A New Cache Monitoring Scheme for Memory-Aware Scheduling and Partitioning

G. Edward Suh
Srinivas Devadas
Larry Rudolph

Massachusetts Institute of Technology

Problem

- Memory system performance is critical
- Everyone thinks about their own application
  - Tuning replacement policies
  - Software/hardware prefetching
- But modern computer systems execute multiple applications concurrently/simultaneously
  - Time-shared systems
    - Context switches cause cold misses
  - Multiprocessors systems sharing memory hierarchy (SMP, SMT, CMP)
    - Simultaneous applications compete for cache space

Solutions: Cache Partitioning & Memory-Aware Scheduling

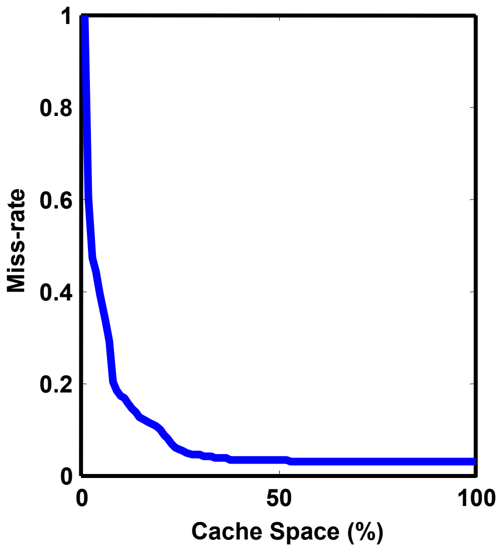
- Cache Partitioning
  - Explicitly manage cache space allocation amongst concurrent/ simultaneous processes
    - Each process gets different benefit from more cache space
    - Similar to main memory partition (e.g.. Stone 1992) in the old days
- Memory-Aware Scheduling
  - Choose a set of simultaneous processes to minimize memory/cache contention
  - Schedule for SMT systems (Snavely 2000)
    - Threads interact in various ways (RUU, functional units, caches, etc)
    - Based on executing various schedules and profiling them
  - Admission control for gang scheduling (Batat 2000)
    - Based on the footprint of a job (total memory usage)

BUT…

- Testing many possible schedules ➔ not viable
  - The number of possible schedules increase exponentially as the number of processes increase
  - Need to decide a good schedule from individual process characteristics → complexity increases linearly
- Footprint-based scheduling ➔ not enough information
  - Footprint of a process is often larger than the cache
  - Processes may not need the entire working set in the cache
- Can we find a good schedule for cache performance?
  - What information do we need for each process?

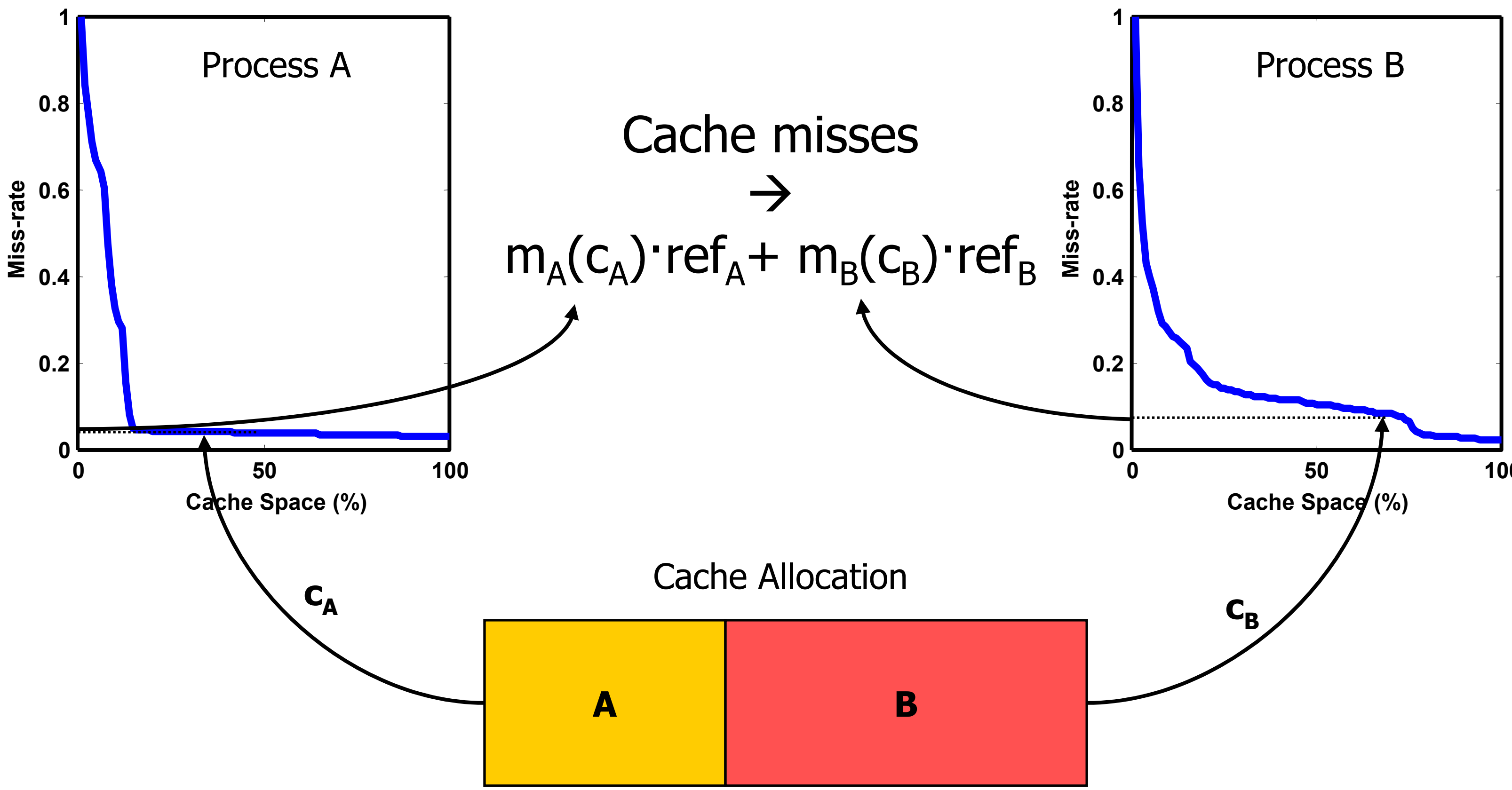
Information a Scheduler/Partitioner Needs

