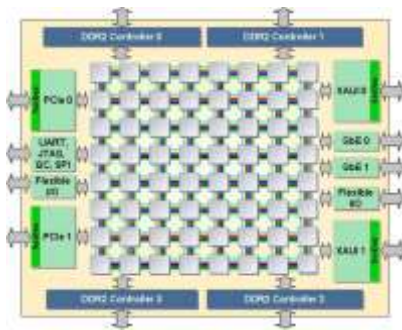


MORC

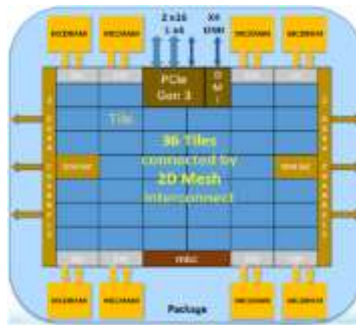
A MANYCORE ORIENTED COMPRESSED CACHE

TRI M. NGUYEN, DAVID WENTZLAFF

Architectures moving toward manycore



Tiler: 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

$$\text{Throughput} = \min(\text{compute_avail}, \text{bandwidth_avail})$$

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**

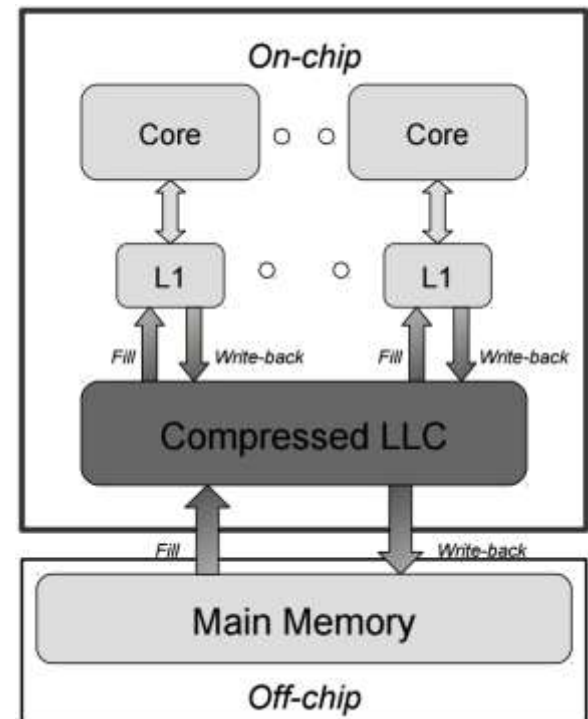
Bandwidth-wall will stall practical manycore scaling

- 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

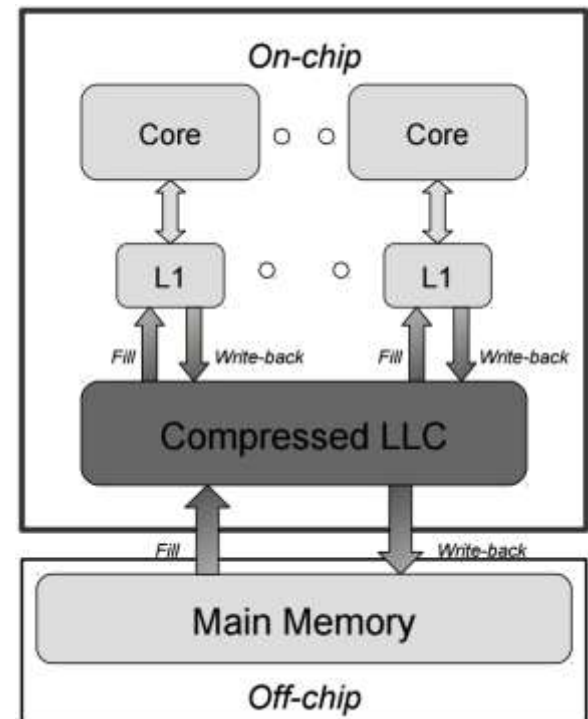
More effective cache through compression correlates with perf.

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)

To compress



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)

To compress



Compress A



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)

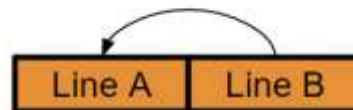
To compress



Compress A



Compress B



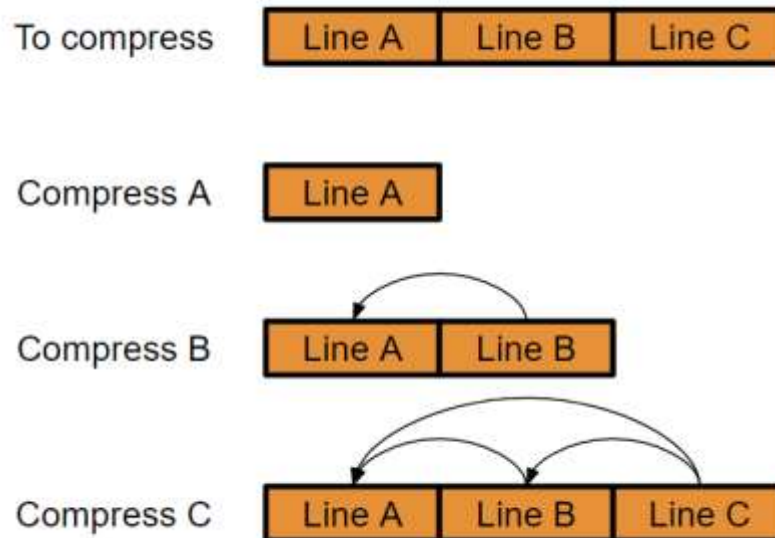
What is stream-based compression?

Common software data compression algorithms

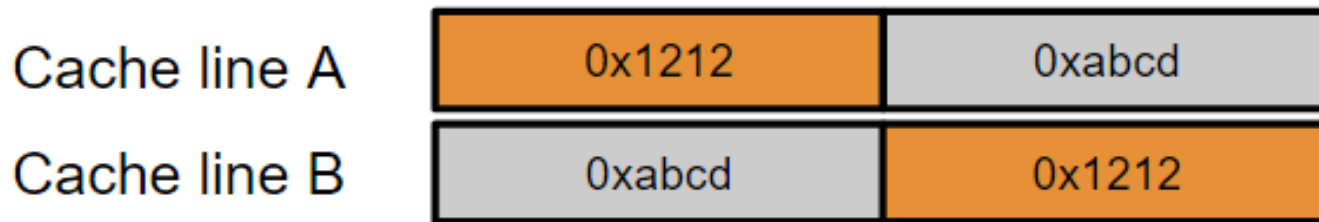
- LZ77, gzip, LZMA

Sequentially compresses cache lines as a single stream

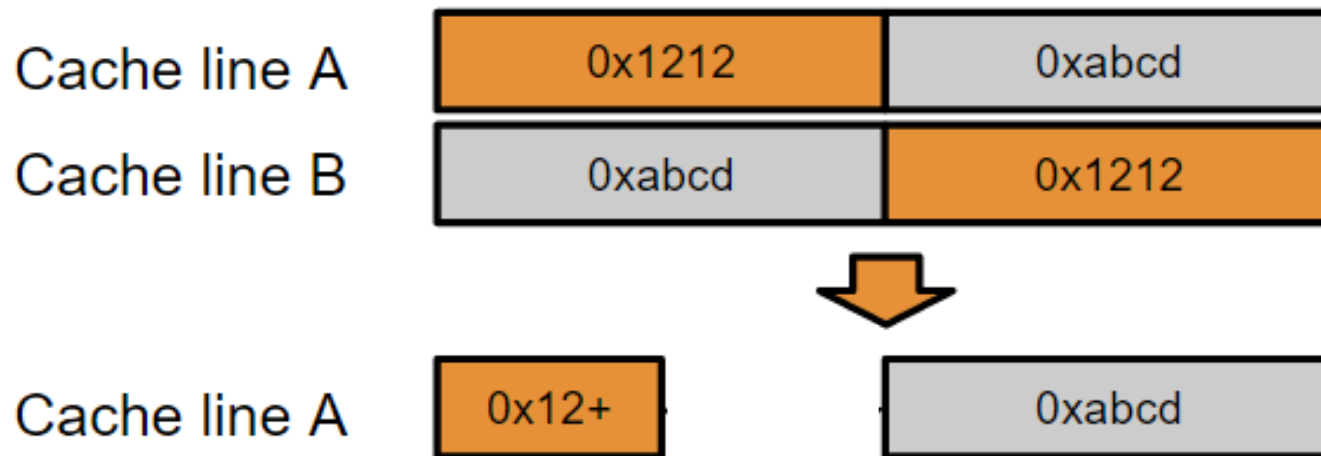
- Compress using pointers to copy repeated string (data)



