Practical Data Compression for On-Chip Caches

November 26, 2019

Problem: CPUs are cheap and fast.

Problem: CPUs are cheap and fast.
Memories are cheap, fast, capacious

Problem: CPUs are cheap and fast.
Memories are cheap, fast, capacious (choose two out of three).

Problem: CPUs are cheap and fast.
Memories are cheap, fast, capacious (choose two out of three).
Fast CPU with slow memory doesn't make sense.

Idea: Put fast memory between CPU and main memory.

Idea: Put fast memory between CPU and main memory.
Fast – so small.

Idea: Put fast memory between CPU and main memory.
Fast – so small.

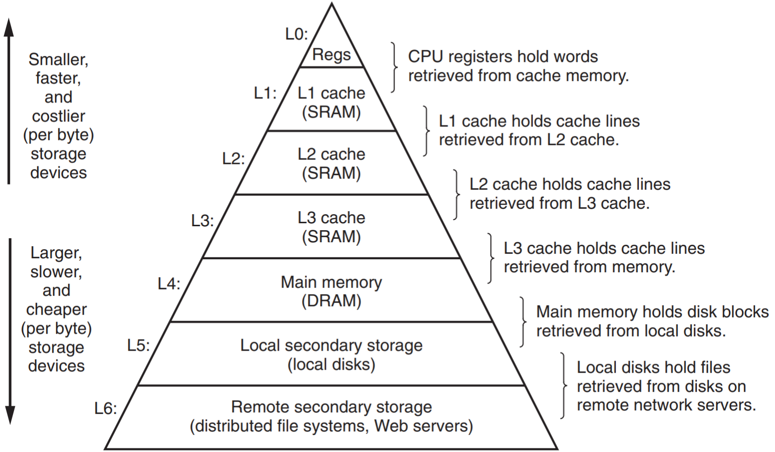
CPU looks for data in the cache;

Idea: Put fast memory between CPU and main memory.
Fast – so small.

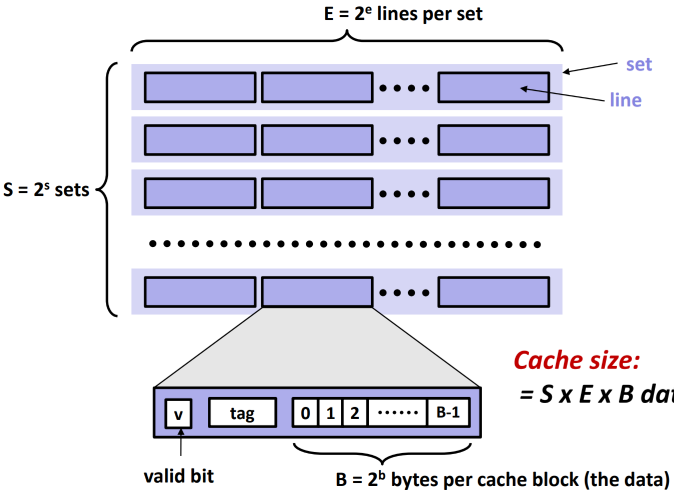
CPU looks for data in the cache; if it finds it, it gets it from there and continues to work;

Idea: Put fast memory between CPU and main memory.
Fast – so small.

CPU looks for data in the cache; if it finds it, it gets it from there and continues to work; if it doesn't find it, it looks for it in main memory.

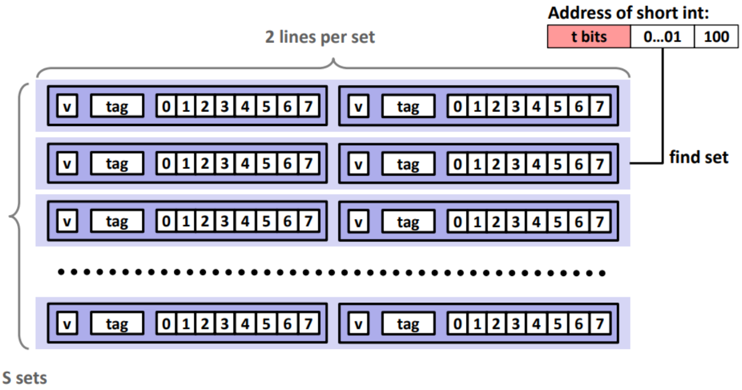

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from L2 cache.
L2 cache holds cache lines retrieved from L3 cache.
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote network servers.

