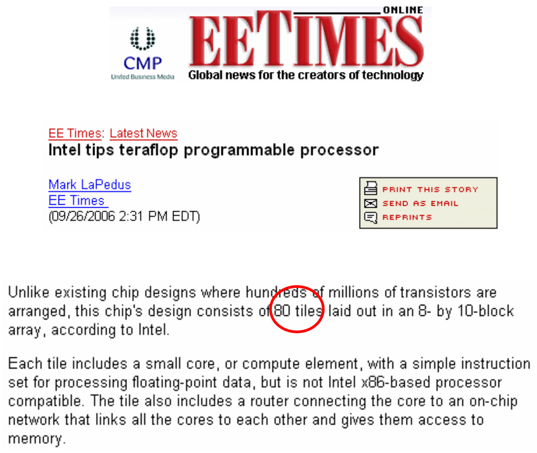
Memory Hierarchy Design Issues in Many-Core Processors

Sangyeun Cho

Dept. of Computer Science
University of Pittsburgh


CMP
United Business Media
EETIMES ONLINE
Global news for the creators of technology

EE Times: Latest News
Intel tips teraflop programmable processor

Mark LaPedus
EE Times
(09/26/2006 2:31 PM EDT)

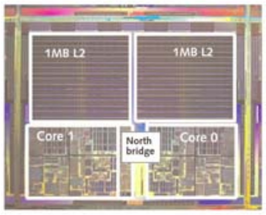
PRINT THIS STORY
SEND AS EMAIL
REPRINTS

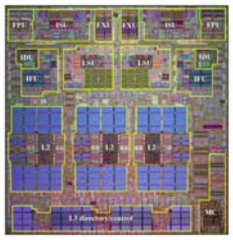
Unlike existing chip designs where hundreds of millions of transistors are arranged, this chip's design consists of 80 tiles laid out in an 8- by 10-block array, according to Intel.

Each tile includes a small core, or compute element, with a simple instruction set for processing floating-point data, but is not Intel x86-based processor compatible. The tile also includes a router connecting the core to an on-chip network that links all the cores to each other and gives them access to memory.

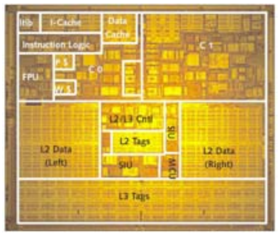
Multicores are Here

AMD Opteron Dual-Core



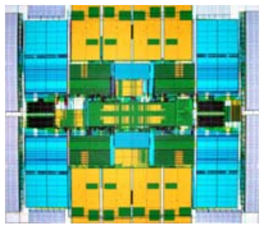


IBM Power5

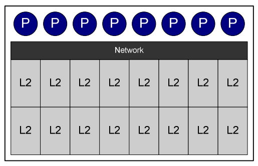


SUN UltraSPARC IV+

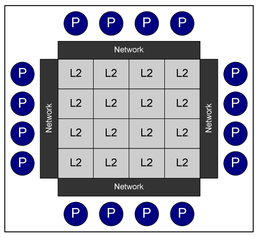
SUN UltraSPARC T1



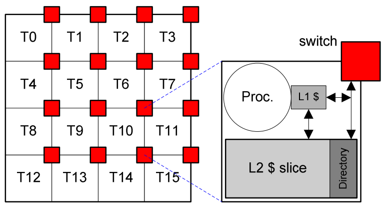
Tomorrow's Processors?



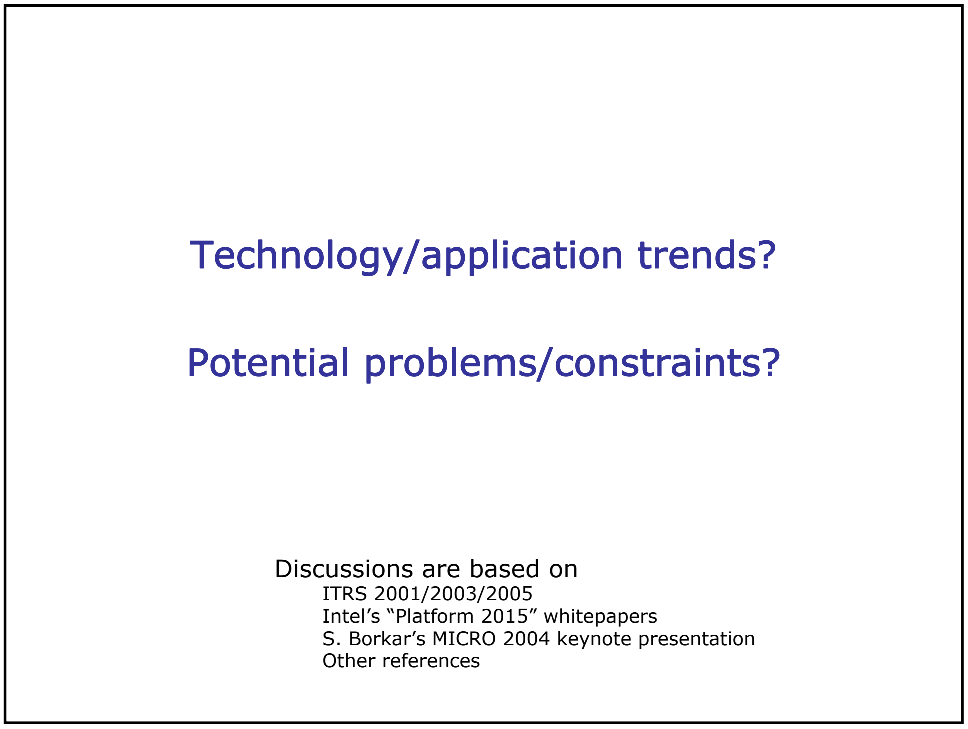
Dance-hall organization



Round-table organization



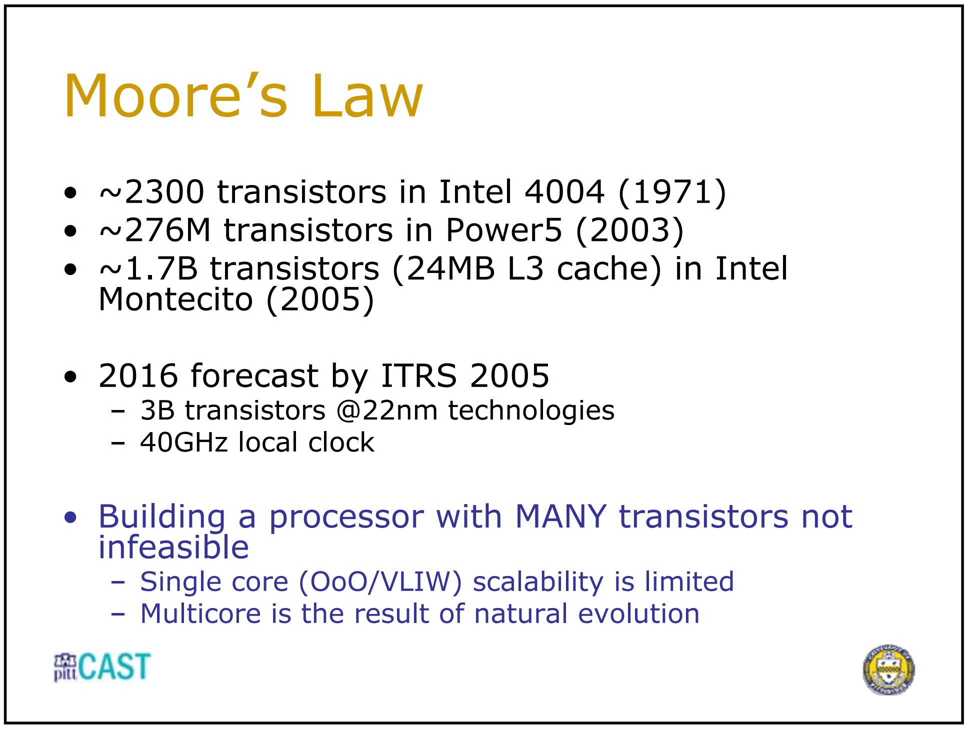
Tiled organization



Technology/application trends?

