Coordinating Accesses to Shared Caches in Multi-core Processors

Software Approach

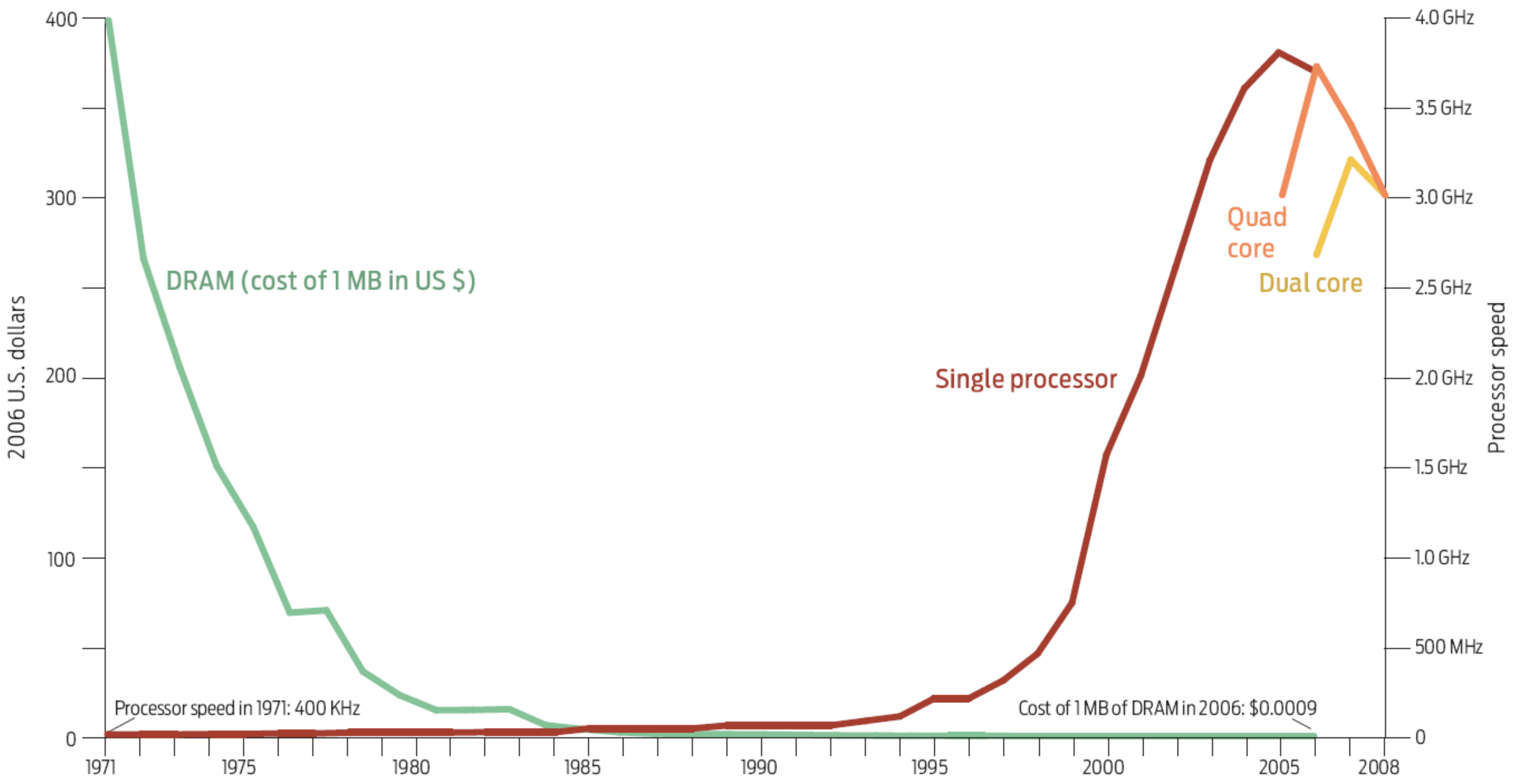
Xiaodong Zhang

Ohio State University

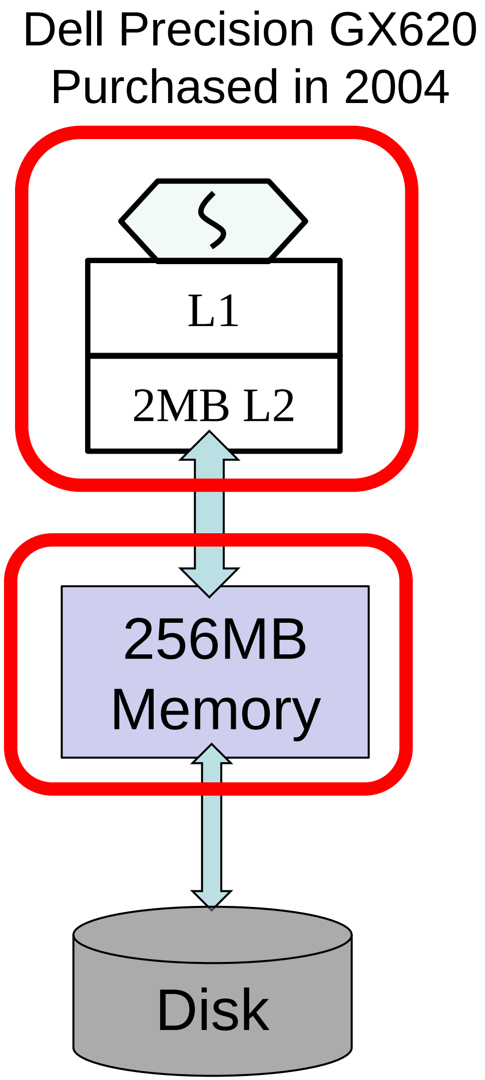
Collaborators: Jiang Lin, Zhao Zhang, Iowa State

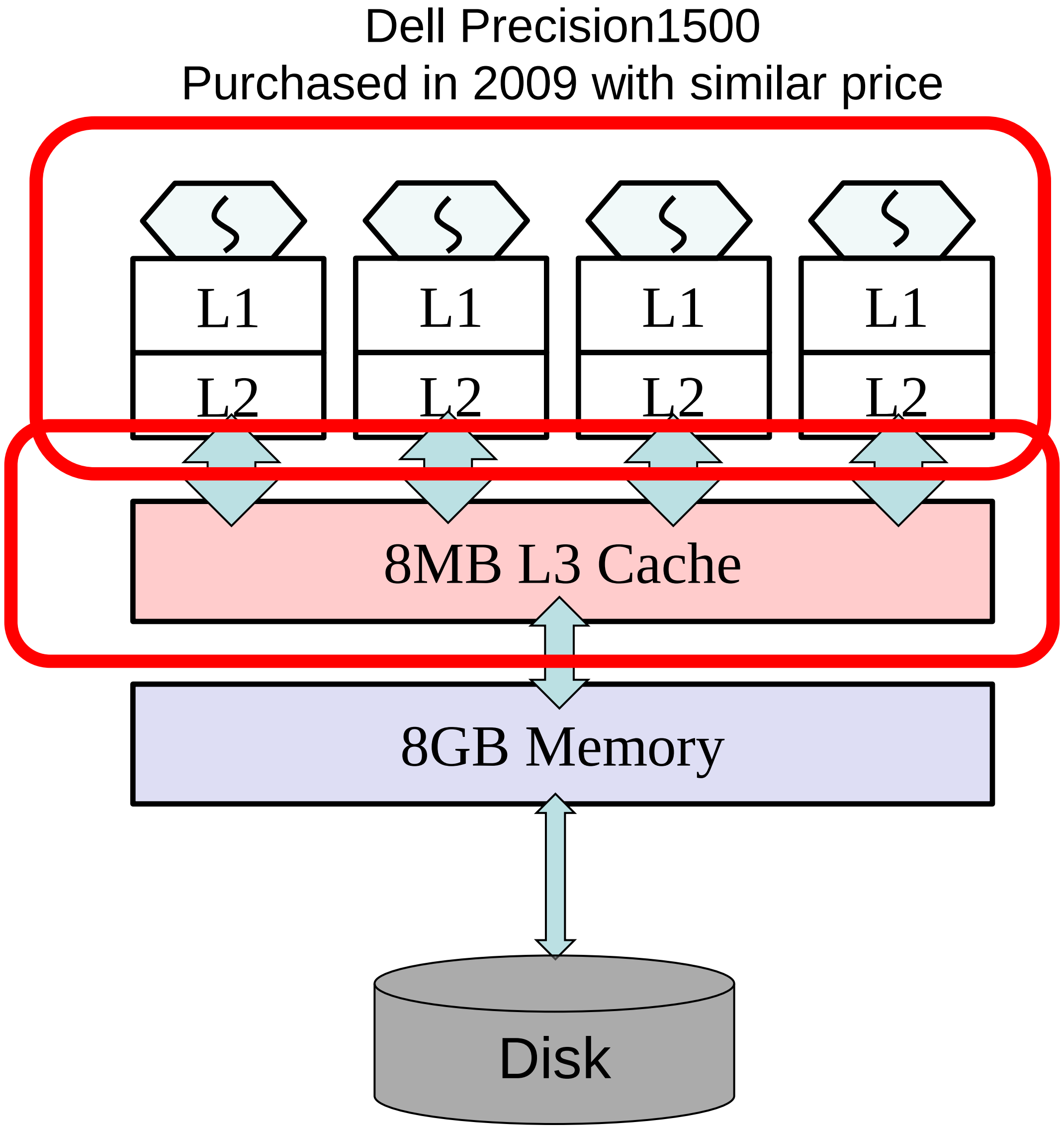
Xiaoning Ding, Qingda Lu, P. Sadayappan, Ohio State

Moore’s Law in 37 Years (IEEE Spectrum May 2008)



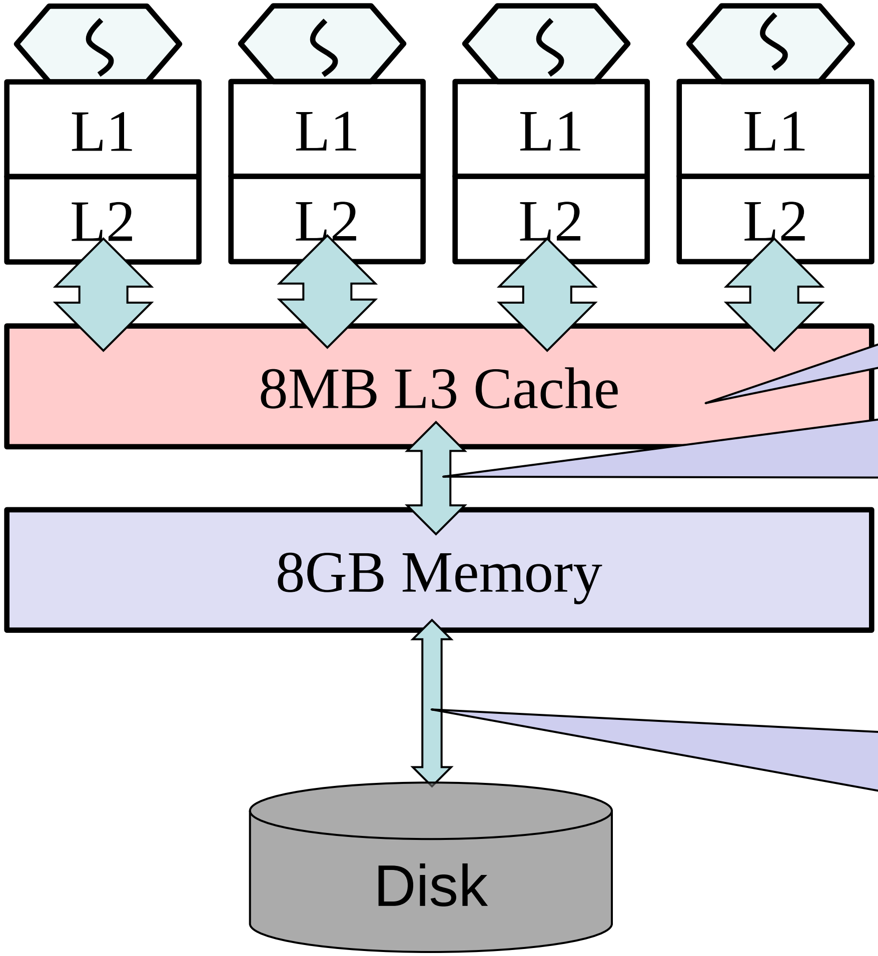
Architectural Changes in Computer Systems





