases the total footprint
  - Puts more pressure on the memory system

Strategy – Partition the Cache

- Control the amount of data for each thread
  ➔ minimizes the number of misses
  - On-line monitoring of thread characteristics
    - Marginal gain; $g_i(x)$: Additional hits by increasing the cache space from *x* blocks to *x+1* blocks
  - Deciding cache allocation to each thread
    - Based on the marginal gain of each thread
  - Partitioning mechanism
    - Augmented LRU replacement policy

Example: Marginal Gains

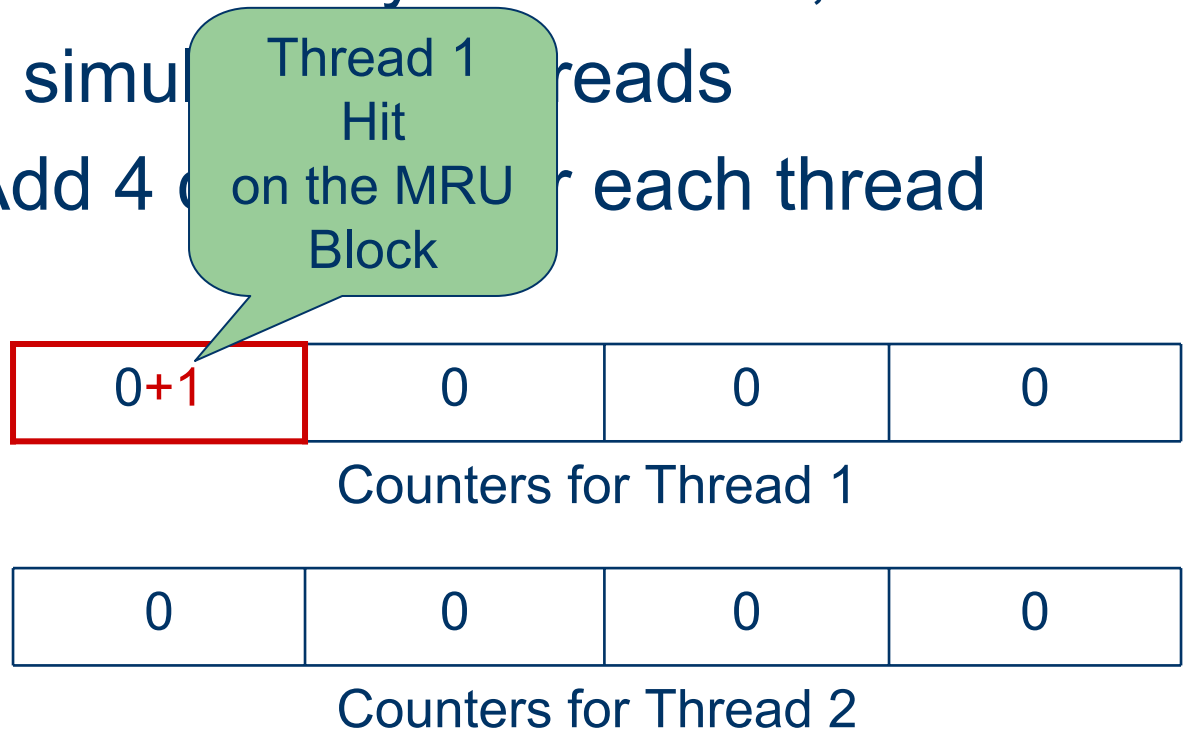
- Cache: 4-way associative, 8192 sets
- 2 simultaneous threads
- Add 4 counters for each thread