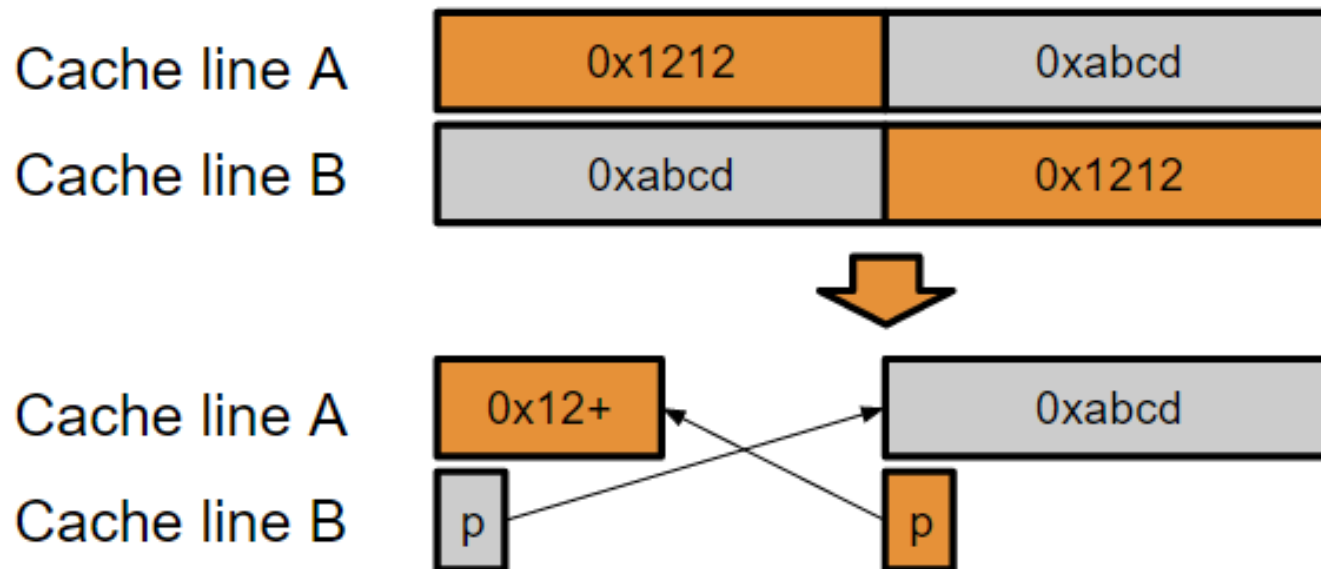
Stream compression example



Stream compression example

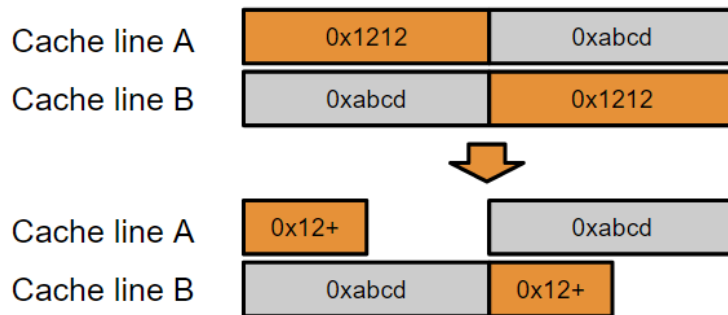


Stream compression example

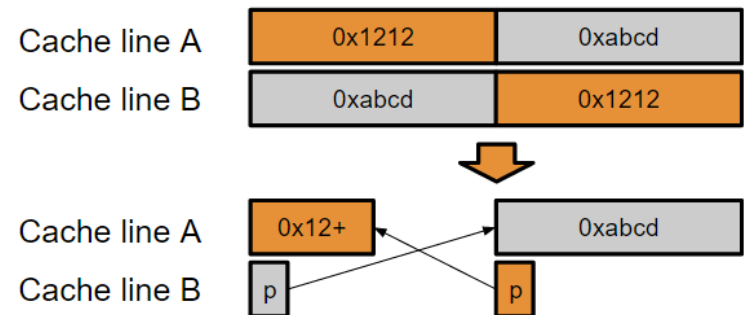


Stream vs block-based compression

Block-based compression



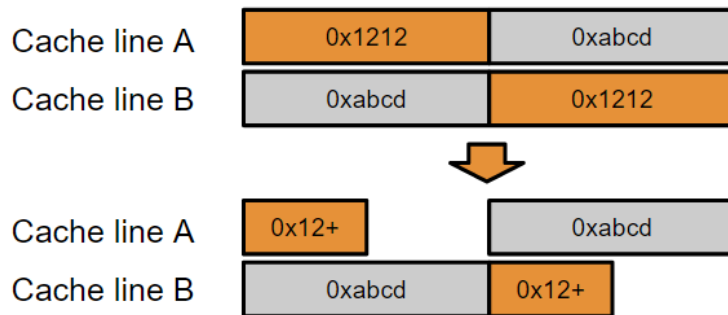
Stream-based compression



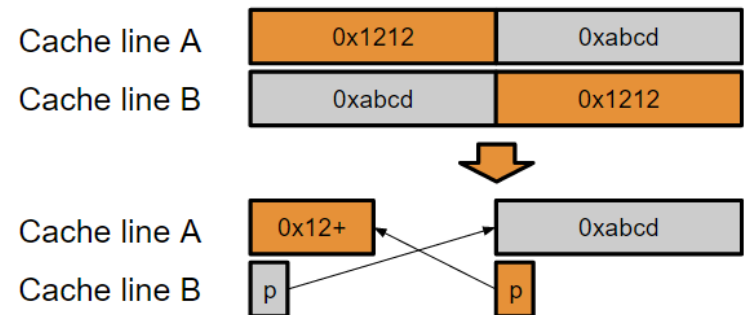
Stream-based compression achieves much higher compression

Stream vs block-based compression

Block-based compression



Stream-based compression



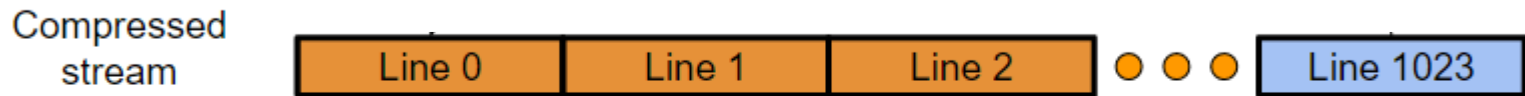
Stream-based compression achieves much higher compression

Many prior-work uses block-based compression

Two reasons: single-threaded performance & implement-ability

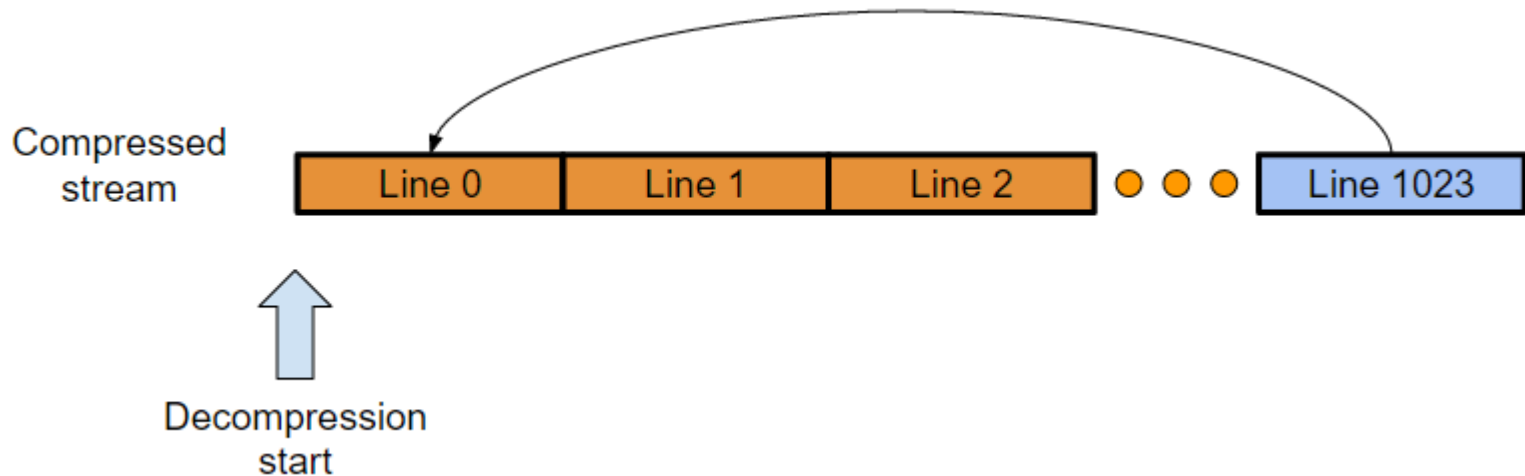
First reason: Well-matched for throughput