Performance Issues w/ the Multicore Architecture

Performance of a multicore system is significantly limited by slow data accesses to memory and disks

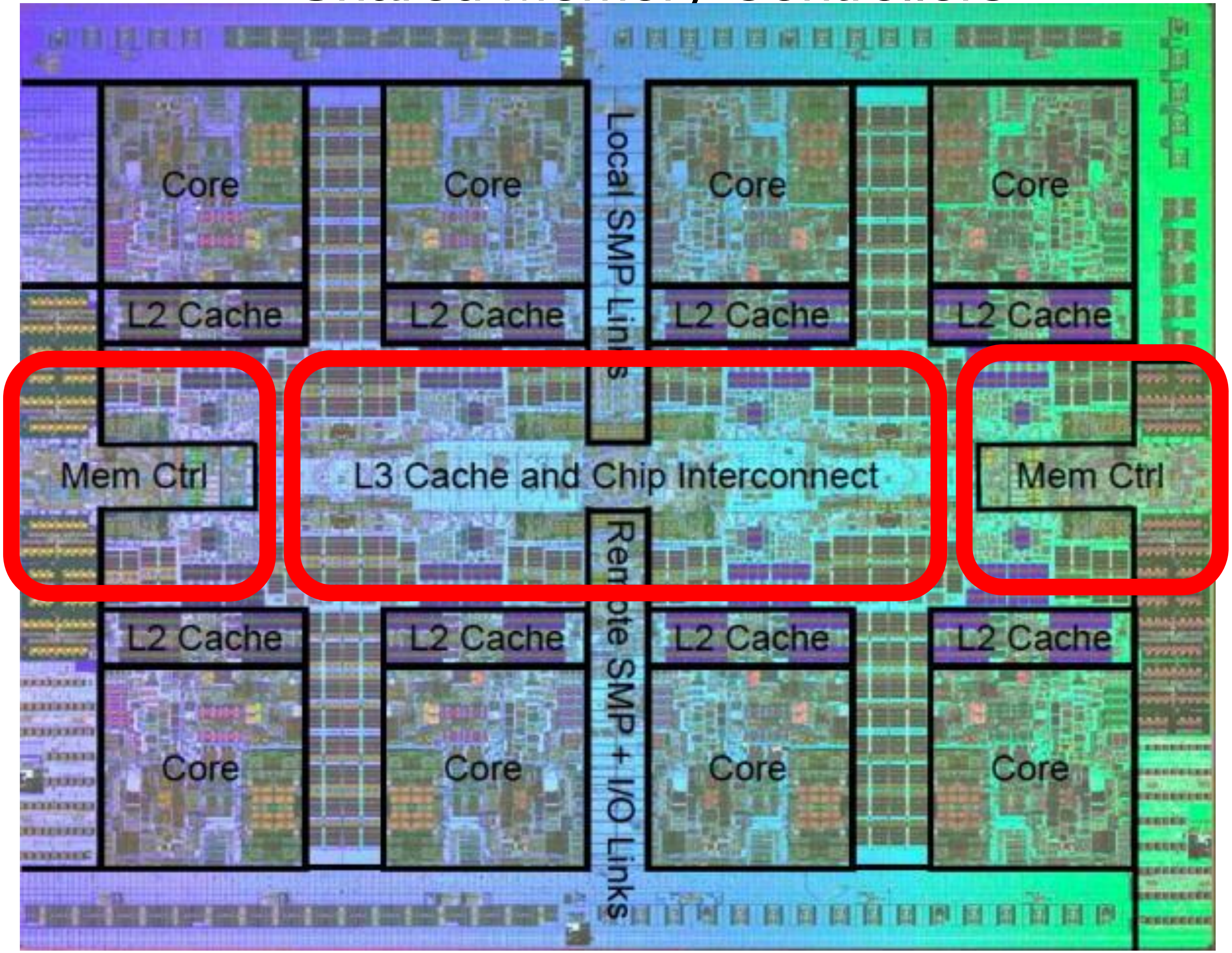


Cache Contention and Pollution:
Uncontrolled contention significantly increases cache misses

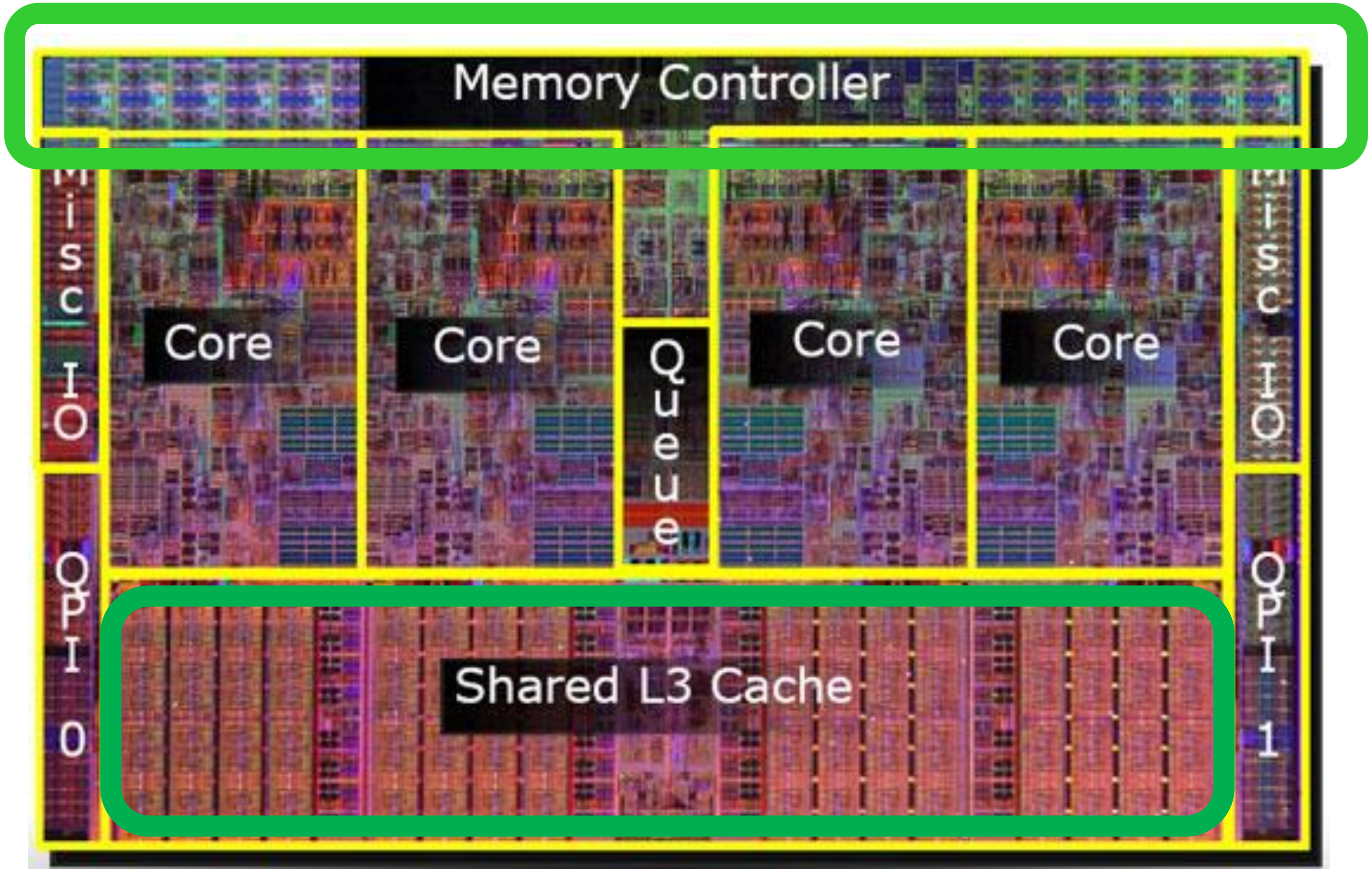
Memory Bus Congestion:
Limited memory bandwidth is competed for by multiple cores

"Disk Wall":
Limited storage throughput is shared by multiple cores

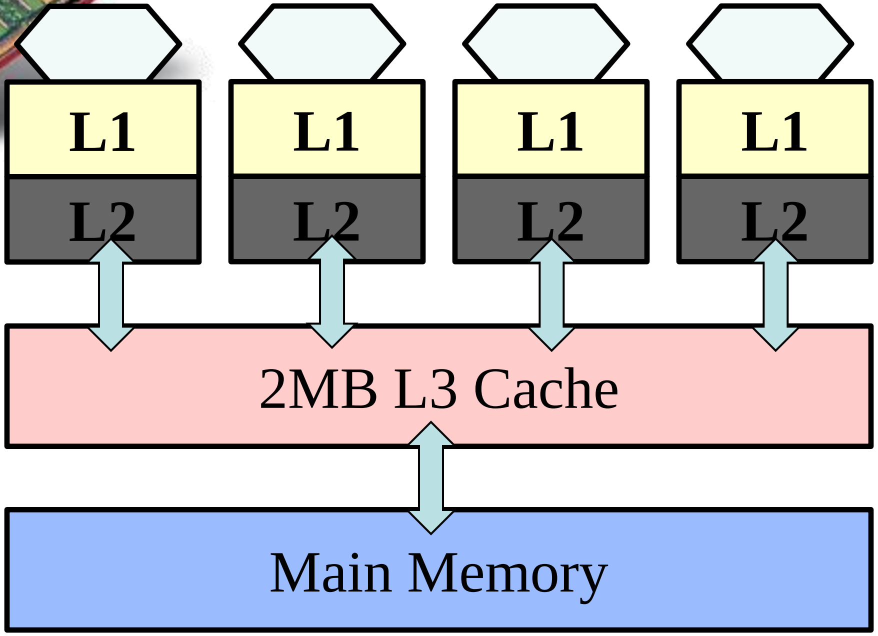
IBM Power 7 w/ Shared Last Level Cache (LLC) & Shared Memory Controllers