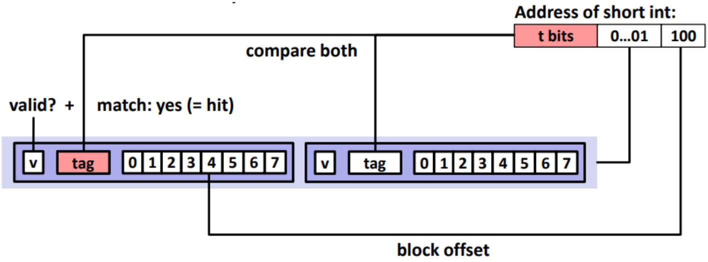
memory	latency	size
L1	4 cycles	32KiB
L2	10 cycles	256KiB
L3	40-75 cycles	8MiB
DRAM	600 cycles	8GiB
HDD	∞	∞

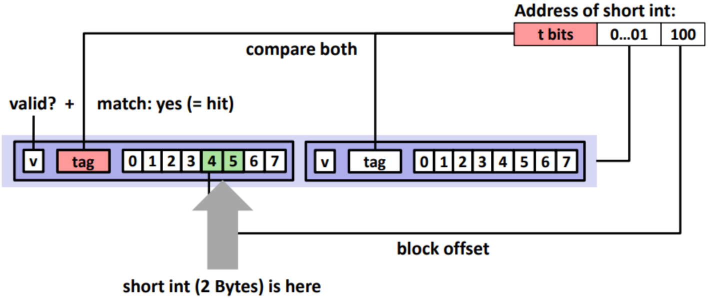


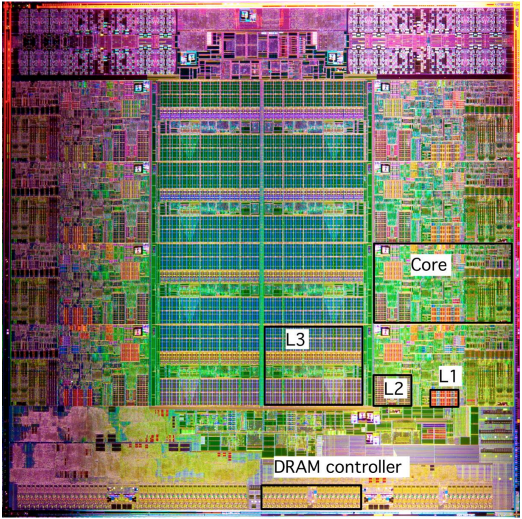
Cache size:

= S x E x B data bytes


Address of short int:
t bits
0…01
100
2 lines per set
v
tag
0 1 2 3 4 5 6 7
find set
S sets


Address of short int:
t bits
0...01
100
compare both
valid? +
match: yes (= hit)
v
tag
0 1 2 3 4 5 6 7
v
tag
0 1 2 3 4 5 6 7
block offset


Address of short int:
t bits
0...01
100
compare both
valid? +
match: yes (= hit)
v
tag
0 1 2 3 4 5 6 7
v
tag
0 1 2 3 4 5 6 7
block offset
short int (2 Bytes) is here


Core
L3
L2
L1
DRAM controller

What can we do?

Compression!

Unfortunately, directly applying well-known compression algorithms (usually implemented in software) leads to high hardware complexity and unacceptable decompression/compression latencies, which in turn can negatively affect performance.

Compression:

Compression: takes place in background upon a cache fill.

Compression: takes place in background upon a cache fill.

Decompression:

Compression: takes place in background upon a cache fill.

Decompression: is on the critical path of a cache hit

Compression: takes place in background upon a cache fill.

Decompression: is on the critical path of a cache hit; we can only consider compression of the L2 caches.