Decompression is inherently expensive



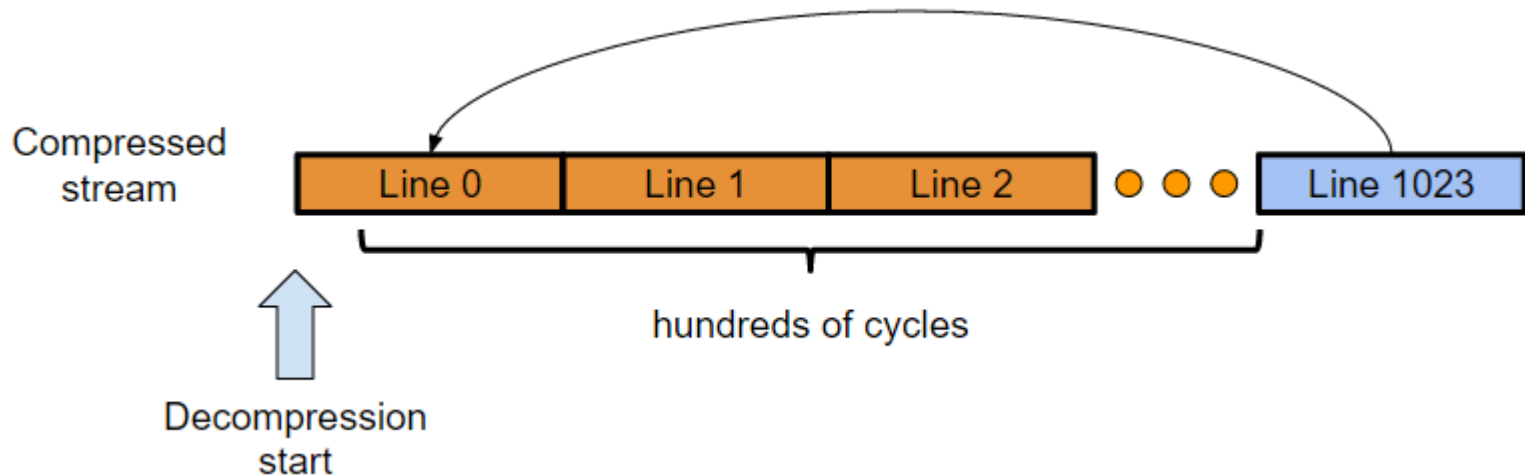
First reason: Well-matched for throughput

Decompression is inherently expensive



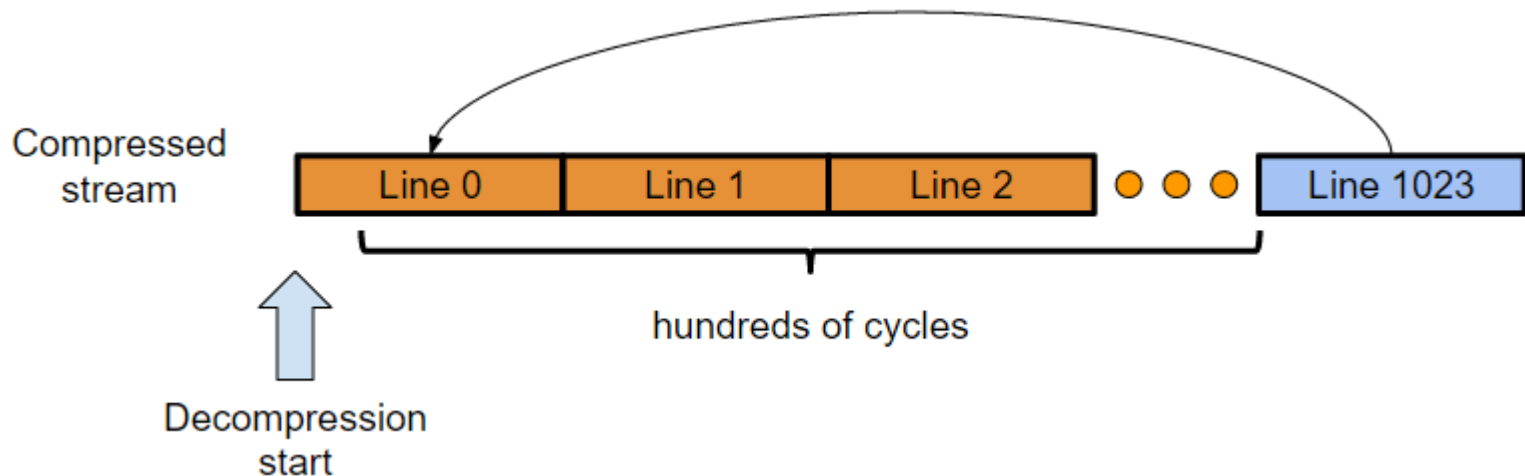
First reason: Well-matched for throughput

Decompression is inherently expensive



First reason: Well-matched for throughput

Decompression is inherently expensive



Insight:

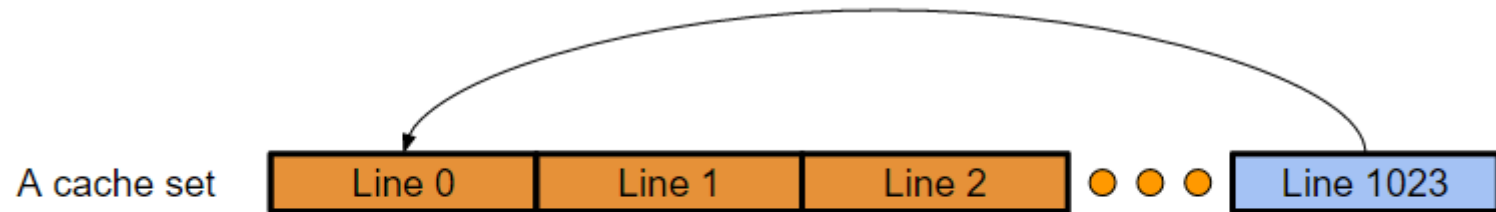
- High latency
- High energy consumption

Memory accesses are expensive!

Second reason: Hard to implement with set-based caches

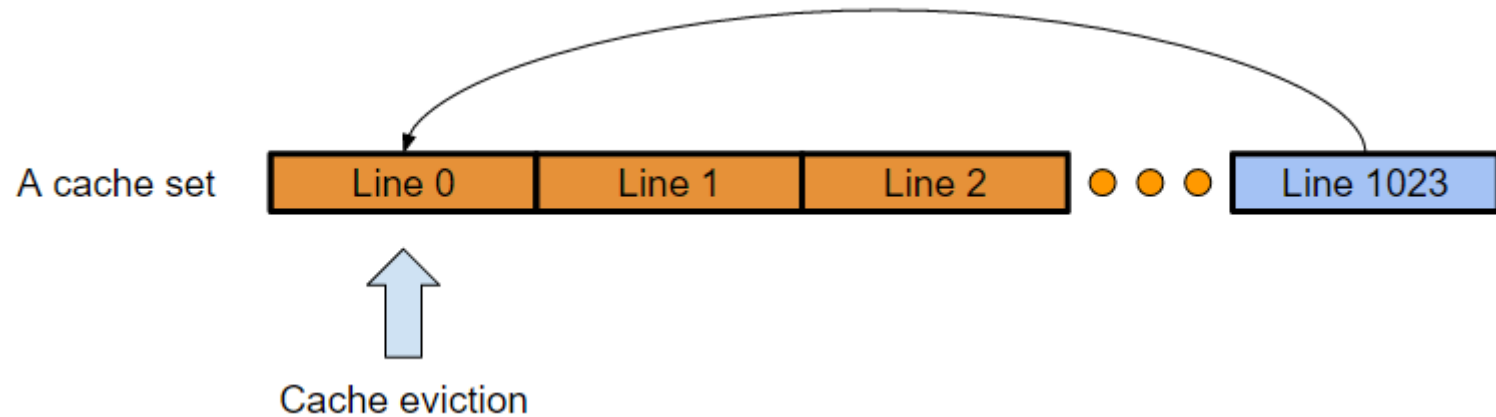
Second reason: Hard to implement with set-based caches