Intel Core i7 with Shared Last Level Cache (LLC) & Shared Memory Controller

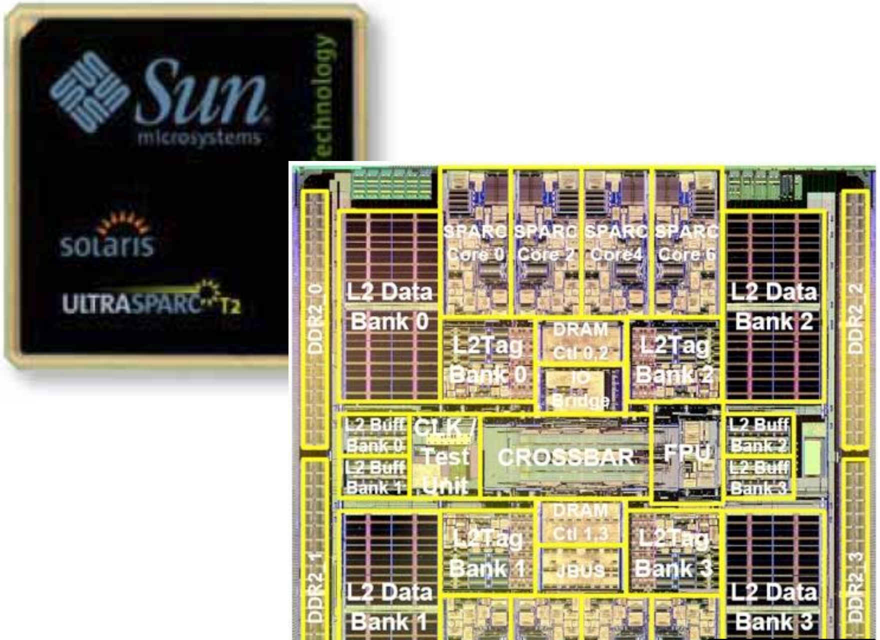


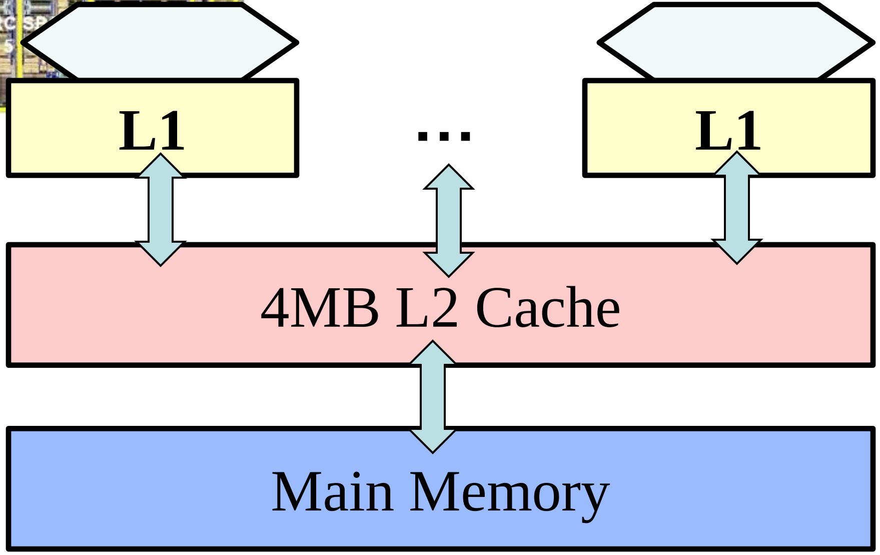
Architecture of AMD Phenom X4



4 Cores share a 2MB last level cache (LLC) and data path

Architecture of Sun Niagara T2





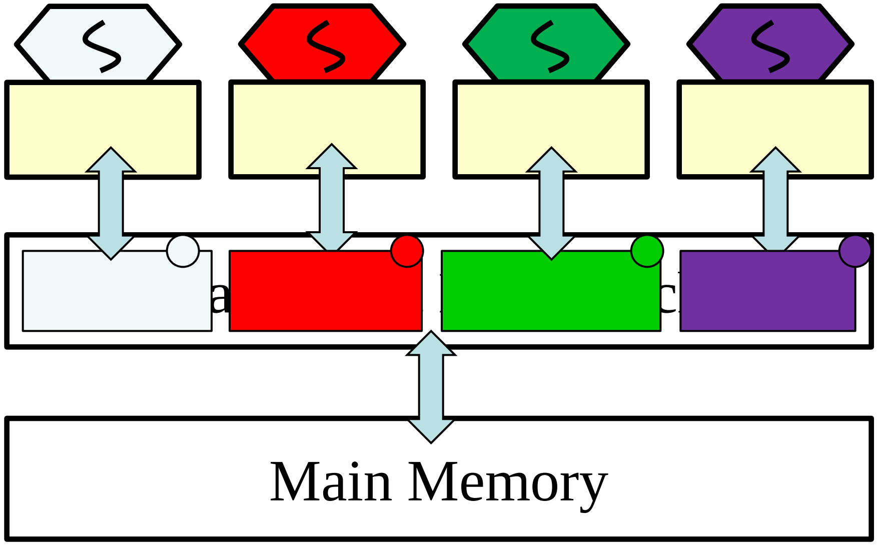
L1 … L1

4MB L2 Cache

Main Memory

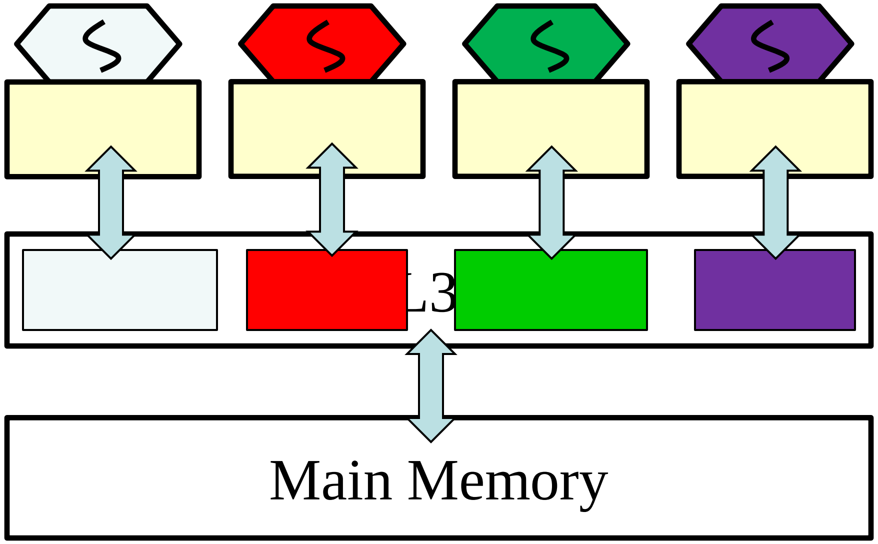
8 Cores share a 4MB last level cache (LLC) and data path

Common Structure of Multi-cores



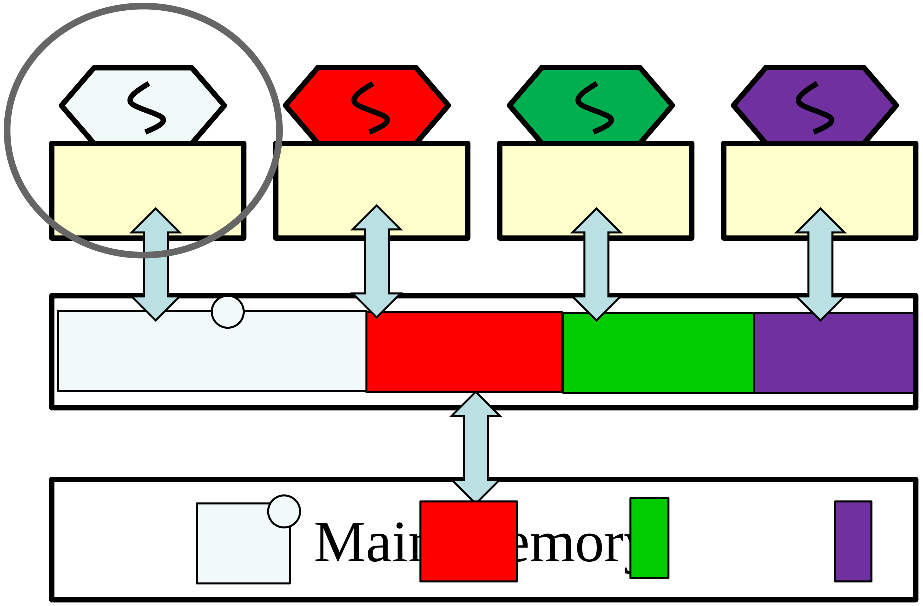
- Last level cache (LLC) and data path to memory (e.g. memory bus and controller) are shared among multiple cores.
- Memory latency is order(s) of magnitude higher than cache latency.
  - LLC is the last line of defense to prevent data accesses hitting memory wall.
  - Hot data (data frequently used) must be cached in LLC for fast accesses

Contention for LLC Space Degrades Performance



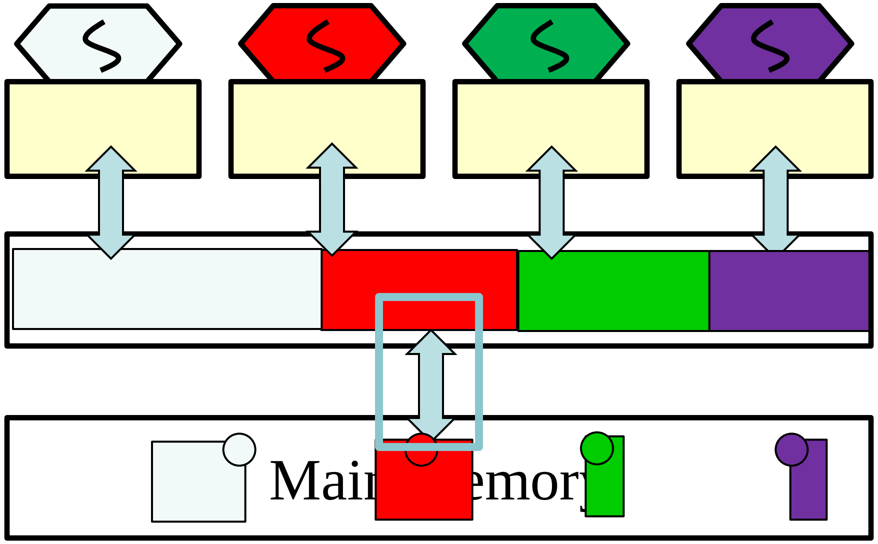
- Multiple running threads contend for the shared LLC space.