Counters for Thread 1

Counters for Thread 2

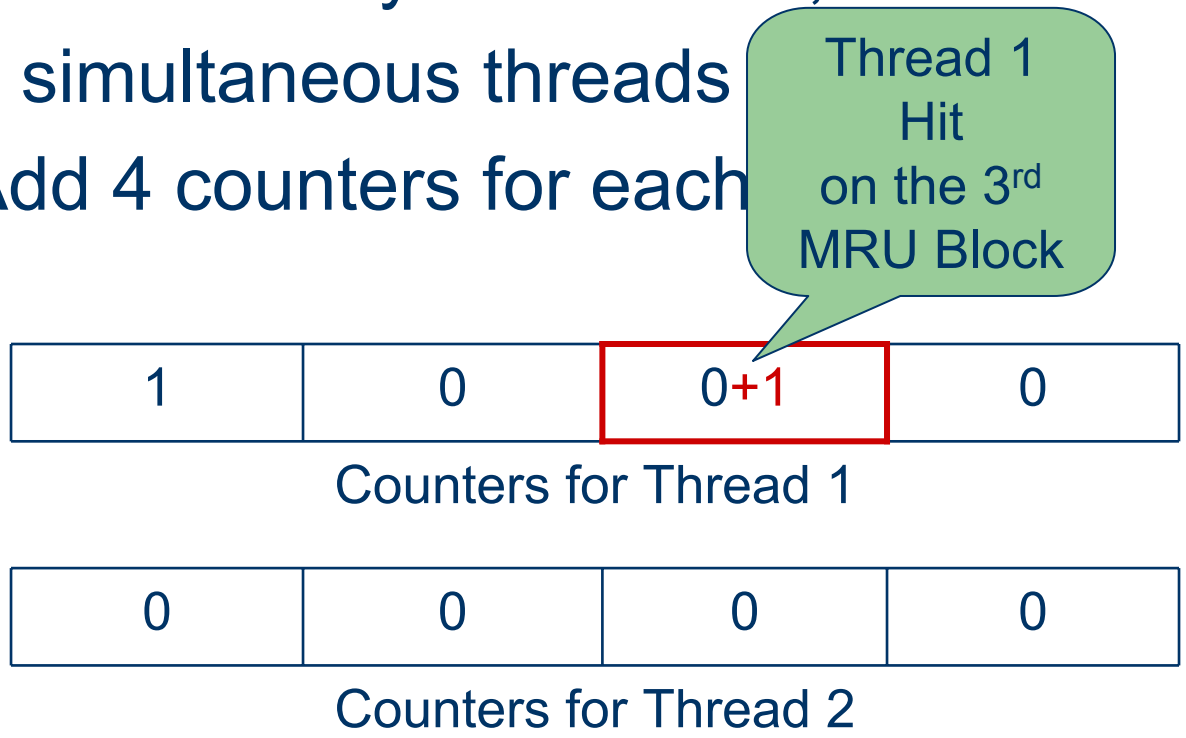
Example: Marginal Gains

- Cache: 4-way associative, 8192 sets
- 2 simul reads
- Add 4 c r each thread



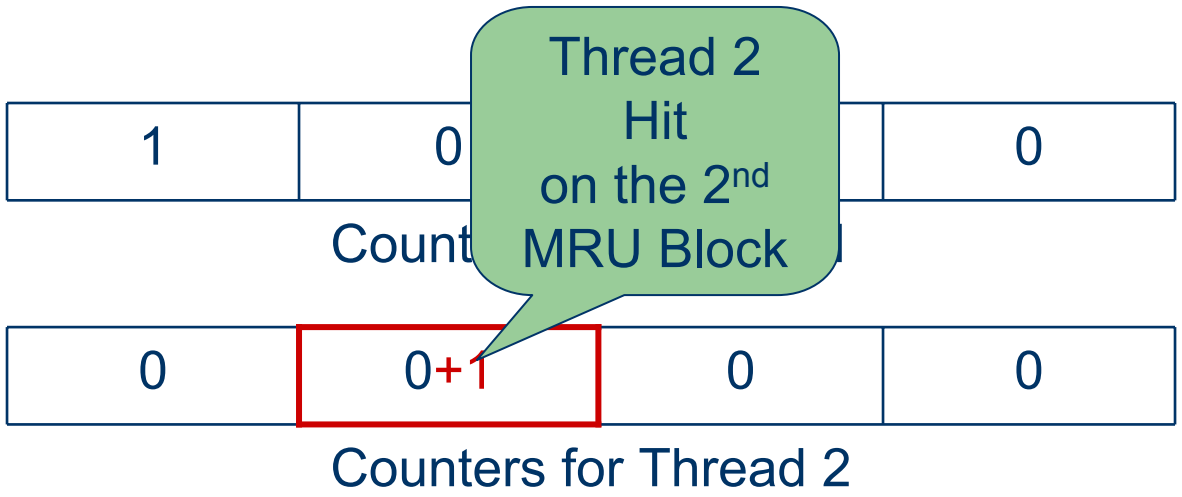
Example: Marginal Gains

- Cache: 4-way associative, 8192 sets
- 2 simultaneous threads
- Add 4 counters for each



