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

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

Technology and Software Trends

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

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

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 (Comp, Pref) > Speedup (Comp) 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
Pin Bottleneck: ITRS 04 Roadmap

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

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

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

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

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

Adaptive Compression
- Use past to predict future

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

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
A Penalized Hit

- Read/Write Address B
  - LRU Stack order = 2 \( \leq 2 \) \( \rightarrow \) Hit regardless of compression
  - Compressed Line \( \rightarrow \) Decompression penalty incurred
  - Compression cost

An Unavoidable Miss

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

An Avoided Miss

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

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 \( < 0 \)

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

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:
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

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

Performance

Adaptive performs similar to the best of Always and Never
Cache Miss Rates

Optimal Adaptive Compression?

Adapting to L2 Size

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

Outline

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

Compressed Cache Hierarchy (CMP)
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

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?

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?

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/normal/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

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) )

Performance
Cache compression provides speedups of up to 18%.

Link compression speeds up bandwidth-limited applications.

Prefetching speeds up all except jbb (up to 21%).
Performance

Compression & Prefetching have up to 51% speedups

Compression & Prefetching have up to 51% speedups

Interactions Between Prefetching and Compression

Interaction (%)

Interaction is positive for seven benchmarks

Positive Interaction: Pin Bandwidth

Compression saves bandwidth consumed by prefetching

Sensitivity to #Cores

Zeus

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

Sensitivity to #Cores

Zeus

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

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

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

Throughput (IPC)
- Cache compression provides speedups of up to 26% (29% when combined with link compression)
- Higher speedup for optimal configuration

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

Simulation (20 GB/sec bandwidth)
- Compression and prefetching combine to significantly improve throughput
**Compression & Prefetching Interaction**

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

**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

**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
  - Hallin and Reinhardt: Use indirect indexed cache for compression

**Related Work (2/2)**

- Link Compression
  - Farrons 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 Buntscher: 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

**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

**Backup Slides**

- Memory Law, CPU vs. Memory Speed
- Moore's Law: CPU vs. Memory Speed
- 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
- Hallin and Reinhardt: Use Indirect Indexed Cache for Compression
- Link Compression
- Farrons 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 Buntscher: 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
Moore’s Law: CPU vs. Memory Speed

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

Decoupled Variable-Segment Cache

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

Software Trends

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

Compression Ratios
**Seg. Compression Ratios - SPECint**

- fpc
- Segmented FPC
- xrl
- Segmented XRL
- brcl
- Segmented BRCL

**Seg. Compression Ratios - SPECfp**

- fpc
- Segmented FPC
- xrl
- Segmented XRL
- brcl
- Segmented BRCL

**Seg. Compression Ratios - Commercial**

- fpc
- Segmented FPC
- xrl
- Segmented XRL
- brcl
- Segmented BRCL

**Frequent Pattern Histogram**

- Uncompressible
- Compr. 16-bits
- Compr. 8-bits
- Compr. 4-bits
- Noise

**Segment Histogram**

- 8 Segments
- 7 Segments
- 6 Segments
- 5 Segments
- 4 Segments
- 3 Segments
- 2 Segments

**(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?

\[
\text{Avoidable Miss}(k) \Leftrightarrow \sum_{|LRU(i)|=1}^{|LRU(i)|=16} CSize(i) \leq 16
\]

- Is not dependent on LRU replacement
  - Any replacement algorithm for top half of tags
  - Any stack algorithm for the remaining tags
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 Power4s
  - Est. ~16 GB/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

CMP Compression: Sensitivity to L2 Size

- CMP Compression: Sensitivity to Memory Latency

- CMP Compression: Pin Bandwidth Demand

- CMP Compression: Sensitivity to Pin Bandwidth
### Prefetching Properties (8p)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>L1 I Cache</th>
<th>L1 D Cache</th>
<th>L2 Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hit rate</td>
<td>Coverage</td>
<td>Accuracy</td>
</tr>
<tr>
<td>apache</td>
<td>4.9</td>
<td>54.0</td>
<td>64.2</td>
</tr>
<tr>
<td>oto</td>
<td>7.1</td>
<td>55.1</td>
<td>67.7</td>
</tr>
<tr>
<td>tftp</td>
<td>13.9</td>
<td>54.8</td>
<td>56.6</td>
</tr>
<tr>
<td>ph</td>
<td>1.8</td>
<td>49.6</td>
<td>42.7</td>
</tr>
<tr>
<td>art</td>
<td>0.55</td>
<td>41.4</td>
<td>50.3</td>
</tr>
<tr>
<td>apsl</td>
<td>0.04</td>
<td>15.7</td>
<td>34.7</td>
</tr>
<tr>
<td>fremd</td>
<td>0.06</td>
<td>7.5</td>
<td>14.4</td>
</tr>
<tr>
<td>mpif</td>
<td>0.06</td>
<td>15.5</td>
<td>26.6</td>
</tr>
</tbody>
</table>

### Analytical Model: IPC

\[
\text{IPC}(N) = \frac{N}{CPI_{\text{REF}} + dp + \text{MissPenalty}(\frac{1}{M}) \sqrt{\frac{N - \text{shr}_{\text{L1}}(N) + 1}{c(1 - k_p N)}}}
\]

\[
\text{MissLatency}_{\text{L1}} = \text{MemoryLatency + LinkLatency}
\]

\[
\text{LinkLatency} = \sum \frac{\text{IPC}(N) \cdot \text{Missrate(S}_{\text{L1}})}{2(1 - \text{IPC}(N) \cdot \text{Missrate(S}_{\text{L1}}))}
\]

### 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)

### Sensitivity to #Cores - OLTP

**Model - Sensitivity to Memory Latency**

- Compression’s impact similar on both extremes
- Compression can shift optimal configuration towards more cores (though not significantly)
Prefetching can degrade throughput for many systems.
Compression alleviates this performance degradation.
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

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.

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.