Contention for LLC Space Degrades Performance



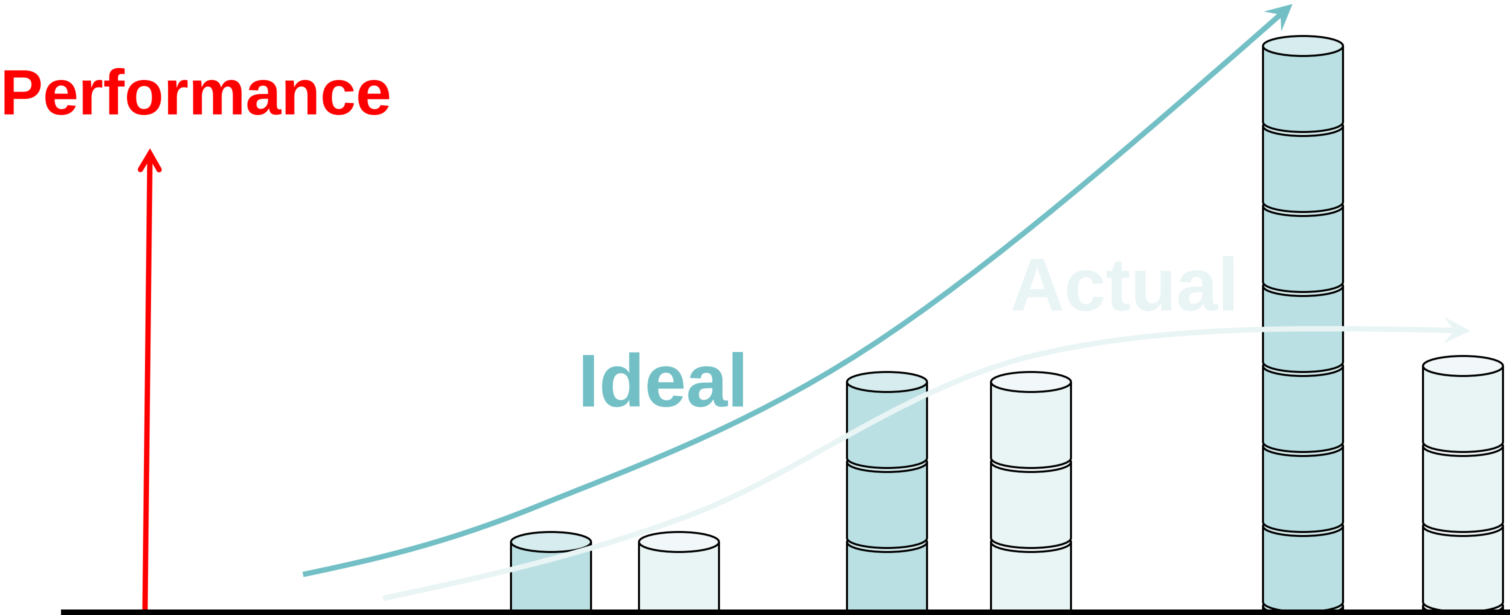
- Multiple running threads contend for the shared LLC space.
- Uncontrolled contention evicts part of hot data from LLC.
- Visiting hot data in memory degrades performance significantly.
  - Increased number of memory accesses with long latency

Contention for LLC Space Degrades Performance



- Multiple running threads contend for the shared LLC space.
- Uncontrolled contention evicts part of hot data from LLC
- Visiting hot data in memory degrades performance significantly.
  - Increased number of memory accesses with long latency

Multi-core Cannot Deliver Expected Performance as It Scales



No mechanism (neither in hardware nor in OS) addresses inter-thread cache contention in LLC

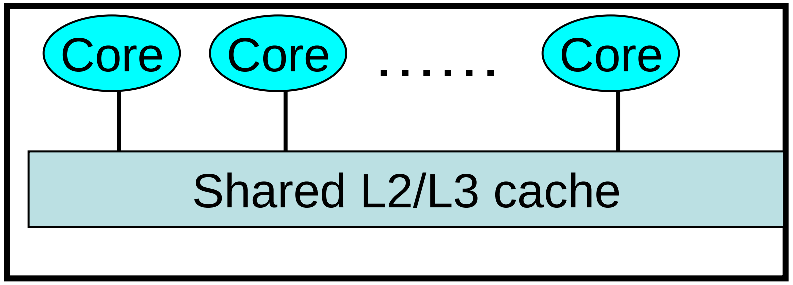
* Finding the Door in the Memory Wall, Erik Hagersten, HPCwire, Mar, 2009
Multicore Is Bad News For Supercomputers, Samuel K. Moore, IEEE Spectrum Nov, 2008

Managing LLC in Multi-cores is Challenging

- Shared cache management becomes NP-complete in multi-core.
  - [Jiang PACT'08, Hassidim ICS 2010]
- Practical Challenges
  - LLC design lacks necessary hardware supports for controlling inter-thread cache contention
    - LLC share the same design with those for uni-core processors
  - Software has limited information and methods to effectively control cache contention
- Working set model can be a pivotal point to address cache contention problem

Shared Caches Can be a Critical Bottleneck

- L2/L3 caches are shared by multiple cores
  - Intel Xeon 51xx (2core/L2)
  - AMD Barcelona (4core/L3)
  - Sun T2, ... (8core/L2)



- Cache partitioning can be effective
- Hardware cache partitioning methods have been proposed with different optimization objectives
  - Performance: [HPCA’02], [HPCA’04], [Micro’06]
  - Fairness: [PACT’04], [ICS’07], [SIGMETRICS’07]
  - QoS: [ICS’04], [ISCA’07]

Limitations of Simulation-Based Studies

- Excessive simulation time
  - Whole programs can not be evaluated. It would take several weeks/months to complete a single SPEC CPU2006 prog
  - As the number of cores continues to increase, simulation ability becomes even more limited
- Absence of long-term OS activities
  - Interactions between processor/OS affect performance significantly
- Proneness to simulation inaccuracy
  - Bugs in simulator
  - Impossible to model many dynamics and details

Our Approach to Address the Issues

Design/implement OS-base Partitioning

- Embedding partitioning *mechanism* in OS
  - By enhancing page coloring technique
  - To support both static and dynamic partitioning
- Evaluate cache partitioning *policies* on commodity processors
  - Execution- and measurement-based
  - Run applications to completion
  - Measure performance with hardware counters

Five Questions to Answer

- Can we confirm the conclusions made by the simulation-based studies?
- Can we provide new insights and findings that simulation is not able to?
- Can we make a case for our OS-based approach as an effective option to evaluate multicore cache partitioning designs?
- What are advantages and disadvantages for OS-based cache partitioning?
- Can the OS-based cache partitioning be used to manage the hardware shared cache?
- HPCA’08, Lin, et. al. (Iowa State and Ohio State)

Outline

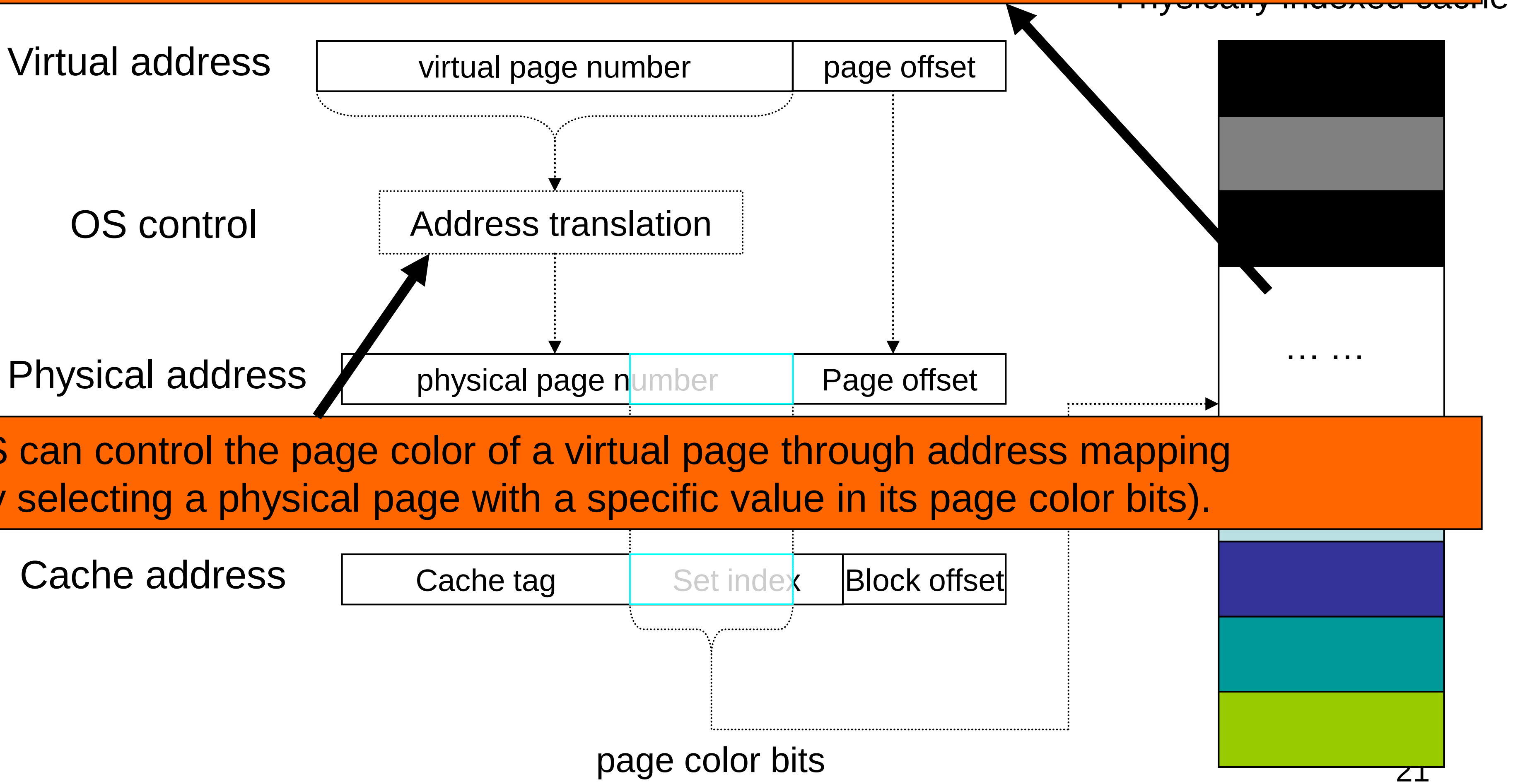
- Introduction
- Design and implementation of OS-based cache partitioning mechanisms
- Evaluation environment and workload construction
- Cache partitioning policies and their results
- Conclusion

OS-Based Cache Partitioning

- Static cache partitioning
  - Predetermines the amount of cache blocks allocated to each program at the beginning of its execution
  - Page coloring enhancement
  - Divides shared cache to multiple regions and partition cache regions through OS page address mapping
- Dynamic cache partitioning
  - Adjusts cache quota among processes dynamically
  - Page re-coloring
  - Dynamically changes processes' cache usage through OS page address re-mapping