Potential problems/constraints?

Discussions are based on
- ITRS 2001/2003/2005
- Intel's "Platform 2015" whitepapers
- S. Borkar's MICRO 2004 keynote presentation
- Other references



Moore's Law

- ~2300 transistors in Intel 4004 (1971)
- ~276M transistors in Power5 (2003)
- ~1.7B transistors (24MB L3 cache) in Intel Montecito (2005)

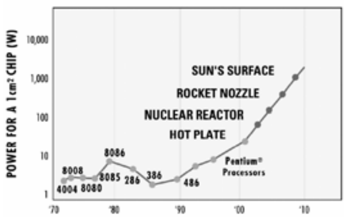
- 2016 forecast by ITRS 2005
  - 3B transistors @22nm technologies
  - 40GHz local clock

- Building a processor with MANY transistors not infeasible
  - Single core (OoO/VLIW) scalability is limited
  - Multicore is the result of natural evolution

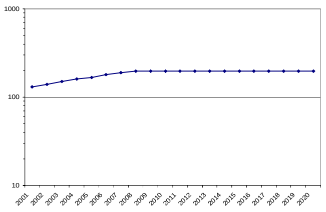
pittCAST

Power Trend

Power trend, unconstrained



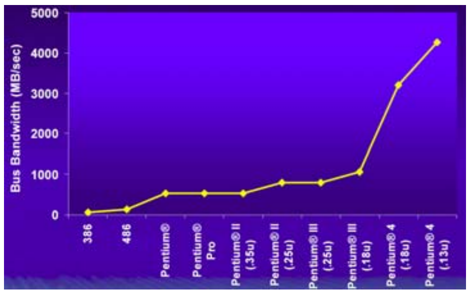
Max. allowed power



- Power drivers
  - # of transistors
  - Faster clock frequency
  - Increased leakage power
- Power density (W/cm2)
  - Related with temperature
  - Becomes more critical
  - Perf. & reliability issues

pittCAST

Bandwidth Trend

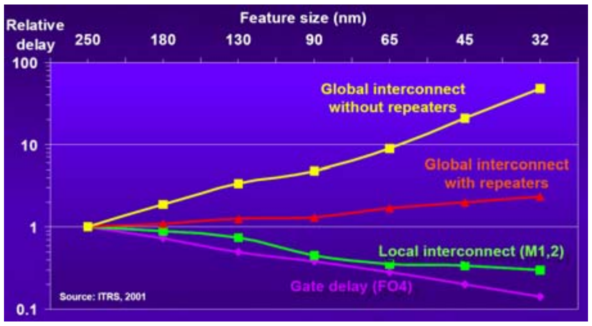


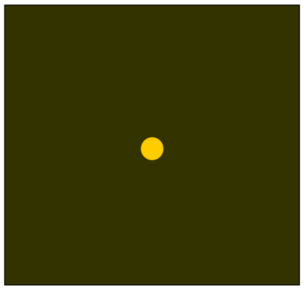
- Today's processor bandwidth is 2~20GB/s.
- Limited by
  - # of pins
  - Bandwidth per pin

- Bandwidth drivers
  - # of processors
  - Faster clock frequency
- Two fronts
  - Off-chip bandwidth
  - On-chip interconnect bandwidth

pittCAST

On-Chip Wire Speed





- Scaling leads to faster devices (transistors)
- Scaling however leads to slower global wires (increased RC delay)

- Possible implications
  - Simpler processor cores
  - On-chip switched network
  - Non-uniform memory access latency

pittCAST

Yield & Reliability Issues

- Errors due to variations (e.g., V_{TH} variation)
  - Run-time dependent
  - Reserving larger margins means lower yield
- Traditional test methods not enough
  - Burn-in/I_{DDQ} less effective
- Time-dependent device degradation
  - ~9% SNM degradation/3yr in SRAM due to NBTI
  - Electromigration, TDDB, …
- Soft error
  - FIT: ~8% degradation (bit/generation)

pittCAST

Application Requirements

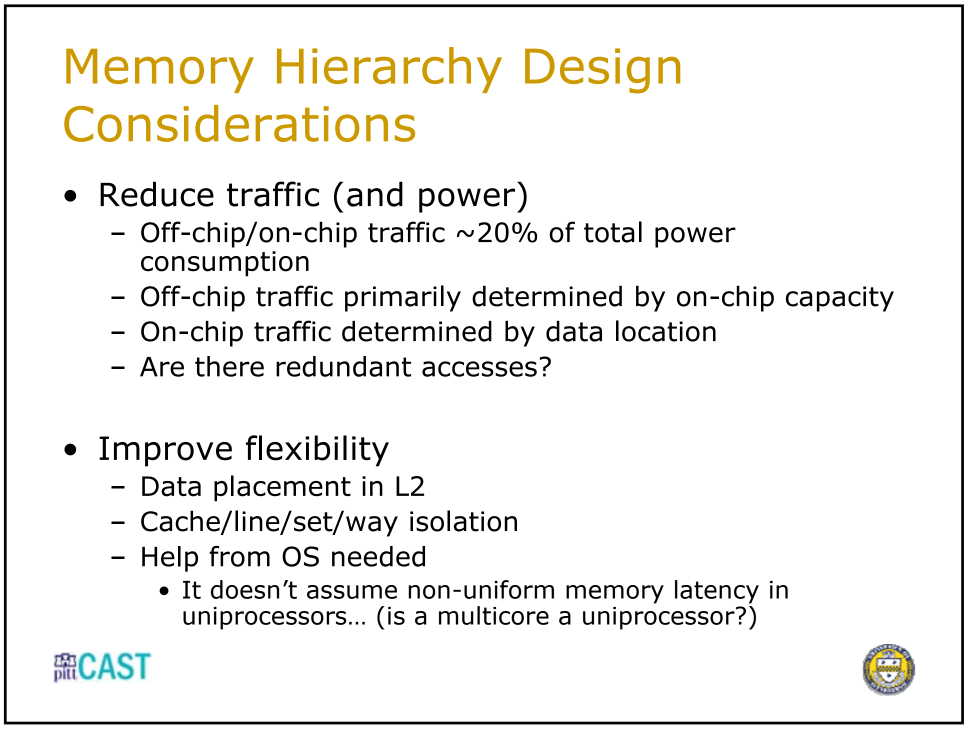
- Applications' performance demand growing
- RMS applications (Intel's term)
  - Recognition
  - Mining
  - Synthesis
- More multimedia applications
  - Games
  - Animations
- Pradip Dubey (Intel)
  - "The era of tera is coming quickly. This will be an age when people need teraflops of computing power, terabits of communication bandwidth, and terabytes of data storage to handle the information all around them."

