Using Compression to Improve Chip Multiprocessor Performance

Alaa R. Alameldeen

Dissertation Defense

Wisconsin Multifacet Project
University of Wisconsin-Madison
http://www.cs.wisc.edu/multifacet

Motivation

- Architectural trends
  - Multi-threaded workloads
  - Memory wall
  - Pin bandwidth bottleneck
- CMP design trade-offs
  - Number of Cores
  - Cache Size
  - Pin Bandwidth
- Are these trade-offs zero-sum?
  - No, compression helps cache size and pin bandwidth

➲ However, hardware compression raises a few questions

2

Thesis Contributions

- Question: Is compression's overhead too high for caches?
- *Contribution #1: Simple compressed cache design*
  - Compression Scheme: Frequent Pattern Compression
  - Cache Design: Decoupled Variable-Segment Cache
- Question: Can cache compression hurt performance?
  - \+ Reduces miss rate
  - – Increases hit latency
- *Contribution #2: Adaptive compression*
  - Adapt to program behavior
  - Cache compression only when it helps

3

Thesis Contributions (Cont.)

- Question: Does compression help CMP performance?
- *Contribution #3: Evaluate CMP cache and link compression*
  - Cache compression improves CMP throughput
  - Link compression reduces pin bandwidth demand
- Question: How does compression and prefetching interact?
- *Contribution #4: Compression interacts positively with prefetching*
  - Speedup (Compr, Pref) > Speedup (Compr) x Speedup (Pref)
- Question: How do we balance CMP cores and caches?
- *Contribution #5: Model CMP cache and link compression*
  - Compression improves optimal CMP configuration

4

Outline

- Background
  - Technology and Software Trends
  - Compression Addresses CMP Design Challenges
- Compressed Cache Design
- Adaptive Compression
- CMP Cache and Link Compression
- Interactions with Hardware Prefetching
- Balanced CMP Design
- Conclusions

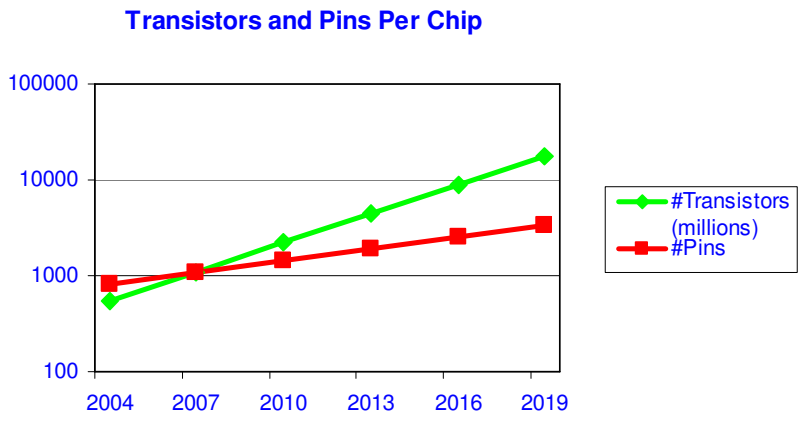
5

Technology and Software Trends

- Technology trends:
  - **Memory Wall:** Increasing gap between processor and memory speeds
  - **Pin Bottleneck:** Bandwidth demand > Bandwidth Supply

6

Pin Bottleneck: ITRS 04 Roadmap



➢ Annual Rates of Increase: Transistors 26%, Pins 10%

7

Technology and Software Trends

- Technology trends:
  - **Memory Wall:** Increasing gap between processor and memory speeds
  - **Pin Bottleneck:** Bandwidth demand > Bandwidth Supply

  ⇒ Favor bigger cache
- Software application trends:
  - **Higher throughput requirements**

  ⇒ Favor more cores/threads

  ⇒ Demand higher pin bandwidth

Contradictory Goals

8

Using Compression

- On-chip Compression
  - **Cache Compression:** Increases effective cache size
  - **Link Compression:** Increases effective pin bandwidth
- Compression Requirements
  - Lossless
  - Low decompression (compression) overhead
  - Efficient for small block sizes
  - Minimal additional complexity

➲ Thesis addresses CMP design with compression support

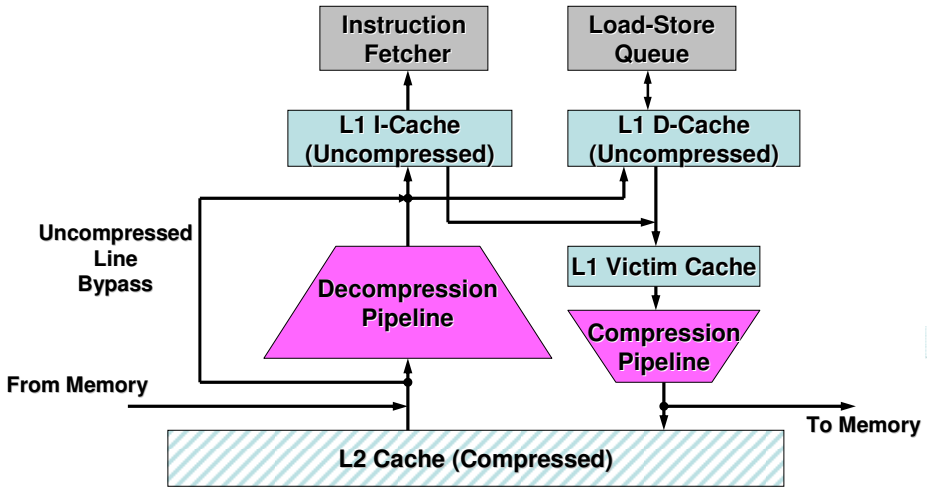
9

Outline

- Background
- Compressed Cache Design
  - Compressed Cache Hierarchy
  - Compression Scheme: FPC
  - Decoupled Variable-Segment Cache
- Adaptive Compression
- CMP Cache and Link Compression
- Interactions with Hardware Prefetching
- Balanced CMP Design
- Conclusions

10

Compressed Cache Hierarchy (Uniprocessor)



11

Frequent Pattern Compression (FPC)

- A significance-based compression algorithm
  - Compresses each 32-bit word separately
  - Suitable for short (32-256 byte) cache lines
  - Compressible Patterns: zeros, sign-ext. 4,8,16-bits, zero-padded half-word, two SE half-words, repeated byte
  - Pattern detected ⇒ Store pattern prefix + significant bits
  - A 64-byte line is decompressed in a five-stage pipeline

12

Decoupled Variable-Segment Cache

- Each set contains twice as many tags as uncompressed lines
- Data area divided into 8-byte segments
- Each tag is composed of:
  - Address tag
  - Permissions

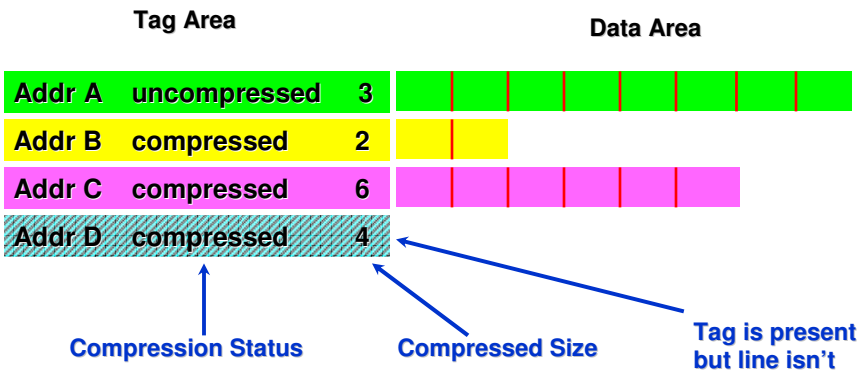
  ← Same as uncompressed cache

  - CStatus : 1 if the line is compressed, 0 otherwise
  - CSize: Size of compressed line in segments
  - LRU/replacement bits

13

Decoupled Variable-Segment Cache

- Example cache set



14

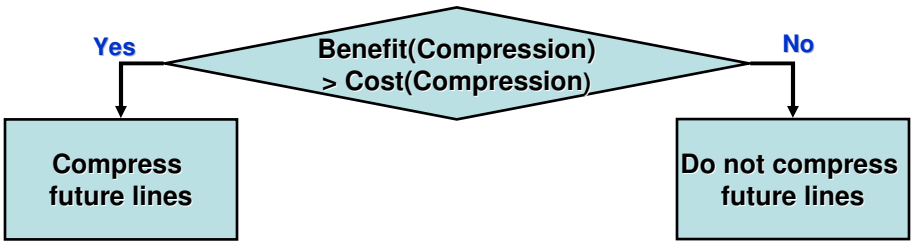
Outline

- Background
- Compressed Cache Design
- Adaptive Compression
  - Key Insight
  - Classification of Cache Accesses
  - Performance Evaluation
- CMP Cache and Link Compression
- Interactions with Hardware Prefetching
- Balanced CMP Design
- Conclusions

15

Adaptive Compression

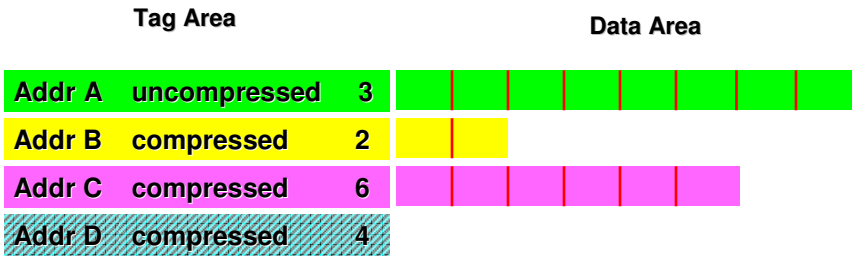
- Use past to predict future



- *Key Insight:*
  - *LRU Stack [Mattson, et al., 1970] indicates for each reference whether compression helps or hurts*

16

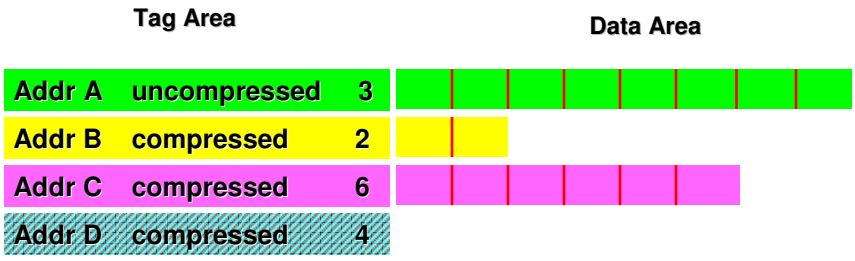
Cost/Benefit Classification



- Classify each cache reference
- Four-way SA cache with space for two 64-byte lines
  - Total of 16 available segments

17

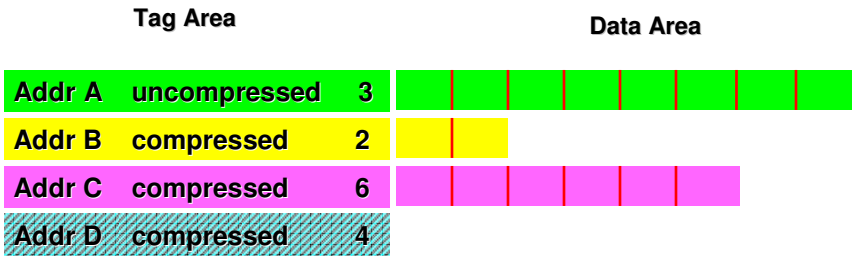
An Unpenalized Hit



- Read/Write Address A
  - LRU Stack order = 1 ≤ 2 ⇨ Hit regardless of compression
  - Uncompressed Line ⇨ No decompression penalty
  - Neither cost nor benefit

18

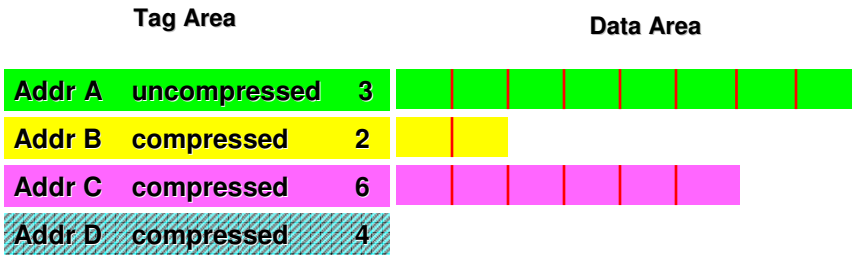
A Penalized Hit



- Read/Write Address B
  - LRU Stack order = 2 ≤ 2 ⇨ Hit regardless of compression
  - Compressed Line ⇨ Decompression penalty incurred
  - Compression cost

19

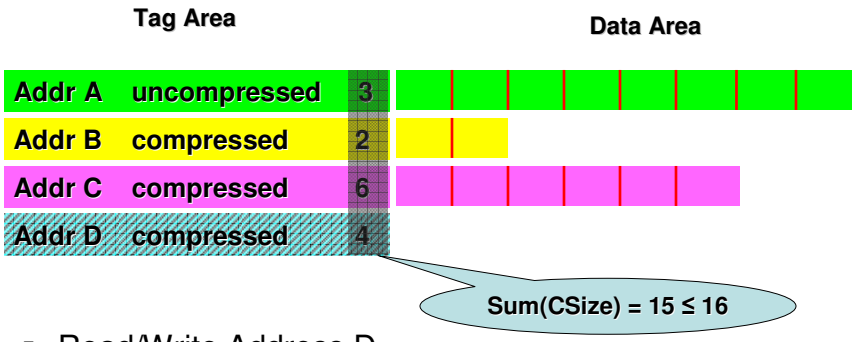
An Avoided Miss



- Read/Write Address C
  - LRU Stack order = 3 > 2 ⇨ Hit only because of compression
  - Compression benefit: Eliminated off-chip miss

20

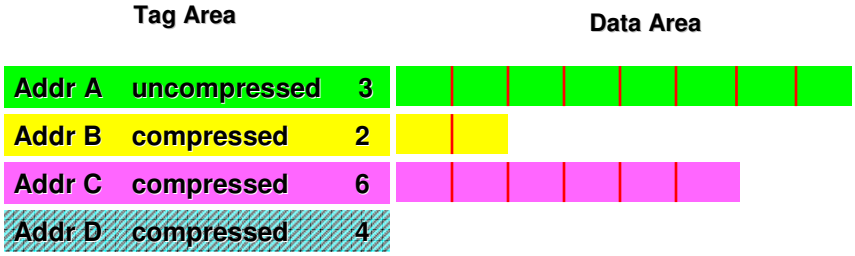
An Avoidable Miss



- Read/Write Address D
  - Line is not in the cache but tag exists at LRU stack order = 4
  - Missed only because some lines are not compressed
  - Potential compression benefit

21

An Unavoidable Miss



- Read/Write Address E
  - LRU stack order > 4 ⇨ Compression wouldn't have helped
  - Line is not in the cache and tag does not exist
  - Neither cost nor benefit

22

Compression Predictor

- Estimate: Benefit(Compression) – Cost(Compression)
- Single counter : Global Compression Predictor (GCP)
  - Saturating up/down 19-bit counter
- GCP updated on each cache access
  - Benefit: Increment by memory latency
  - Cost: Decrement by decompression latency
  - Optimization: Normalize to memory_lat / decompression_lat, 1
- Cache Allocation
  - Allocate compressed line if GCP ≥ 0
  - Allocate uncompressed lines if GCP < 0

23

Simulation Setup

- Workloads:
  - Commercial workloads [Computer'03, CAECW'02] :
    - OLTP: IBM DB2 running a TPC-C like workload
    - SPECJBB
    - Static Web serving: Apache and Zeus
  - SPEC2000 benchmarks:
    - SPECint: bzip, gcc, mcf, twolf
    - SPECfp: ammp, applu, equake, swim
- Simulator:
  - Simics full system simulator; augmented with:
  - Multifacet General Execution-driven Multiprocessor Simulator (GEMS) [Martin, et al., 2005, http://www.cs.wisc.edu/gems/]

24

System configuration

- **Configuration parameters:**

L1 Cache	Split I&D, 64KB each, 4-way SA, 64B line, 3-cycles/access
L2 Cache	Unified 4MB, ***8-way*** SA, 64B line, access latency 15 cycles + *5-cycle decompression latency* (if needed)
Memory	4GB DRAM, *400-cycle access time*, 16 outstanding requests
Processor	Dynamically scheduled SPARC V9, 4-wide superscalar, 64-entry Instruction Window, 128-entry reorder buffer