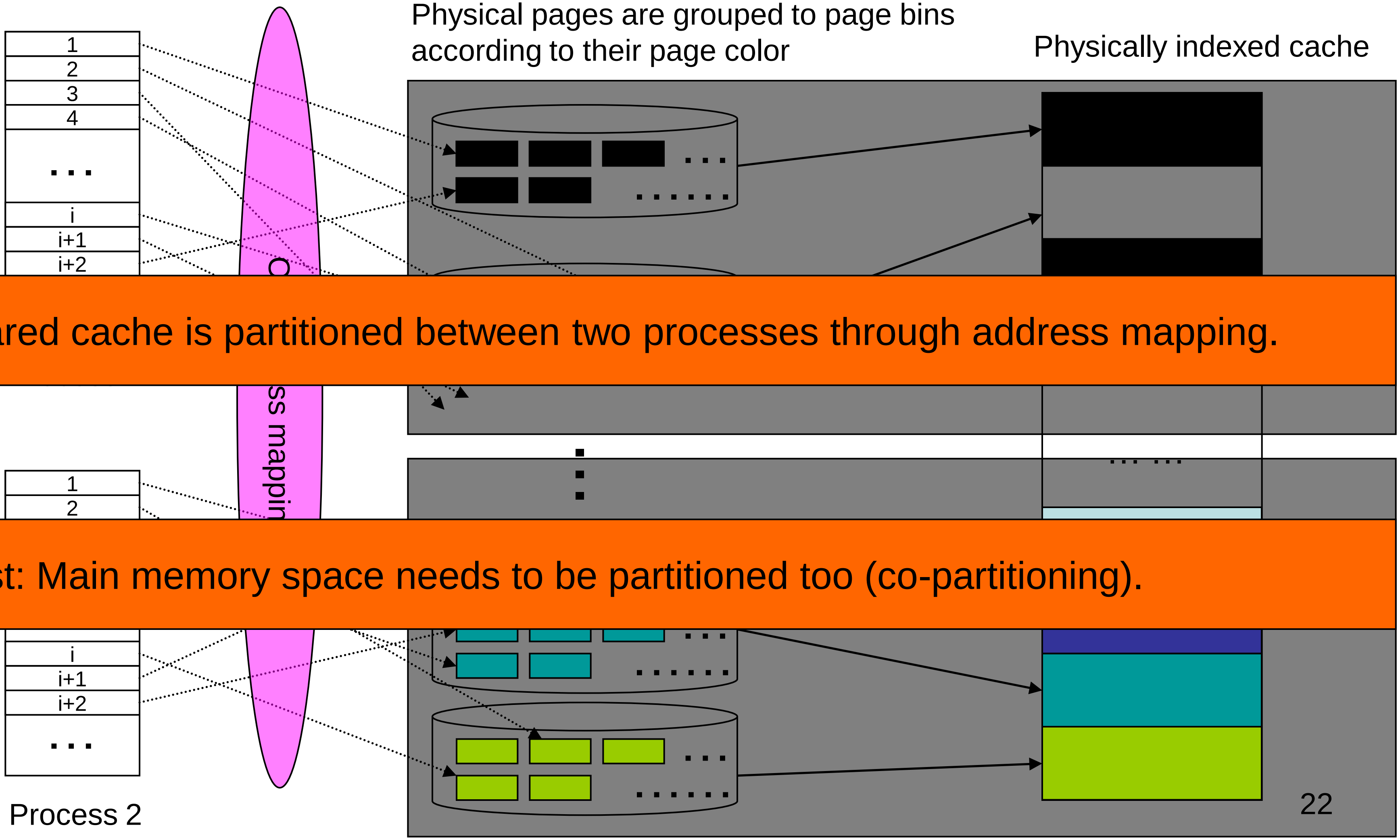
Page Coloring

- Physically indexed caches are divided into multiple regions (colors).
- All cache lines in a physical page are cached in one of those regions (colors).



OS can control the page color of a virtual page through address mapping (by selecting a physical page with a specific value in its page color bits).

Enhancement for Static Cache Partitioning



Physical pages are grouped to page bins according to their page color

Physically indexed cache

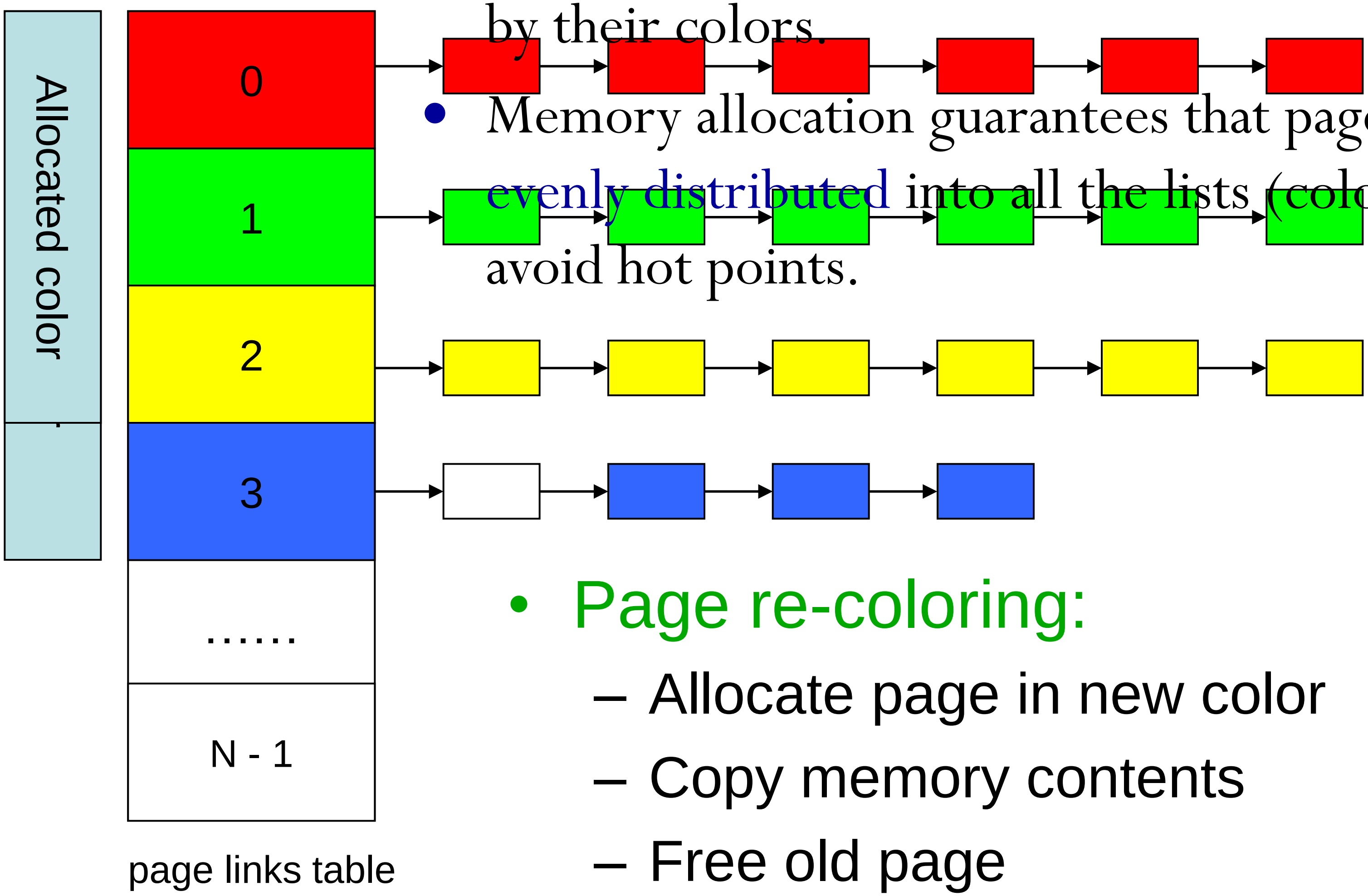
Shared cache is partitioned between two processes through address mapping.

Cost: Main memory space needs to be partitioned too (co-partitioning).

Dynamic Cache Partitioning

- To respond dynamic program behavior
  - hardware cache reallocations are proposed
  - OS-based approach: Page re-coloring
- Software Overhead
  - Measure overhead by performance counter
  - Remove overhead in result (emulating hardware schemes)

Page Re-Coloring for Dynamic Partitioning



- Pages of a process are organized into linked lists by their colors.
- Memory allocation guarantees that pages are evenly distributed into all the lists (colors) to avoid hot points.

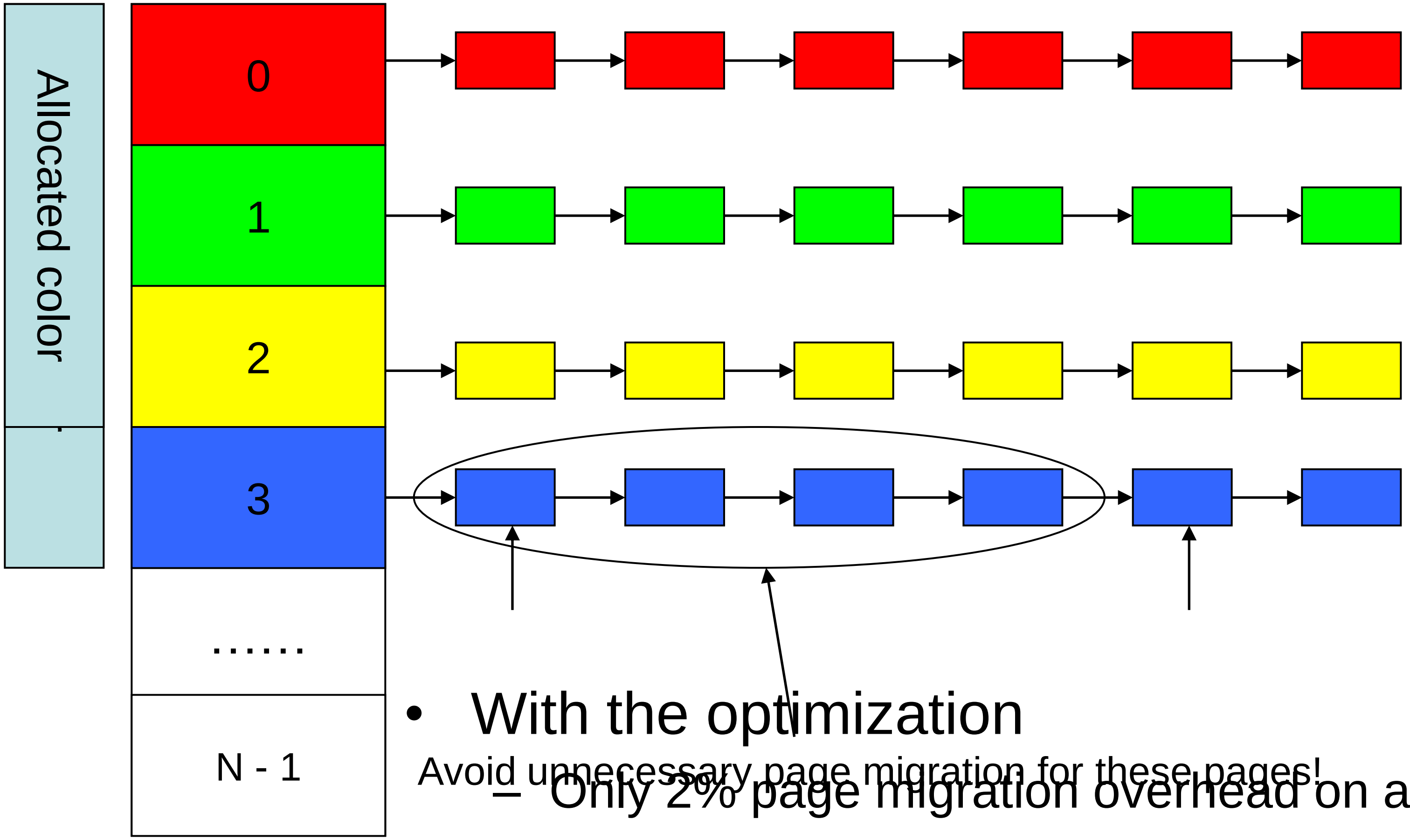
- Page re-coloring:
  - – Allocate page in new color
  - – Copy memory contents
  - – Free old page

Reduce Page Migration Overhead

- Control the frequency of page migration
  - Frequent enough to capture phase changes
  - Reduce large page migration frequency

- Lazy migration: avoid unnecessary migration
  - Observation: Not all pages are accessed between their two migrations.
  - Optimization: do not migrate a page until it is accessed

Lazy Page Migration



Avoid unnecessary page migration for these pages!