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

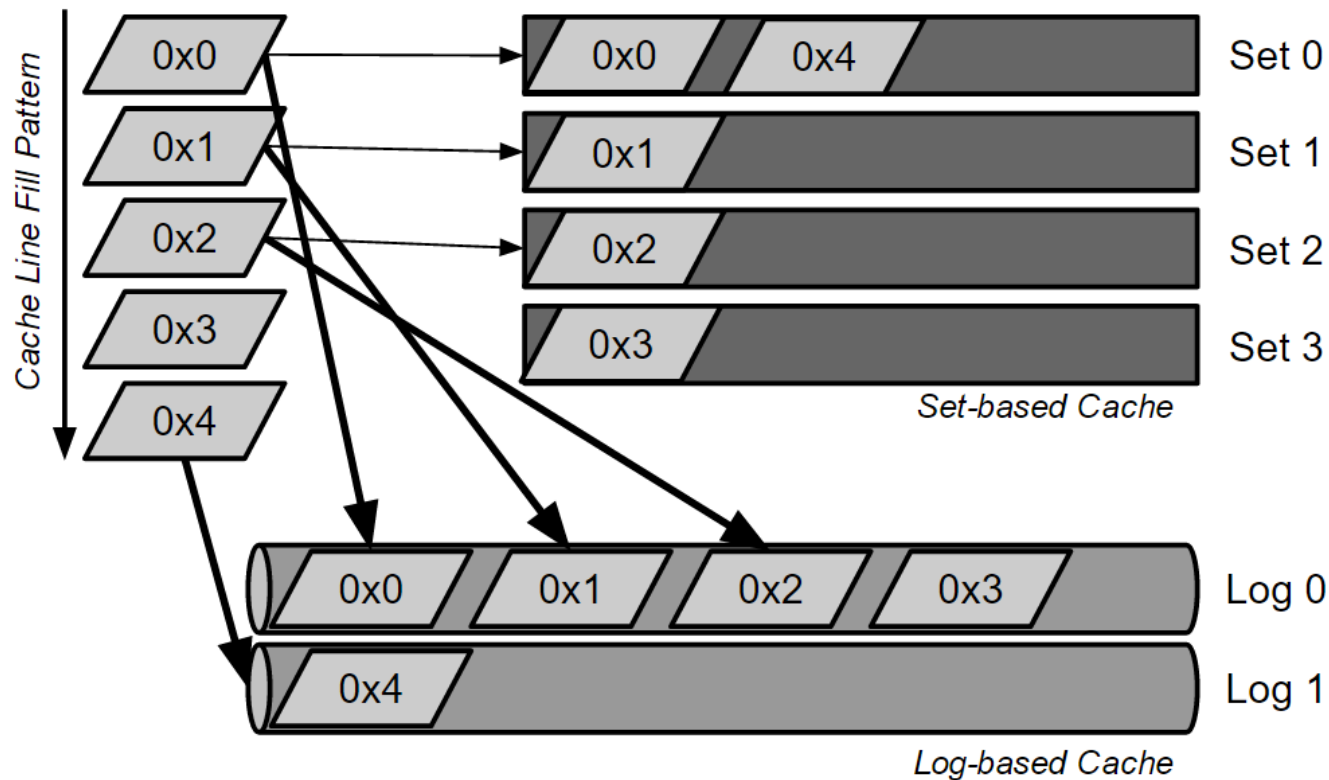


Cache sets are unsuited for stream-based compression

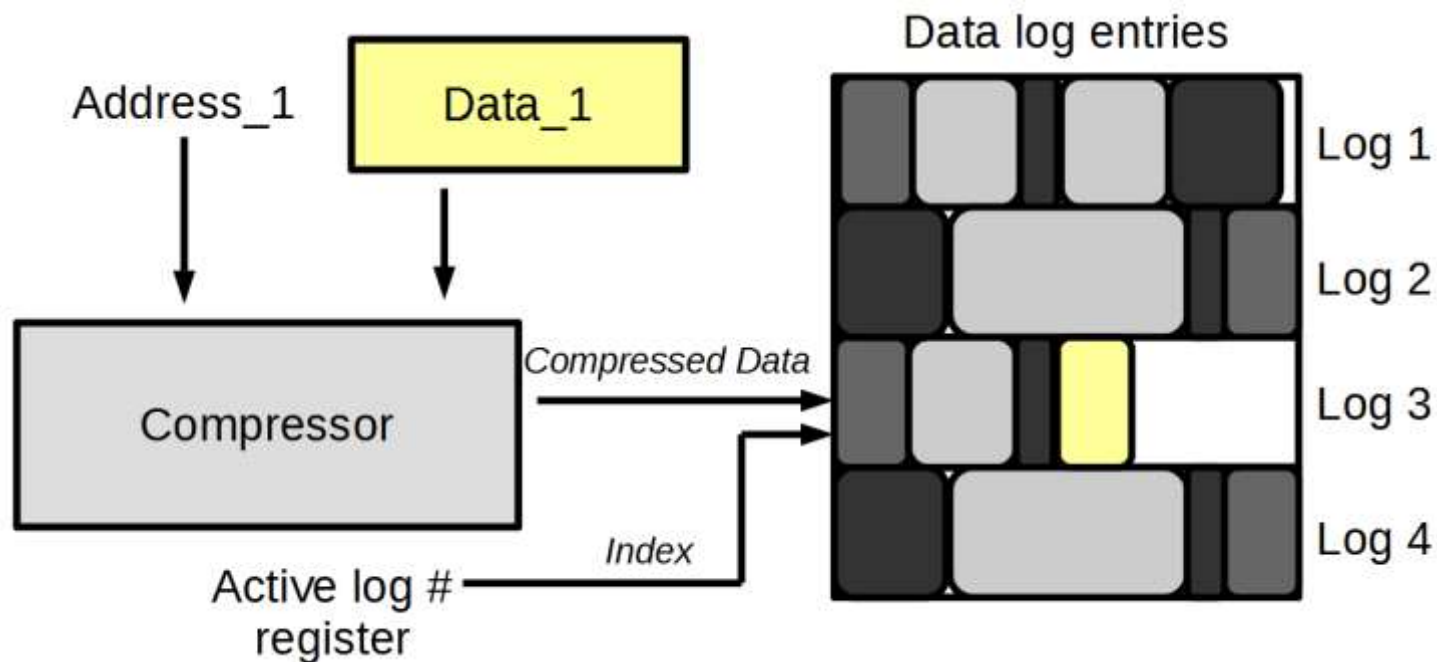
- Evictions and write-backs **corrupt** the compression stream