pittCAST

Issues Summary

- We must keep scaling performance @Moore's law
- Power consumption
  - Every component design must (re-)consider power consumption
- Power density
  - Thermal management a must (but not sufficient)
  - Design/software methods for low temperature further needed
- Off-chip/on-chip bandwidth requirement
  - High-speed/low-power I/O
  - Larger on-chip memory (e.g., L2)
  - Package-level memory integration may become more interesting
- Wire delay dominance
  - Smaller cores
  - Non-uniform memory latency (i.e., hierarchy at same level)
- Yield/reliability
  - Microarchitectural provisions for yield/reliability improvement a must
  - Dynamic self-test/diagnosis/reconfiguration/adapt

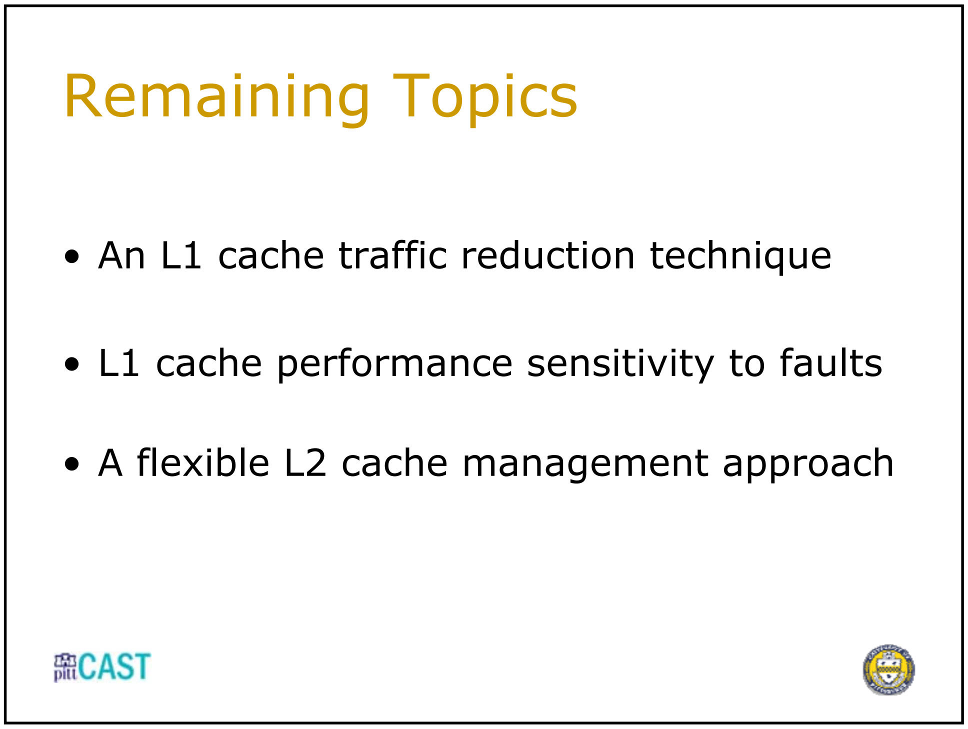
pittCAST



Memory Hierarchy Design Considerations

- Reduce traffic (and power)
  - Off-chip/on-chip traffic ~20% of total power consumption
  - Off-chip traffic primarily determined by on-chip capacity
  - On-chip traffic determined by data location
  - Are there redundant accesses?
- Improve flexibility
  - Data placement in L2
  - Cache/line/set/way isolation
  - Help from OS needed
    - It doesn't assume non-uniform memory latency in uniprocessors… (is a multicore a uniprocessor?)

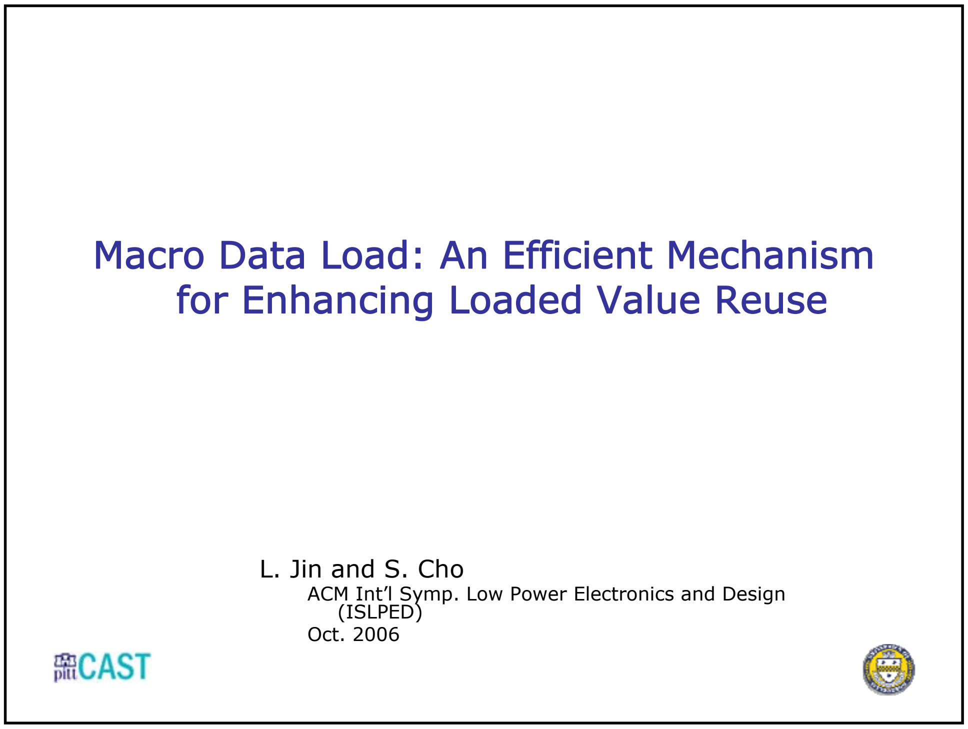
CAST



Remaining Topics

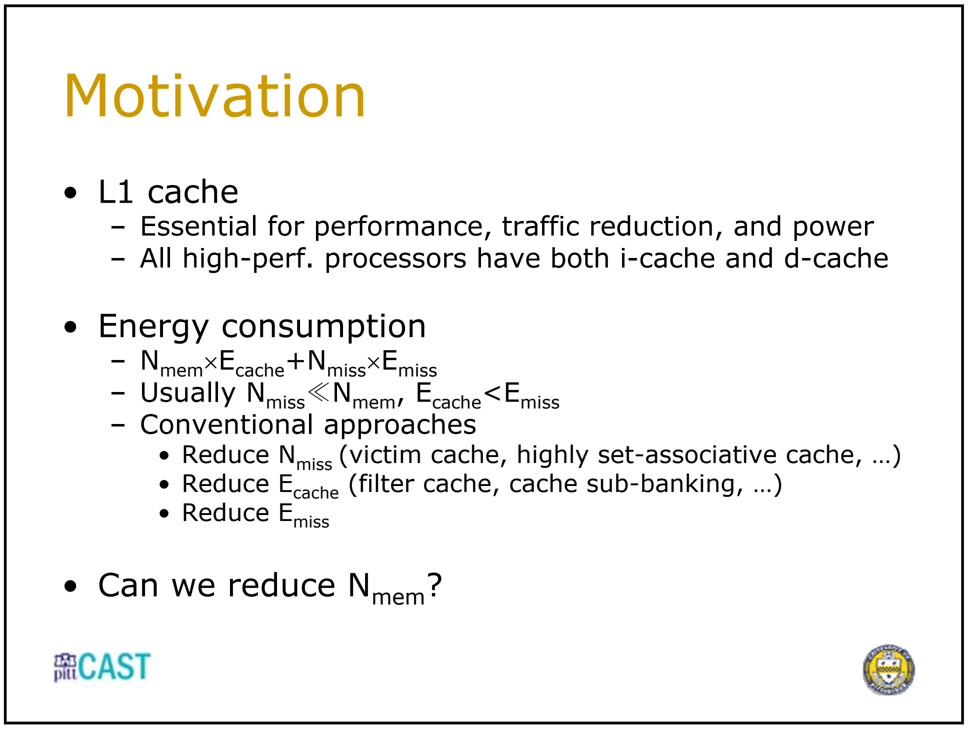
- An L1 cache traffic reduction technique
- L1 cache performance sensitivity to faults
- A flexible L2 cache management approach