- Characterizing a process
  - For scheduling and partitioning, need to know the effect of varying cache size
    - Multiple performance numbers for different cache sizes
    - Ignore other effects than cache size
- Miss-rate curves; m(c)
  - Cache miss-rates as a function of cache size (cache blocks)
    - Assume a process is isolated
    - Assume the cache is FULLY-ASSOCIATIVE
  - Provides essential information for scheduling and partitioning



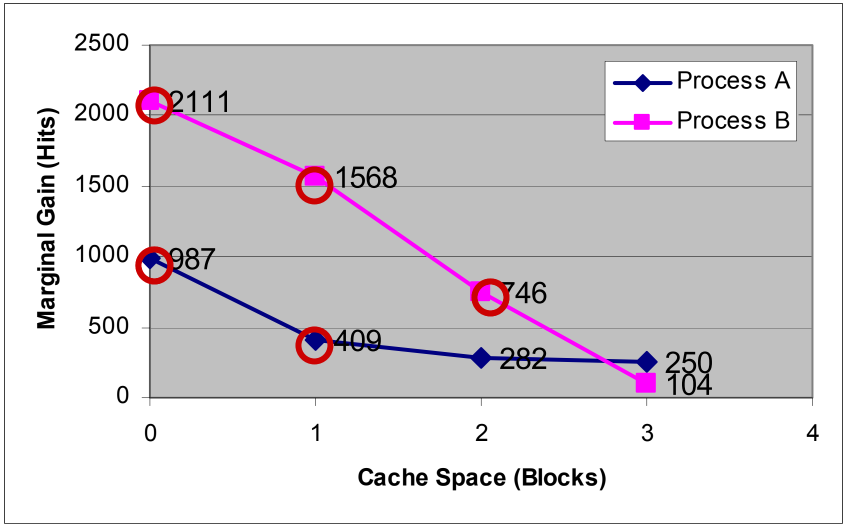
Using Miss-Rate Curves for Partitioning

- What do miss-rate curves tell about cache allocation?

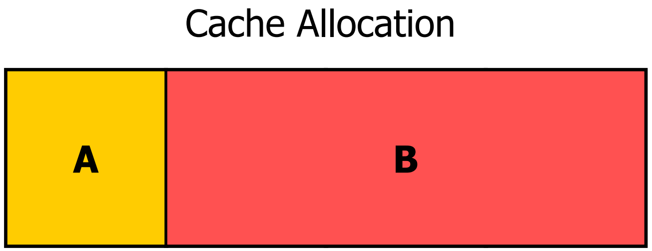


Finding the best allocation

- Use marginal gain; g(c) = m(c) ·ref - m(c+1)·ref
  - Gain in the number of misses by increasing the cache space
- Allocate cache blocks to each process in a greedy manner
  - Guaranteed to result in the optimal partition if m(c) are convex