Introducing log-based caches

Log-based caches organize cache lines by temporal fill order

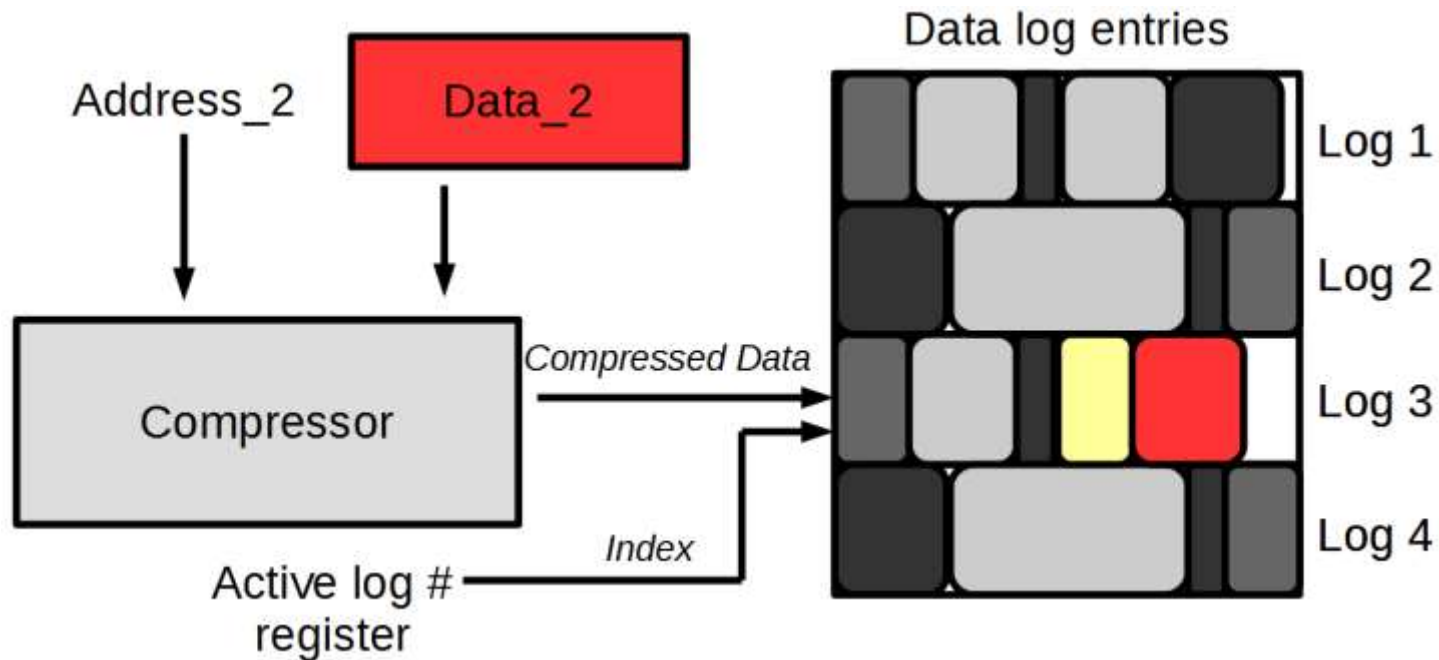


Fill data-path architecture



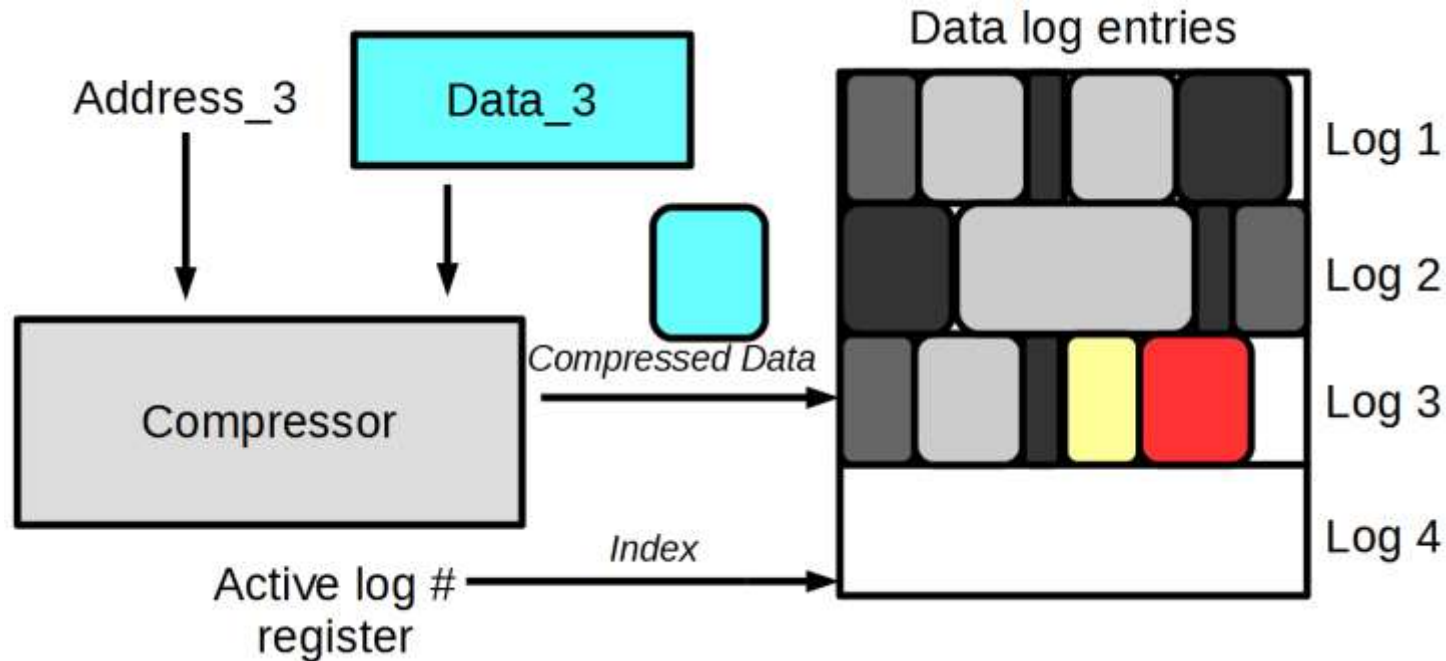
- Lines stream to one active log sequentially
- Record *address_1* to *log_3* in a table

Fill data-path architecture



- Lines stream to one active log sequentially
- Record *address_2* to *log_3* in a table

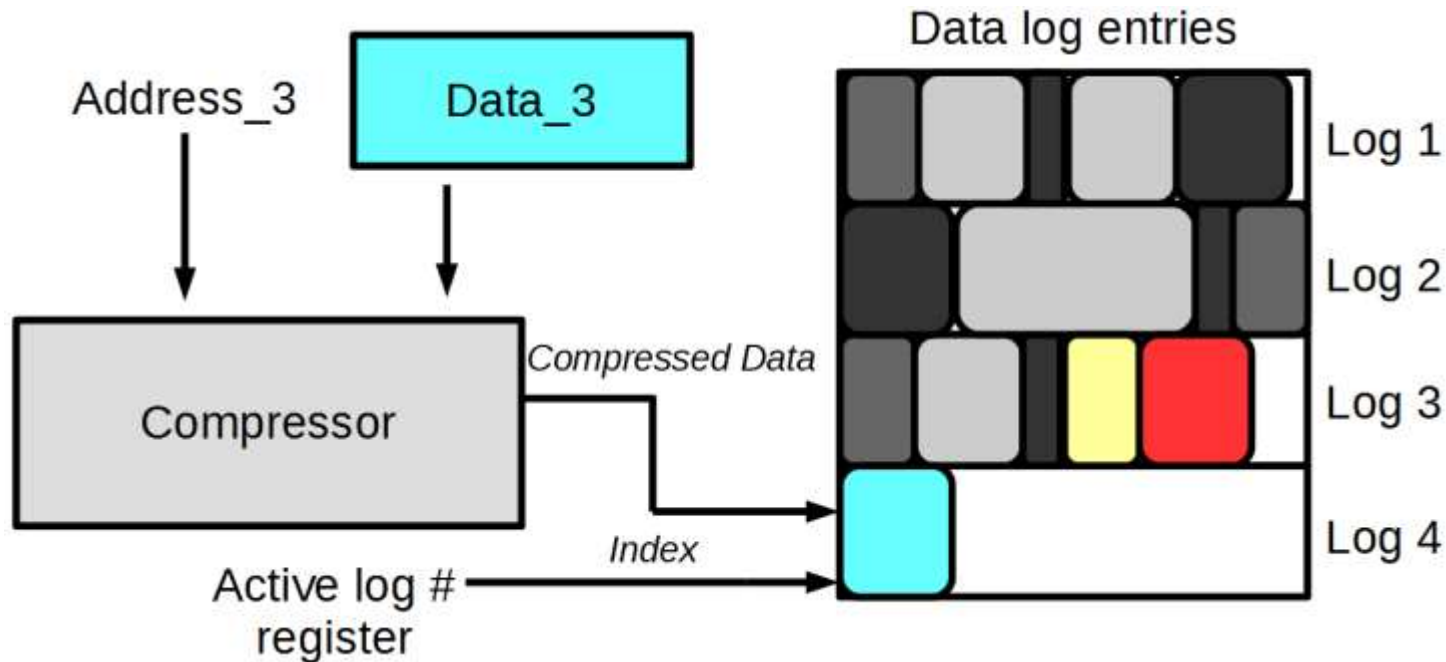
Fill data-path architecture



Log-flush happens when not enough space

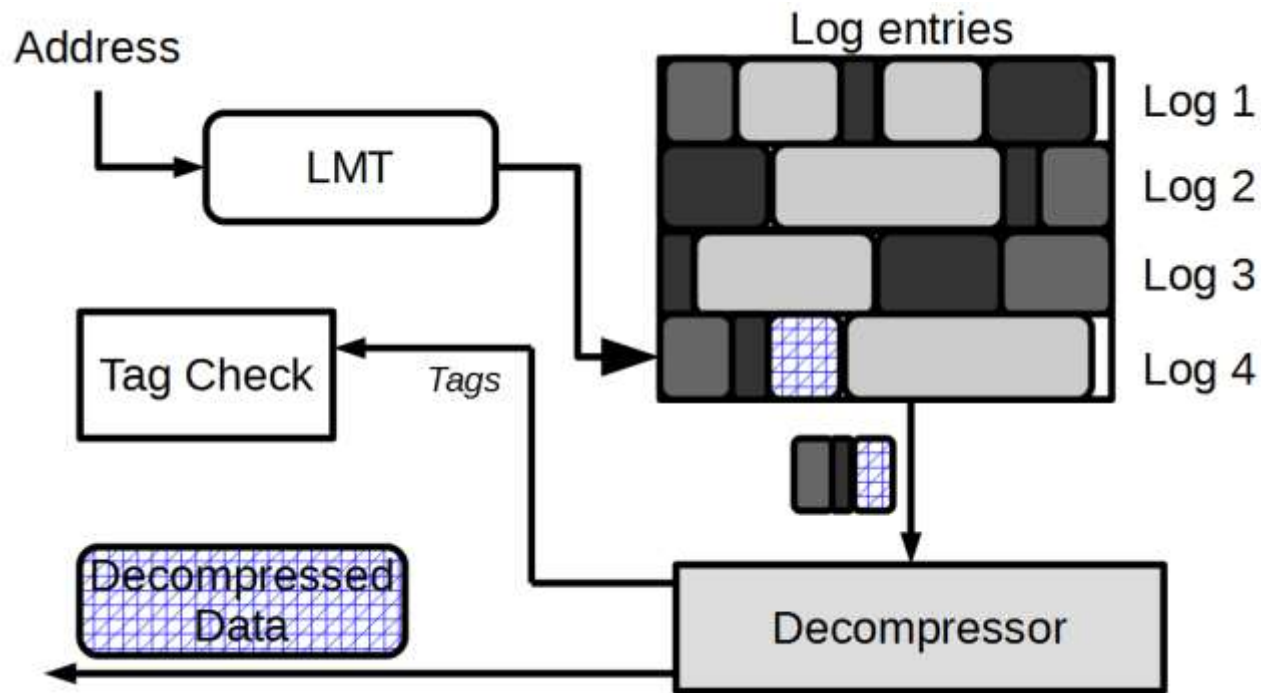
- Not in critical-path
- Only writes back dirty cache lines

Fill data-path architecture



- Lines stream to one active log sequentially
- Record *address_3* to **log_4** in a table