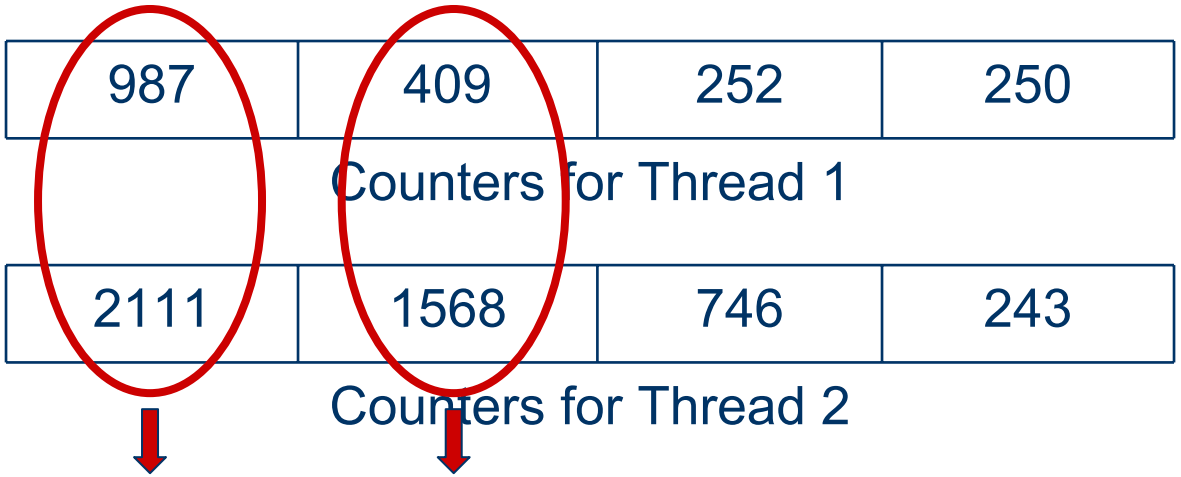
Example: Marginal Gains

- Cache: 4-way associative, 8192 sets
- 2 simultaneous threads
- Add 4 counters for each thread



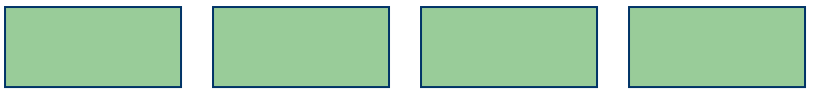
Example: Marginal Gains

- Cache: 4-way associative, 8192 sets
- 2 simultaneous threads
- Add 4 counters for each thread