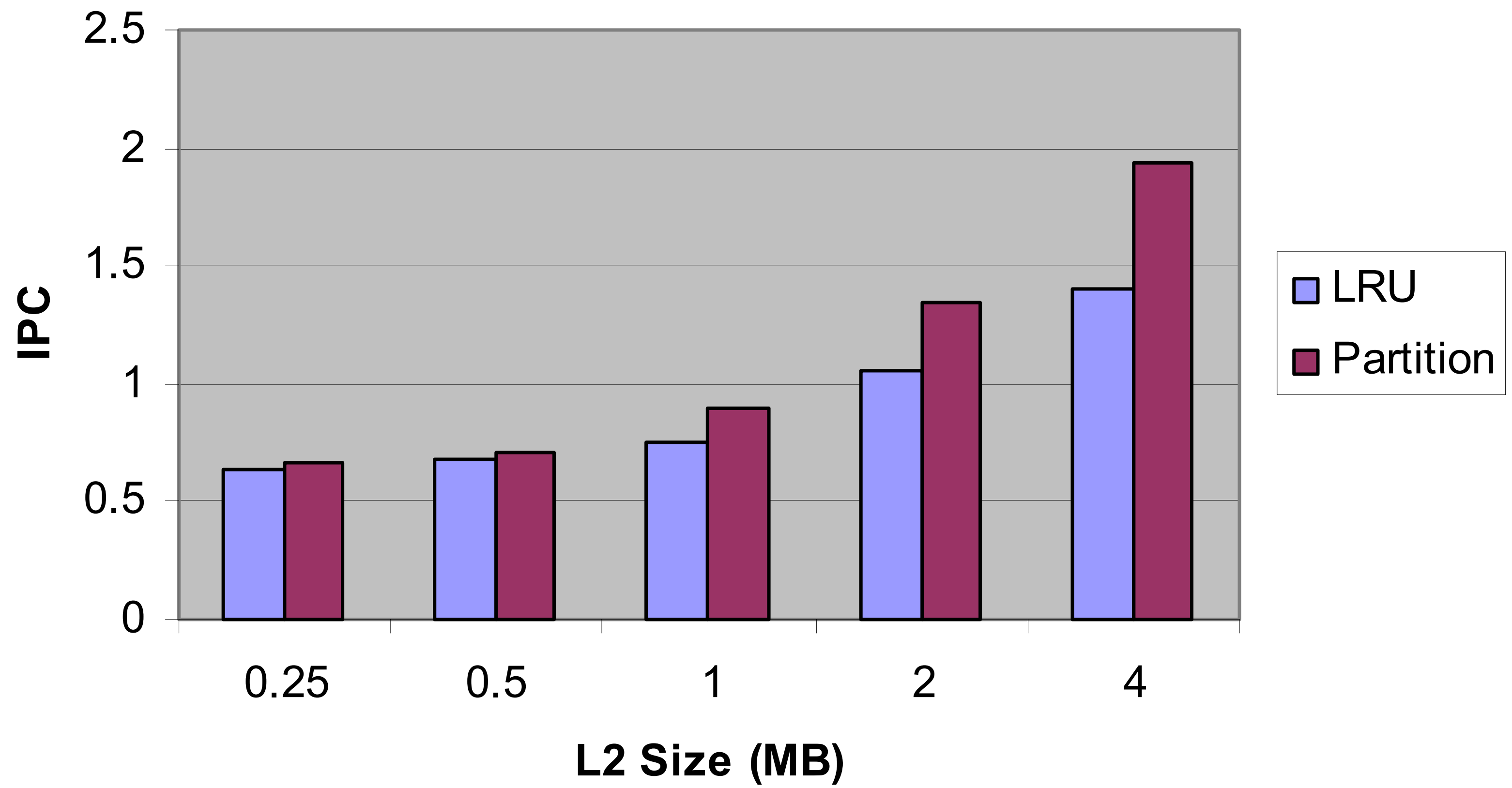


Allocate a block to
Process B



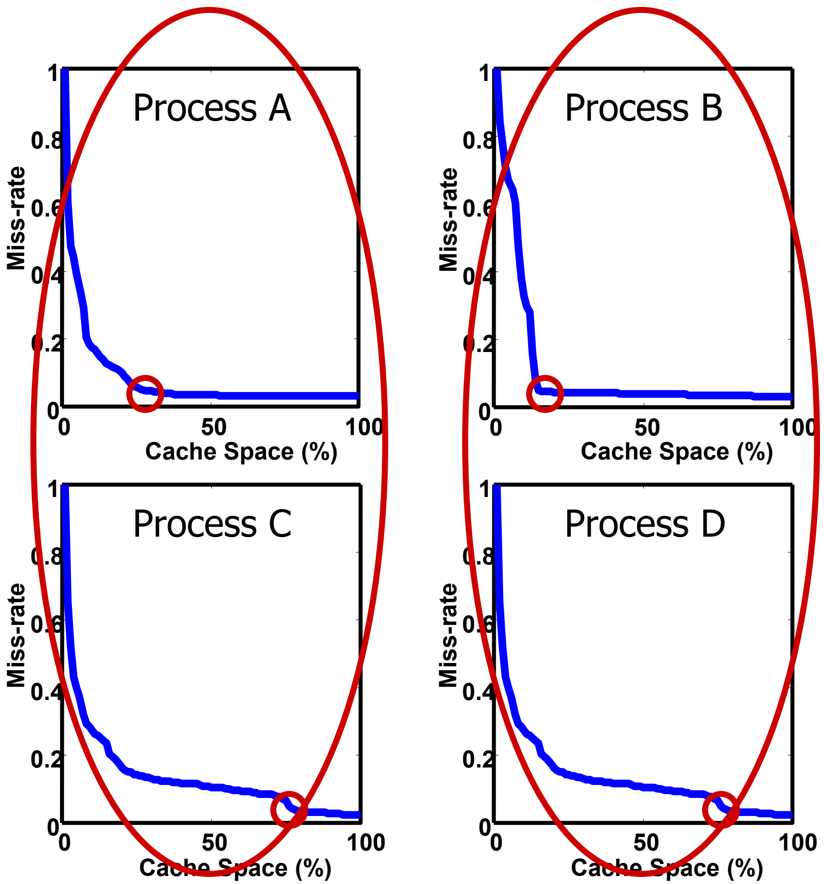
Partitioning Results

- Partition the L2 cache amongst two simultaneous processes (spec2000 benchmarks: *art* and *mcf*)



Intuition for Memory-Aware Scheduling

- How to schedule 4 processes on 2 processor system using individual miss-rate curves?



Curves tend to have a knee
→ The amount of cache space where the marginal gain diminishes a lot