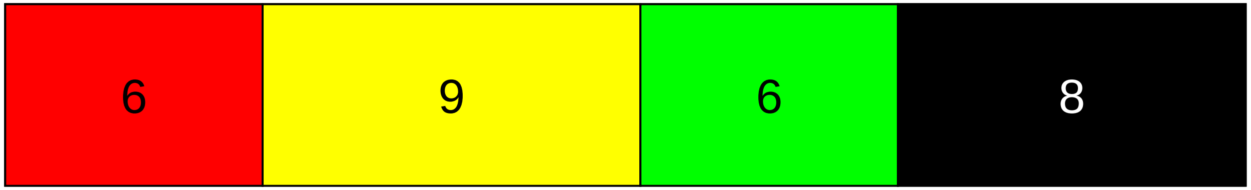
Process page links

- With the optimization
  - Only 2% page migration overhead on average
  - Up to 7%.

Experimental Environment

- Dell PowerEdge1950
  - Two-way SMP, Intel dual-core Xeon 5160
  - Shared 4MB L2 cache, 16-way
  - 8GB Fully Buffered DIMM
- Red Hat Enterprise Linux 4.0
  - 2.6.20.3 kernel
  - Performance counter tools from HP (Pfmon)
  - Divide L2 cache into 16 colors

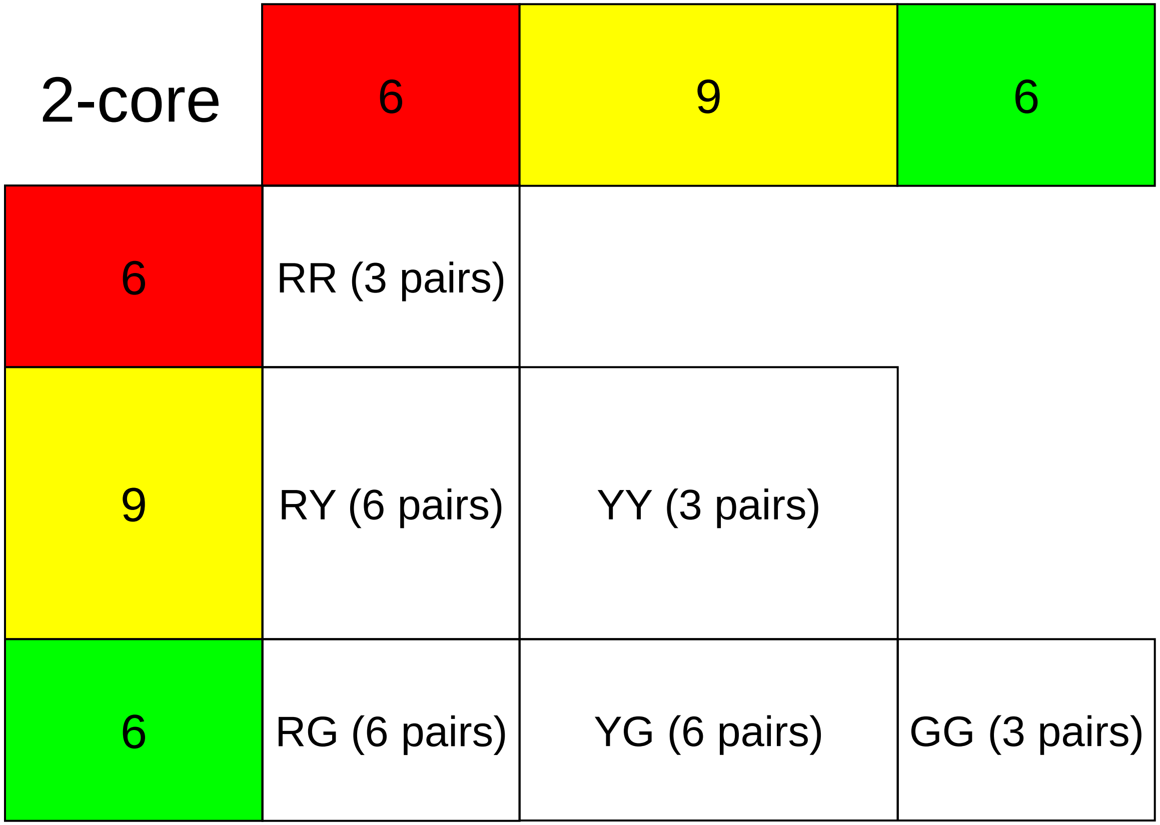
Benchmark Classification



29 benchmarks from SPEC CPU2006

- Is it sensitive to L2 cache capacity?
  - Red group: IPC(1M L2 cache)/IPC(4M L2 cache) < 80%
    - Give red benchmarks more cache: big performance gain
  - Yellow group: 80% <IPC(1M L2 cache)/IPC(4M L2 cache) < 95%
    - Give yellow benchmarks more cache: moderate performance gain
- Else: Does it extensively access L2 cache?
  - Green group: > = 14 accesses / 1K cycle
    - Give it small cache
  - **Black** group: < 14 accesses / 1K cycle
    - Cache insensitive

Workload Construction

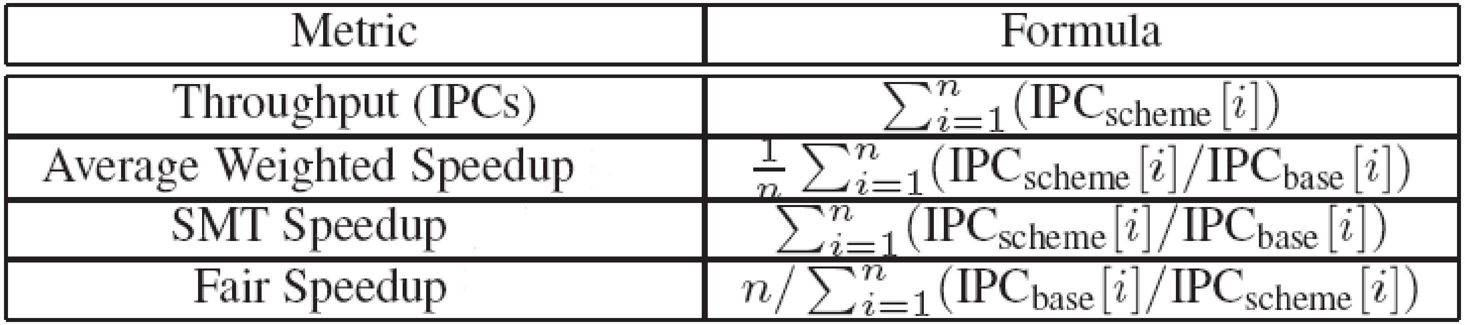


2-core	🟥 6	🟨 9	🟩 6
🟥 6	RR (3 pairs)		
🟨 9	RY (6 pairs)	YY (3 pairs)	
🟩 6	RG (6 pairs)	YG (6 pairs)	GG (3 pairs)

27 workloads: representative benchmark combinations

Performance – Metrics

- Divide metrics into evaluation metrics and policy metrics [PACT’06]
  - Evaluation metrics:
    - Optimization objectives, not always available at run-time



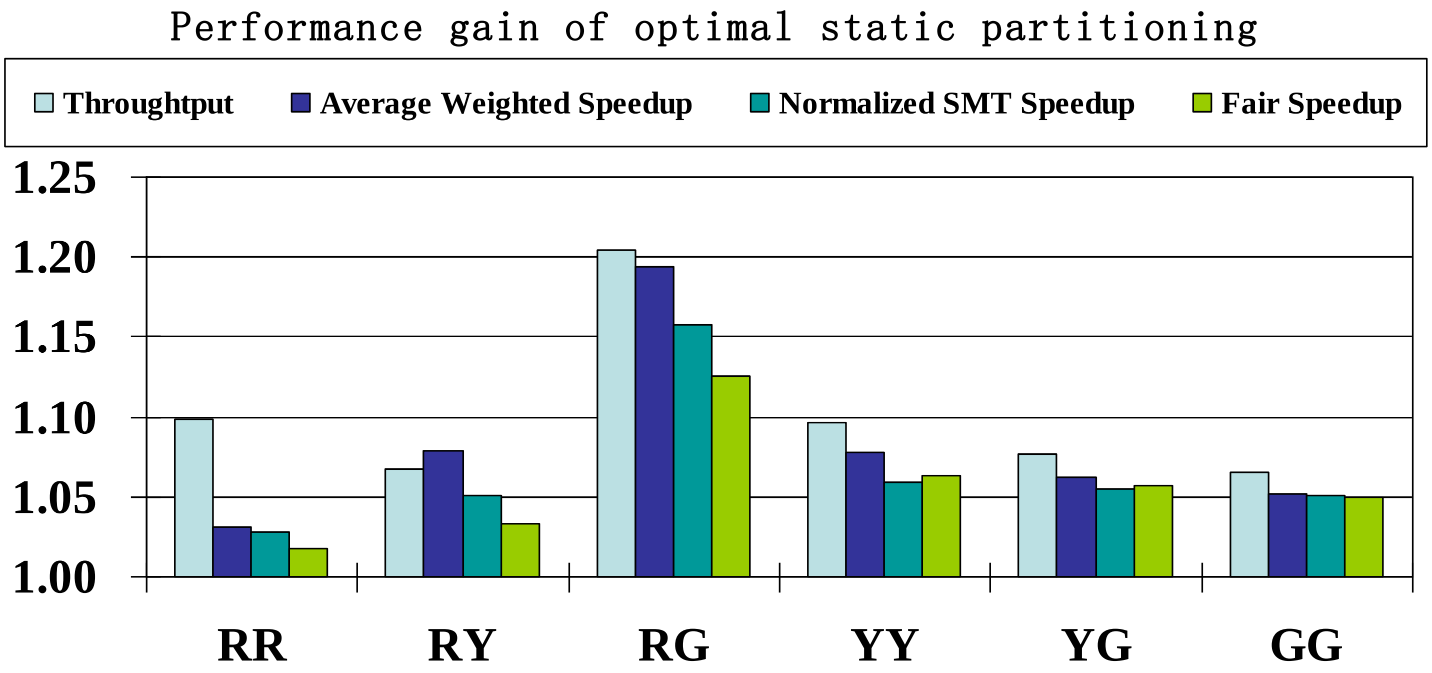
Metric	Formula
Throughput (IPCs)	$\sum_{i=1}^{n}(\text{IPC}_{\text{scheme}}[i])$
Average Weighted Speedup	$\frac{1}{n}\sum_{i=1}^{n}(\text{IPC}_{\text{scheme}}[i]/\text{IPC}_{\text{base}}[i])$
SMT Speedup	$\sum_{i=1}^{n}(\text{IPC}_{\text{scheme}}[i]/\text{IPC}_{\text{base}}[i])$
Fair Speedup	$n/\sum_{i=1}^{n}(\text{IPC}_{\text{base}}[i]/\text{IPC}_{\text{scheme}}[i])$

