MORC

A MANYCORE ORIENTED COMPRESSED CACHE

TRI M. NGUYEN, DAVID WENTZLAFF
Architectures moving toward manycore

Tilera: 64-72 cores (2007)

Intel MIC: 288 threads (2015)

NVIDIA GPGPUs: 3072 threads (2015)

Increasing thread aggregation
- Cloud computing
- Massive warehouse scale center
Motivation: off-chip bandwidth scalability

Throughput = min(compute_avail, bandwidth_avail)

Throughput is already bandwidth-bound
- Assumption: 1000 threads, 1GB/s per thread
- Demand: 1000GB/s
- Supply: 102.4GB/s (four DDR4 channels)
- Oversubscribed ratio: ~10x

Bandwidth-wall will stall practical manycore scaling
- Economy of high pin-count packaging
- Pin size hard to be smaller even in high cost chips
- Frequency does not scale well
Compressing LLC as a solution

More on-chip cache correlates with higher performance

More *effective* cache through *compression* correlates with perf.
Compressing LLC as a solution

More on-chip cache correlates with higher performance

More effective cache through compression correlates with perf.

**MORC:**
- Manycore-oriented compressed cache
- Compresses the LLC (last level cache) to reduce off-chip misses

**Insight:**
- throughput over single-threaded
- expensive stream-based compression algorithms
Outline

◦ Stream compression is great!
  ◦ ...but is hard with set-based caches
  ◦ ...and is not for single-threaded performance
◦ Stream compression with log-based caches
◦ Architecture of log-based compressed cache
◦ Results
  ◦ Performance
  ◦ Energy
What is stream-based compression?

Common software data compression algorithms
- LZ77, gzip, LZMA

Sequentially compresses cache lines as a single stream
- Compress using pointers to copy repeated string (data)
What is stream-based compression?

Common software data compression algorithms

- LZ77, gzip, LZMA

Sequentially compresses cache lines as a single stream

- Compress using pointers to copy repeated string (data)
What is stream-based compression?

Common software data compression algorithms
- LZ77, gzip, LZMA

Sequentially compresses cache lines as a single stream
- Compress using pointers to copy repeated string (data)
What is stream-based compression?

Common software data compression algorithms

- LZ77, gzip, LZMA

Sequentially compresses cache lines as a single stream

- Compress using **pointers** to copy repeated string (data)
Stream compression example
Stream compression example

Cache line A
0x1212
0xabcd

Cache line B
0xabcd
0x1212

Cache line A
0x12+
0xabcd
Stream compression example

![Diagram showing cache lines and memory blocks]

- **Cache line A**: 0x1212, 0xabcd
- **Cache line B**: 0xabcd, 0x1212

- **Cache line A**: 0x12+, p
- **Cache line B**: p
Stream vs block-based compression

Block-based compression

<table>
<thead>
<tr>
<th>Cache line A</th>
<th>0x1212</th>
<th>0xabcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache line B</td>
<td>0xabcd</td>
<td>0x1212</td>
</tr>
</tbody>
</table>

Stream-based compression

<table>
<thead>
<tr>
<th>Cache line A</th>
<th>0x1212</th>
<th>0xabcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache line B</td>
<td>0xabcd</td>
<td>0x1212</td>
</tr>
</tbody>
</table>

Stream-based compression achieves much higher compression
Stream vs block-based compression

Many prior-work uses block-based compression

Two reasons: single-threaded performance & implement-ability
First reason: Well-matched for throughput

Decompression is inherently expensive
First reason: Well-matched for throughput

Decompression is inherently expensive
First reason:
Well-matched for throughput

Decompression is inherently expensive
First reason: Well-matched for throughput

Decompression is inherently expensive

Insight:
- Memory accesses are expensive!
  - High latency
  - High energy consumption
Second reason: Hard to implement with set-based caches
Second reason: Hard to implement with set-based caches

Implementation: compress each cache set as a compressed stream
Second reason: Hard to implement with set-based caches

Implementation: compress each cache set as a compressed stream

Cache sets are unsuited for stream-based compression
- Evictions and write-backs corrupt the compression stream
Introducing log-based caches

Log-based caches organize cache lines by temporal fill order
Fill data-path architecture

- Lines stream to one active log sequentially
- Record address_1 to log_3 in a table
Fill data-path architecture