Example: Partitioning Decision

Unassigned Blocks : 8192*4



Allocation to Thread 1 : 0

Allocation to Thread 2 : 0

Example: Partitioning Decision

Unassigned Blocks : 8192*3

Allocation to Thread 1 : 0

Allocation to Thread 2 : 8192

Example: Partitioning Decision

Unassigned Blocks : 8192*2

Allocation to Thread 1 : 0

Allocation to Thread 2 : 16384

Example: Partitioning Decision

Unassigned Blocks : 8192

Allocation to Thread 1 : 8192

Allocation to Thread 2 : 16384

Example: Partitioning Decision

987	409	282	250

Counters for Thread 1

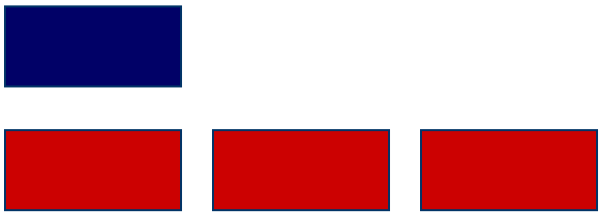
2111	1568	746	243

Counters for Thread 2

Unassigned Blocks : 0

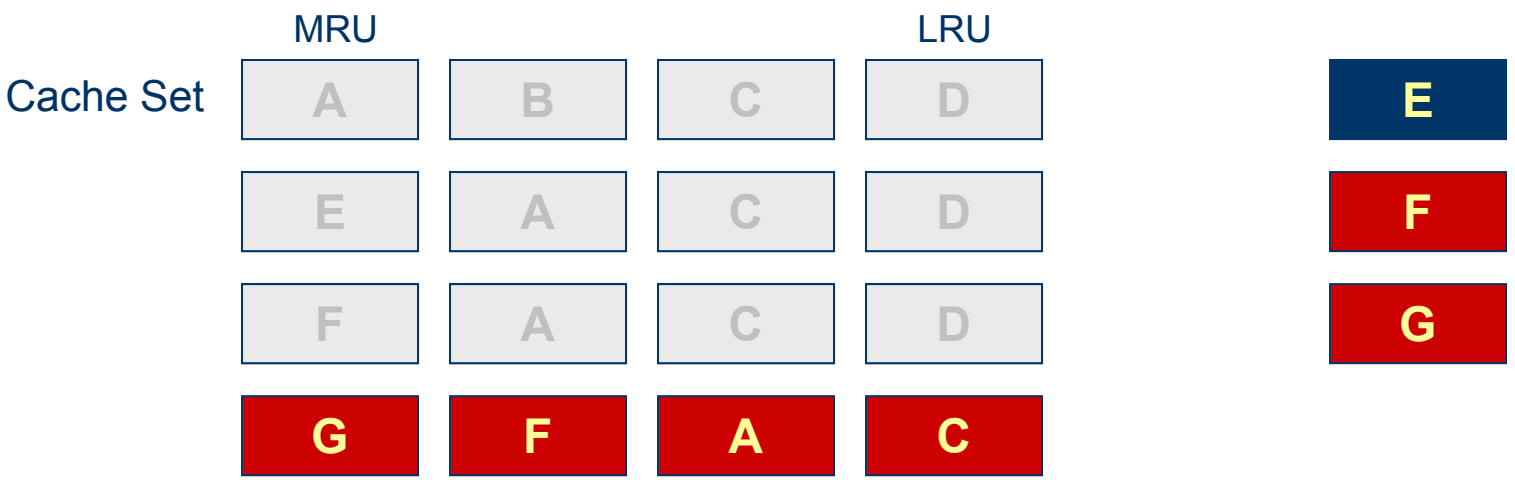
Allocation to Thread 1 : 8192

Allocation to Thread 2 : 24576



Example: Augmented LRU

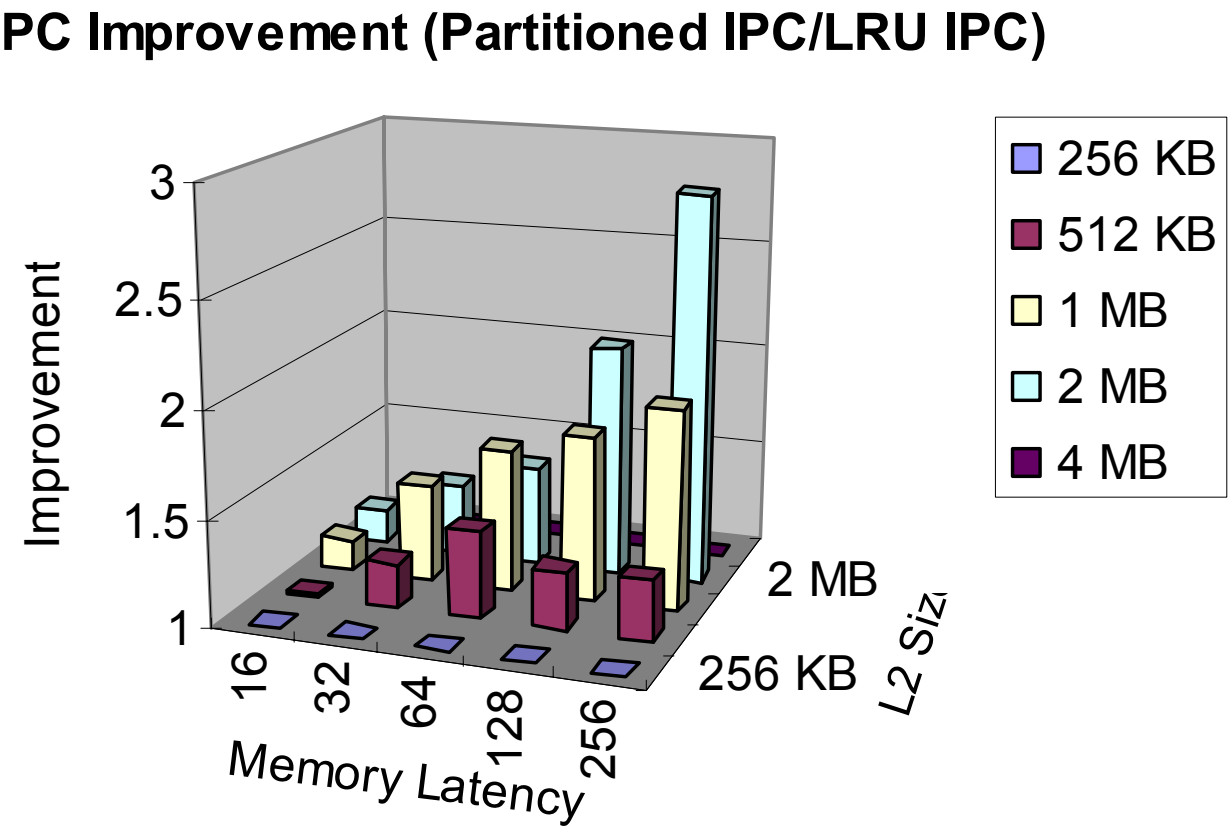
	Allocation	Actual
Thread 1	8192	10326
Thread 2	24576	22442