- Working set size is larger than the cache for all processes

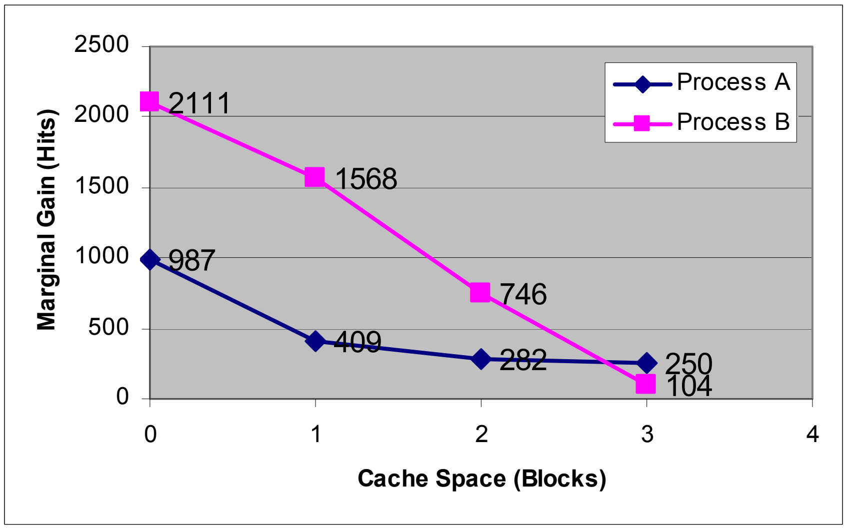
Group processes based on the knees

- All processes result in similar miss-rate if they have the entire cache

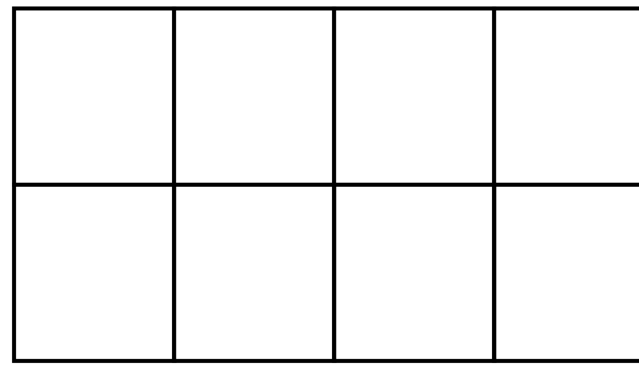
Schedule A and C, and B and D together

Determining the Knee of the Curve

- Use partitioning technique