CAST



Macro Data Load: An Efficient Mechanism for Enhancing Loaded Value Reuse

L. Jin and S. Cho
ACM Int’l Symp. Low Power Electronics and Design (ISLPED)
Oct. 2006



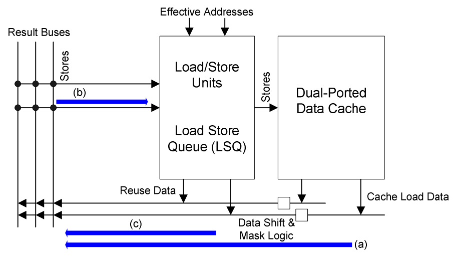
Motivation

- L1 cache
  - Essential for performance, traffic reduction, and power
  - All high-perf. processors have both i-cache and d-cache
- Energy consumption
  - $N_{mem} \times E_{cache} + N_{miss} \times E_{miss}$
  - Usually $N_{miss} \ll N_{mem}$, $E_{cache} < E_{miss}$
  - Conventional approaches
    - Reduce N_{miss} (victim cache, highly set-associative cache, …)
    - Reduce E_{cache} (filter cache, cache sub-banking, …)
    - Reduce E_{miss}
- Can we reduce N_{mem}?

L1 Traffic Reduction Ideas

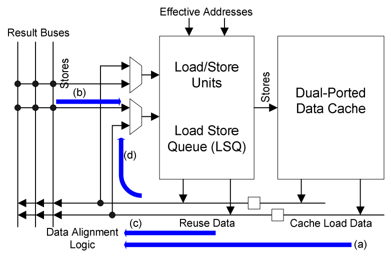
- Store-to-load forwarding
  - Usually needed for correctness in OoO engine
  - Implemented in LSQ
  - Design pipeline in such a way that cache is not accessed if the desired value is in LSQ
- Load-to-load forwarding ("loaded value reuse")
  - A loaded value may be necessary again soon
  - Use a separate structure or LSQ
- Silent stores
  - Stores that write a same value again
  - Identify, track, and eliminate silent stores
  - Lepak and Lipasti, ASPLOS 2002

Store-to-Load Forwarding



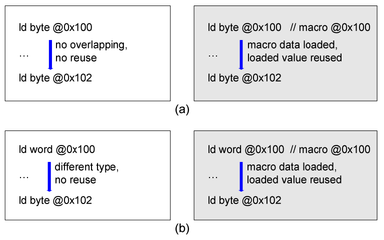
- Basic idea
  - Stores are kept in Load Store Queue (LSQ) until they are committed
  - A load dependent on a previous store may find the value in LSQ
- Often, a load accesses LSQ and cache together for higher performance
  - One can re-design pipeline so that LSQ is looked up before cache is accessed
  - How to deal with performance impact?

Load-to-Load Forwarding



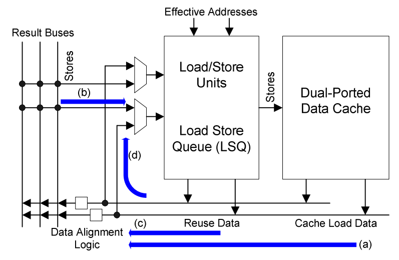
- Basic idea
  - Loaded values are kept in Load Store Queue (LSQ)
  - A load targeting a value previously loaded may find the value in LSQ
- Related work
  - Nicolaescu et al., ISLPED 2003

Macro Data Load



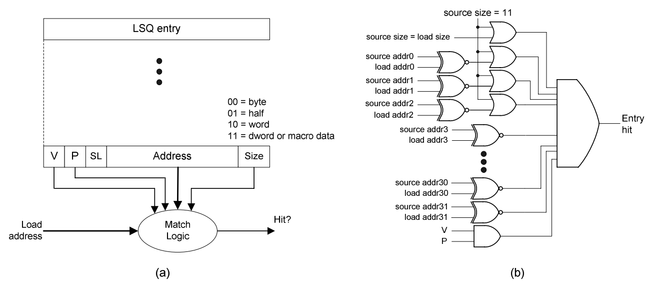
- Goal
  - Maximize loaded value reuse
- Idea
  - Bring full data (64 bits) regardless of load size
  - Keep it in LSQ
  - Use partial matching and data alignment
- Essentially, we want to exploit spatial locality present in cache line

Macro Data Load, cont'd



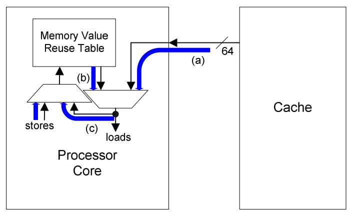
- Architectural changes
  - Relocated data alignment logic
  - Sequential LSQ-cache access
- Net impact
  - LSQ becomes a small fully associative cache with FIFO replacement

Macro Data Load, cont'd



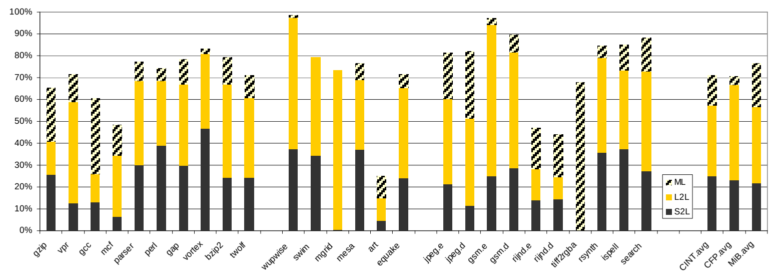
- Architectural changes
  - Relocated data alignment logic
  - Sequential LSQ-cache access
- Net impact
  - LSQ becomes a small fully associative cache with FIFO replacement

Idealized Limit Study



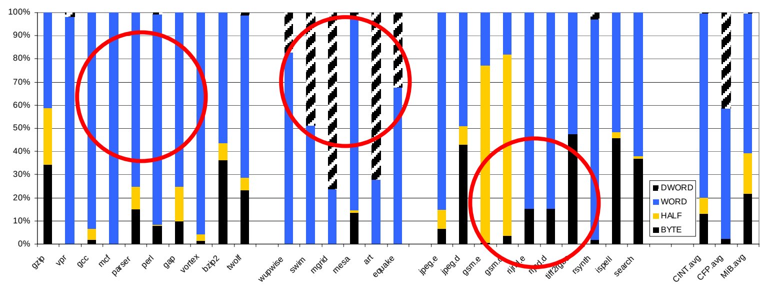
- MVRT (Memory Value Reuse Table)
  - N entries (parameter)
  - Tracks store-to-load (S2L), load-to-load (L2L), and macro data load (ML)
- Simple, idealized processor model
  - No branch mis-prediction; single-issue pipeline