- Lines stream to one active log sequentially
- Record `address_2` to `log_3` in a table
Fill data-path architecture

Log-flush happens when not enough space
- Not in critical-path
- Only writes back dirty cache lines
Fill data-path architecture

- Lines stream to one active log sequentially
- Record address_3 to log_4 in a table
Request data-path

LMT: Line-Map Table (redirection table)
- Indexed by addresses
- Points to logs
Request data-path

LMT: Line-Map Table (redirection table)
- Indexed by addresses
- Points to logs

1. Stream compressor
2. LMT
3. Eviction policy (flush)
Content-aware compression with logs

Multiple active logs enable content aware compression
- Dynamically chooses the best stream based on similarity
- Better than strict sequential compression
Prior work in LLC compression

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Internal fragmentation</th>
<th>External fragmentation</th>
<th>Tags overhead</th>
<th>Requiring software</th>
<th>Set-based</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive[1]</td>
<td>Yes</td>
<td>Yes</td>
<td>Medium</td>
<td>No</td>
<td>Yes</td>
<td>Block</td>
</tr>
<tr>
<td>Decoupled[2]</td>
<td>Yes</td>
<td>No</td>
<td>Low</td>
<td>No</td>
<td>Yes</td>
<td>Block</td>
</tr>
<tr>
<td>SC2[3]</td>
<td>Yes</td>
<td>Yes</td>
<td>High</td>
<td>Yes</td>
<td>Yes</td>
<td>Centralized</td>
</tr>
<tr>
<td>MORC</td>
<td>Very little</td>
<td>No</td>
<td>Low</td>
<td>No</td>
<td>Log-based</td>
<td>Stream</td>
</tr>
</tbody>
</table>

Internal-fragmentation in compression blocks
- Decreases absolute compression ratio as much as 12.5%

External fragmentation
- Increase LLC energy by as much as 200% (studied in [2])

Simulation methodology

Simulator: PriME[1]
  ◦ Execution driven, x86 inorder

SPEC2006 benchmarks

Future manycore system
  ◦ 1024 cores in a single chip
  ◦ 128MB LLC (128KB per core)
  ◦ 100GB/s off-chip bandwidth (100MB/s per core)

Compression results
Compression results

Max average comp. ratio: 6x
Arithmetic mean: 3x
Throughput improvements

Max average comp. ratio: 6x
Arithmetic mean: 3x
Throughput improvements

Max average comp. ratio: 6x
Arithmetic mean: 3x
Throughput improvements: 40%
Best prior work: 20%
Throughput improvements

Max average comp. ratio: 6x
Arithmetic mean: 3x

Throughput improvements: 40%
Best prior work: 20%
Throughput improvements

Max average comp. ratio: 6x
Arithmetic mean: 3x
Throughput improvements: 40%
Best prior work: 20%
Improvements depends on working set sizes
Energy

Two questions:
- DRAM access energy savings 😊
- Compression/decompression energy concern 😞
Energy

Expensive DRAM accesses
Negligible compression energy
Small decompression energy

Memory subsystem energy normalized to uncompressed baseline
Energy

Expensive DRAM accesses
Negligible compression energy
Small decompression energy

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>64b comparison (65nm)</td>
<td>2pJ</td>
<td>1x</td>
</tr>
<tr>
<td>64b access 128KB SRAM (32nm)</td>
<td>4pJ</td>
<td>2x</td>
</tr>
<tr>
<td>64b floating point op (45nm)</td>
<td>45pJ</td>
<td>22.5x</td>
</tr>
<tr>
<td>64b transfer across 15mm on-chip</td>
<td>375pJ</td>
<td>185x</td>
</tr>
<tr>
<td>64b transfer across main-board</td>
<td>2.5nJ</td>
<td>1250x</td>
</tr>
<tr>
<td>64b access to DDR3</td>
<td>9.35nJ</td>
<td>4675x</td>
</tr>
</tbody>
</table>

Memory subsystem energy normalized to uncompressed baseline
Summary

Stream compression is much better versus block-based
  ◦ ...but is hard with set-based caches
  ◦ ...and is not right approach for single-threaded performance

Log-based caches efficiently support stream-based compression
  ◦ Sequential cache line placements

Architecture
  ◦ Stream compressor, LMT, eviction policy

Results
  ◦ 50% better compression, 100% better throughput improvements
  ◦ Better energy efficiency