Compression must be fast, simple, effective

Compression must be fast, simple, effective, the challenge is to find the right balance.

Base+Delta Encoding (B+Δ)

Base+Delta Encoding (B+Δ)
Base-Delta-Immediate (BΔI)

Observation:

Observation: for many cache lines, the data values stored within the line have a low dynamic range: i.e., the relative difference between values is small.

Observation: for many cache lines, the data values stored within the line have a low dynamic range: i.e., the relative difference between values is small. In such cases, the cache line can be represented in a compact form using a common base value plus an array of relative differences.

- **Zeros:** Zero is by far the most frequently seen value in application data. For example, zero is most commonly used to initialize data, to represent NULL pointers or false boolean values.

- **Zeros:** Zero is by far the most frequently seen value in application data. For example, zero is most commonly used to initialize data, to represent NULL pointers or false boolean values.
- **Repeated Values:** A large contiguous region of memory may contain a single value repeated multiple times. This pattern is widely present in applications that use a common initial value for a large array, or in multimedia applications where a large number of adjacent pixels have the same color.

- **Zeros:** Zero is by far the most frequently seen value in application data. For example, zero is most commonly used to initialize data, to represent NULL pointers or false boolean values.
- **Repeated Values:** A large contiguous region of memory may contain a single value repeated multiple times. This pattern is widely present in applications that use a common initial value for a large array, or in multimedia applications where a large number of adjacent pixels have the same color.
- **Narrow Values:** A narrow value is a small value stored using a large data type: e.g., a one-byte value stored as a four-byte integer.

Benchmarks:

- libquantum – Physics: Quantum Computing
- lbm – Fluid Dynamics
- mcf – Combinatorial Optimization
- sjeng – Artificial Intelligence: chess
- omnetpp – Discrete Event Simulation
- sphinx3 – Speech recognition
- xalancbmk – XML Processing
- bzip2 – Compression
- leslie3d – Fluid Dynamics
- apache – Web server
- gromacs – Biochemistry/Molecular Dynamics

Benchmarks:

- astar – Path-finding Algorithms
- gobmk – Artificial Intelligence: go
- soplex – Linear Programming, Optimization
- gcc - C Compiler
- hmmer – Search Gene Sequence
- wrf – Weather Prediction
- h264ref – Video Compression
- zeusmp – Physics / CFD
- cacutsADM – Physics / General Relativity
- GemsFDTD – Computational Electromagnetics

	Characteristics			Compressible data patterns			
	Decomp. Lat.	Complex.	C. Ratio	Zeros	Rep. Val.	Narrow	LDR
ZCA [8]	**Low**	**Low**	Low	✔	×	×	×
FVC [33]	High	High	Modest	✔	Partly	×	×
FPC [2]	High	High	**High**	✔	✔	✔	×
B∆I	**Low**	Modest	**High**	✔	✔	✔	✔

Table 1: Qualitative comparison of B∆I with prior work. LDR: Low dynamic range. Bold font indicates desirable characteristics.

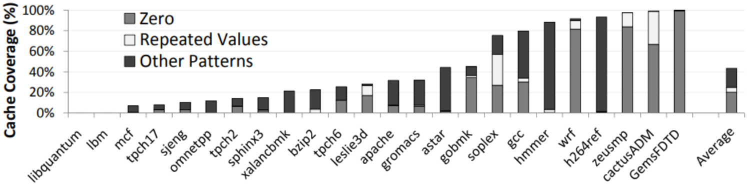


Figure 1: Percentage of cache lines with different data patterns in a 2MB L2 cache. "Other Patterns" includes "Narrow Values".