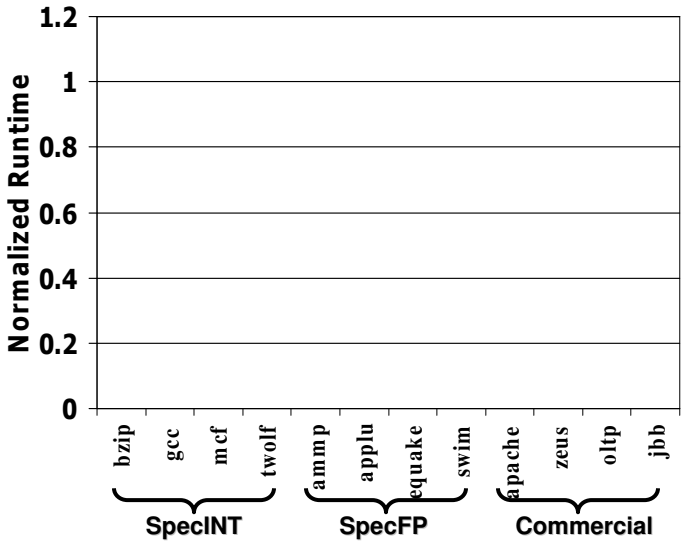
25

Simulated Cache Configurations

- *Always:* All compressible lines are stored in compressed format
  - Decompression penalty for all compressed lines
- *Never:* All cache lines are stored in uncompressed format
  - Cache is 8-way set associative with half the number of sets
  - Does not incur decompression penalty
- *Adaptive:* Adaptive compression scheme

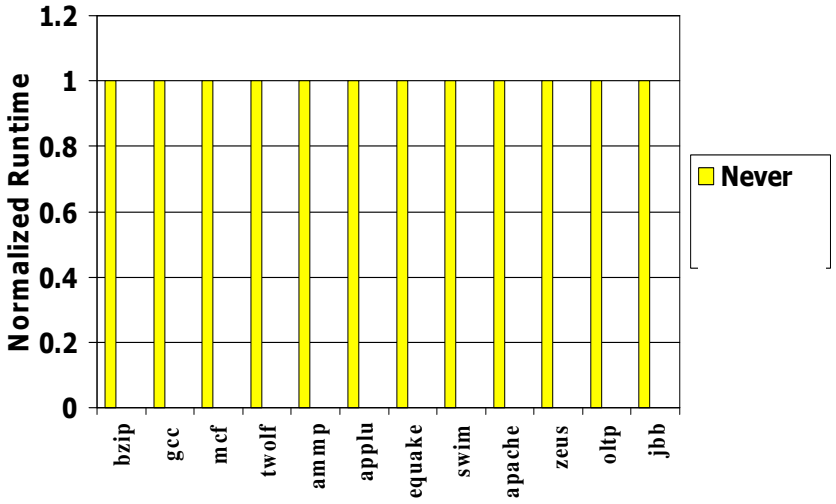
26

Performance



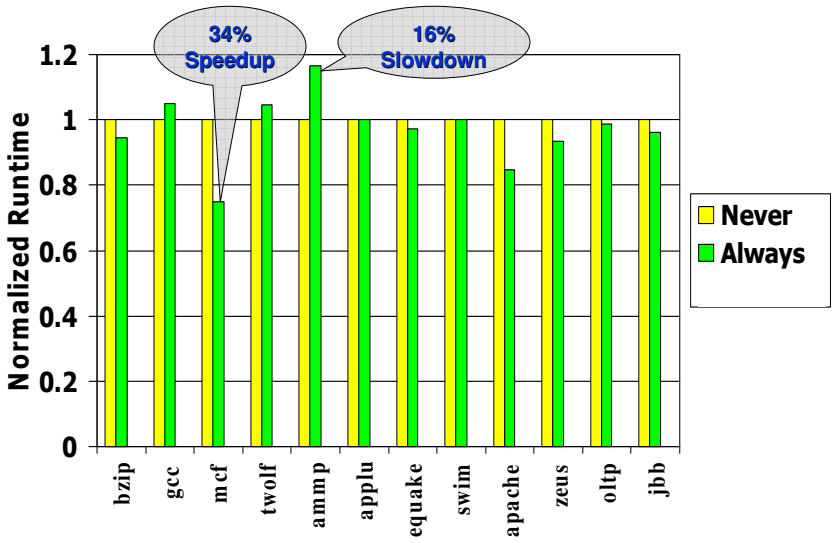
27

Performance



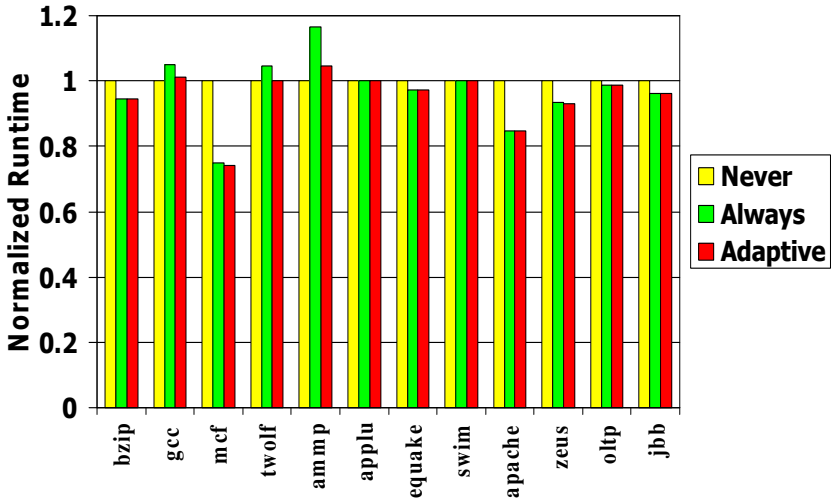
28

Performance



29

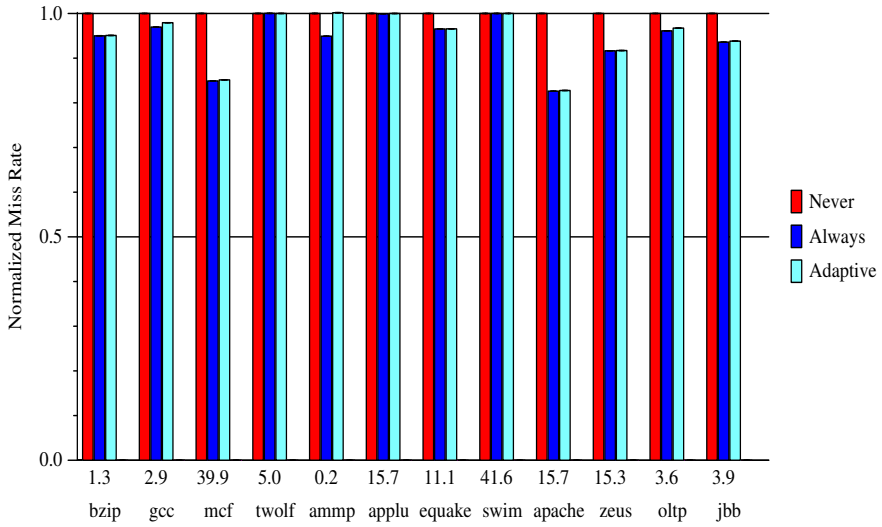
Performance



➲ Adaptive performs similar to the best of Always and Never

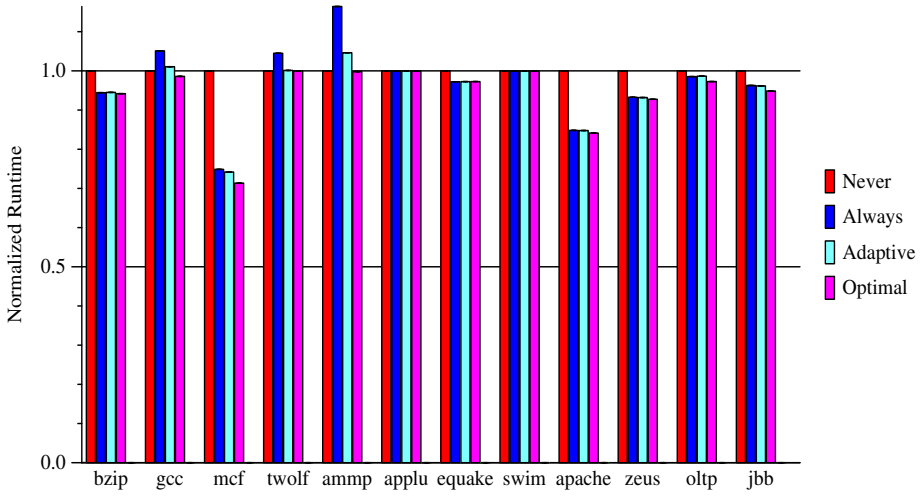
30

Cache Miss Rates



31

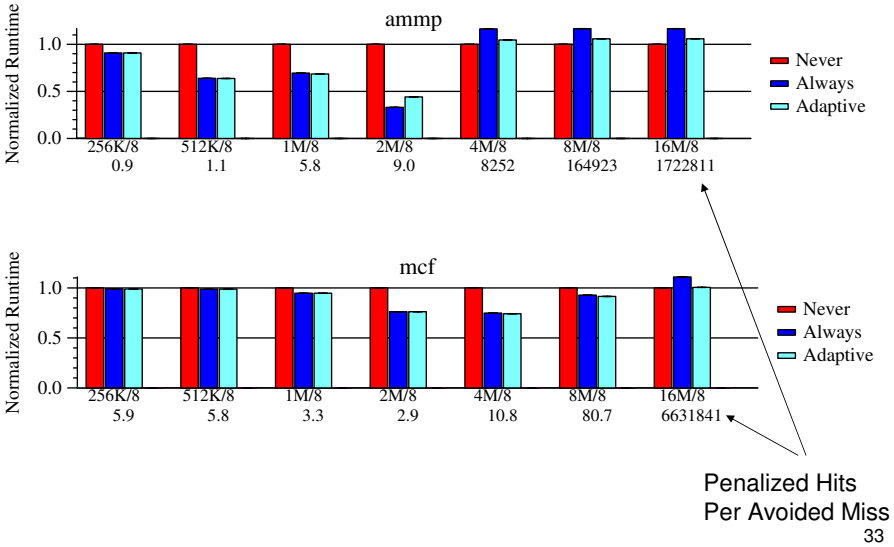
Optimal Adaptive Compression?



- Optimal: Always with no decompression penalty

32

Adapting to L2 Size



33

Adaptive Compression: Summary

- Cache compression increases cache capacity but slows down cache hit time
  - Helps some benchmarks (e.g., apache, mcf)
  - Hurts other benchmarks (e.g., gcc, ammp)
- Adaptive compression
  - Uses (LRU) replacement stack to determine whether compression helps or hurts
  - Updates a single global saturating counter on cache accesses
- Adaptive compression performs similar to the better of *Always Compress* and *Never Compress*

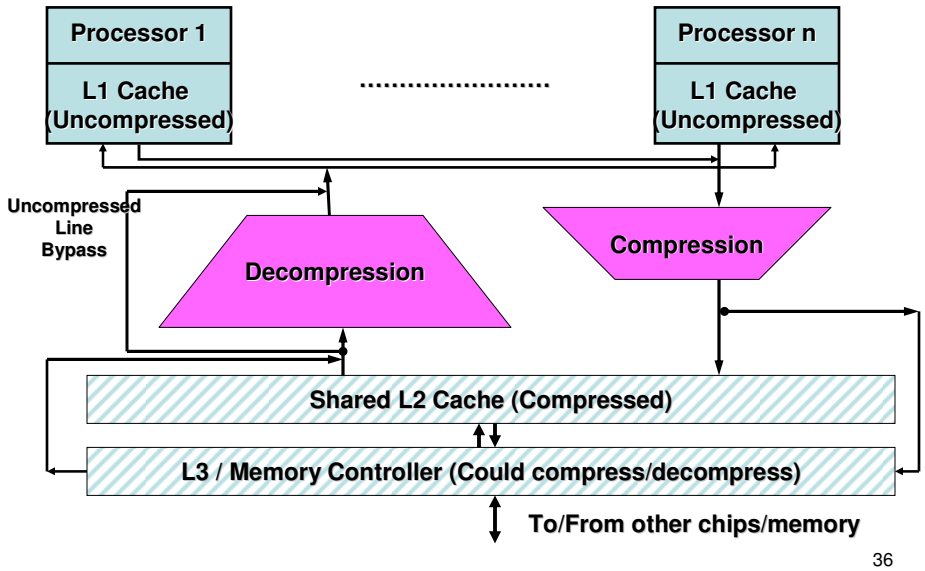
34

Outline

- Background
- Compressed Cache Design
- Adaptive Compression
- CMP Cache and Link Compression
- Interactions with Hardware Prefetching
- Balanced CMP Design
- Conclusions

35

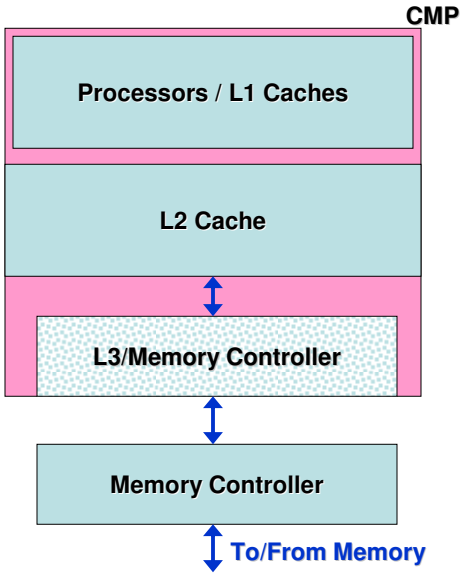
Compressed Cache Hierarchy (CMP)



36

Link Compression

- On-chip L3/Memory Controller transfers compressed messages
- Data Messages
  - 1-8 sub-messages (flits), 8-bytes each
- Off-chip memory controller combines flits and stores to memory



37

Hardware Stride-Based Prefetching

- L2 Prefetching
  - \+ Hides memory latency
  - \- Increases pin bandwidth demand
- L1 Prefetching
  - \+ Hides L2 latency
  - \- Increases L2 contention and on-chip bandwidth demand
  - \- Triggers L2 fill requests ⇒ Increases pin bandwidth demand
- **Questions:**
  - Does compression interfere positively or negatively with hardware prefetching?
  - How does a system with both compare to a system with only compression or only prefetching?

38

Interactions Terminology

- Assume a base system S with two architectural enhancements A and B, All systems run program P

Speedup(A) = Runtime(P, S) / Runtime(P, A)

Speedup(B) = Runtime(P, S) / Runtime (P, B)

Speedup(A, B) = Speedup(A) x Speedup(B)

x (1 + Interaction(A,B))

39

Compression and Prefetching Interactions

- **Positive Interactions:**
  - + L1 prefetching hides part of decompression overhead
  - + Link compression reduces increased bandwidth demand because of prefetching
  - + Cache compression increases effective L2 size, L2 prefetching increases working set size
- **Negative Interactions:**
  - – L2 prefetching and L2 compression can eliminate the same misses

➲ Is Interaction(Compression, Prefetching) positive or negative?