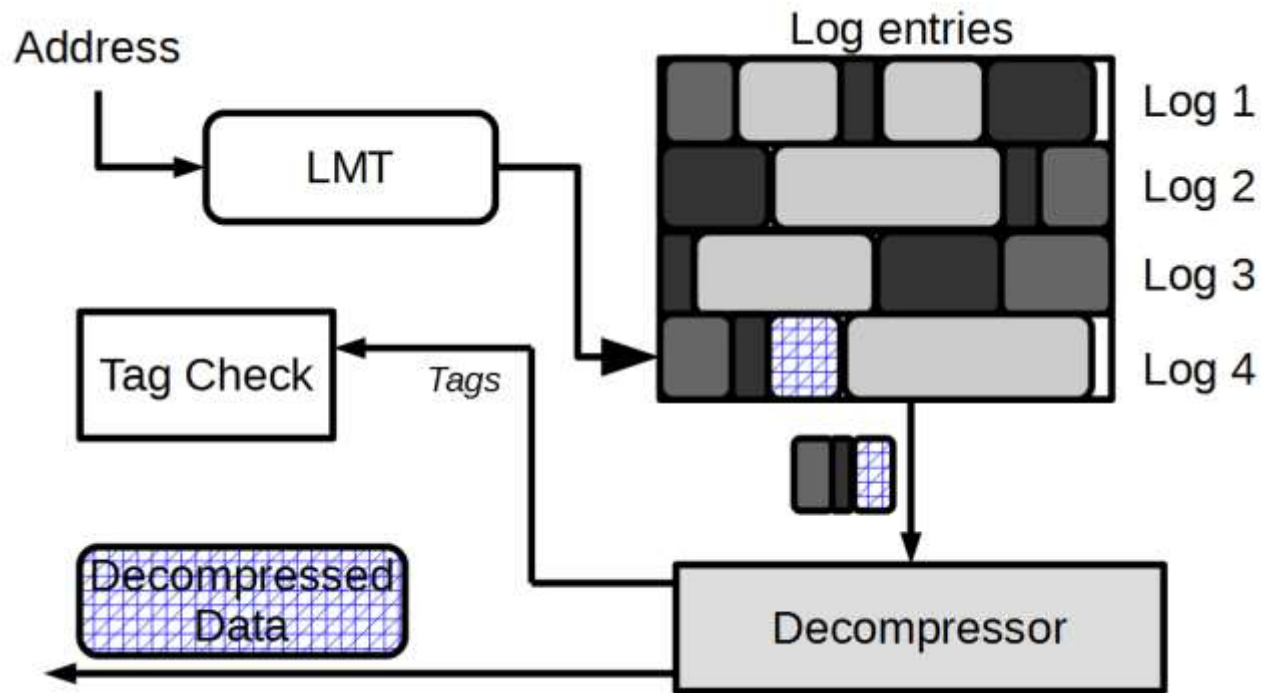
Request data-path



LMT: Line-Map Table (redirection table)

- Indexed by addresses
- Points to logs

Request data-path



LMT: Line-Map Table (redirection table)

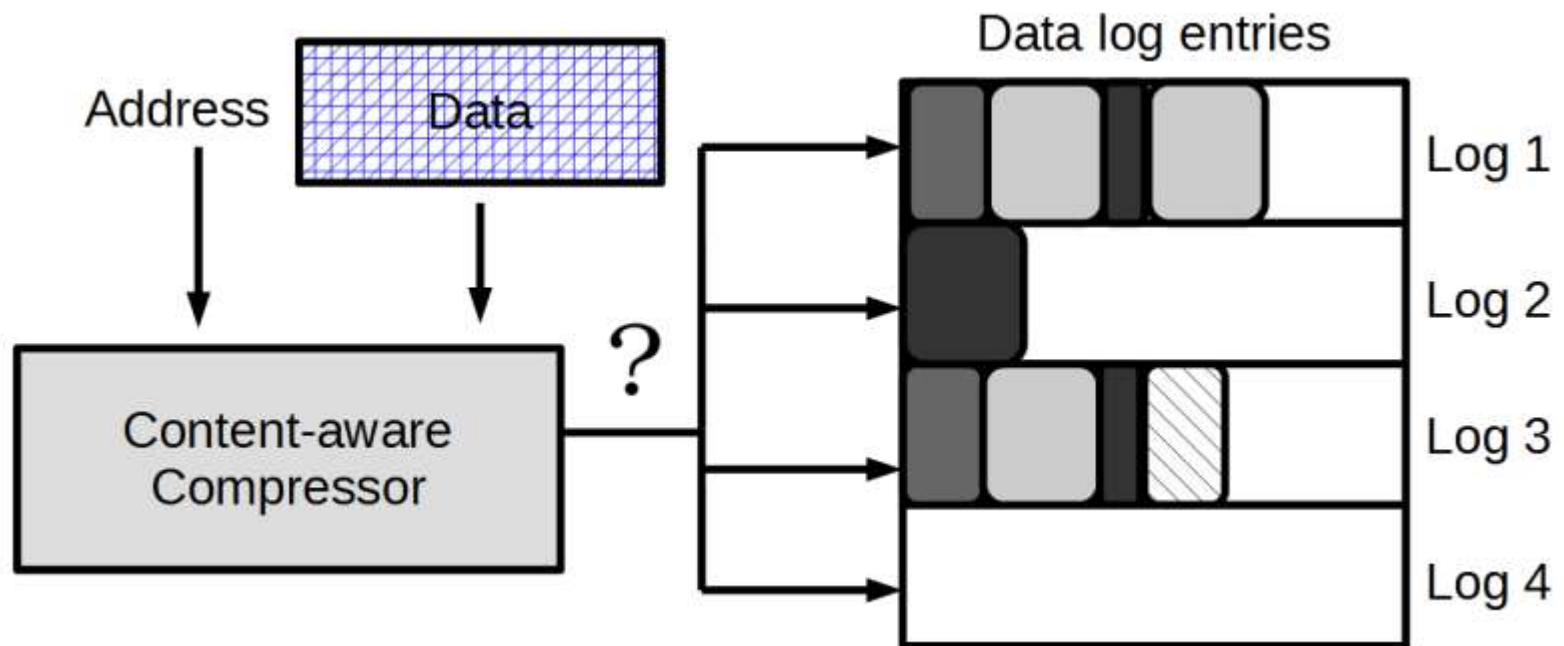
- Indexed by addresses
- Points to logs

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

Scheme	Internal fragmentation	External fragmentation	Tags overhead	Requiring software	Set-based	Algorithm
Adaptive[1]	Yes	Yes	Medium	No	Yes	Block
Decoupled[2]	Yes	No	Low	No	Yes	Block
SC2[3]	Yes	Yes	High	Yes	Yes	Centralized
MORC	Very little	No	Low	No	Log-based	Stream

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])

[1] Alameldeen et al, "Adaptive cache compression for high-performance processors," ISCA'04

[2] Sardashti et al, "Decoupled compressed cache: exploiting spatial locality for energy-optimized compressed caching," MICRO'13

[3] Arelakis et al, "SC2: A statistical compression cache scheme," ISCA'14

Simulation methodology

Simulator: PriME[1]

- Execution driven, x86 inorder

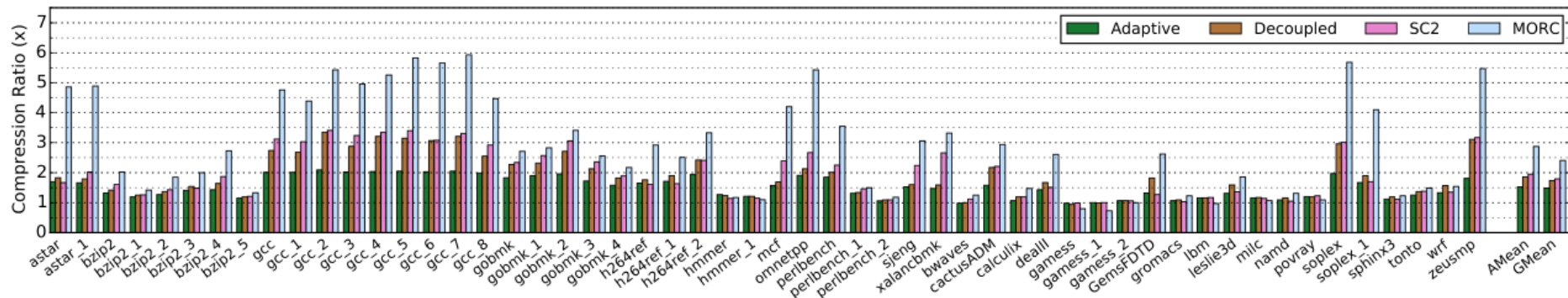
SPEC2006 benchmarks

Future manycore system

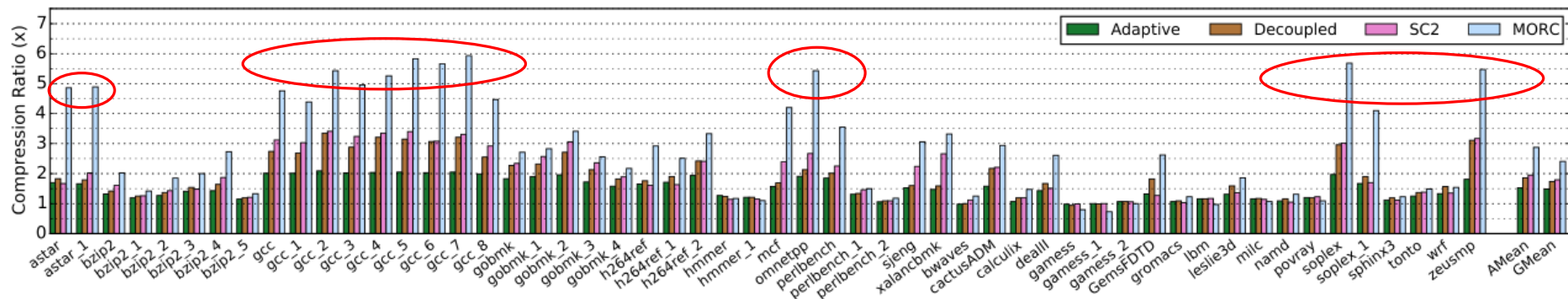
- 1024 cores in a single chip
- 128MB LLC (128KB per core)
- 100GB/s off-chip bandwidth (100MB/s per core)

[1] Y. Fu et al, "PriME: A parallel and distributed simulator for thousand-core chips," ISPASS 2014