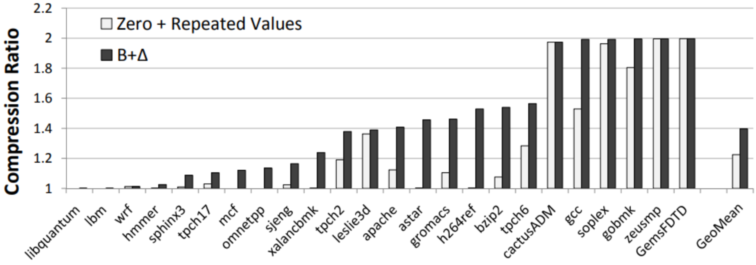


Figure 2: Effective compression ratio with different value patterns

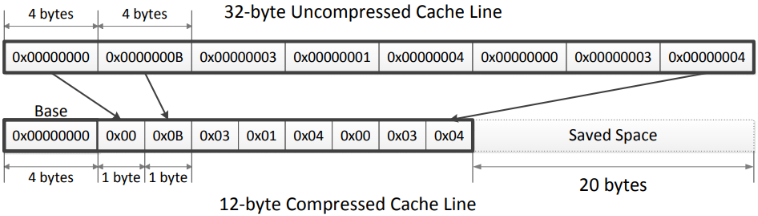


Figure 3: Cache line from *h264ref* compressed with B+Δ

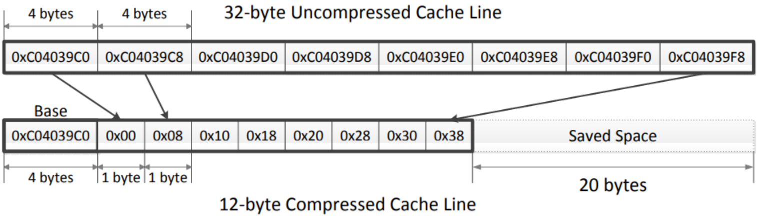


Figure 4: Cache line from *perlbench* compressed with B+Δ

Decompression:

Decompression:

B^* – base value

Decompression:

B^* – base value

$\Delta = \Delta_1, \Delta_2, \ldots, \Delta_n$ – array of differences

Decompression:

B^* – base value
$\Delta = \Delta_1, \Delta_2, \ldots, \Delta_n$ – array of differences
$S = (v_1, v_2, \ldots, v_n)$ – set of real values

Decompression:

B^* – base value
$\Delta = \Delta_1, \Delta_2, \ldots, \Delta_n$ – array of differences
$S = (v_1, v_2, \ldots, v_n)$ – set of real values
$v_i = B^* + \Delta_i$

Decompression:

B^* – base value
$\Delta = \Delta_1, \Delta_2, \ldots, \Delta_n$ – array of differences
$S = (v_1, v_2, \ldots, v_n)$ – set of real values
$v_i = B^* + \Delta_i$ – SIMD-style vector adder.

Compression:

Compression:

Algorithm views a cache line as a set of fixed-size values i.e., 8 8-byte, 16 4-byte, or 32 2-byte values for a 64-byte cache line.

Compression:

Algorithm views a cache line as a set of fixed-size values i.e., 8 8-byte, 16 4-byte, or 32 2-byte values for a 64-byte cache line.

Assume that the size of each value in the set is k bytes and the set of values to be compressed is $S = (v_1, v_2, \ldots, v_n)$.

Compression:

Algorithm views a cache line as a set of fixed-size values i.e., 8 8-byte, 16 4-byte, or 32 2-byte values for a 64-byte cache line.

Assume that the size of each value in the set is k bytes and the set of values to be compressed is $S = (v_1, v_2, \ldots, v_n)$.

The goal of the algorithm is to determine the value of the base, B^* and the size of values in the set, k, that provide maximum compressibility.

Compression:

The cache line is compressible only if $\max(\text{size}(\Delta_i)) < k$.

Compression:

The cache line is compressible only if $\max(\text{size}(\Delta_i)) < k$.

To determine the value of B^*, either the value of $\min(S)$ or $\max(S)$ needs to be found.

Compression:

The cache line is compressible only if $\max(\text{size}(\Delta_i)) < k$.

To determine the value of B^*, either the value of $\min(S)$ or $\max(S)$ needs to be found.

The optimum can be reached only for $\min(S)$, $\max(S)$, or exactly in between them.

Compression:

The cache line is compressible only if $\max(\text{size}(\Delta_i)) < k$.

To determine the value of B^*, either the value of $\min(S)$ or $\max(S)$ needs to be found.

The optimum can be reached only for $\min(S)$, $\max(S)$, or exactly in between them.