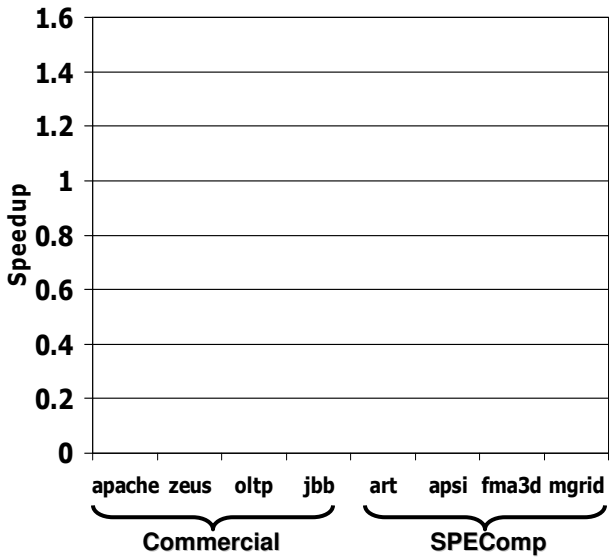
40

Evaluation

- *8-core* CMP
- **Cores**: single-threaded, out-of-order superscalar with a 64-entry IW, 128-entry ROB, 5 GHz clock frequency
- **L1 Caches**: 64K instruction, 64K data, 4-way SA, 320 GB/sec total on-chip bandwidth (to/from L1), 3-cycle latency
- **Shared L2 Cache**: 4 MB, 8-way SA (uncompressed), 15-cycle uncompressed latency, 128 outstanding misses
- **Memory**: 400 cycles access latency, 20 GB/sec memory bandwidth
- **Prefetching:**
  - Similar to prefetching in IBM's Power4 and Power5
  - 8 unit/negative/non-unit stride streams for L1 and L2 for each processor
  - Issue 6 L1 prefetches on L1 miss
  - Issue 25 L2 prefetches on L2 miss

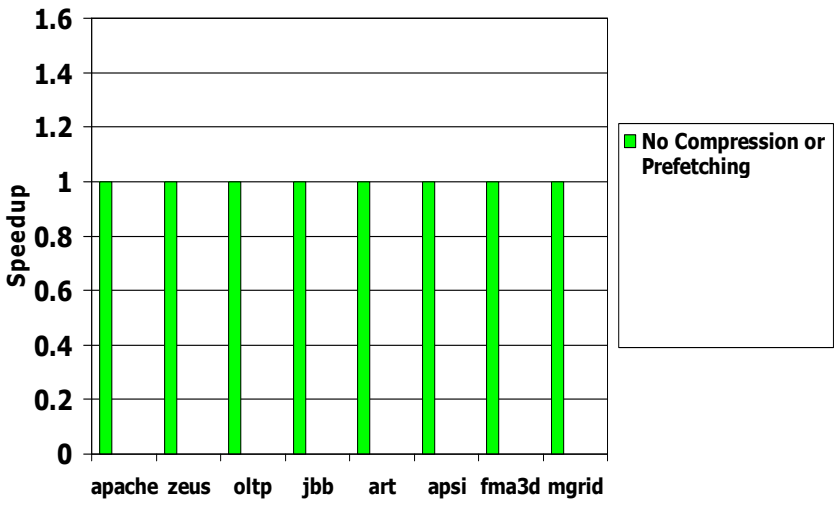
41

Performance



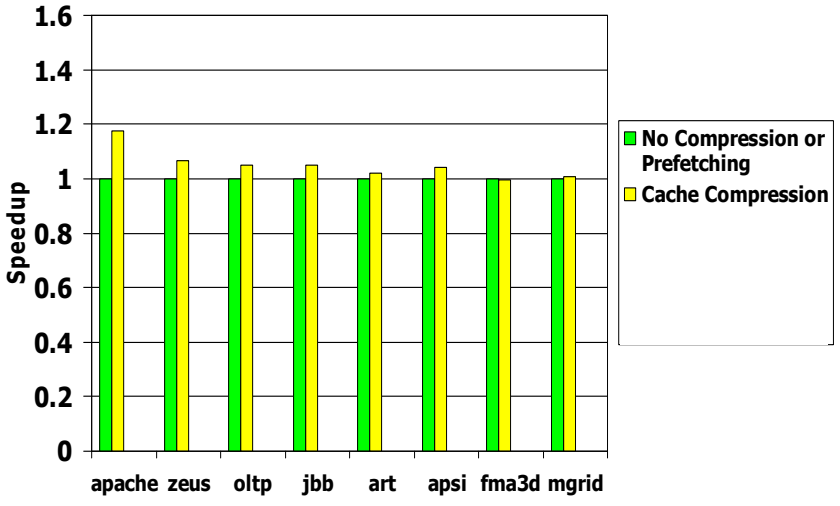
42

Performance



43

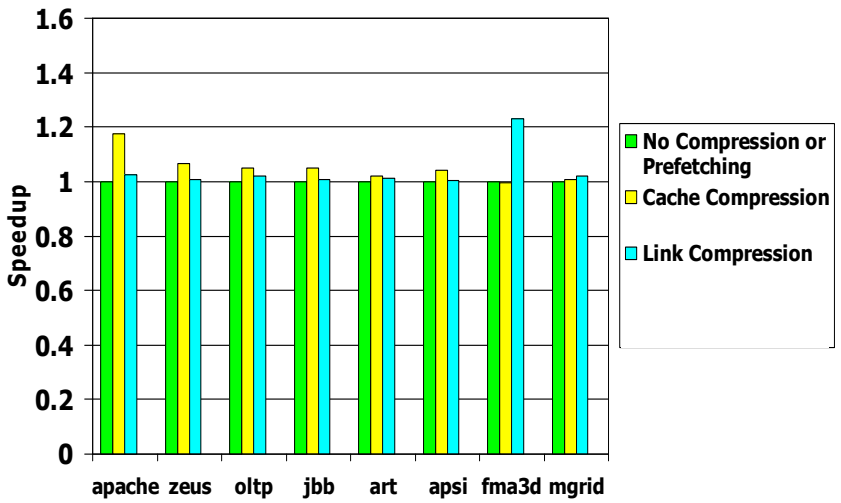
Performance



➲ **Cache Compression provides speedups of up to 18%**

44

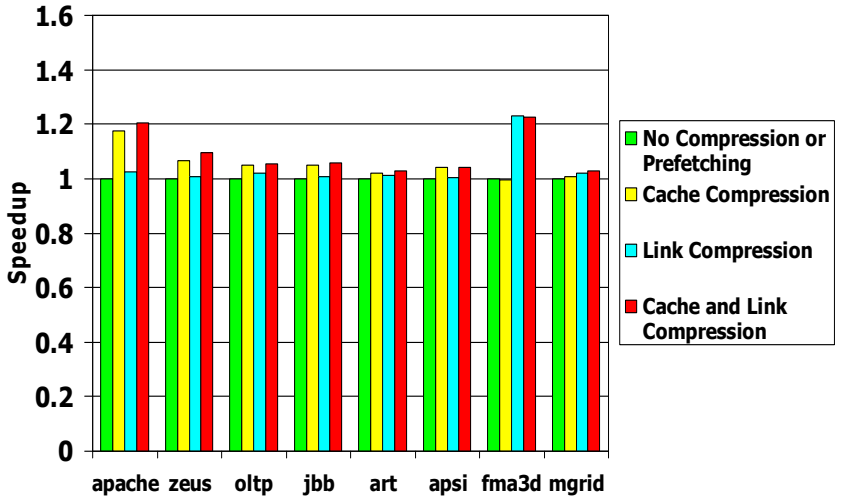
Performance



➲ Link compression speeds up bandwidth-limited applications

45

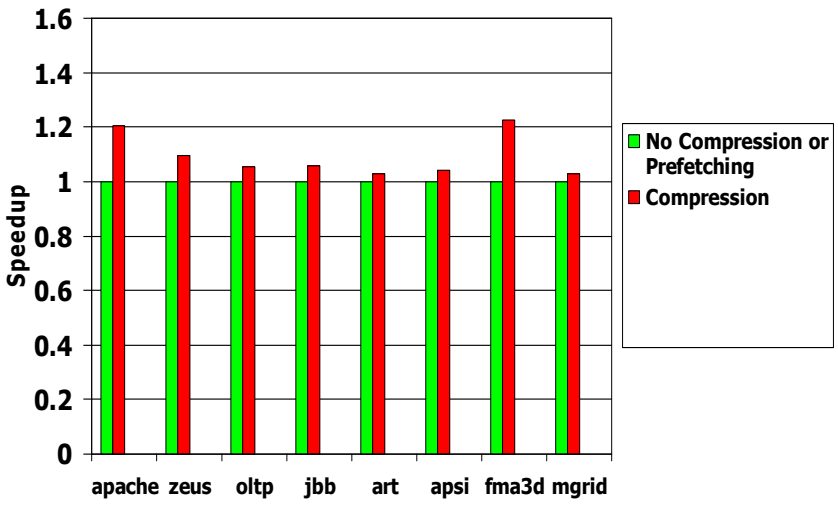
Performance



➲ Cache&Link compression provide speedups up to 22%

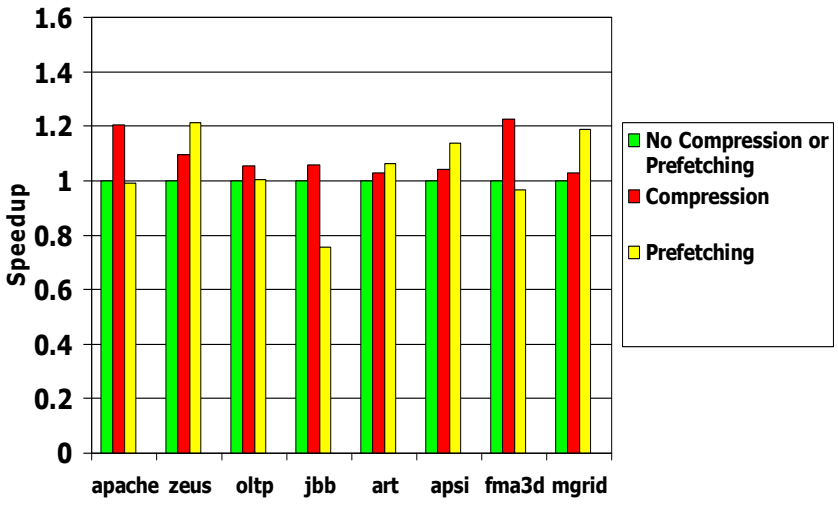
46

Performance



47

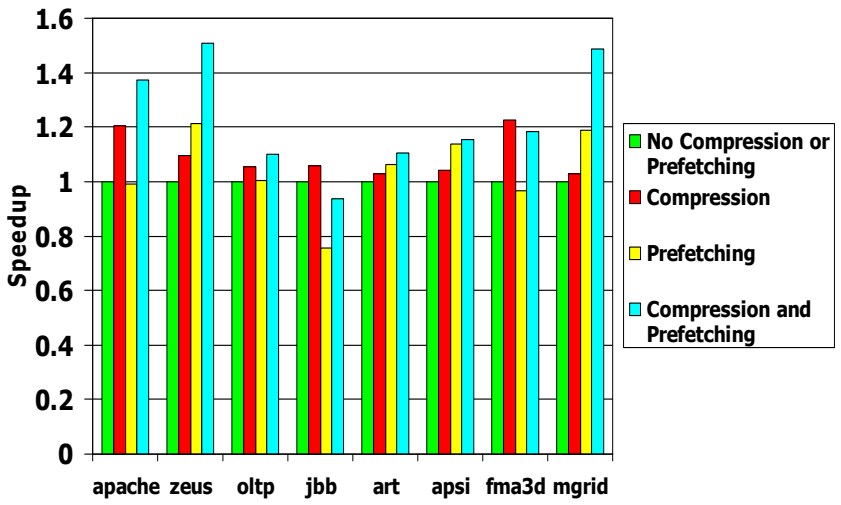
Performance



➲ **Prefetching speeds up all except jbb (up to 21%)**

48

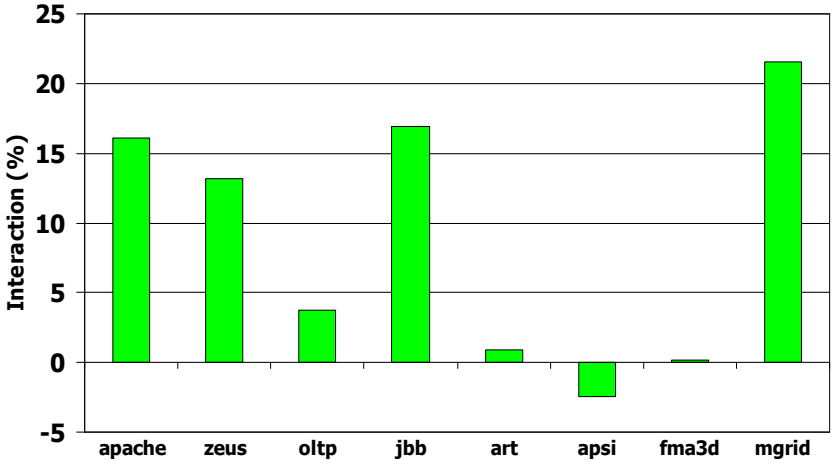
Performance



➲ **Compression&Prefetching have up to 51% speedups**

49

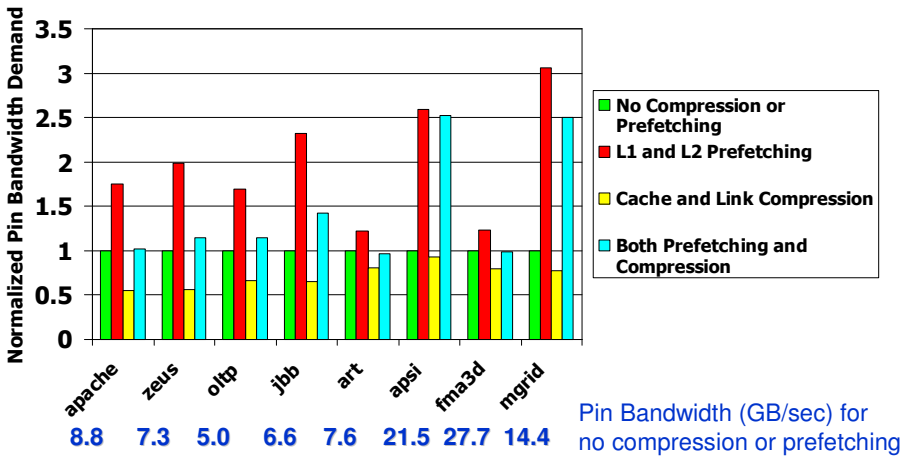
Interactions Between Prefetching and Compression