Compression results

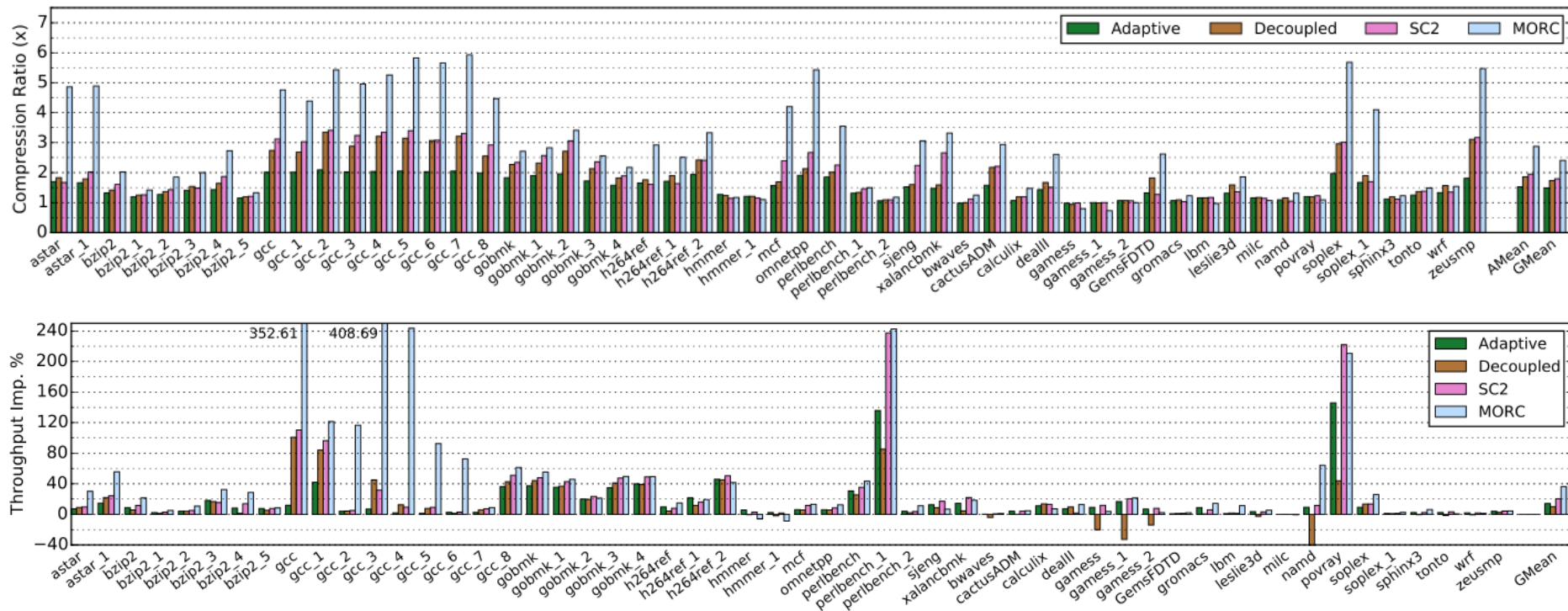


Compression results



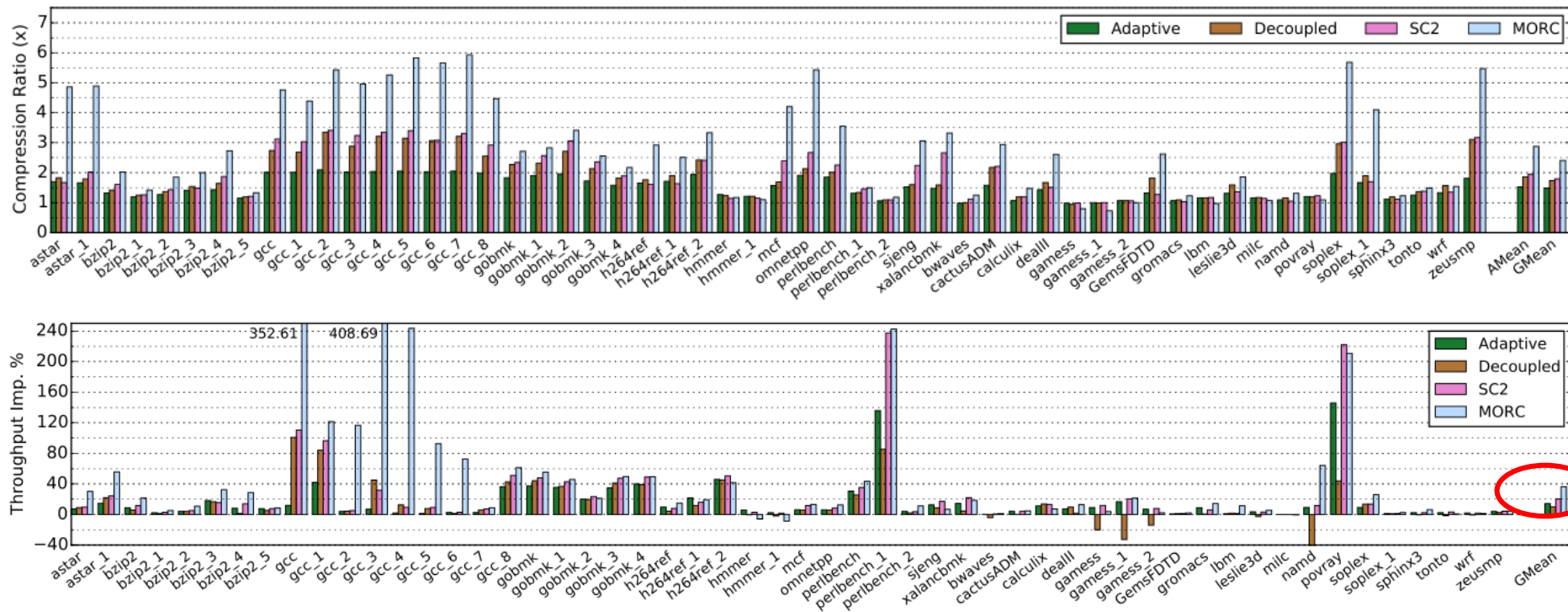
Max average comp. ratio: 6x
Arithmetic mean: 3x