Overall Result



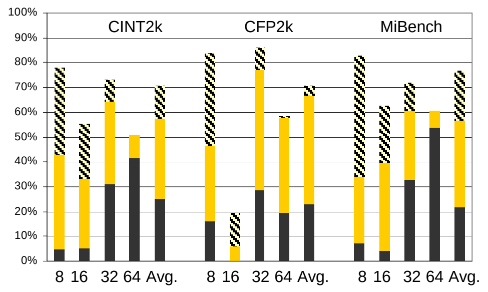
- Assuming 256-entry buffer size (maximum in our study)
- Up to over 70% of accesses are redundant
- Most programs have significant reuse opportunities
  - In certain cases, reuse distance is short and data footprint is small (wupwise)
- ML consistently boosts loaded value reuse (40~60% in CINT and MiBench)

Load Size Mix



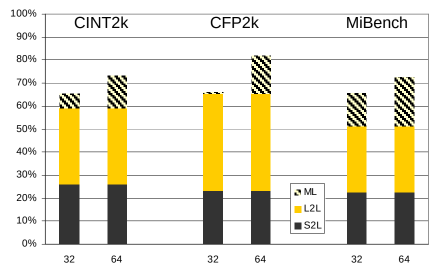
- CINT2k
  - Many word (32-bit) accesses
- CFP2k
  - Relatively frequent long-word (64-bit) accesses
- MiBench
  - More frequent half (16-bit) and byte (8-bit) accesses

Per-Type Reuse



- 8-/16-bit macro data reuse is high
  - Many word (32-bit) accesses
- CFP2k
  - Relatively frequent long-word (64-bit) accesses
- MiBench
  - More frequent half (16-bit) and byte (8-bit) accesses

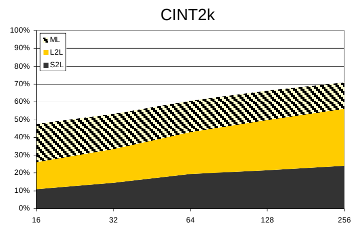
On Different Machine Width

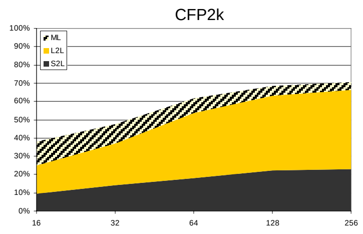


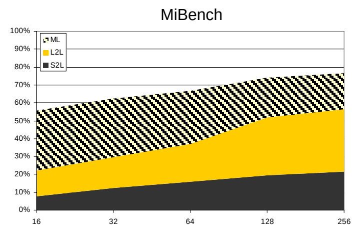
- We considered running a 64-bit binary on a 64-bit machine
- Consider two cases:
  - Running a 32-bit binary on a 32-bit machine
  - Running a 32-bit binary on a 64-bit machine

pittCAST

Sensitivity to Buffer Size







- When MVRT size = 16…
- When MVRT size = 32…
- When MVRT size = 64…

pittCAST

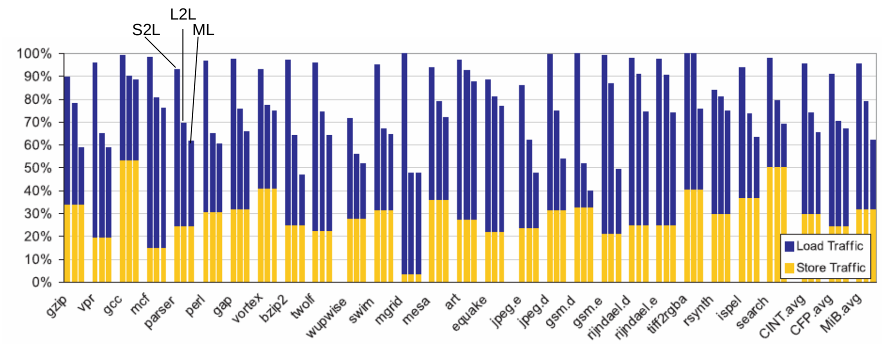
Performance Study Setup

- Processor
  - 4-issue OoO w/ 64-entry
- Caches
  - L1: 32kB, 2-way, 64B line, 2-cycle access
  - L2: 2MB, 4-way, 128B line, 10-cycle access
- Memory
  - 120 cycle latency

- Studied models
  - Traffic-optimized pipeline
    - Access LSQ before cache
  - Performance-optimized pipeline
    - Access cache and LSQ simultaneously

pittCAST

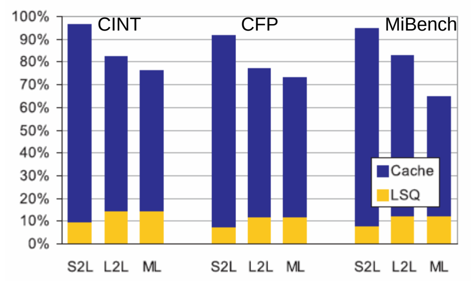
Traffic Reduction



- Significant reduction in load traffic
- Lost opportunities
  - Branch misspeculation (LSQ drain)
  - OoO execution (simultaneous LSQ accesses)

pittCAST

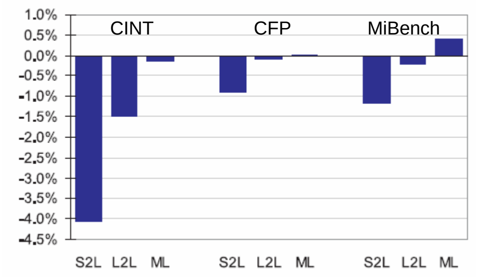
Energy Reduction



- LSQ energy should be considered
  - Previous works didn't…
- Up to 35% (MiBench) energy reduction

pittCAST

Performance Impact



- Performance impact should be minimized
  - Increased time = unwanted energy consumption
- ML performance impact is minimal

pittCAST

Summary

- L1 cache traffic reduction techniques evaluated
- ML (macro data load) outperforms previously proposed S2L and L2L schemes
- Smart traffic reduction techniques at all levels of memory hierarchy will become more and more important

pittCAST

Assessing the Impact of Defects in Cache Memory

H. Lee, S. Cho, and B. R. Childers

Submitted to 2nd Workshop on Architectural Reliability (WAR-2)
Sep. 2006

pittCAST

Motivation

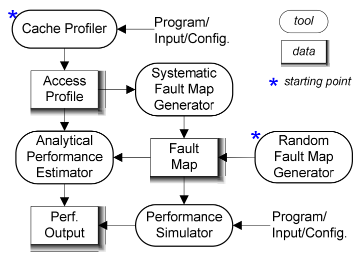
- Extreme technologies suffer…
  - Process variation – V_{TH}, leakage, …
  - Lifetime reliability – EM, NBTI, TDDB, …
  - Deteriorated testing capability – I_{DDQ}, burn-in, …
- Yield may stagger without large design & manufacturing margins
  - Burden on keeping Moore’s law
  - Profitability threatened
- Resilient designs will become critically important
  - Faults are masked
  - Graceful performance degradation

Our Approach

- Our focus is in microarchitecture & system
- Examine simple “delete” schemes
  - E.g., cache lines, sets, ways, etc.
  - Predictable performance degradation @low cost
- Explore “remap” schemes (still simple)
  - Share existing resource on demand
  - Slight performance degradation @small cost
- Devise system-level “management” schemes
  - Deleting & remapping decisions made intelligently
  - With little/no hardware addition
  - Synergistic architectural & system collaboration