➲ **Interaction is positive for seven benchmarks**

50

Positive Interaction: Pin Bandwidth

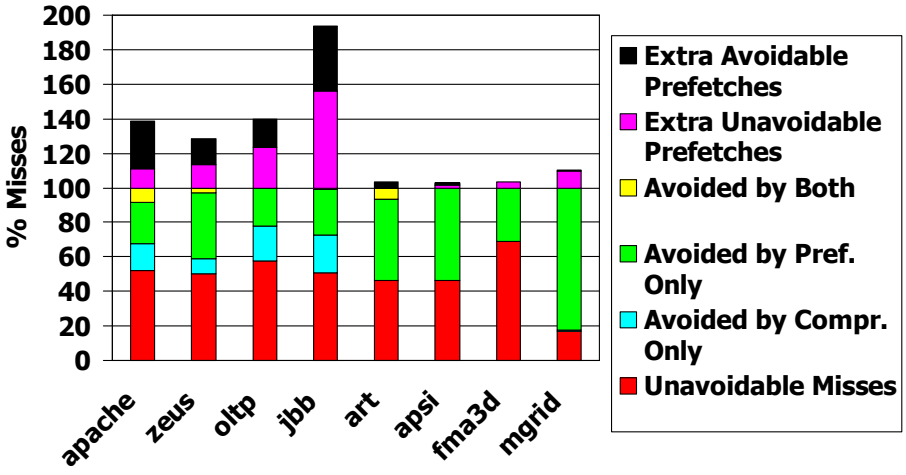


Pin Bandwidth (GB/sec) for no compression or prefetching

➲ **Compression saves bandwidth consumed by prefetching**

51

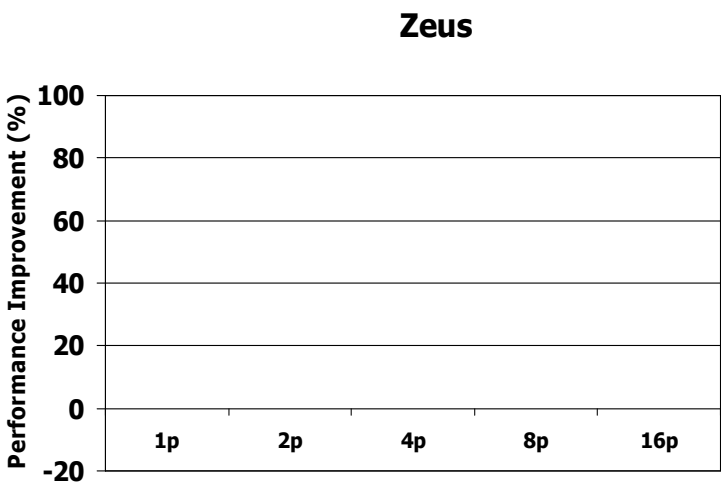
Negative Interaction: Avoided Misses



➲ **Small fraction of misses (<9%) avoided by both**

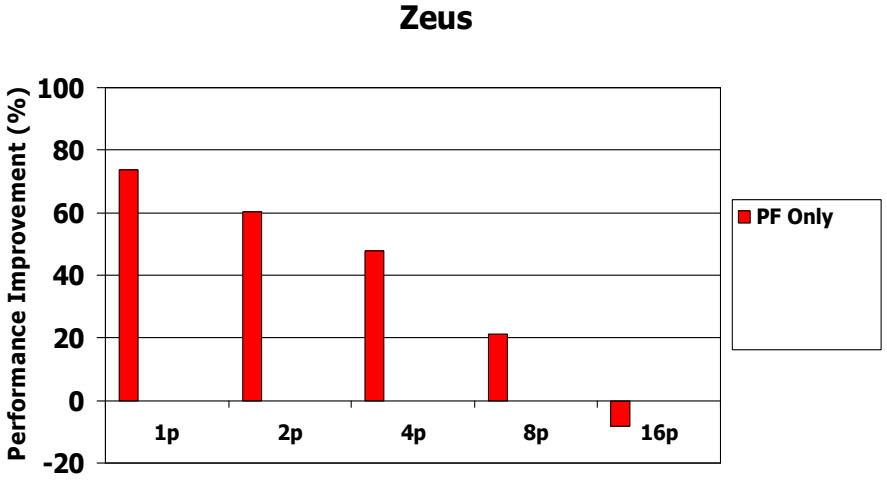
52

Sensitivity to #Cores



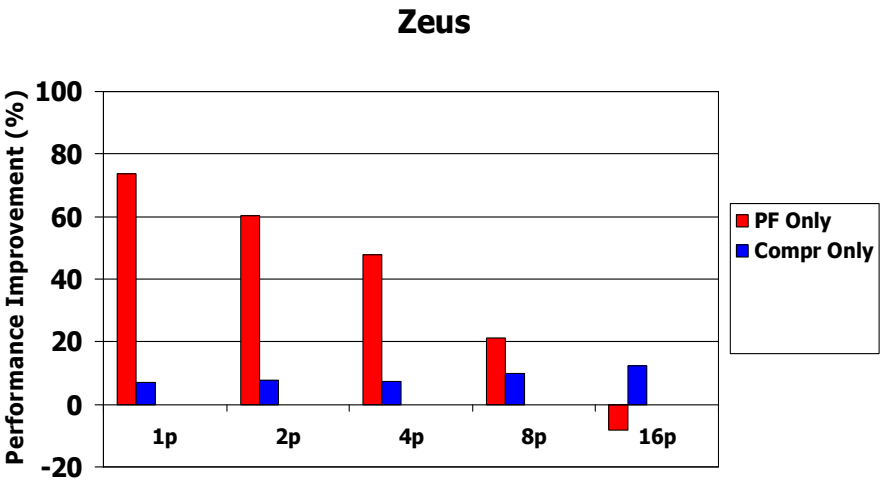
53

Sensitivity to #Cores



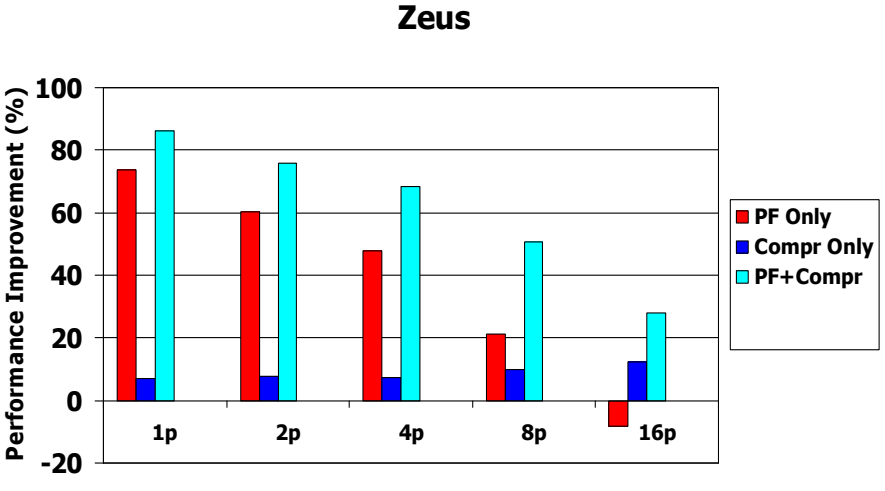
54

Sensitivity to #Cores



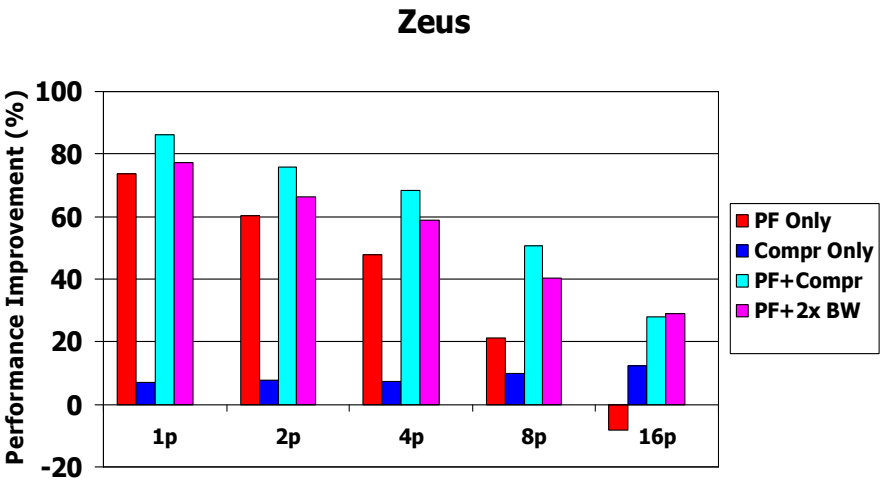
55

Sensitivity to #Cores



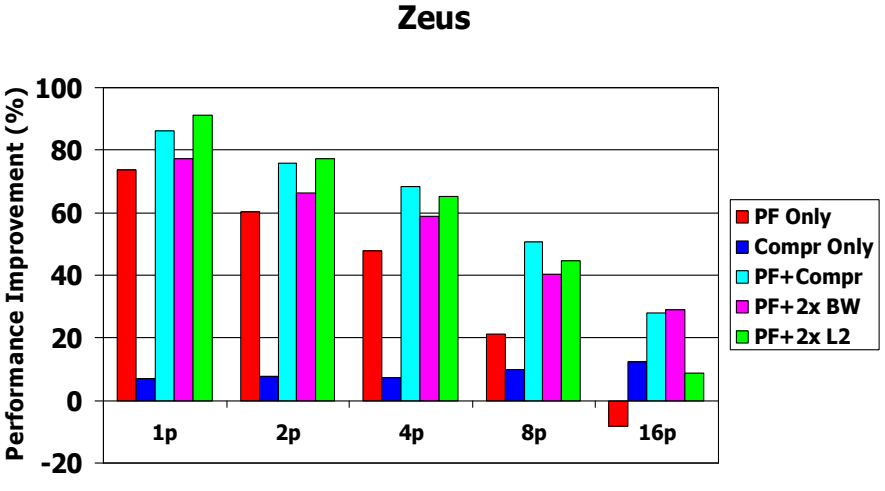
56

Sensitivity to #Cores



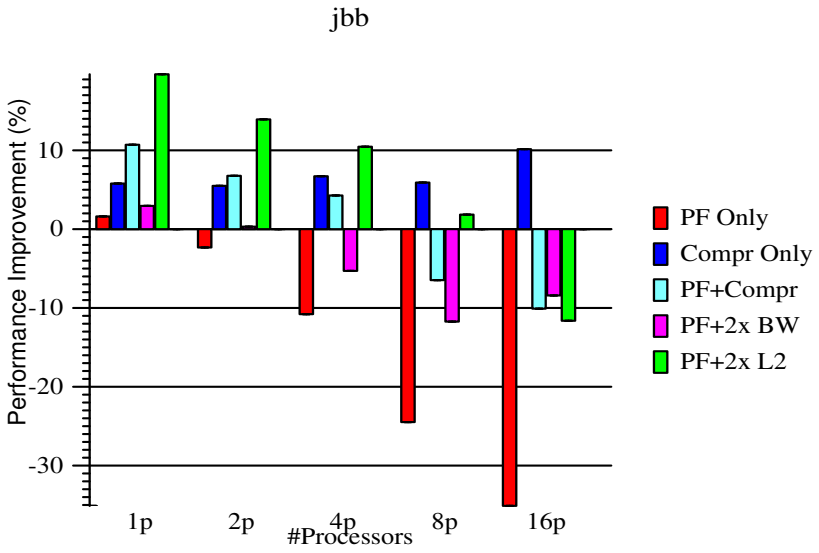
57

Sensitivity to #Cores



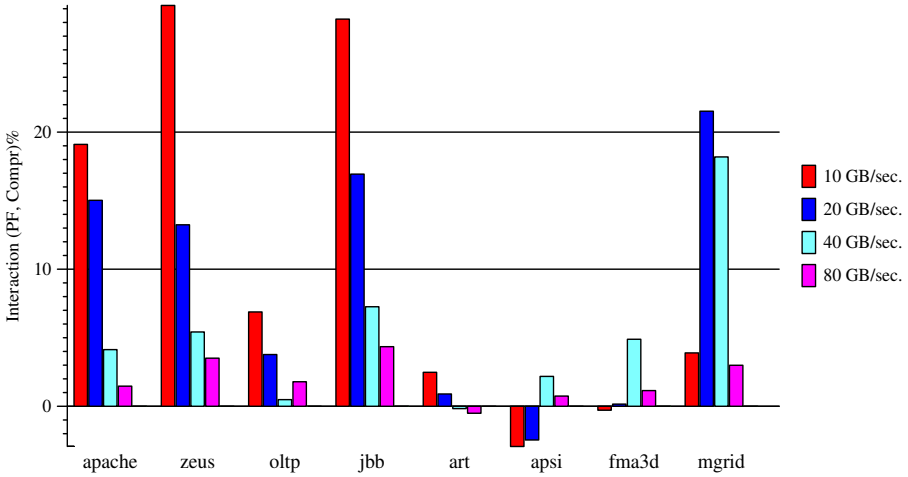
58

Sensitivity to #Cores



59

Sensitivity to Pin Bandwidth



➲ **Positive interactions for most configurations**

60

Compression and Prefetching: Summary

- More cores on a CMP increase demand for:
  - *On-chip (shared) caches*
  - *Off-chip pin bandwidth*
- Prefetching further increases demand on both resources
- Cache and link compression alleviate such demand
- Compression interacts positively with hardware prefetching

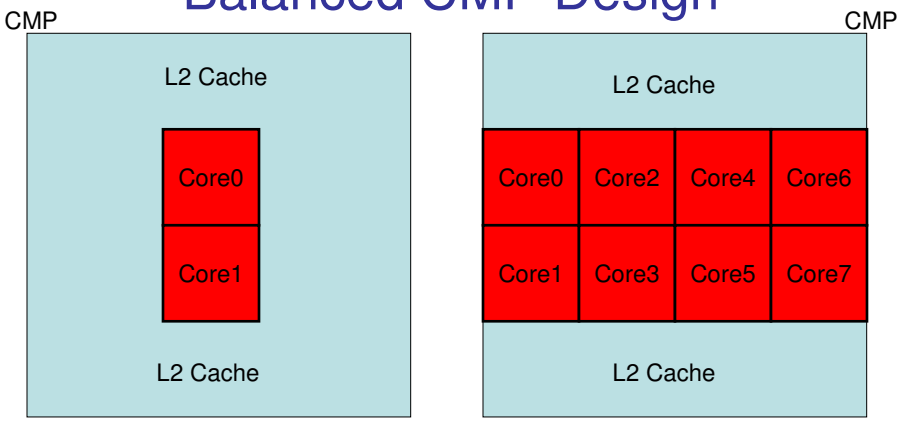
61

Outline

- Background
- Compressed Cache Design
- Adaptive Compression
- CMP Cache and Link Compression
- Interactions with Hardware Prefetching
- Balanced CMP Design
  - Analytical Model
  - Simulation
- Conclusions

62

Balanced CMP Design



- Compression can shift this balance
  - Increases effective cache size (small area overhead)
  - Increases effective pin bandwidth
  - Can we have more cores in a CMP?
- Explore by analytical model & simulation

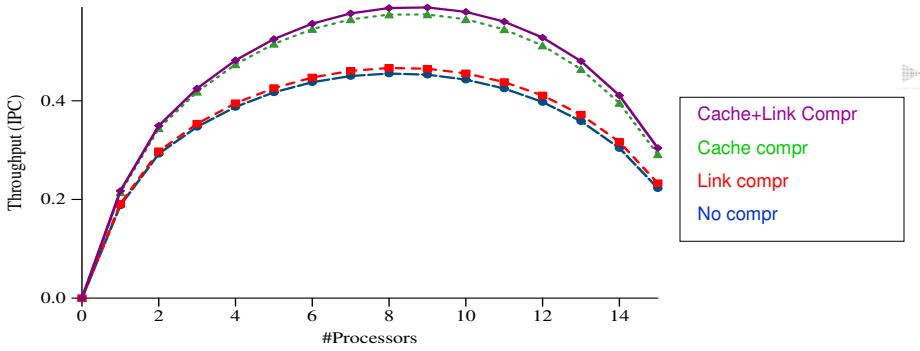
63

Simple Analytical Model

- Provides intuition on core vs. cache trade-off
- Model simplifying assumptions:
  - Pin bandwidth demand follows an M/D/1 model
  - Miss rate decreases with square root of increase in cache size
  - Blocking in-order processor
  - Some parameters are fixed with change in #processors
  - Uses IPC instead of a work-related metric

64

Throughput (IPC)

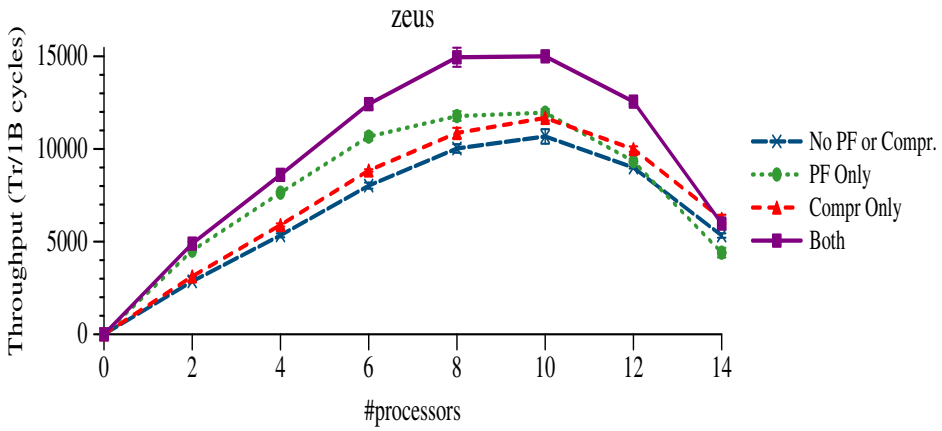


➲ **Cache compression provides speedups of up to 26% (29% when combined with link compression**

➲ **Higher speedup for optimal configuration**

65

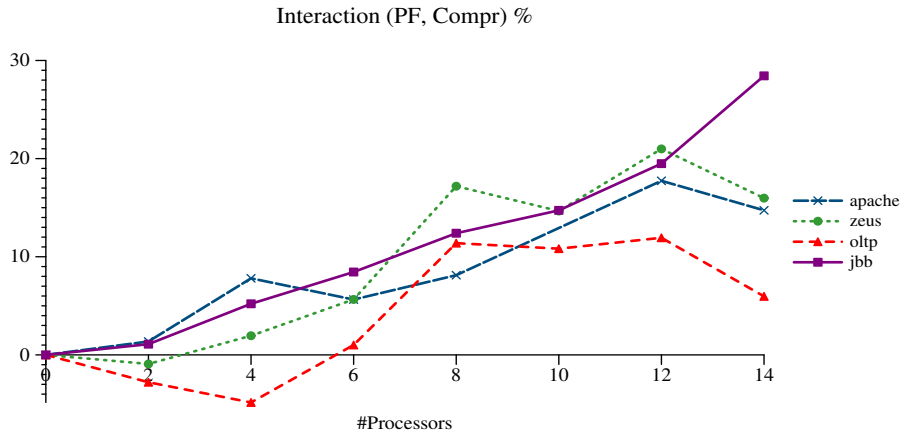
Simulation (20 GB/sec bandwidth)



➲ **Compression and prefetching combine to significantly improve throughput**

66

Compression & Prefetching Interaction



➲ **Interaction is positive for most configurations (and all "optimal" configurations)**

67

Balanced CMP Design: Summary

- Analytical model can qualitatively predict throughput
  - Can provide intuition into trade-off
  - Quickly analyzes sensitivity to CMP parameters
  - Not accurate enough to estimate throughput
- Compression improves throughput across all configurations
  - Larger improvement for "optimal" configuration
- Compression can shift balance towards more cores
- Compression interacts positively with prefetching for most configurations

68

Related Work (1/2)

- Memory Compression
  - IBM MXT technology
  - Compression schemes: X-Match, X-RL
  - Significance-based compression: Ekman and Stenstrom
- Virtual Memory Compression
  - Wilson et al.: varying compression cache size
- Cache Compression
  - Selective compressed cache: compress blocks to half size
  - Frequent value cache: frequent L1 values stored in cache
  - Hallnor and Reinhardt: Use indirect indexed cache for compression