  - Policy metrics
    - Used to drive dynamic partitioning policies: available during run-time
    - Sum of IPC, Combined cache miss rate, Combined cache misses

Static Partitioning

- Total #color of cache: 16
- Give at least two colors to each program
  - Make sure that each program get 1GB memory to avoid swapping (because of co-partitioning)
- Try all possible partitionings for all workloads
  - (2:14), (3:13), (4:12) ……. (8,8), ……, (13:3), (14:2)
  - Get value of evaluation metrics
  - Compared with performance of all partitionings with performance of shared cache

Optimal Static Partitioning



- Confirm that cache partitioning has significant performance impact
- Different evaluation metrics have different performance gains
- RG-type of workloads have largest performance gains (up to 47%)
- Other types of workloads also have performance gains (2% to 10%)

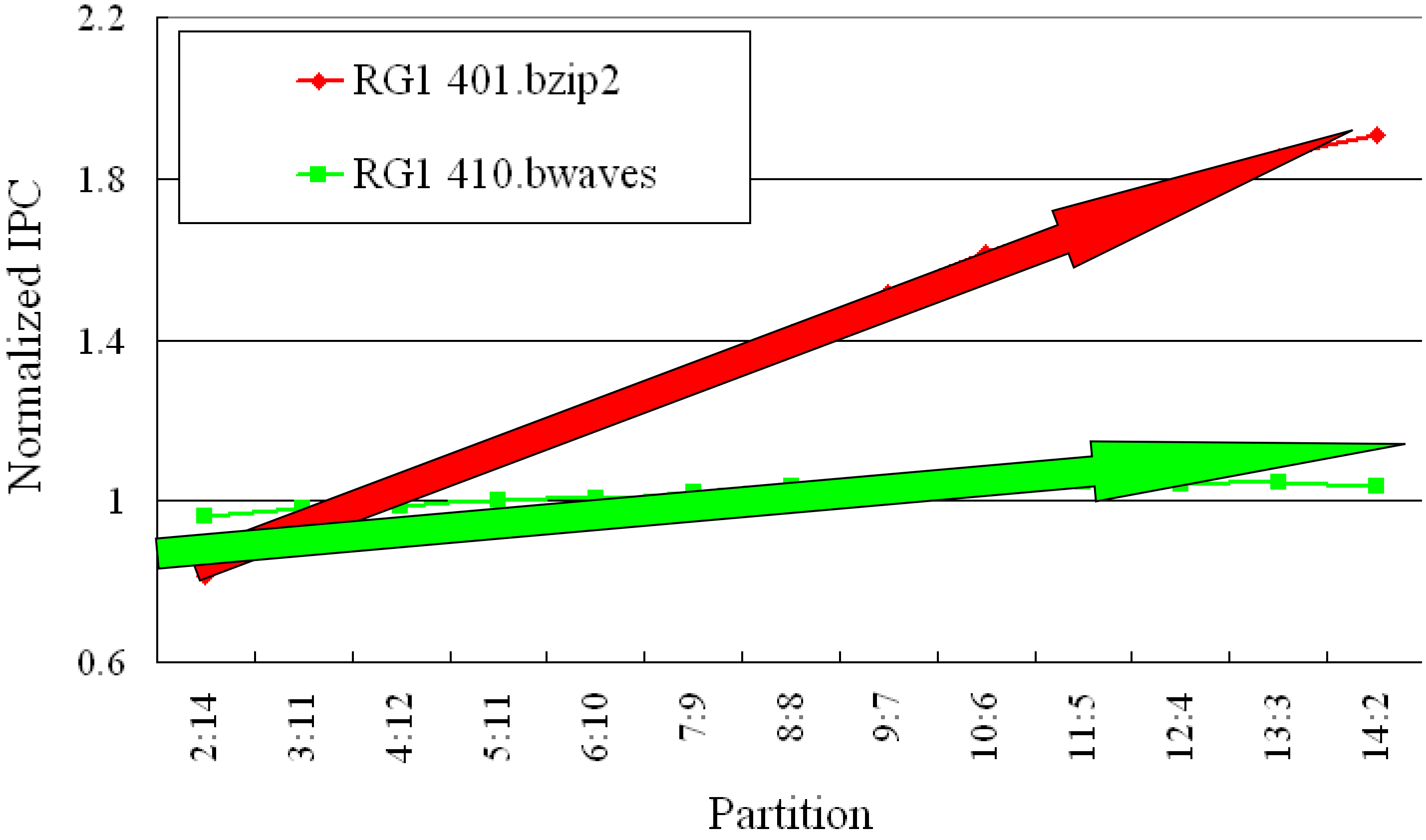
New Finding I

- Workload RG1: 401.bzip2 (cache demanding) + 410.bwaves (less cache demanding)

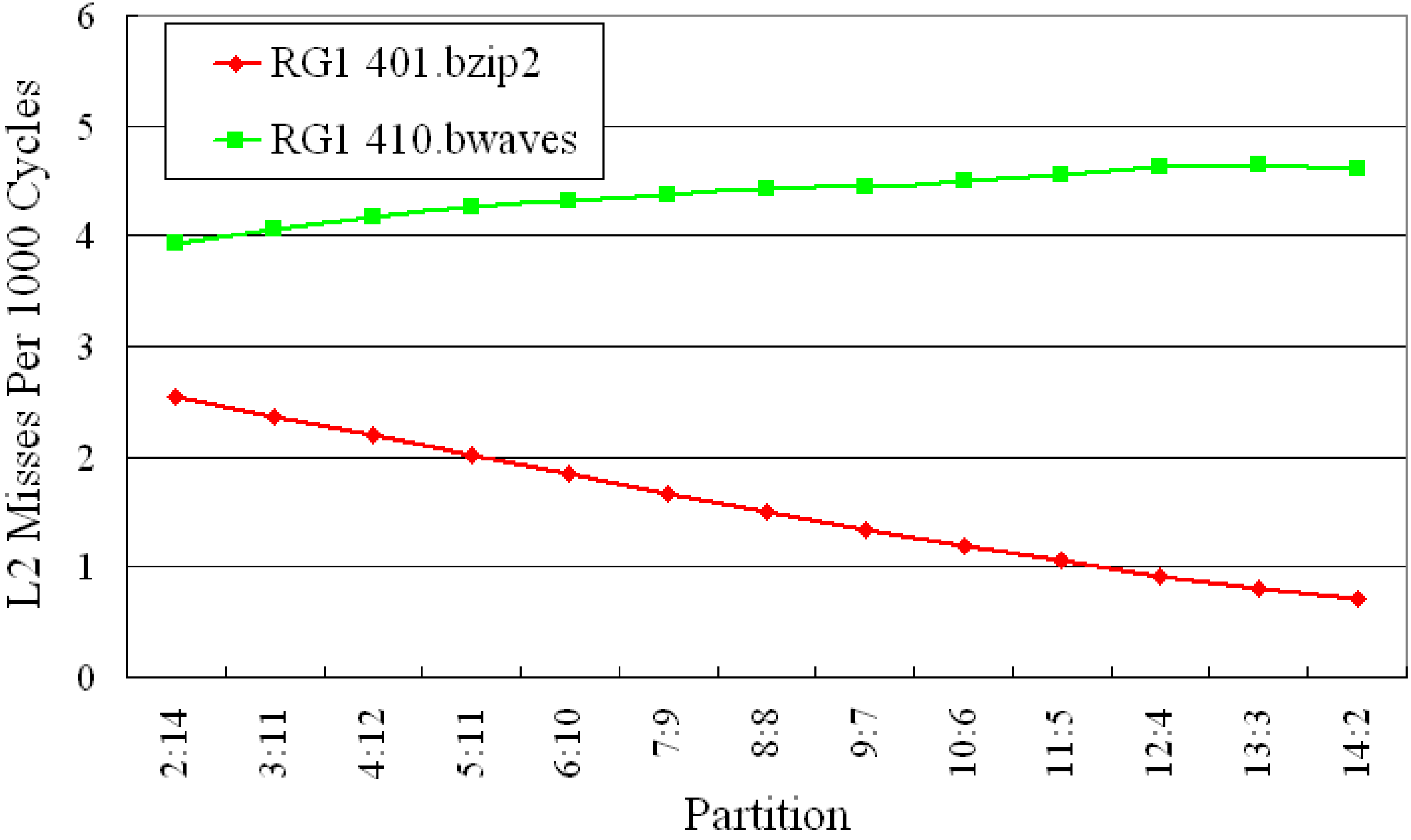
- Intuitively, giving more cache space to 401.bzip2 (cache demanding)
    - Increases the performance of 401.bzip2 largely
    - Decreases the performance of 410.bwaves slightly