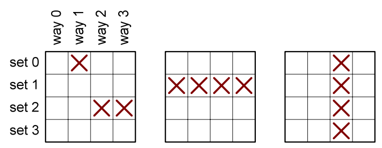
CAFÉ

- CAFÉ (Cache Fault Evaluation tool set)



(a)

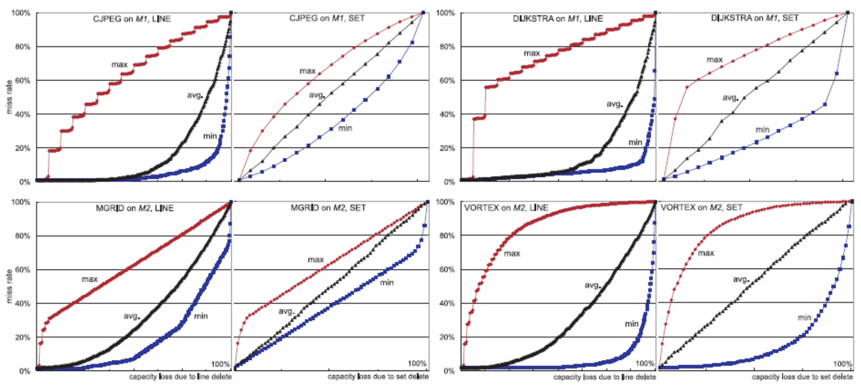
Evaluation flow

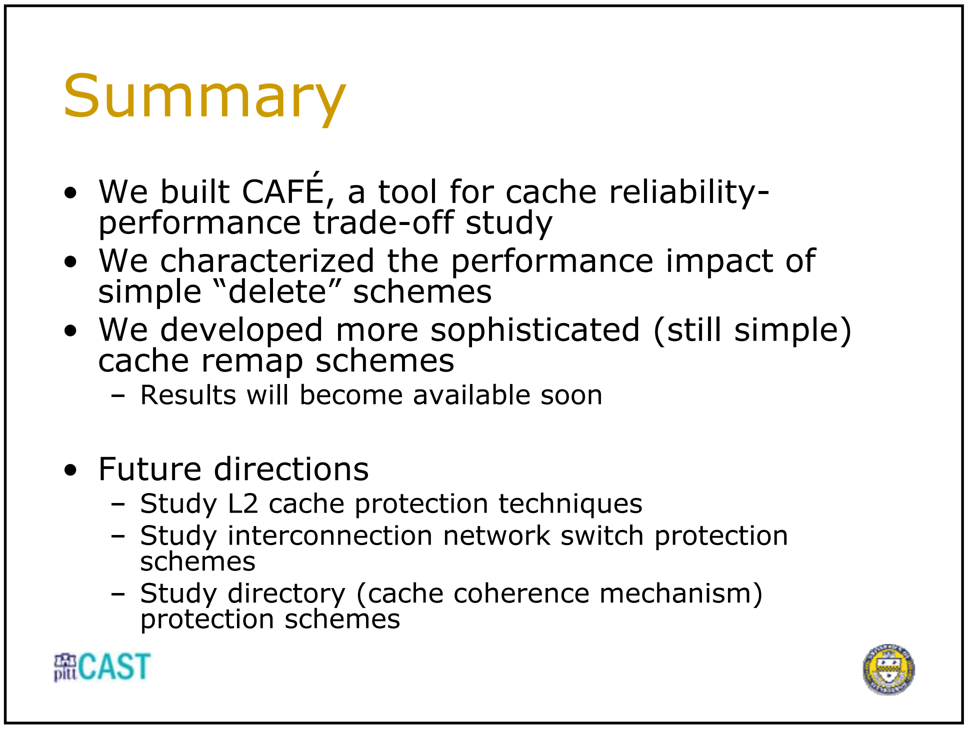


(b)

Fault map

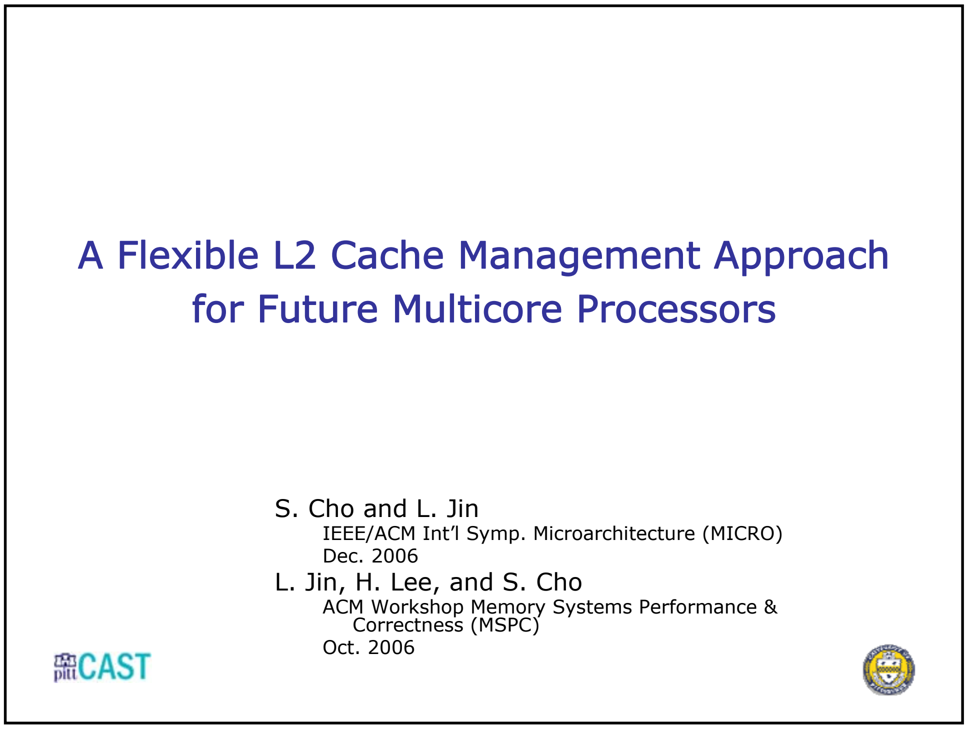
Performance of "Delete"





Summary

- We built CAFÉ, a tool for cache reliability-performance trade-off study
- We characterized the performance impact of simple "delete" schemes
- We developed more sophisticated (still simple) cache remap schemes
  - Results will become available soon
- Future directions
  - Study L2 cache protection techniques
  - Study interconnection network switch protection schemes
  - Study directory (cache coherence mechanism) protection schemes



A Flexible L2 Cache Management Approach for Future Multicore Processors

S. Cho and L. Jin
IEEE/ACM Int'l Symp. Microarchitecture (MICRO)
Dec. 2006

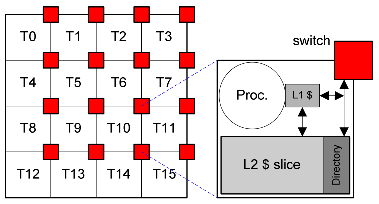
L. Jin, H. Lee, and S. Cho
ACM Workshop Memory Systems Performance & Correctness (MSPC)
Oct. 2006