69

Related Work (2/2)

- Link Compression
  - Farrens and Park: address compaction
  - Citron and Rudolph: table-based approach for address & data
- Prefetching in CMPs
  - IBM's Power4 and Power5 stride-based prefetching
  - Beckmann and Wood: prefetching improves 8-core performance
  - Gunasov and Burtscher: One CMP core dedicated to prefetching
- Balanced CMP Design
  - Huh et al.: Pin bandwidth a first-order constraint
  - Davis et al.: Simple Chip multi-threaded cores maximize throughput

70

Conclusions

- CMPs increase demand on caches and pin bandwidth
  - Prefetching further increases such demand
- Cache Compression
  - \+ Increases effective cache size - Increases cache access time
- Link Compression decreases bandwidth demand
- *Adaptive Compression*
  - Helps programs that benefit from compression
  - Does not hurt programs that are hurt by compression
- *CMP Cache and Link Compression*
  - Improve CMP throughput
  - Interact positively with hardware prefetching

➢ Compression improves CMP performance

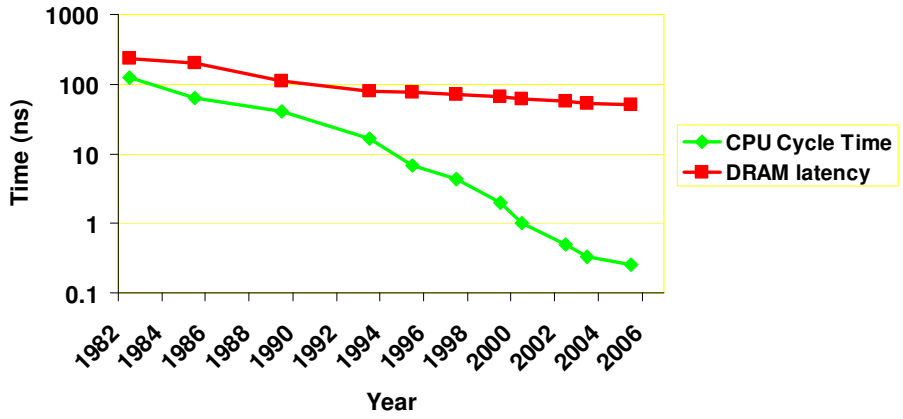
71

Backup Slides

- Moore's Law: CPU vs. Memory Speed
- Moore's Law (1965)
- Software Trends
- Decoupled Variable-Segment Cache
- Classification of L2 Accesses
- Compression Ratios
- Seg. Compr. Ratios SPECint SPECfp Commercial
- Frequent Pattern Histogram
- Segment Histogram
- (LRU) Stack Replacement
- Cache Bits Read or Written
- Sensitivity to L2 Associativity
- Sensitivity to Memory Latency
- Sensitivity to Decompression Latency
- Sensitivity to Cache Line Size
- Phase Behavior
- Commercial CMP Designs
- CMP Compression – Miss Rates
- CMP Compression: Pin Bandwidth Demand
- CMP Compression: Sensitivity to L2 Size
- CMP Compression: Sensitivity to Memory Latency
- CMP Compression: Sensitivity to Pin Bandwidth
- Prefetching Properties (8p)
- Sensitivity to #Cores – OLTP
- Sensitivity to #Cores – Apache
- Analytical Model: IPC
- Model Parameters
- Model - Sensitivity to Memory Latency
- Model - Sensitivity to Pin Bandwidth
- Model - Sensitivity to L2 Miss rate
- Model-Sensitivity to Compression Ratio
- Model - Sensitivity to Decompression Penalty
- Model - Sensitivity to Perfect CPI
- Simulation (20 GB/sec bandwidth) – apache
- Simulation (20 GB/sec bandwidth) – oltp
- Simulation (20 GB/sec bandwidth) – jbb
- Simulation (10 GB/sec bandwidth) – zeus
- Simulation (10 GB/sec bandwidth) – apache
- Simulation (10 GB/sec bandwidth) – oltp
- Simulation (10 GB/sec bandwidth) – jbb
- Compression & Prefetching Interaction – 10 GB/sec pin bandwidth
- Model Error apache, zeus oltp, jbb
- Online Transaction Processing (OLTP)
- Java Server Workload (SPECjbb)
- Static Web Content Serving: Apache

72

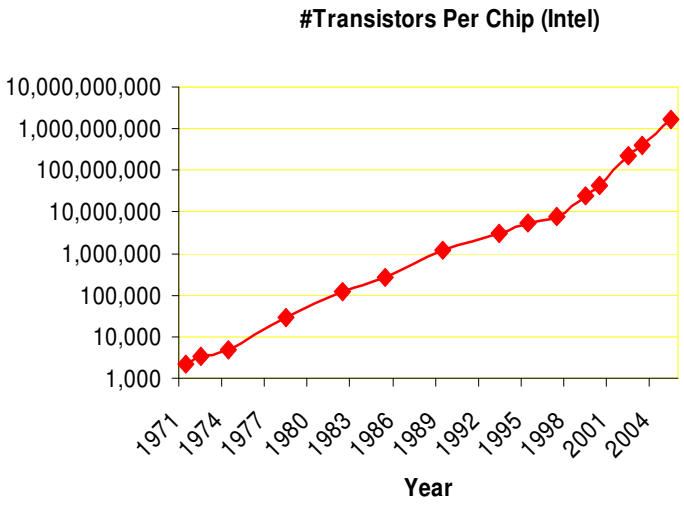
Moore's Law: CPU vs. Memory Speed



- CPU cycle time: 500 times faster since 1982
- DRAM Latency: Only ~5 times faster since 1982

73

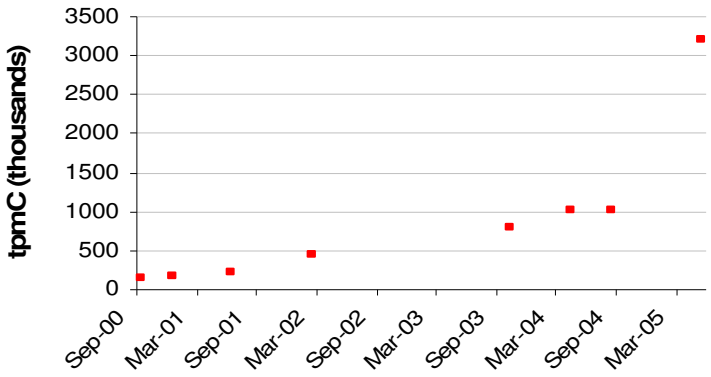
Moore's Law (1965)



Almost 75% increase per year

74

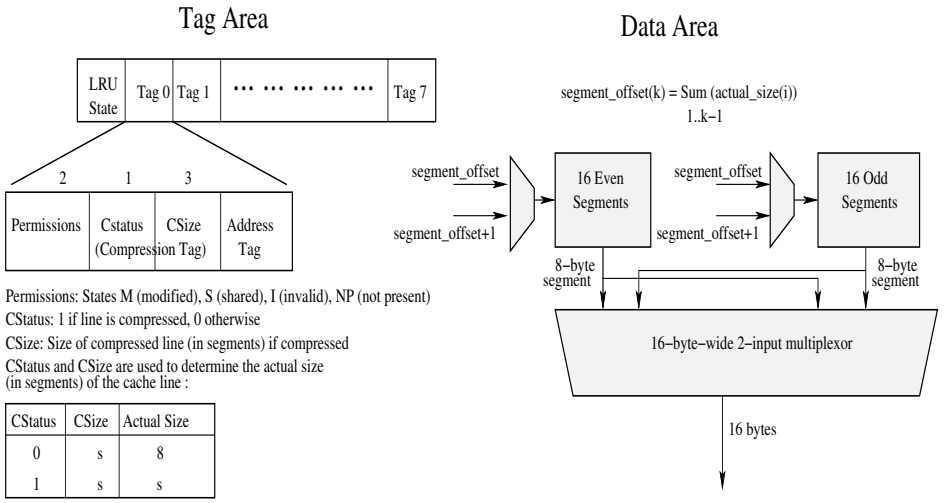
Software Trends



- Software trends favor more cores and higher off-chip bandwidth

75

Decoupled Variable-Segment Cache



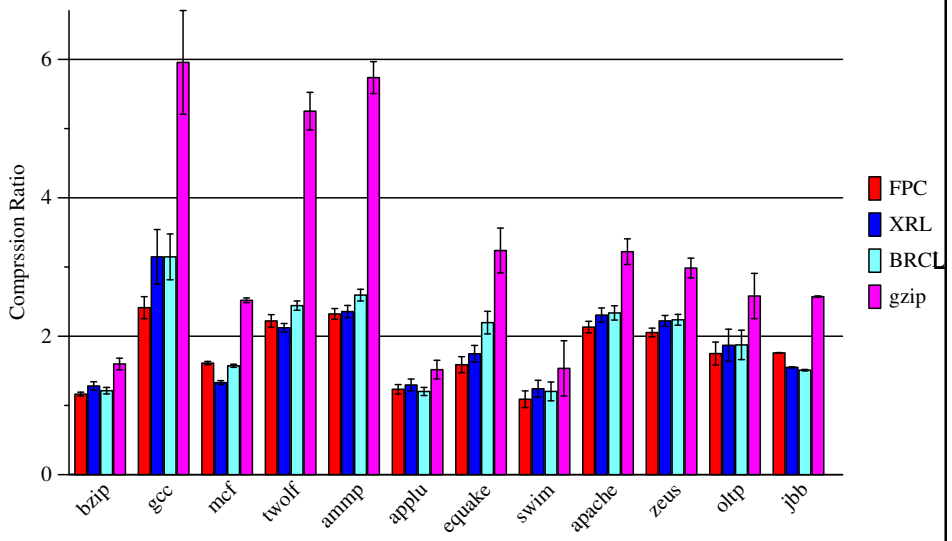
76

Classification of L2 Accesses

- Cache hits:
  - **Unpenalized hit:** Hit to an **uncompressed** line that **would have hit** without compression
  - − **Penalized hit:** Hit to a **compressed** line that **would have hit** without compression
  - \+ **Avoided miss:** Hit to a line that **would NOT have hit** without compression
- Cache misses:
  - \+ **Avoidable miss:** Miss to a line that **would have hit** with compression
  - **Unavoidable miss:** Miss to a line that **would have missed** even with compression