Check all possible value of $k \in \{2, 4, 8\}$ and compute B^*.

Compression:

To avoid compression latency increase and reduce hardware complexity use the first value from the set of values as an approximation for the B^*.

Surprise!

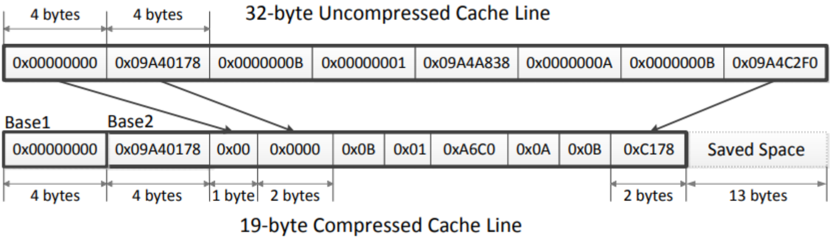
Surprise!

Choosing the first value as the base instead of computing the optimum base value reduces the average compression ratio only by 0.4%.

Some of applications can mix data of different types in the same cache line.

Some of applications can mix data of different types in the same cache line.

Idea: multiple bases.


4 bytes
4 bytes
32-byte Uncompressed Cache Line
0x00000000
0x09A40178
0x0000000B
0x00000001
0x09A4A838
0x0000000A
0x0000000B
0x09A4C2F0
Base1
Base2
0x00000000
0x09A40178
0x00
0x0000
0x0B
0x01
0xA6C0
0x0A
0x0B
0xC178
Saved Space
4 bytes
4 bytes
1 byte
2 bytes
2 bytes
13 bytes
19-byte Compressed Cache Line

Why not more bases?

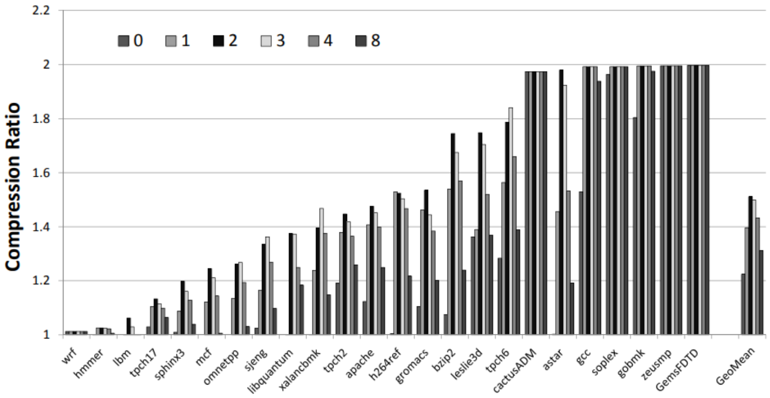


Figure 6: Effective compression ratio with different number of bases. "0" corresponds to zero and repeated value compression.

One simple trick!

One simple trick!

0

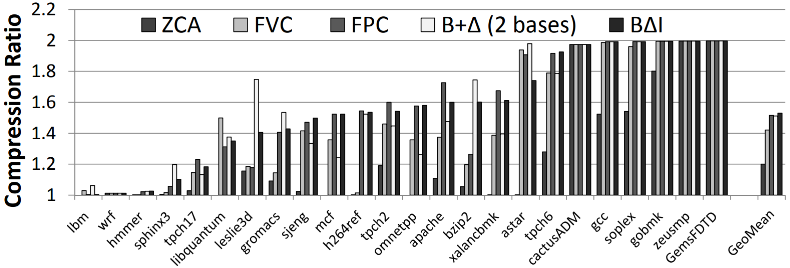


Figure 7: Compression ratio comparison of different algorithms: ZCA [8], FVC [33], FPC [2], B+Δ (two arbitrary bases), and BΔI. Results are obtained on a cache with twice the tags to accommodate more cache lines in the same data space as an uncompressed cache.