Experimental Setup

- On-line L2 cache partitioning
- Combine SimpleScalar with a cache simulator
- System configuration
  - Executes up to 4 threads simultaneously
  - 4 ALUs and 1 Multiplier
  - 32-KB 8-way L1 caches (latency 1 cycle)
  - Various size 8-way L2 caches (latency 10 cycles)
- Benchmarks
  - SPEC CPU2000; **art** and **mcf**

Experimental Results



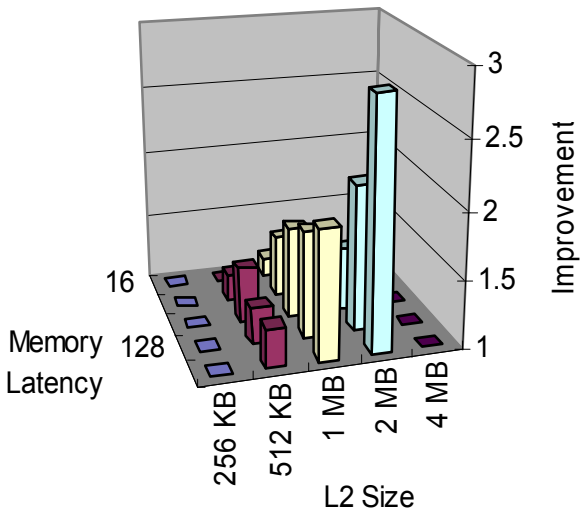
Discussion of Results

- Small caches
  - Nothing helps: should change the workload
- Medium caches
  - Partitioning helps
  - Improvement related to latency (more than linear)
- Large caches
  - Partitioning does not help: All workloads fit into the cache

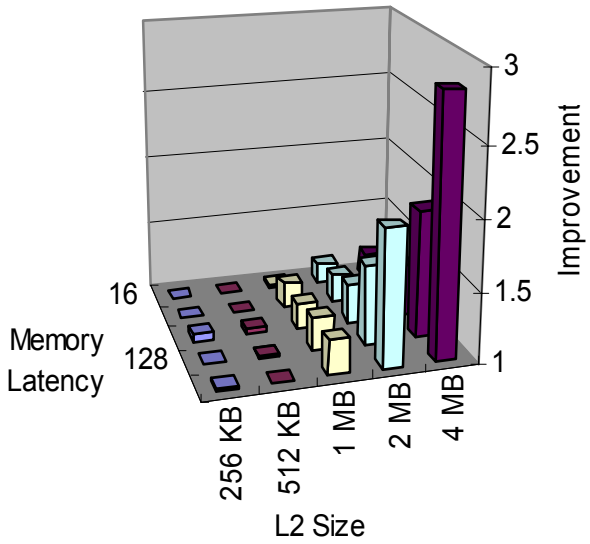
Relevant Cache Sizes

- Partitioning helps for medium size caches
- Relevant cache sizes depend on the characteristics of threads and the number of active threads

IPC Improvement: 2 threads



IPC Improvement: 4 threads



Summary

- Simultaneous Multithreading may significantly degrade the cache performance
- Smart partitioning can relieve the problem for medium size caches
  - The relevant size varies depending on the characteristics and the number of threads
- Cache-Aware thread scheduling is needed for small caches