Throughput improvements



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

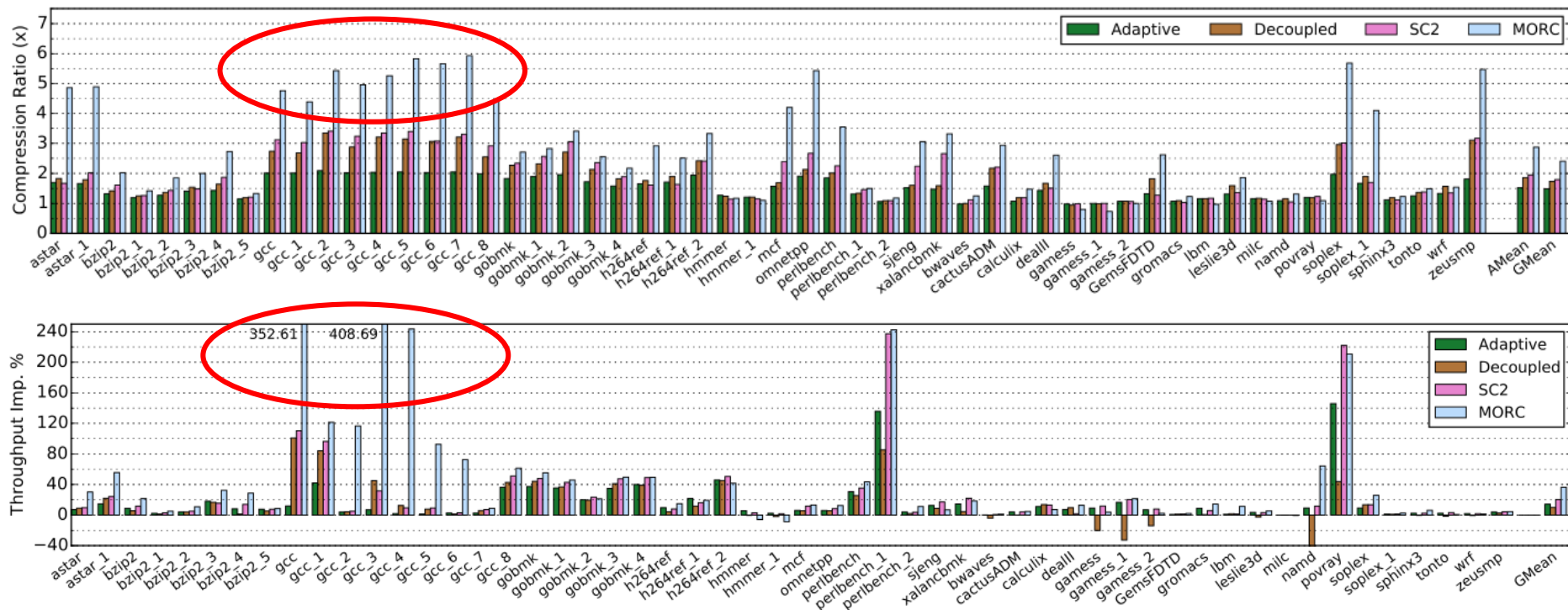
Throughput improvements



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

Throughput improvements: 40%
Best prior work: 20%

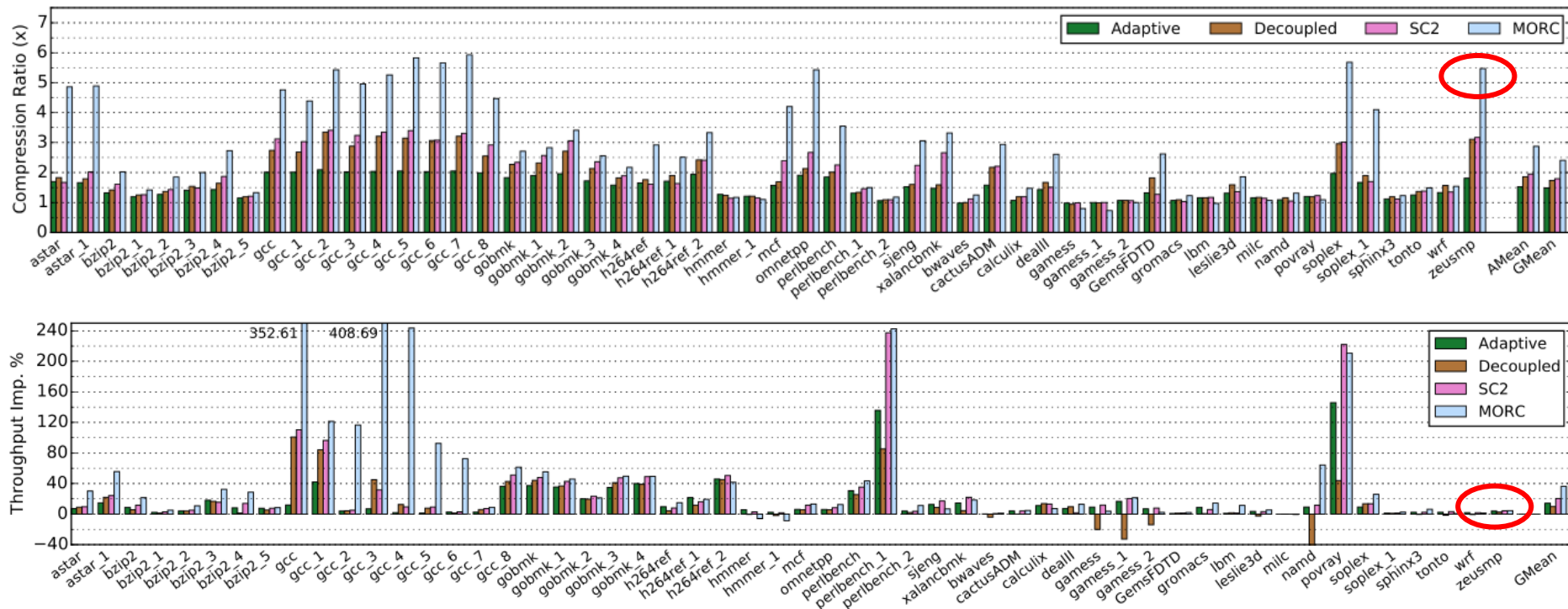
Throughput improvements



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:

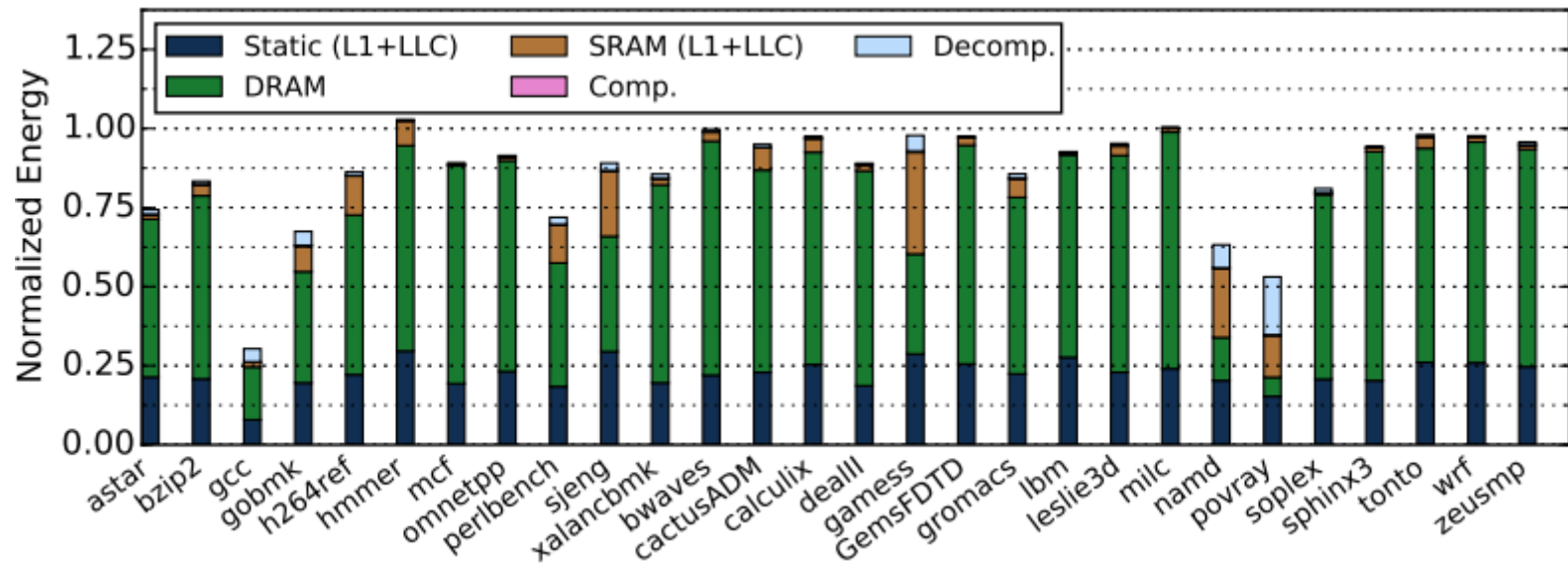
- DRAM access energy savings 
- Compression/decompression energy concern 

Energy

Expensive DRAM accesses

Negligible compression energy

Small decompression energy



Memory subsystem energy normalized to uncompressed baseline

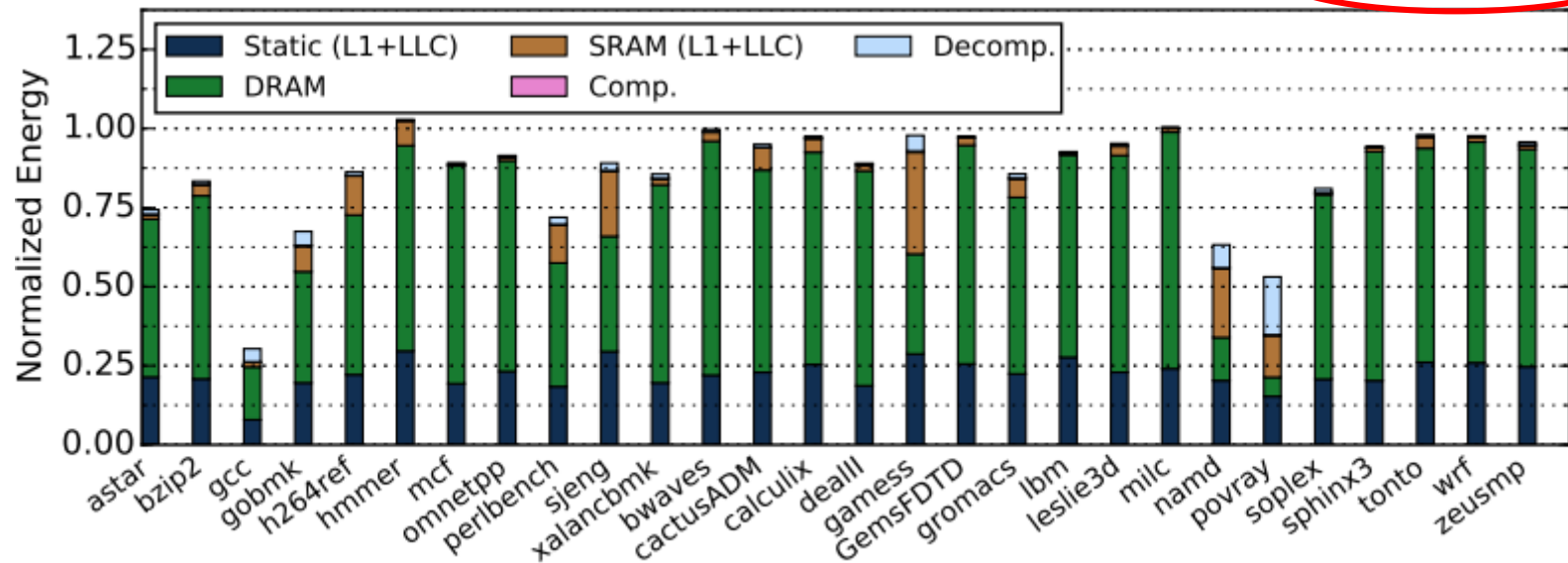
Energy

Expensive DRAM accesses

Negligible compression energy

Small decompression energy

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



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