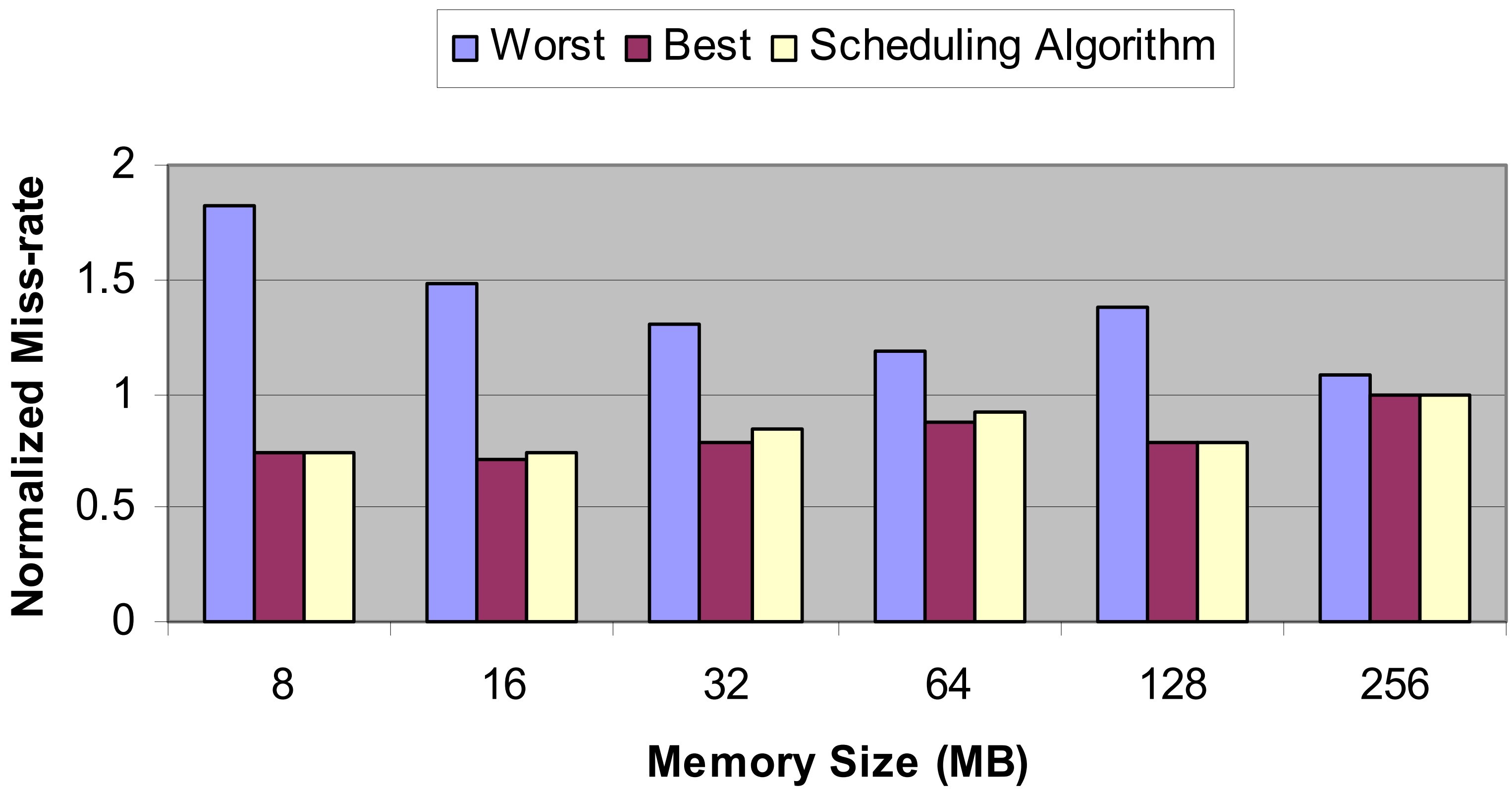
Cache Allocation



- However, now we may need multiple time slices to schedule processes (2 time slices in our example)
- Available cache resource should be doubled

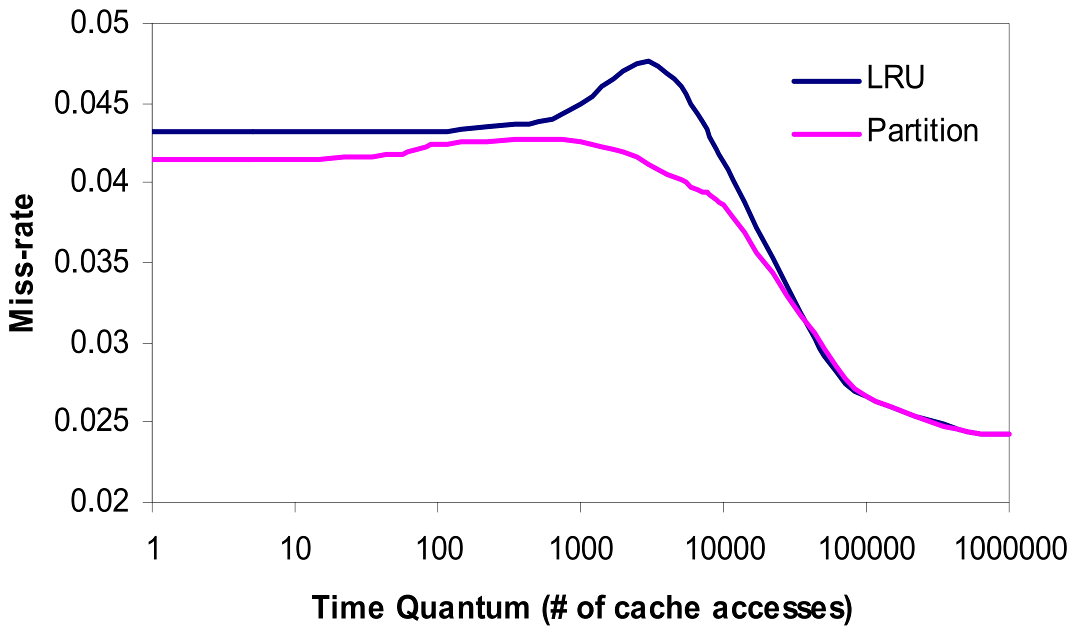
Scheduling Results

- Schedule 6 SPEC CPU benchmarks for 2 Processors



Analytical Model (ICS`01)