L2 Cache Basics

- Physically indexed, physically tagged
  - Physical address determines location within cache
- In the range of 256kB~2MB in single core
  - 64B~256B cache line size
  - 4~8-way set associative
- OS approach to conflict miss reduction in L2 cache
  - Page coloring
  - Bin hopping
  - Best bin

pittCAST

CMP L2 Cache Management



- Tile-based multicore
- Conventional L2 cache management strategies
  - Treat each cache slice private to a core (“private” design)
  - Treat all cache slices shared by all (“shared” design)
    - Non-Uniform Cache Architectures (NUCA)
  - Hybrid private/shared design (“hybrid” design)
    - Group slices

pittCAST

Private Caching Scheme

- Assume L1 is "private"
- On an L1 cache miss
  - Access local L2 slice (always)
  - If hit
    - Be happy
  - If miss
    - Go to directory
    - See if wanted data is on-chip
      - Perform coherence action and get data
    - If data is missing, go to main memory, update directory
- (+) Low average hit latency
  - "Data attraction"
- (-) Low on-chip hit rate
  - Each processor core has limited caching space
- (-) Complex coherence protocol
  - Replication, unknown data location

Shared Caching Scheme

- Assume L1 is "private"
- On an L1 cache miss
  - Determine which cache slice to access
    - Usually, slice id is directly derived from address (e.g., cache line address % # slices)
  - Access data
    - If data is missing, go to main memory
- (+) Low on-chip miss rate
  - Fine distribution of data onto available cache slices
- (+) Simple and efficient coherence protocol
  - Data location is deterministic
- (-) Low average hit latency
  - Wanted data may be found in cache slice far off
- Current generation processors use a shared cache model
  - IBM Power4/5
  - Sun Microsystems Niagara
  - Intel Core Duo

Other Variants

- Clusters of private caches
  - Each private cache cluster comprises multiple shared caches
  - Huh et al., ICS 2005
- "Victim replication" – based on a shared design
  - Victims from L1 are copied to local L2 slice
  - Zhang and Asanovic, ISCA 2005
- "Cooperative caching" – based on a private design
  - Limit degree of sharing, evict globally unused blocks, exploit cache-to-cache transfer
  - Chang and Sohi, ISCA 2006

Proximity vs. Miss Rate

- Different schemes make different trade-off between proximity and on-chip miss rate
- Private cache
  - Favors proximity
- Shared cache
  - Favors miss rate
- Hybrid/variant schemes
  - Attempts to achieve both good proximity and low miss rate

What About Scalability/Flexibility?

- Future processors may include 100’s of cores and 100’s of cache slices
- Private cache
  - Scaling will not help single program – it won’t get more capacity
- Shared cache
  - More caches increase overall capacity
  - Average latency increases!
- How can we manage so many cache slices
  - Performance
  - Power
  - Reliability

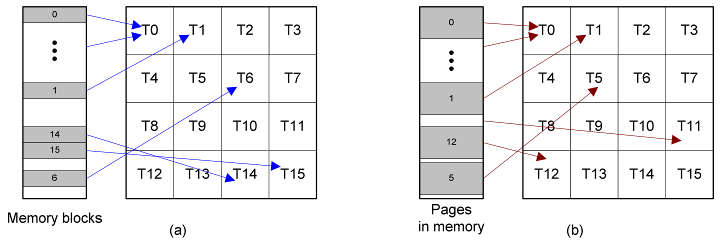
Our Goal

- Provide a flexible scheme
  - Good proximity (~private cache)
  - Good on-chip hit rate (~shared cache)
  - Cache slice isolation
    - Unnecessary slices can be turned off
    - Unreliable slices should be pulled out
- Architectural support
  - Mapping information look-up and maintenance
  - Light-weight performance monitor
- (System implementation)
  - Production-quality OS
  - Full-system simulation environment

Revisiting a Shared Design

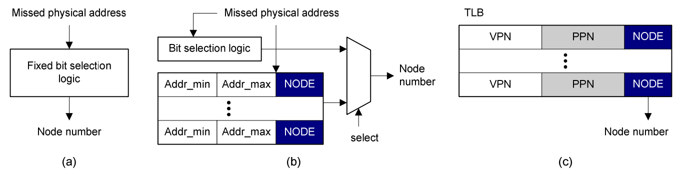
- L1 cache is private
  - Coherence information is distributed to L2 cache tags
  - Each L2 tag entry keeps a bit-map showing nodes keeping data
- L2 cache slices form a logical single cache
- Data to cache slice mapping
  - Cache line granularity
    - A very balanced way

Changing Granularity



- Benefit of mapping at page granularity
  - OS can map program's virtual pages to decide data location
  - Mapping creation at page allocation time
  - Mapping information size is more manageable
  - Data access behavior (e.g., sequential access) preserved

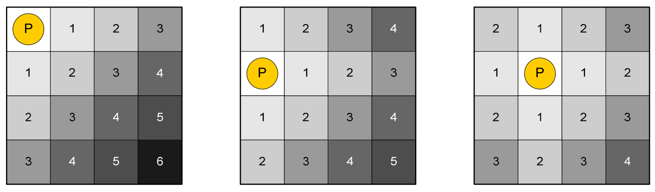
Carrying Mapping Information



- Simple bit selection method
- Region-based method
- Page table (TLB) method

pittCAST

Program-Data Proximity

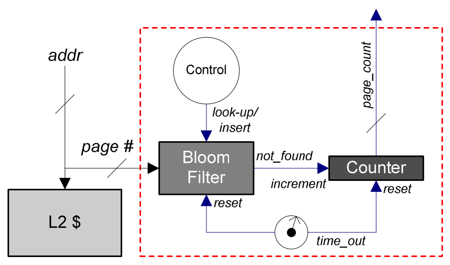


Case 1 (avg. distance = 3) Case 2 (avg. distance = 2.5) Case 3 (avg. distance = 2)

- Program & data location determine min. latency to bridge them
- "Tiers"
- Page spreading
  - Borrow cache space from nearby neighbors

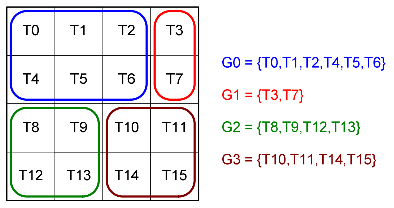
pittCAST

Cache Pressure



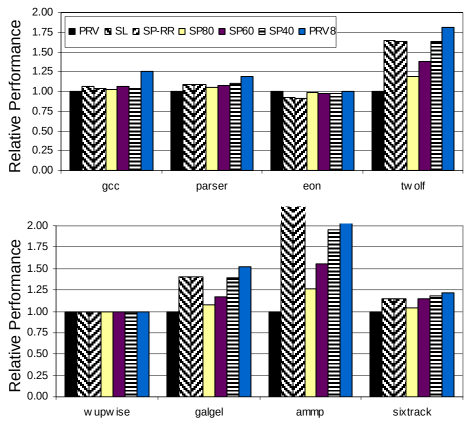
- We don’t want to overload a single slice
- Cache pressure = # of “active” pages × page size / cache slice size
- A performance monitor to estimate cache pressure is necessary

Virtual Multicore

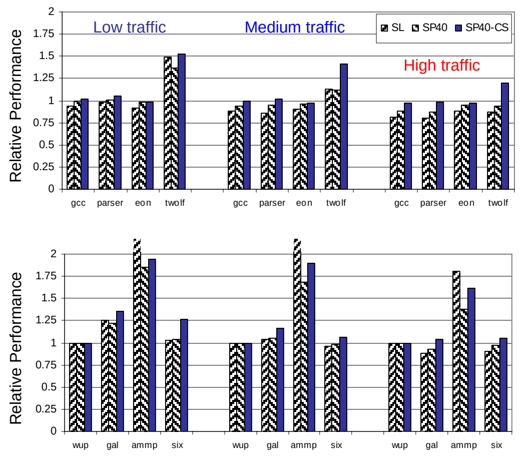


- Cluster processor & cache slices
- Processors in VM share their cache slices
- Coherence and data transfer traffic contained within VM