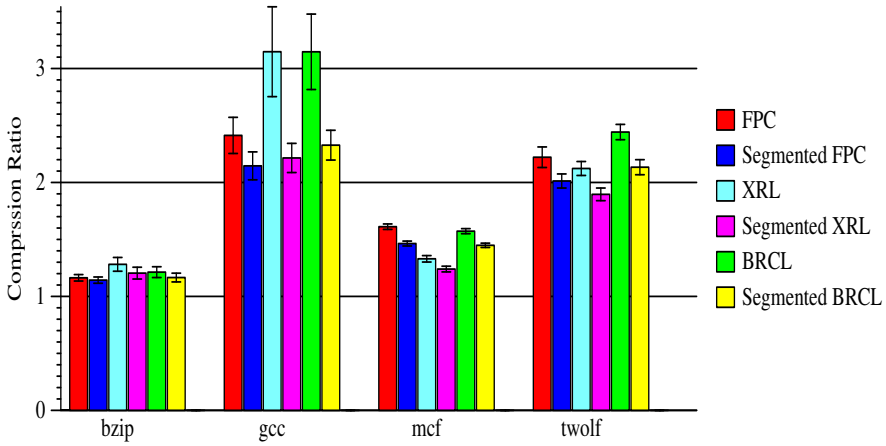
77

Compression Ratios



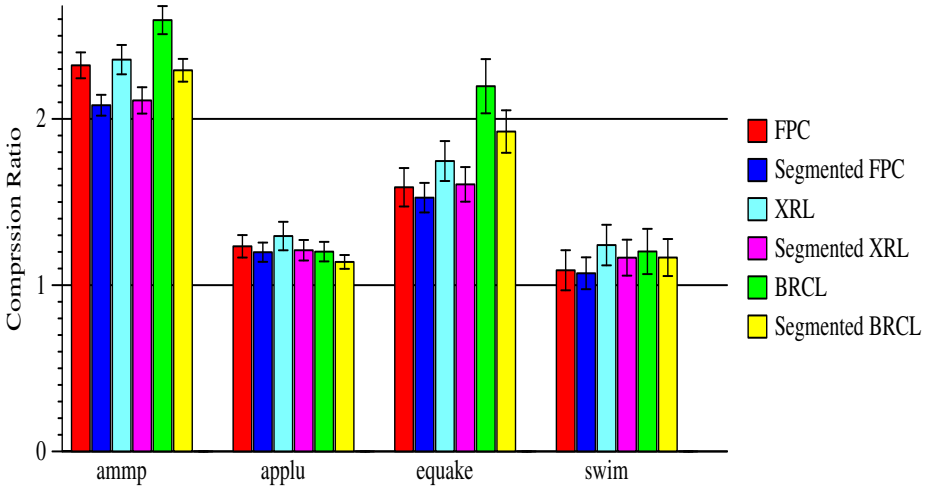
78

Seg. Compression Ratios - SPECint



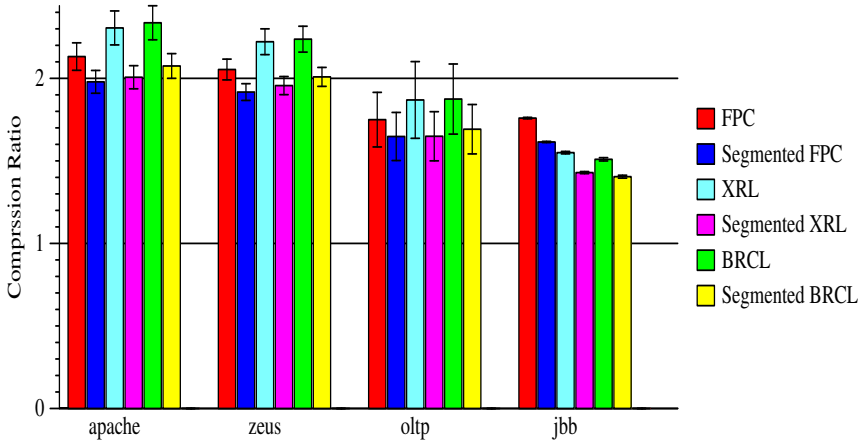
79

Seg. Compression Ratios - SPECfp



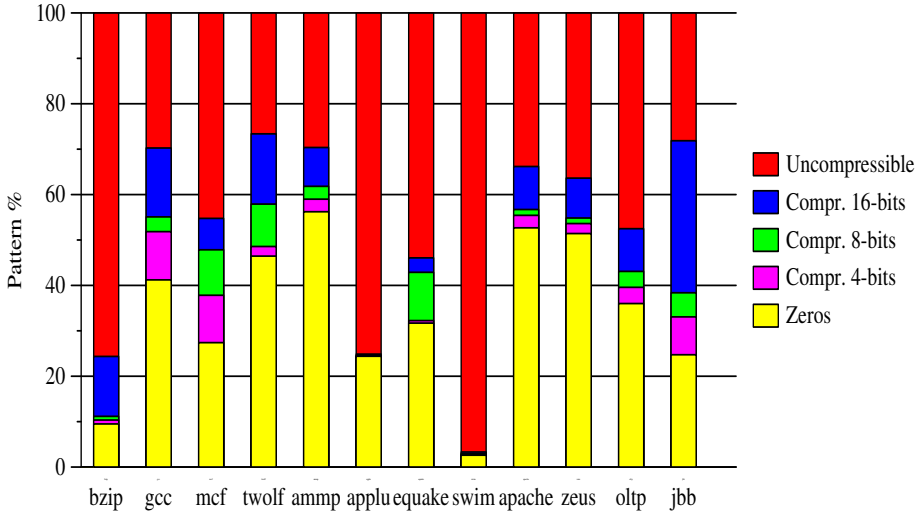
80

Seg. Compression Ratios - Commercial



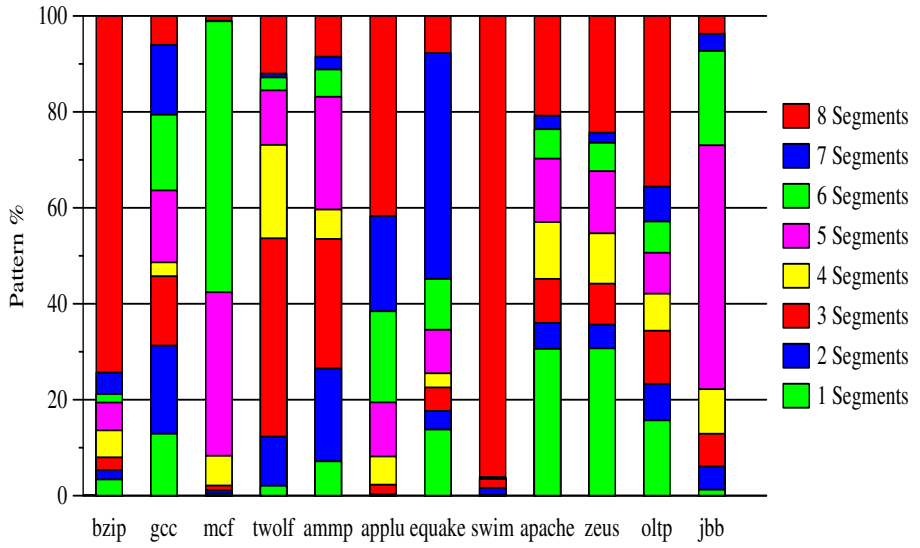
81

Frequent Pattern Histogram



82

Segment Histogram



83

(LRU) Stack Replacement

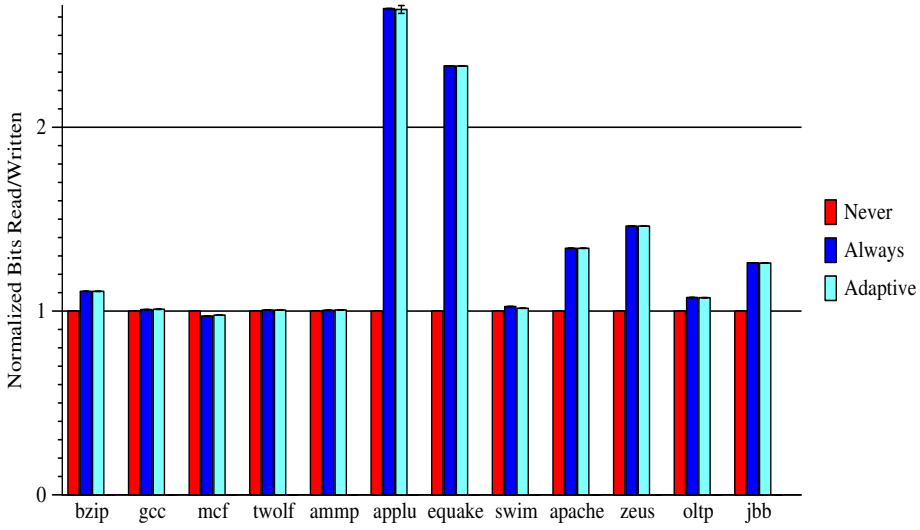
- Differentiate penalized hits and avoided misses?
  - Only hits to top half of the tags in the LRU stack are penalized hits
- Differentiate avoidable and unavoidable misses?

$$Avoidable\_Miss(k) \Leftrightarrow \sum_{LRU(i)=1}^{LRU(i)=LRU(k)} CSize(i) \leq 16$$

- Is not dependent on LRU replacement
  - Any replacement algorithm for top half of tags
  - Any stack algorithm for the remaining tags

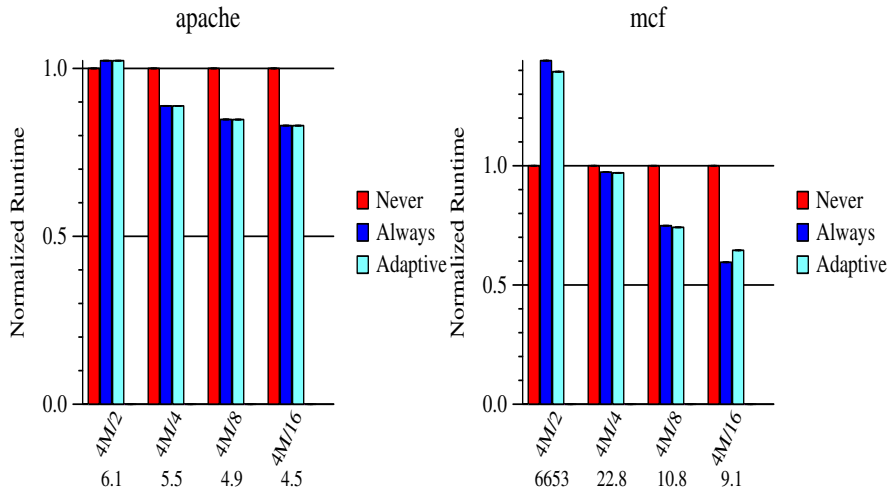
84

Cache Bits Read or Written



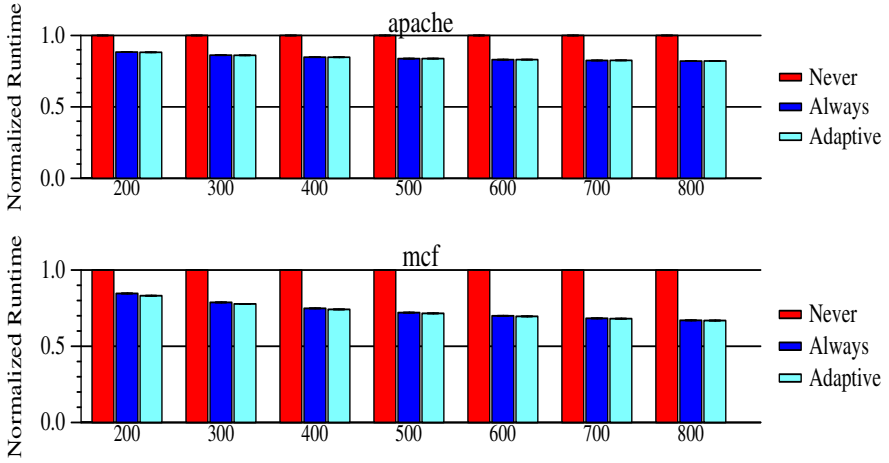
85

Sensitivity to L2 Associativity



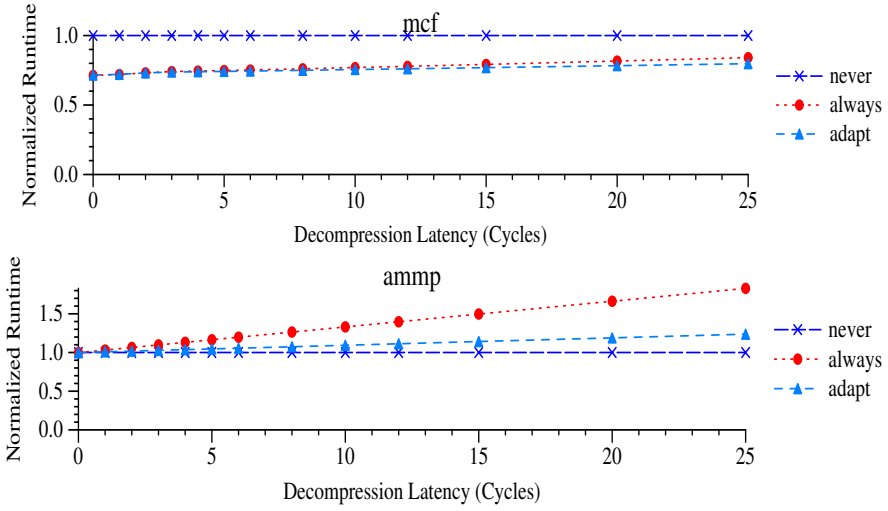
86

Sensitivity to Memory Latency



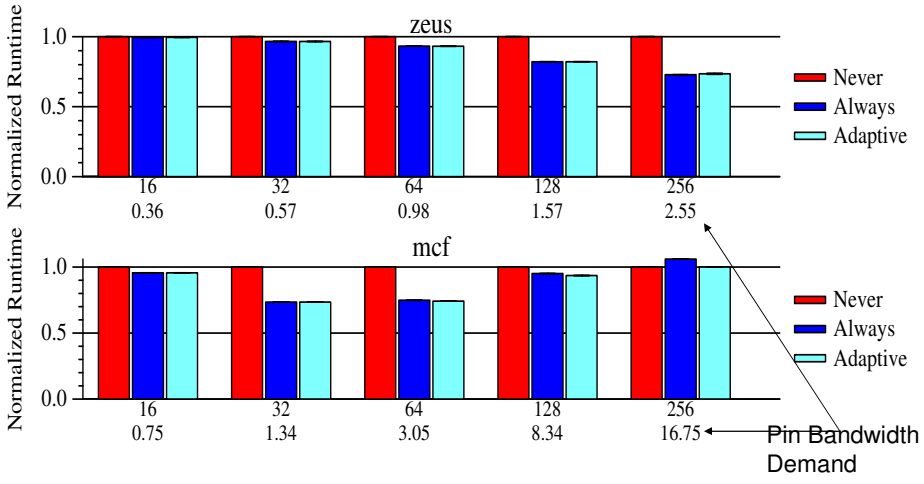
87

Sensitivity to Decompression Latency



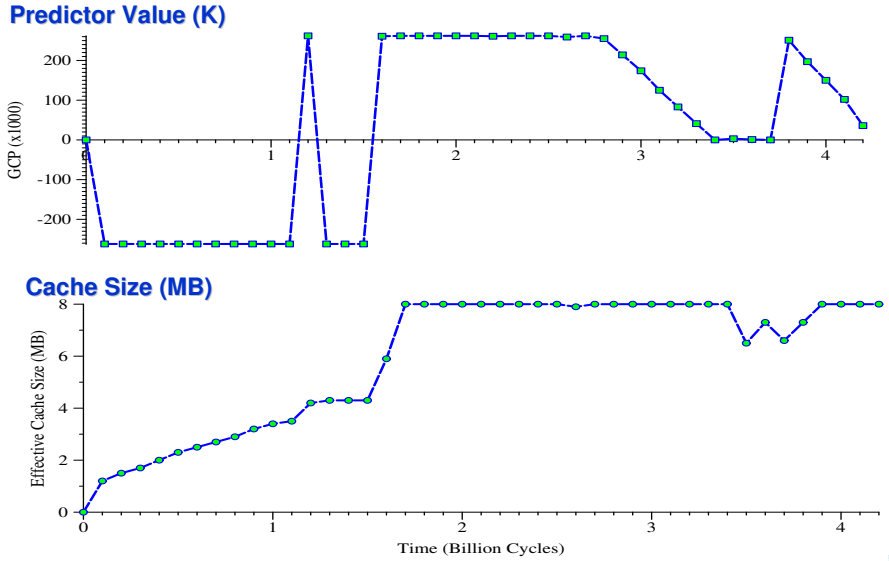
88

Sensitivity to Cache Line Size



89

Phase Behavior



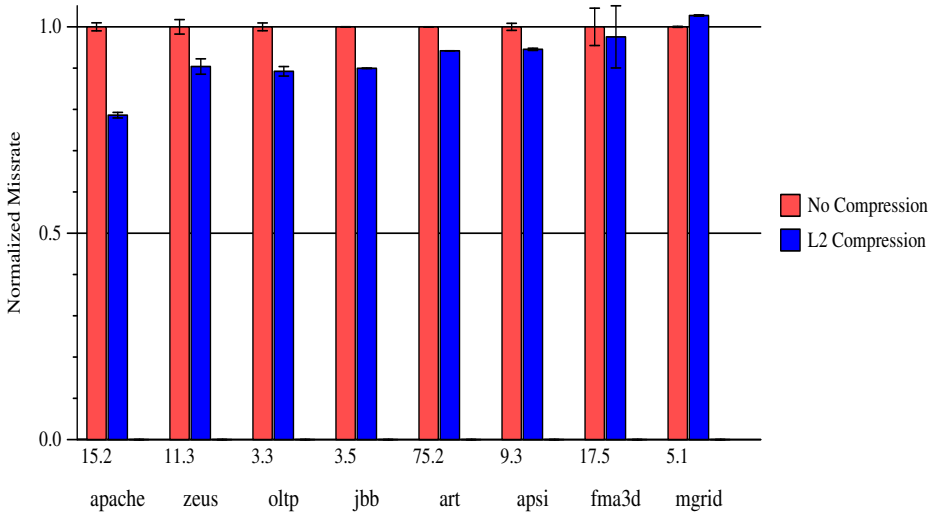
90

Commercial CMP Designs

- IBM Power5 Chip:
  - Two processor cores, each 2-way multi-threaded
  - ~1.9 MB on-chip L2 cache
    - < 0.5 MB per thread with no sharing
    - Compare with 0.75 MB per thread in Power4+
  - Est. ~16GB/sec. max. pin bandwidth
- Sun's Niagara Chip:
  - Eight processor cores, each 4-way multi-threaded
  - 3 MB L2 cache
    - < 0.4 MB per core, < 0.1 MB per thread with no sharing
  - Est. ~22 GB/sec. pin bandwidth