- Miss-rate curves (or marginal gains) alone may not be enough for optimizing time-shared systems
  - Partitioning amongst concurrent processes
  - Scheduling considering the effects of context switches
- Use analytical model to predict cache-sharing effects



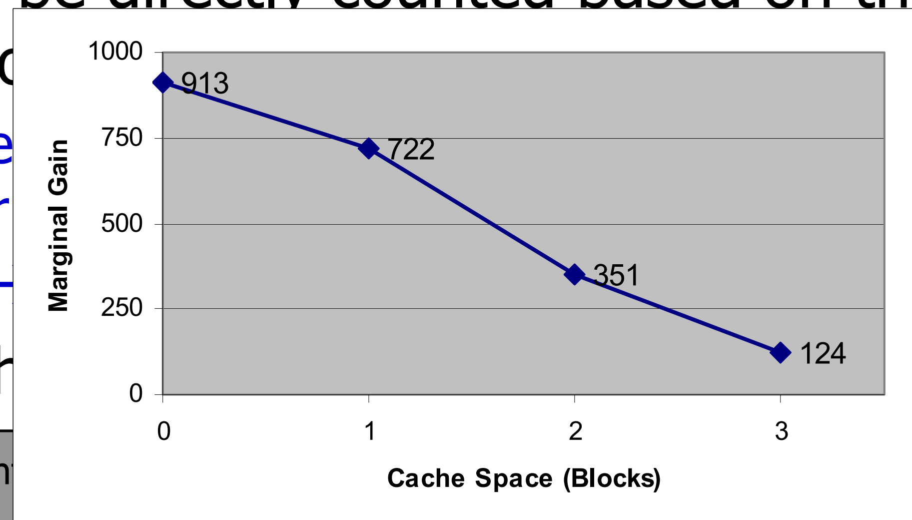
32-KB 8-way Set-Associative (bzip2+gcc+swim+mesa+vortex+vpr+twolf+iu)

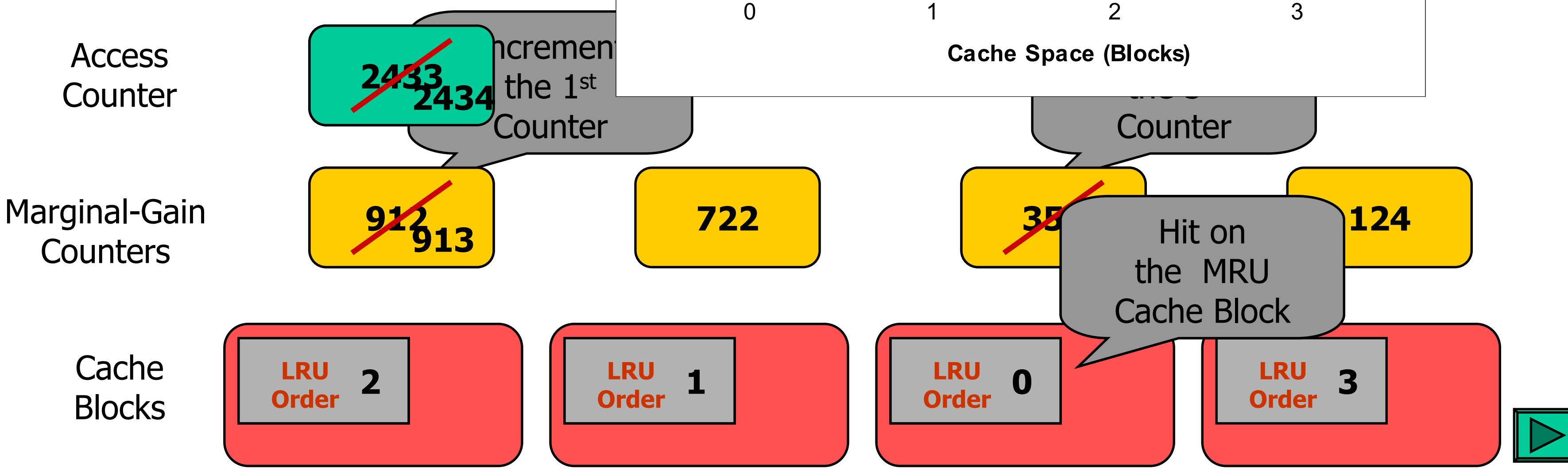
BUT…

- Processes to execute are only known at run-time
  - Users decide what applications to run
  - Scheduling/Partitioning decisions should be made at run-time
- The behavior of a process changes over time
  - Applications have different phases
  - Miss-rates curves (and marginal gains) may change over an execution
- Cache configurations are different for systems
  - Miss-rate curves (and marginal gains) are different for systems
- Need an on-line estimation of miss-rate curves (and marginal gains)