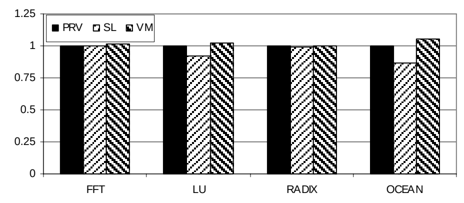
Single Program Performance



Under Different Traffic Load

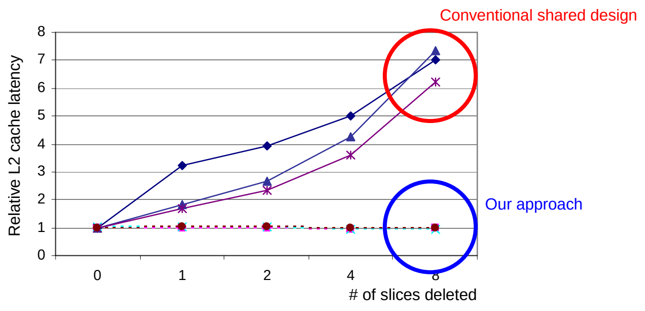


Parallel Workloads



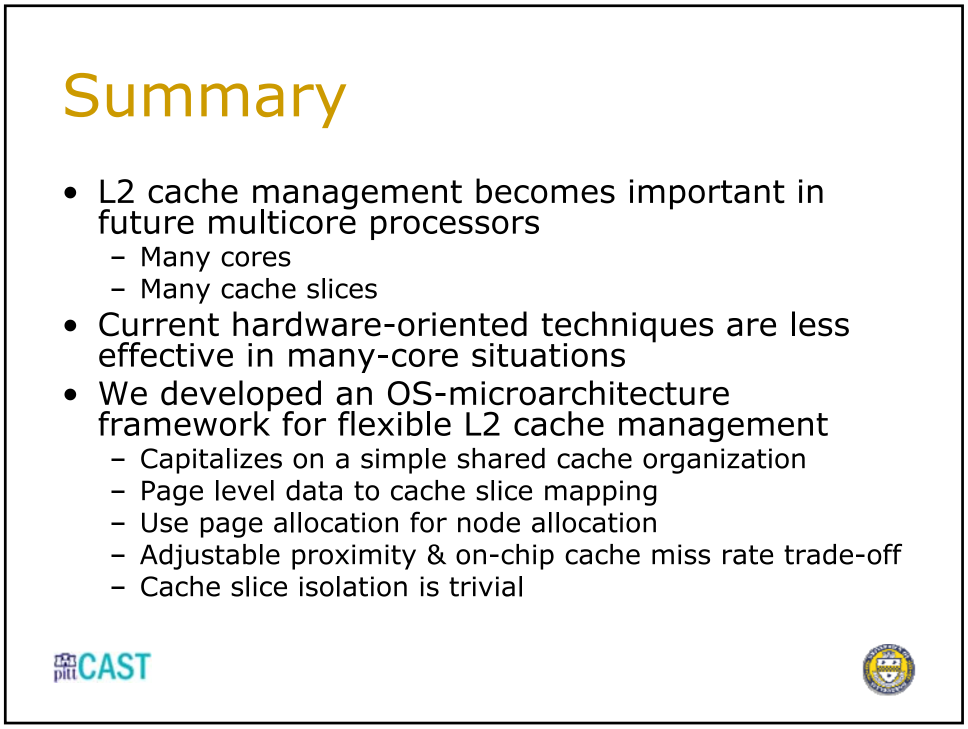
pittCAST

Deleting Cache Slice



- When cache slices need be deleted
  - Reliability issues (e.g., faults)
  - Power issues
- Conventional design suffers
- Our approach can simply avoid using certain cache slices

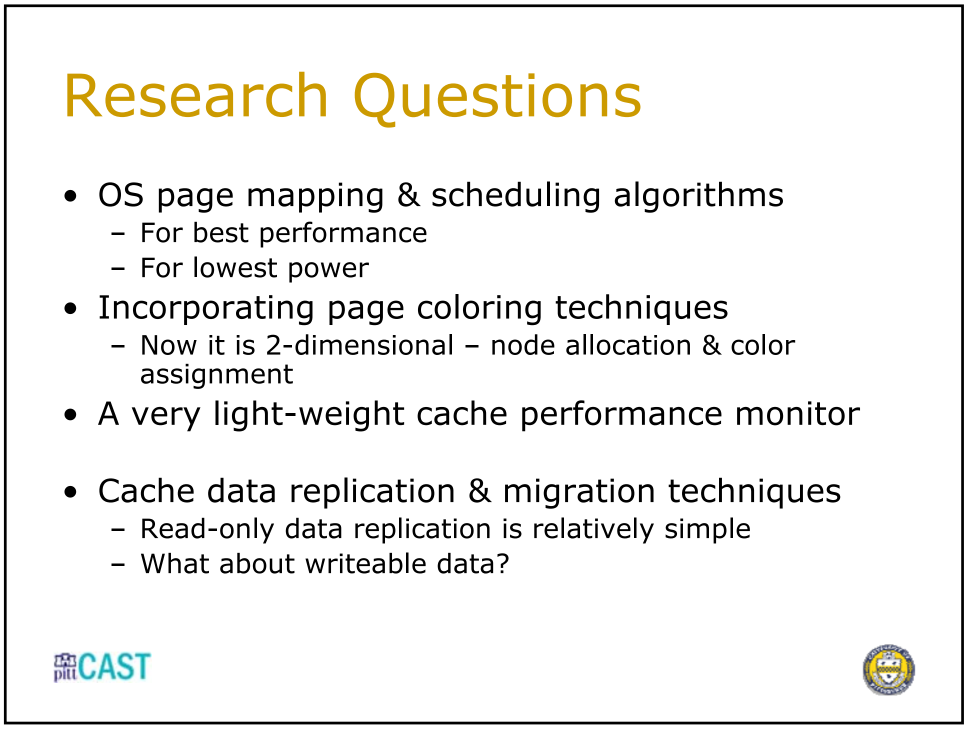
pittCAST



Summary

- L2 cache management becomes important in future multicore processors
  - Many cores
  - Many cache slices
- Current hardware-oriented techniques are less effective in many-core situations
- We developed an OS-microarchitecture framework for flexible L2 cache management
  - Capitalizes on a simple shared cache organization
  - Page level data to cache slice mapping
  - Use page allocation for node allocation
  - Adjustable proximity & on-chip cache miss rate trade-off
  - Cache slice isolation is trivial

pittCAST



Research Questions

- OS page mapping & scheduling algorithms
  - For best performance
  - For lowest power
- Incorporating page coloring techniques
  - Now it is 2-dimensional – node allocation & color assignment
- A very light-weight cache performance monitor
- Cache data replication & migration techniques
  - Read-only data replication is relatively simple
  - What about writeable data?

pittCAST

Homework

- Write a quicksort program and run it
  - Start from 100 integer numbers, 100, 99, …, 2, 1
  - Run the quicksort to sort the numbers to 1, 2, …, 99, 100
- Your job is to estimate the # of memory accesses of your program
- Refer to the description on the course (2001) website