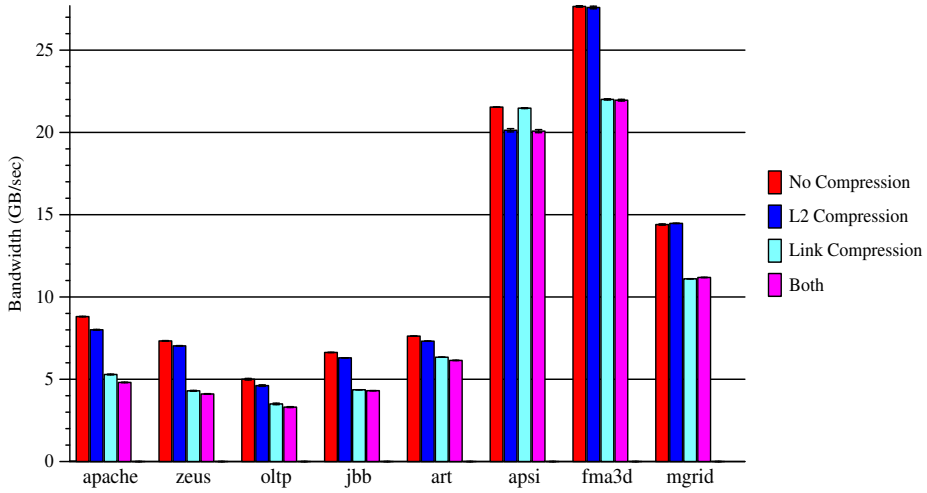
91

CMP Compression – Miss Rates



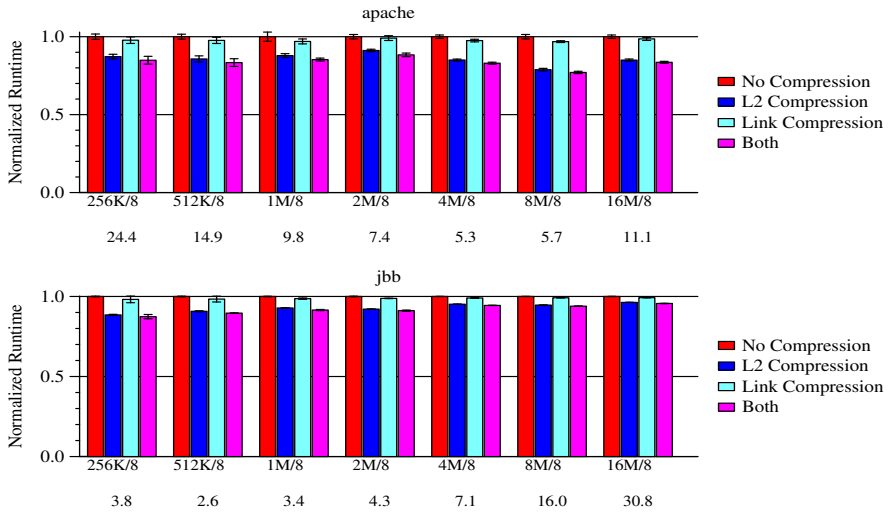
92

CMP Compression: Pin Bandwidth Demand



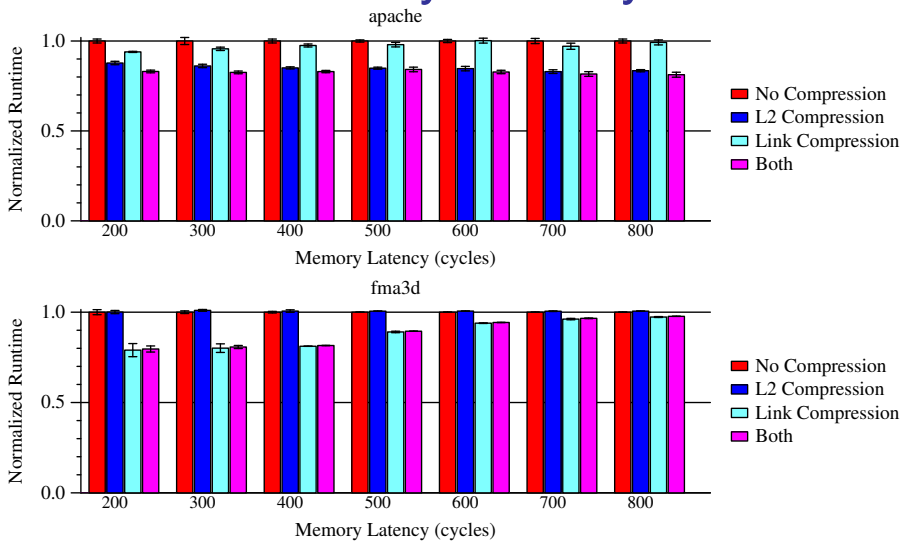
93

CMP Compression: Sensitivity to L2 Size



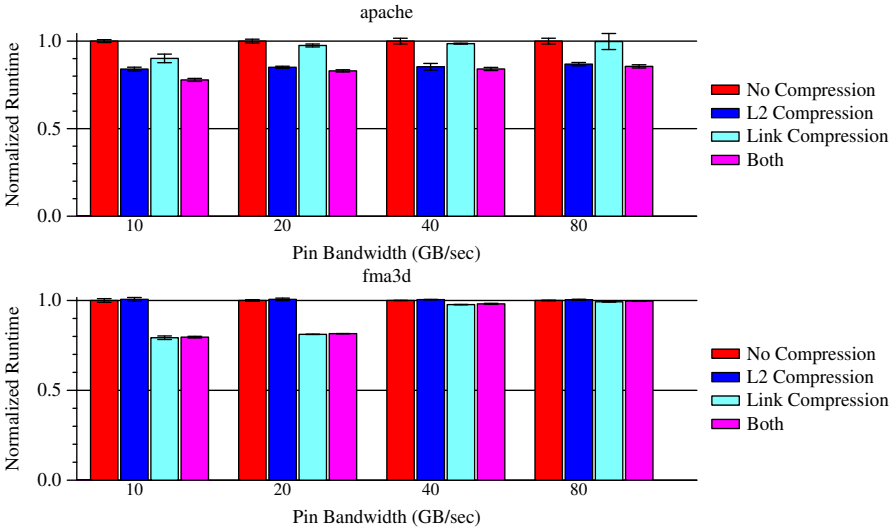
94

CMP Compression: Sensitivity to Memory Latency



95

CMP Compression: Sensitivity to Pin Bandwidth



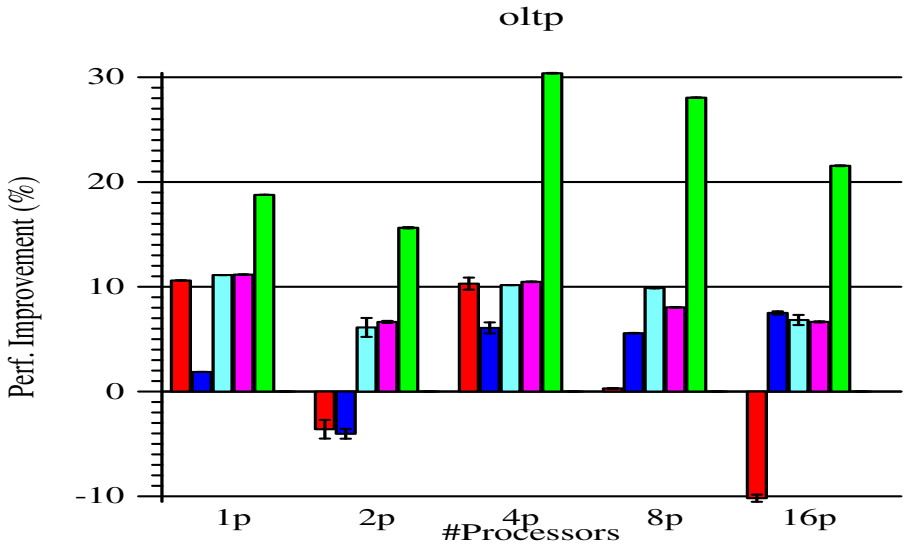
96

Prefetching Properties (8p)

Benchmark	L1 I Cache			L1 D Cache			L2 Cache		
	PF rate	coverage	accuracy	PF rate	coverage	accuracy	PF rate	coverage	accuracy
apache	4.9	16.4	42.0	6.1	8.8	55.5	10.5	37.7	57.9
zeus	7.1	14.5	38.9	5.5	17.7	79.2	8.2	44.4	56.0
oltp	13.5	20.9	44.8	2.0	6.6	58.0	2.4	26.4	41.5
jbb	1.8	24.6	49.6	4.2	23.1	60.3	**5.5**	**34.2**	**32.4**
art	0.05	9.4	24.1	56.3	30.9	81.3	49.7	56.0	85.0
apsi	0.04	15.7	30.7	8.5	25.5	96.9	4.6	95.8	97.6
fma3d	0.06	7.5	14.4	7.3	27.5	80.9	8.8	44.6	73.5
mgrid	0.06	15.5	26.6	8.4	80.2	94.2	6.2	89.9	81.9

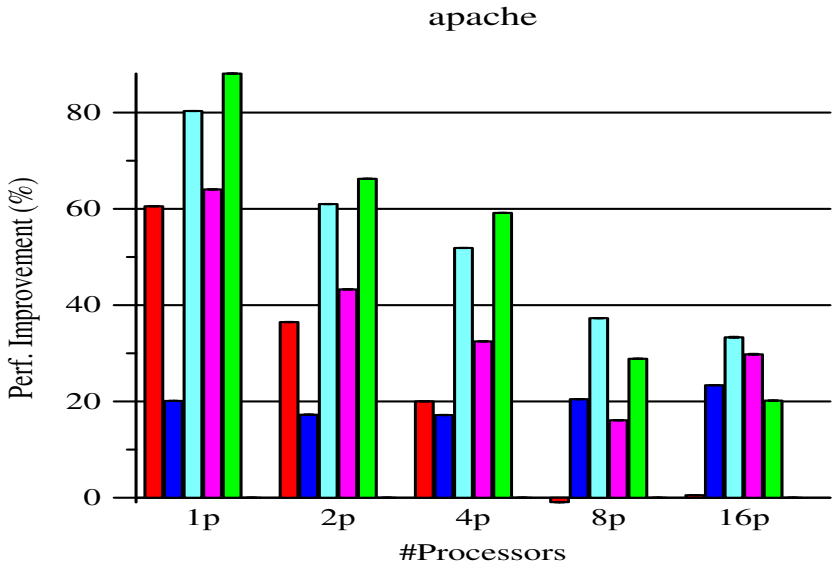
97

Sensitivity to #Cores - OLTP



98

Sensitivity to #Cores - Apache



99

Analytical Model: IPC

$$IPC(N) = \frac{N}{CPI_{PerfectL2} + dp + MissPenalty_{L2}.\alpha.\sqrt{\frac{N - sharers_{av}(N) + 1}{c.(m - k_p.N)}}}$$

$$MissLatency_{L2} = MemoryLatency + LinkLatency$$

$$LinkLatency = \overline{X} + \frac{IPC(N).Missrate(S_{L2p}).\overline{X}^2}{2.(1 - IPC(N).Missrate(S_{L2p}).\overline{X})}$$

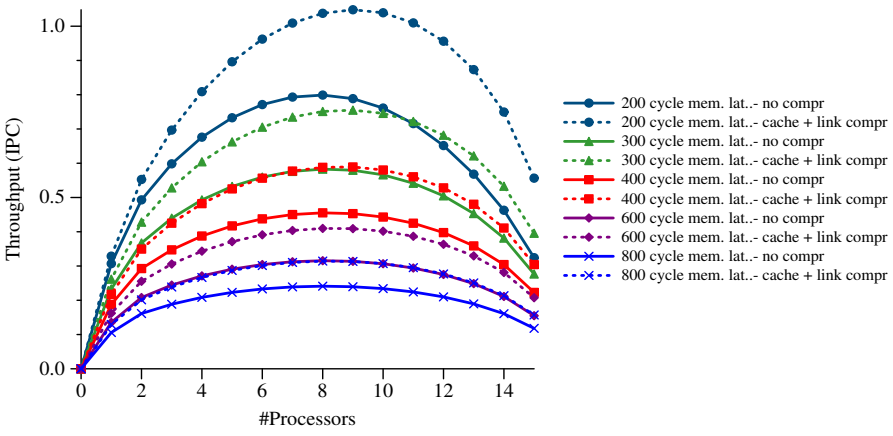
100

Model Parameters

- Divide chip area between cores and caches
  - Area of one (in-order) core = 0.5 MB L2 cache
  - Total chip area = 16 cores, or 8 MB cache
  - Core frequency = 5 GHz
  - Available bandwidth = 20 GB/sec.
- Model Parameters (hypothetical benchmark)
  - Compression Ratio = 1.75
  - Decompression penalty = 0.4 cycles per instruction
  - Miss rate = 10 misses per 1000 instructions for 1proc, 8 MB Cache
  - IPC for one processor, perfect cache = 1
  - Average #sharers per block = 1.3 (for #proc > 1)

101

Model - Sensitivity to Memory Latency

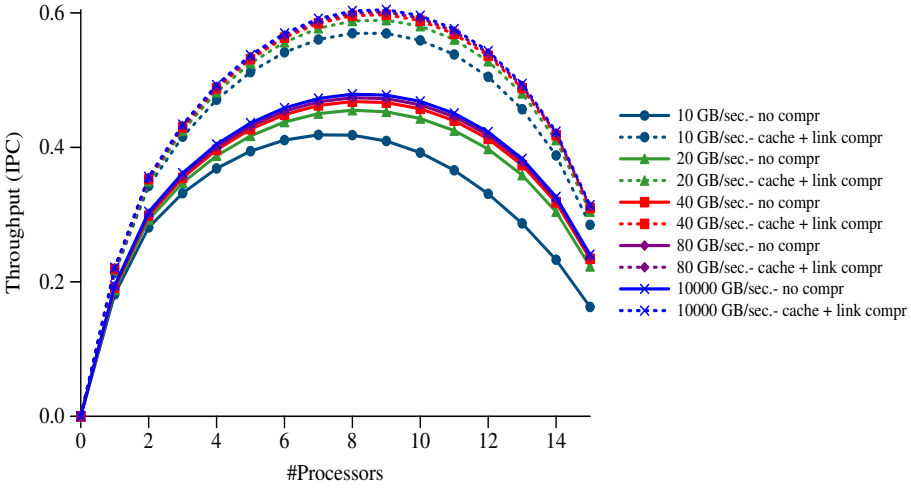


➲ **Compression's impact similar on both extremes**

➲ **Compression can shift optimal configuration towards more cores (though not significantly)**