On-Line Estimation of Marginal Gains: Fully-Associative Caches

- Marginal gains can be directly counted based on the temporal ordering c)
  - Use one counter pe e blocks) and one for
  - Hit on the i^{th} MRU –
- Example: a FA cach



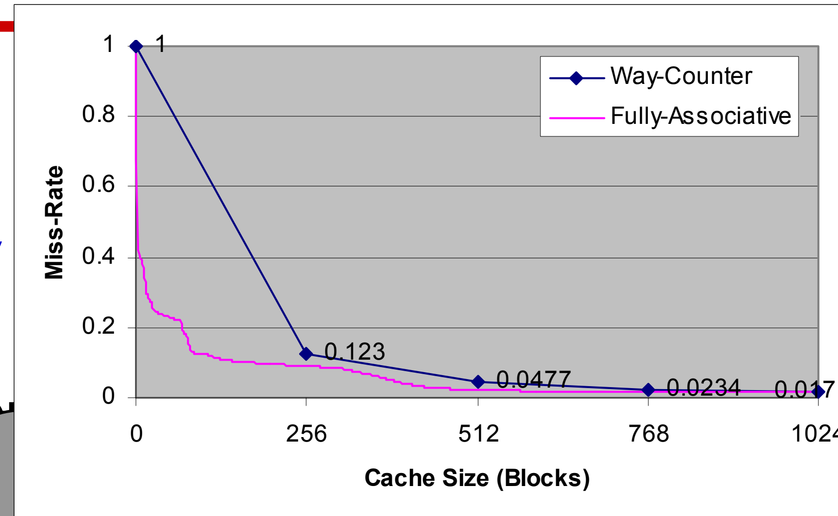


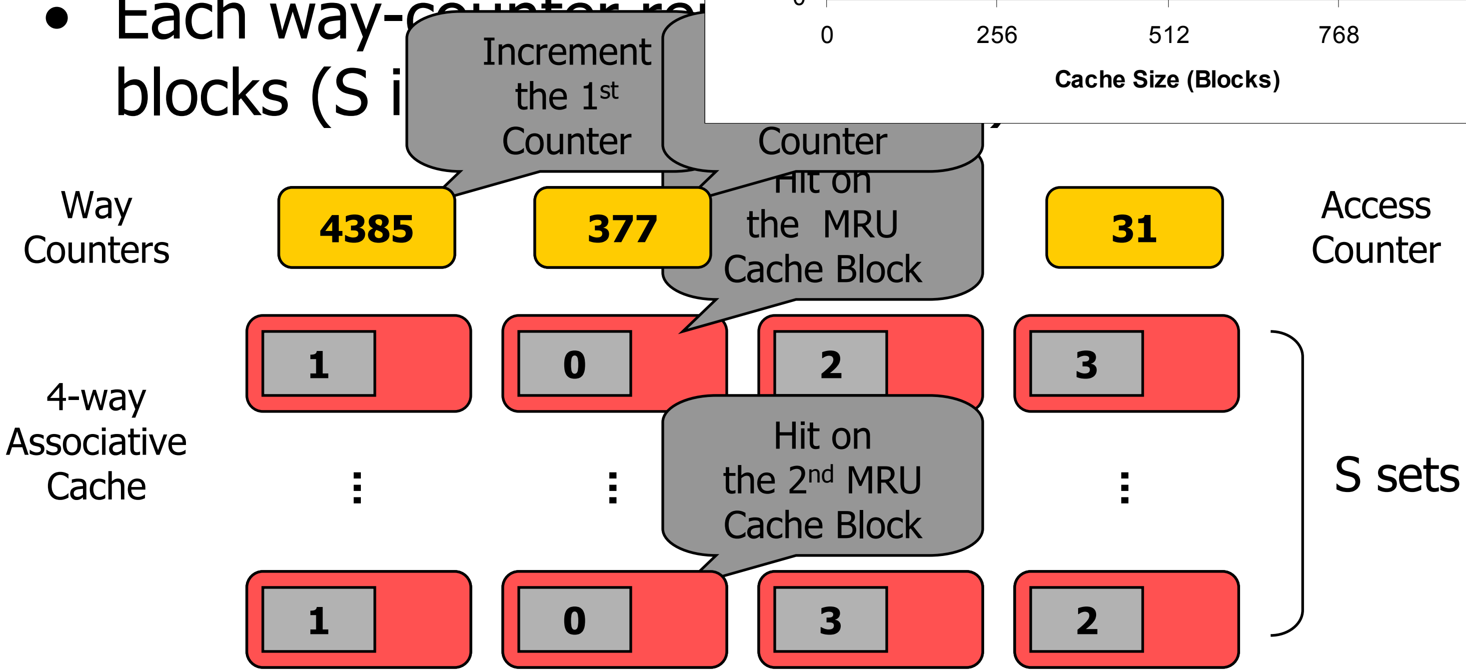
BUT…

- Most caches are SET-ASSOCIATIVE
  - Except main memory
  - Usually up to 8-way associative
- Set-associative caches only maintain temporal ordering within a set
  - No global temporal ordering
- Cannot use block-by-block temporal ordering to obtain marginal gains for fully-associative caches