- Our experiments give different answers

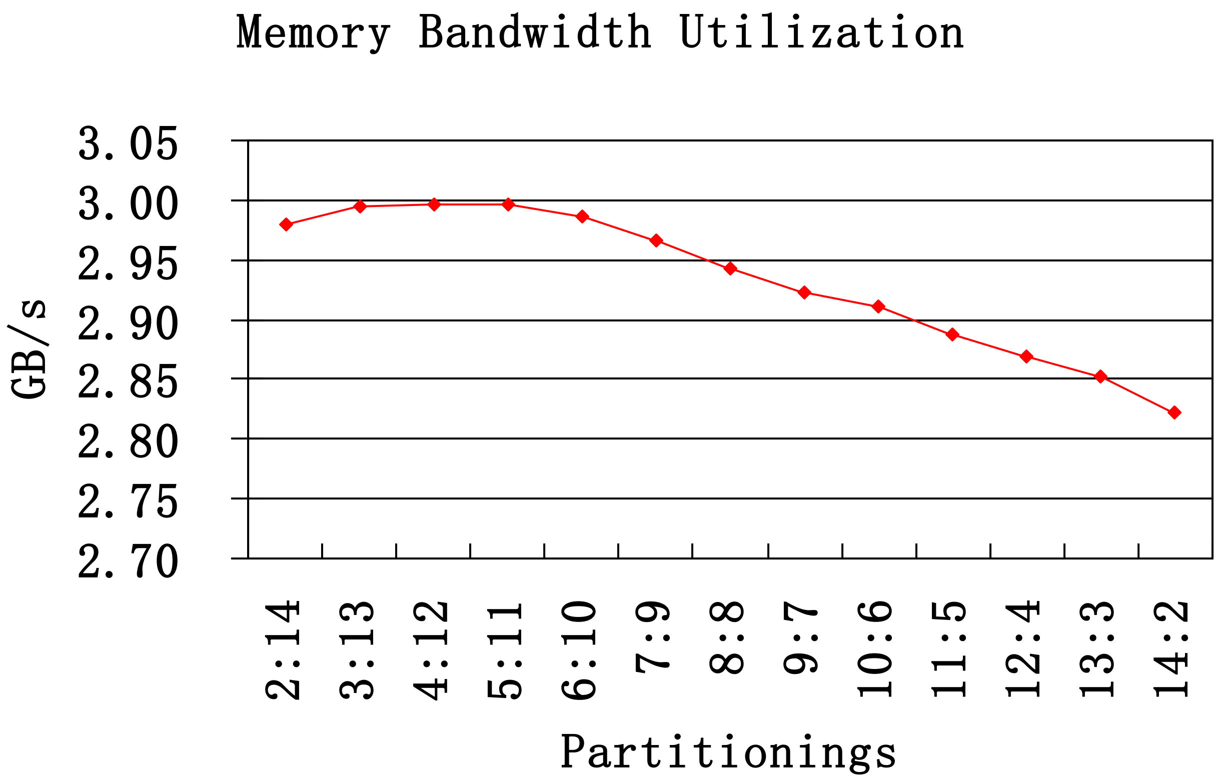
Performance Gains for Both



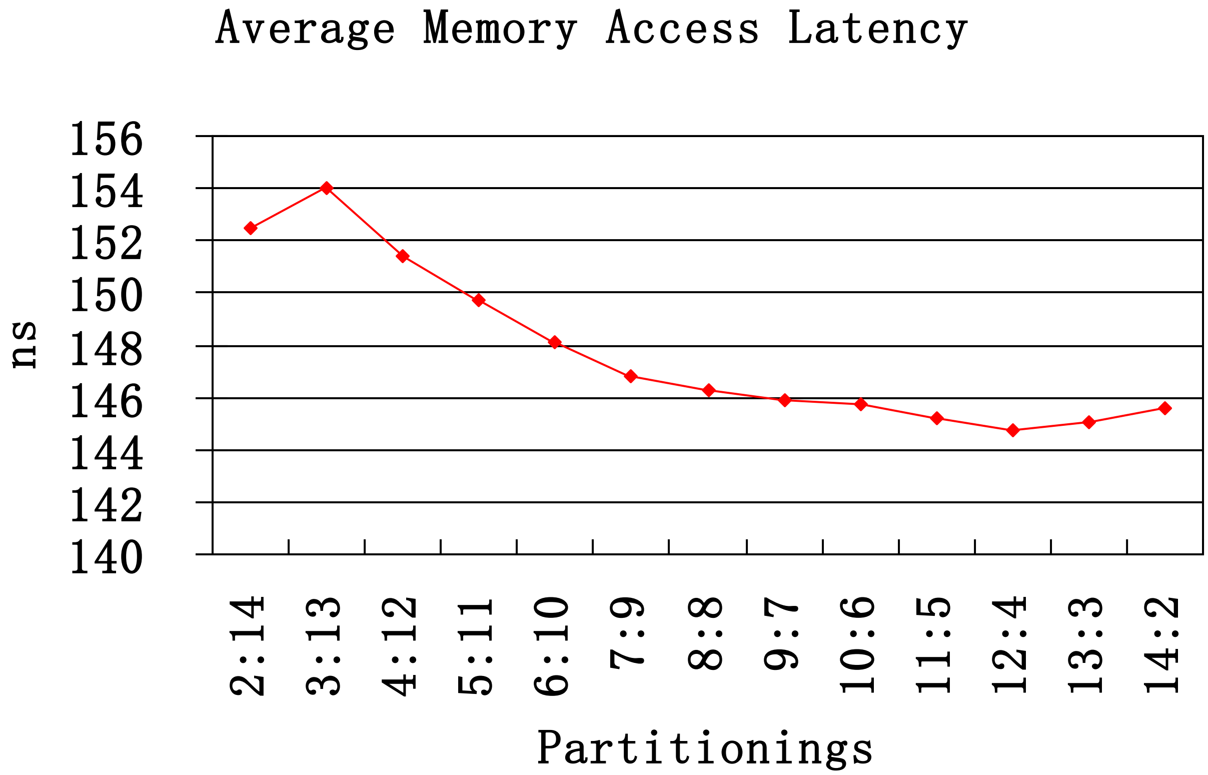
Cache Misses



Memory Bus Pressure is Reduced



Average Latency is Reduced



Insight into Our Finding

- Coordination between cache utilization and memory bandwidth is a key for performance

- This has not been shown by simulation
  - Not model main memory sub-system in detail
    - Assumed fixed memory access latency

- Advantages of execution- and measurement-base study

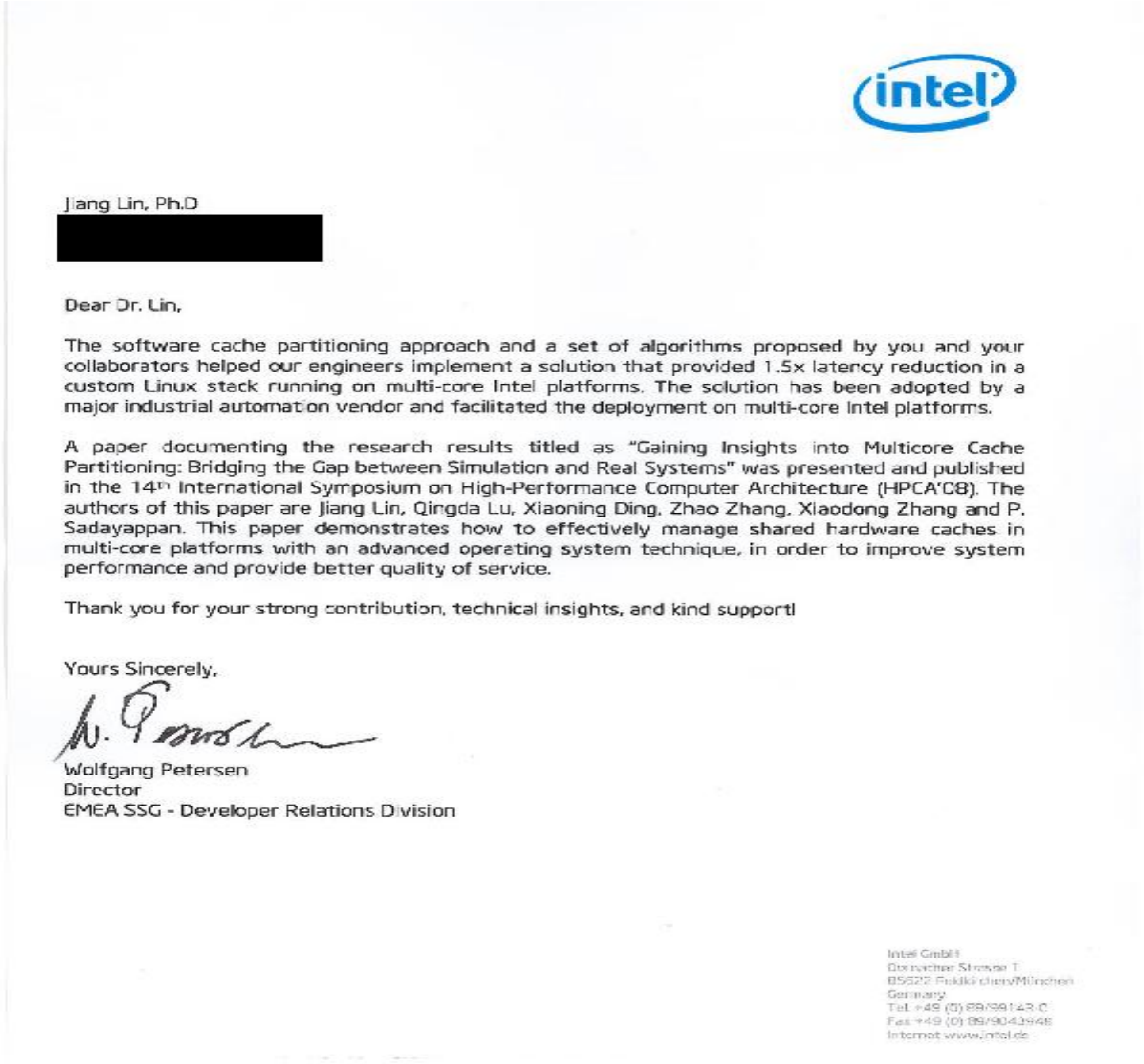
Impact of the Work

- Intel Software and Service Group (SSG) has adopted the OS-based cache partitioning methods (static and dynamic) as software solutions to manage the multi-core shared cache.

- The solution has been merged into a production system in a major automation industry (multi-core based motion controller)

- The software cache partitioning is becoming a standard method in any OS for multi-cores, such as Windows

An Acknowledgment Letter from Intel



intel

Jiang Lin, Ph.D
[redacted]

Dear Dr. Lin,

The software cache partitioning approach and a set of algorithms proposed by you and your collaborators helped our engineers implement a solution that provided 1.5x latency reduction in a custom Linux stack running on multi-core Intel platforms. The solution has been adopted by a major industrial automation vendor and facilitated the deployment on multi-core Intel platforms.

A paper documenting the research results titled as "Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems" was presented and published in the 14th International Symposium on High-Performance Computer Architecture (HPCA'08). The authors of this paper are Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang and P. Sadayappan. This paper demonstrates how to effectively manage shared hardware caches in multi-core platforms with an advanced operating system technique, in order to improve system performance and provide better quality of service.

Thank you for your strong contribution, technical insights, and kind support!

Yours Sincerely,

W. Petersen

Wolfgang Petersen
Director
EMEA SSG - Developer Relations Division

Intel GmbH
Dornacher Strasse 1
85622 Feldkirchen/München
Germany
Tel. +49 (0) 89/99143-0
Fax +49 (0) 89/9043948
Internet www.intel.de

Quotes from the Intel Letter

- The software cache partitioning approach and a set of algorithms helped our engineers implement a solution that provided 1.5 times latency reduction in a custom Linux stack running on multi-core Intel platforms.

- This solution has been adopted by a major industrial automation vendor and facilitated the deployment on multi-core platforms.

- Thanks for your strong contribution, technical insights, and kind support!

Conclusion

- Confirmed some conclusions made by simulations
- Provided new insights and findings
  - Coordinating usage of cache and memory bandwidth
  - Poor correlation between evaluation and policy metrics for fairness
- Made a case for our OS-based approach as an effective option for evaluating cache partitioning
- Advantages of OS-based cache partitioning
  - Working on commodity processors for an execution- and measurement-based study
  - Shared hardware caches can be managed by OS
- Disadvantages of OS-based cache partitioning
  - Co-partitioning (may under-utilize memory), migration overhead
- Have been adopted as a software solution for Intel.