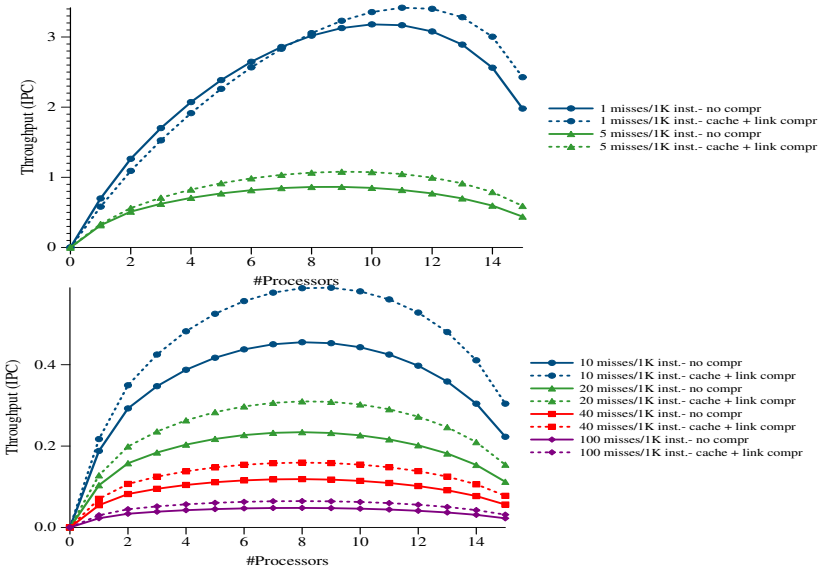
102

Model - Sensitivity to Pin Bandwidth



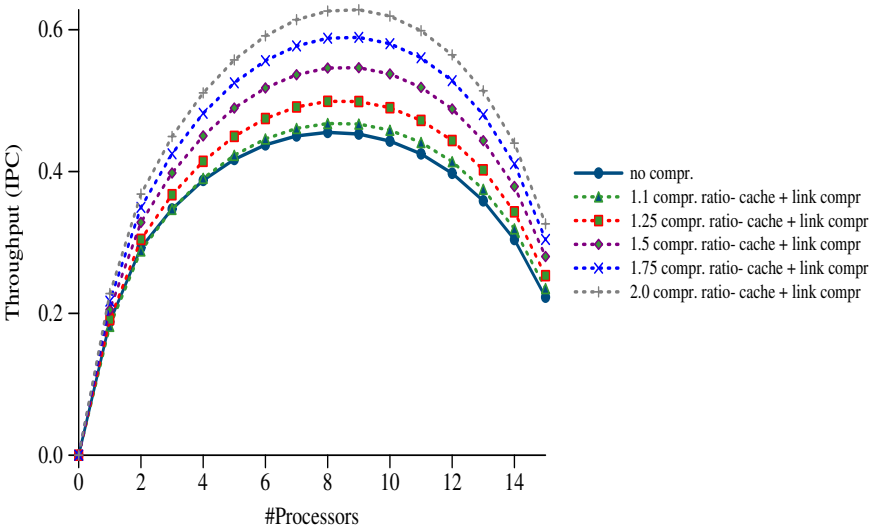
103

Model - Sensitivity to L2 Miss rate



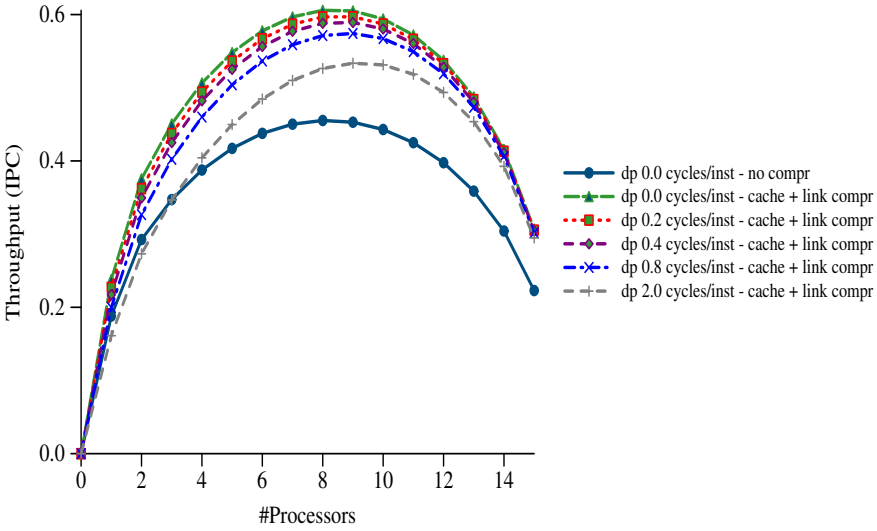
104

Model-Sensitivity to Compression Ratio



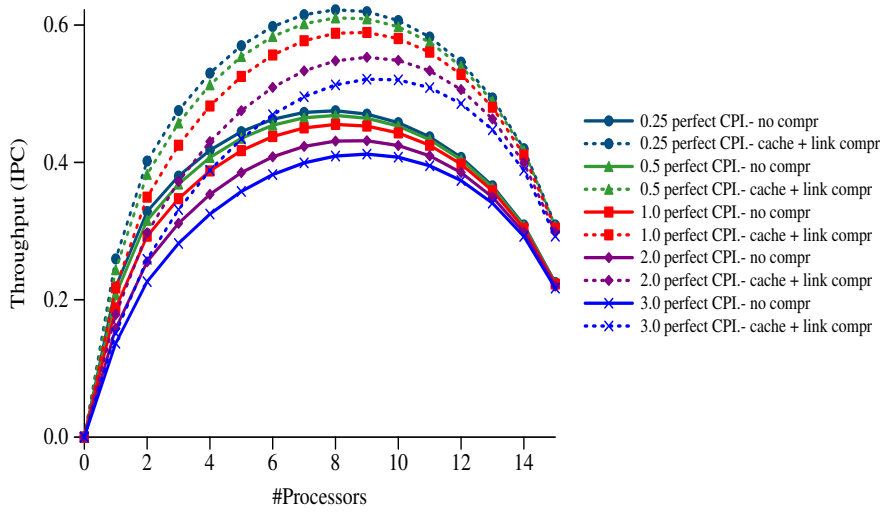
105

Model - Sensitivity to Decompression Penalty



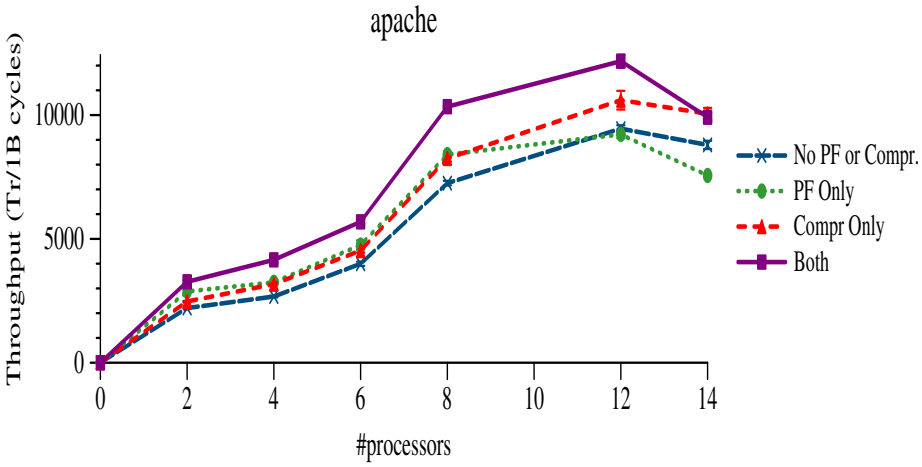
106

Model - Sensitivity to Perfect CPI



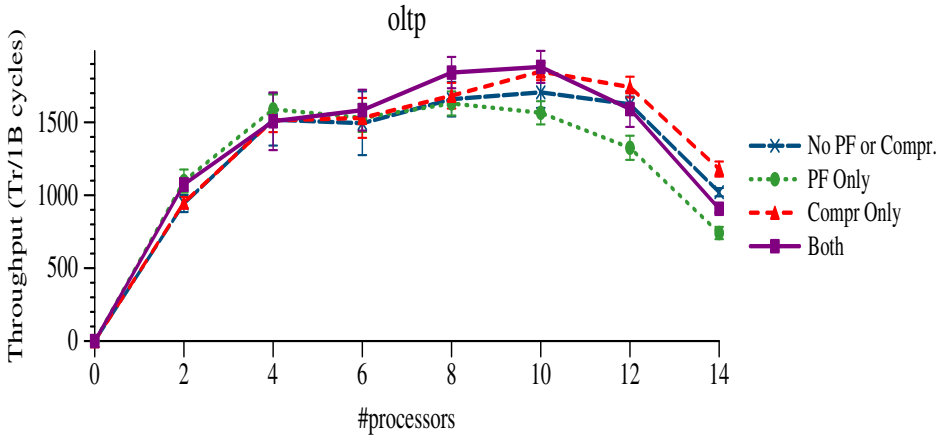
107

Simulation (20 GB/sec bandwidth) - apache



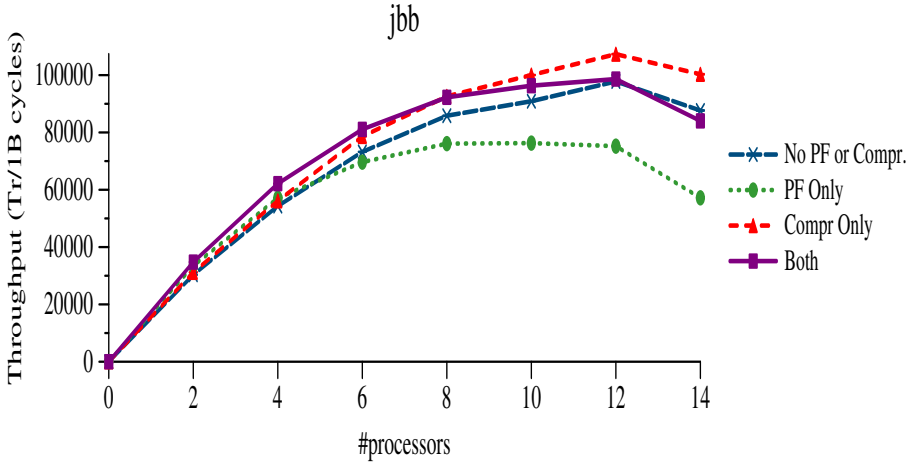
108

Simulation (20 GB/sec bandwidth) - oltp



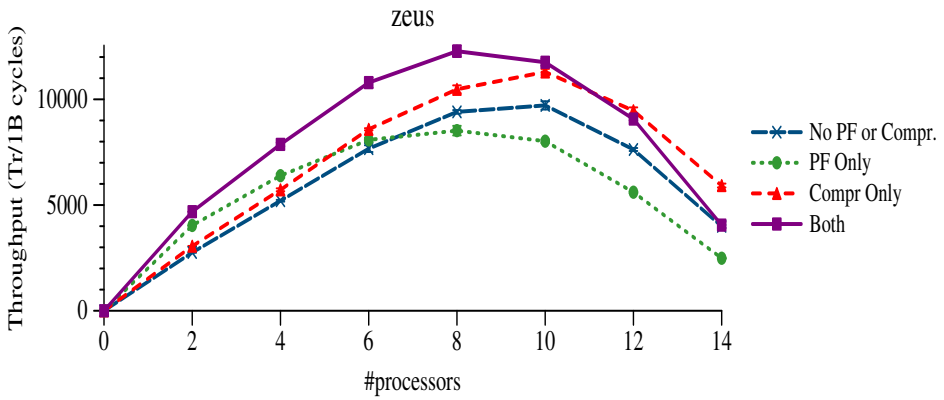
109

Simulation (20 GB/sec bandwidth) - jbb



110

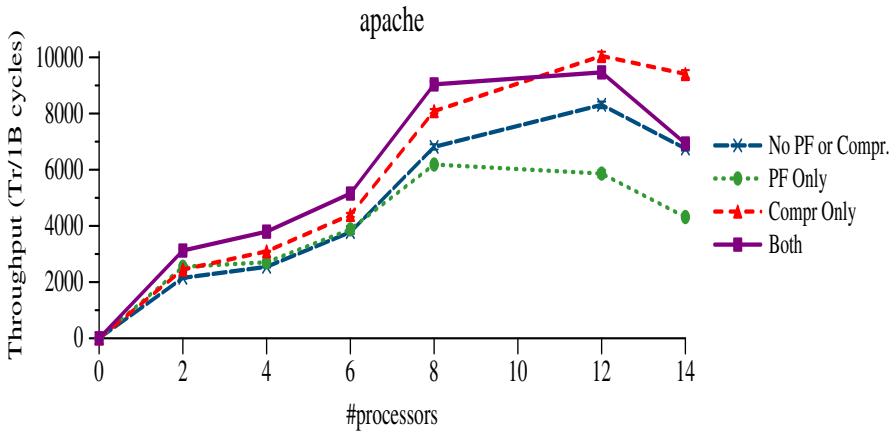
Simulation (10 GB/sec bandwidth) - zeus



- **Prefetching can degrade throughput for many systems**
- **Compression alleviates this performance degradation**

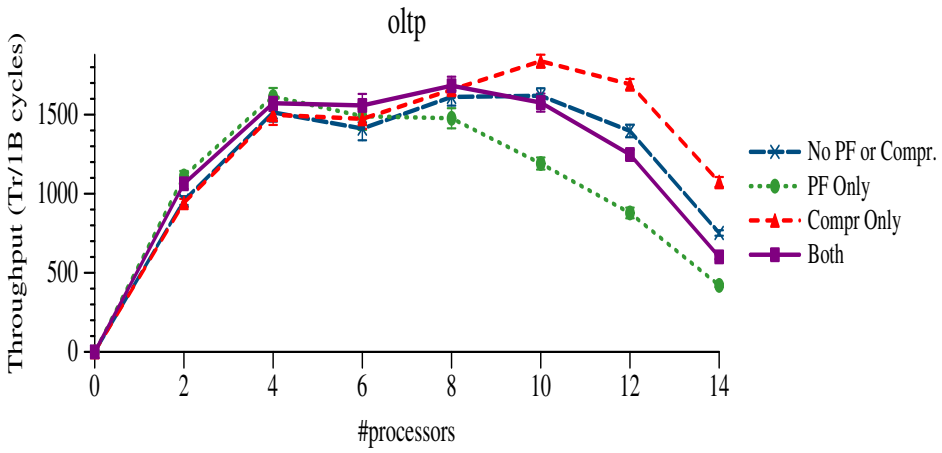
111

Simulation (10 GB/sec bandwidth) - apache



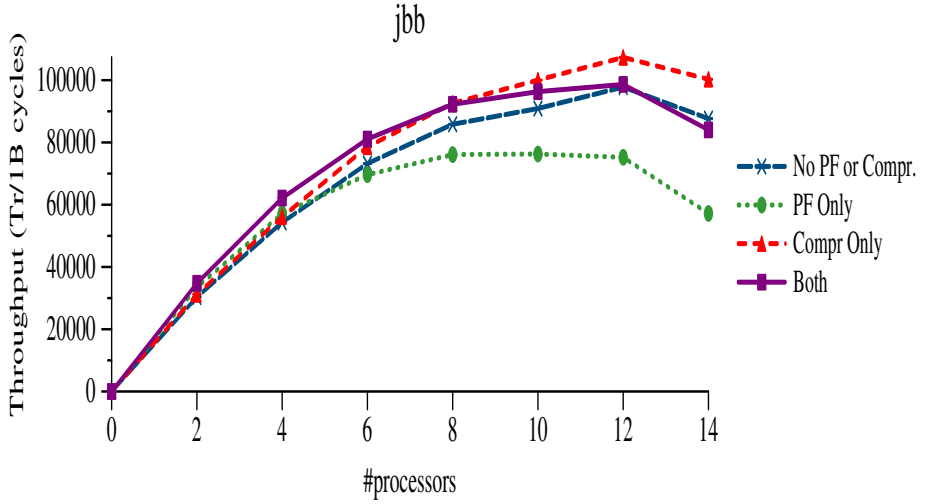
112

Simulation (10 GB/sec bandwidth) - oltp



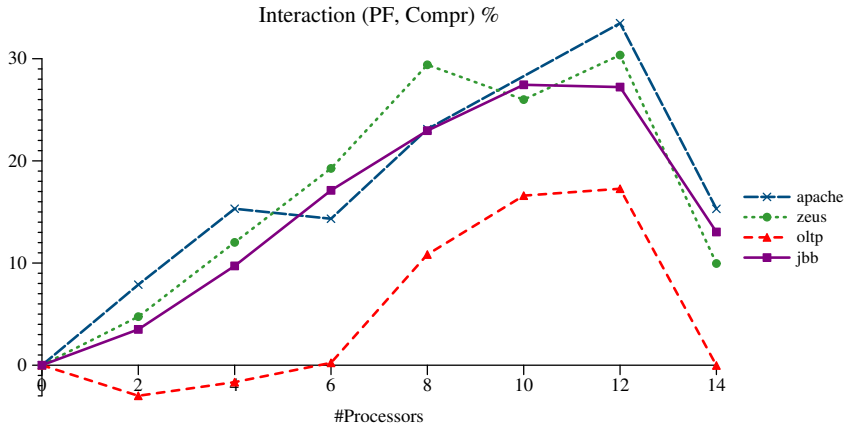
113

Simulation (10 GB/sec bandwidth) - jbb



114

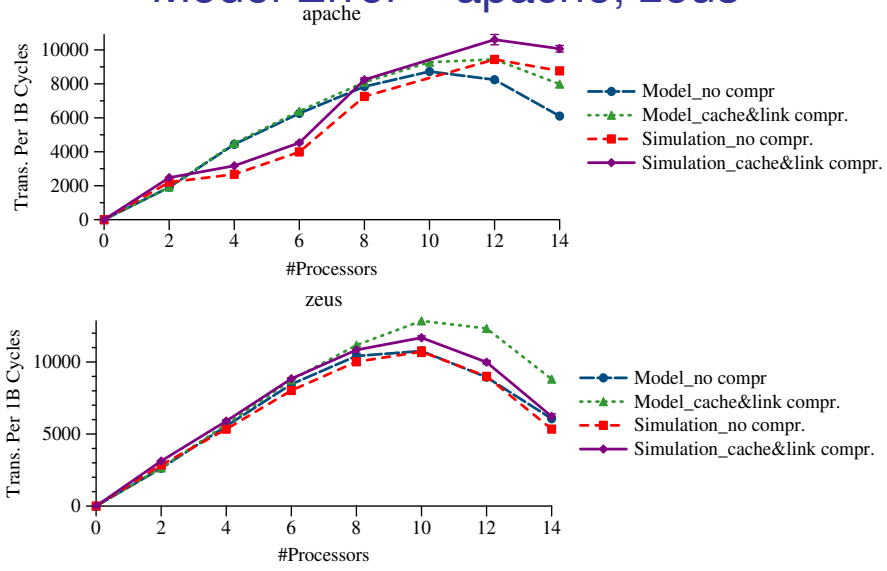
Compression & Prefetching Interaction – 10 GB/sec pin bandwidth



➲ **Interaction is positive for most configurations (and all "optimal" configurations)**

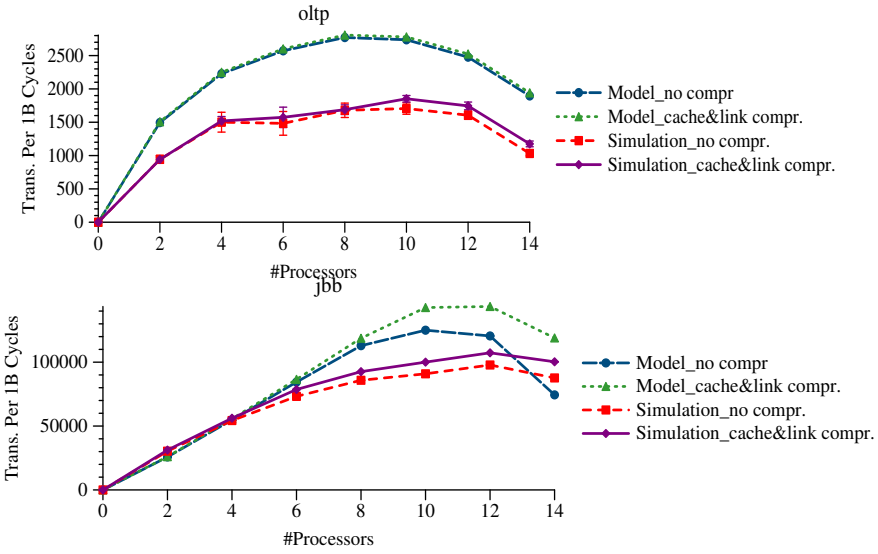
115

Model Error – apache, zeus



116

Model Error – oltp, jbb



117

Online Transaction Processing (OLTP)

- **DB2 with a TPC-C-like workload.**
  - Based on the TPC-C v3.0 benchmark.
  - We use IBM's DB2 V7.2 EEE database management system and an IBM benchmark kit to build the database and emulate users.
  - 5 GB 25000-warehouse database on eight raw disks and an additional dedicated database log disk.
  - We scaled down the sizes of each warehouse by maintaining the reduced ratios of 3 sales districts per warehouse, 30 customers per district, and 100 items per warehouse (compared to 10, 30,000 and 100,000 required by the TPC-C specification).
  - Think and keying times for users are set to zero.
  - 16 users per processor
  - Warmup interval: 100,000 transactions

118

Java Server Workload (SPECjbb)

- **SpecJBB.**
  - We used Sun's HotSpot 1.4.0 Server JVM and Solaris's native thread implementation
  - The benchmark includes driver threads to generate transactions
  - System heap size to 1.8 GB and the new object heap size to 256 MB to reduce the frequency of garbage collection
  - 24 warehouses, with a data size of approximately 500 MB.

119

Static Web Content Serving: Apache

- **Apache.**
  - We use Apache 2.0.39 for SPARC/Solaris 9 configured to use pthread locks and minimal logging at the web server
  - We use the Scalable URL Request Generator (SURGE) as the client.
  - SURGE generates a sequence of static URL requests which exhibit representative distributions for document popularity, document sizes, request sizes, temporal and spatial locality, and embedded document count
  - We use a repository of 20,000 files (totaling ~500 MB)
  - Clients have zero think time
  - We compiled both Apache and Surge using Sun's WorkShop C 6.1 with aggressive optimization

120