Way-Counters

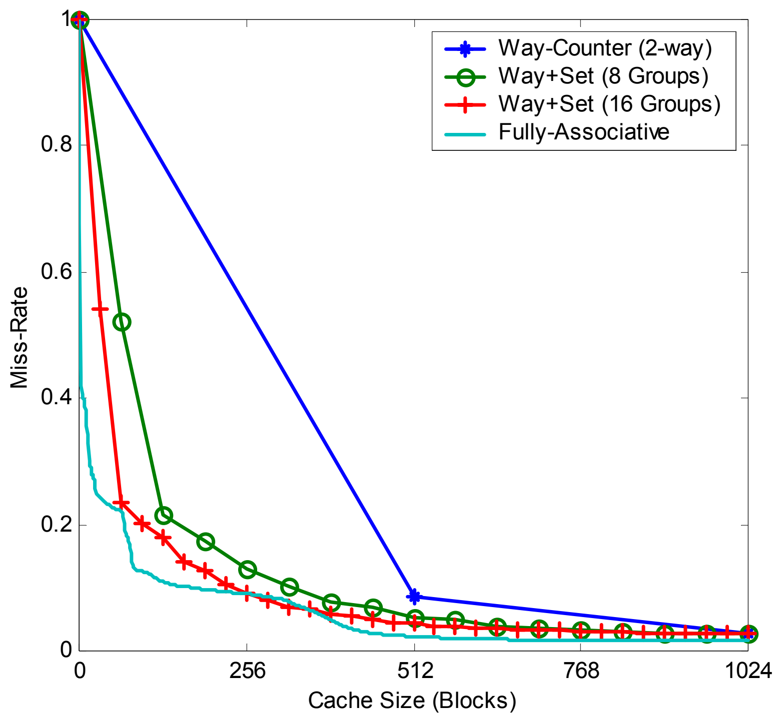
- Way-Counters
  - Use the existing LRU
  - One counter per way
  - Hit on the i^{th} MRU →
- Each way-counter re... more blocks (S i...

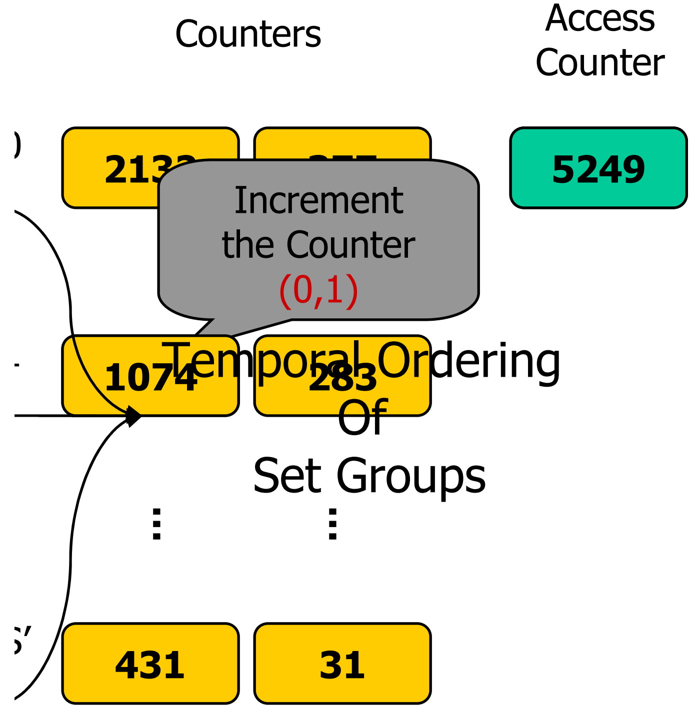


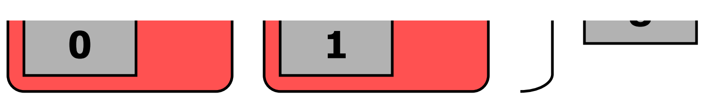


Way+Set Counters

- Use more counters for more detailed information
  - Maintain the LRU information of sets
  - MRU set → Increment counter(i,j)







Summary

- Caches should be managed more carefully considering the effect of space/time-sharing
  - Cache Partitioning
  - Memory-Aware Scheduling
- Miss-rate curves provide very relevant information for scheduling and partitioning
  - Enables us to predict the effect of varying the cache space
  - Useful for any tradeoff between performance and space (power)
- On-line counters can estimate miss-rate curves at run-time
  - Use the temporal ordering of blocks to predict miss-rates for smaller caches
  - Works for both fully-associative and set-associative caches