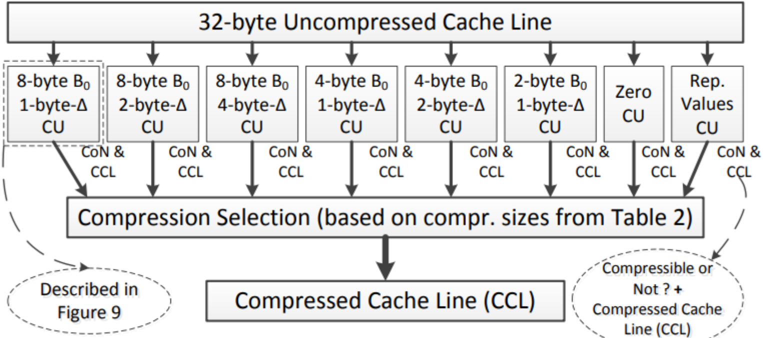


Figure 8: Compressor design. CU: Compressor unit.

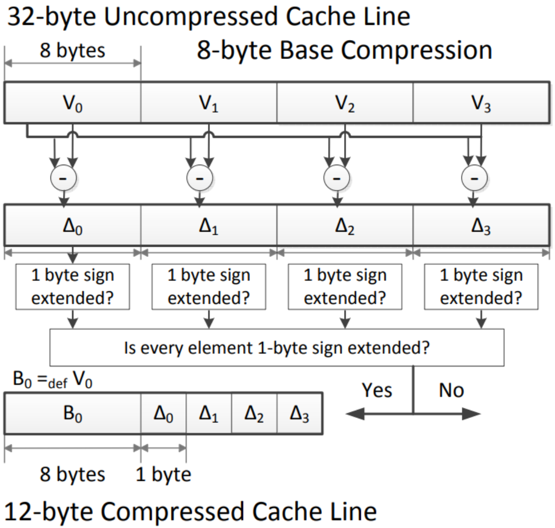


Figure 9: Compressor unit for 8-byte base, 1-byte Δ

Name	Base	Δ	Size	Enc.	Name	Base	Δ	Size	Enc.
Zeros	1	0	1/1	0000	Rep.Values	8	0	8/8	0001
Base8-Δ1	8	1	12/16	0010	Base8-Δ2	8	2	16/24	0011
Base8-Δ4	8	4	24/40	0100	Base4-Δ1	4	1	12/20	0101
Base4-Δ2	4	2	20/36	0110	Base2-Δ1	2	1	18/34	0111
NoCompr.	N/A	N/A	32/64	1111					

Table 2: BΔI encoding. All sizes are in bytes. Compressed sizes (in bytes) are given for 32-/64-byte cache lines.

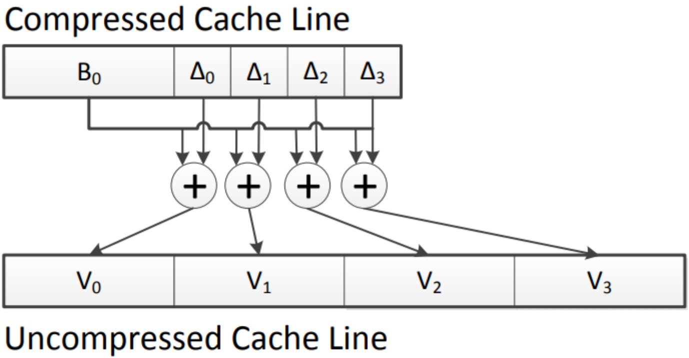


Figure 10: Decompressor design

Is it all?

Is it all?

No :(

Is it all?

No :(

Problems:

Is it all?

No :(

Problems: Unused space in compressed cache lines.

Is it all?

No :(

Problems: Unused space in compressed cache lines.

Ideas: Put data in unused space.

Is it all?

No :(

Problems: Unused space in compressed cache lines. We need to address cache lines.

Ideas: Put data in unused space.

Is it all?

No :(

Problems: Unused space in compressed cache lines. We need to address cache lines.

Ideas: Put data in unused space. Change cache organisation.

Is it all?

No :(

Problems: Unused space in compressed cache lines. We need to address cache lines. Cache eviction policy.

Ideas: Put data in unused space. Change cache organisation.

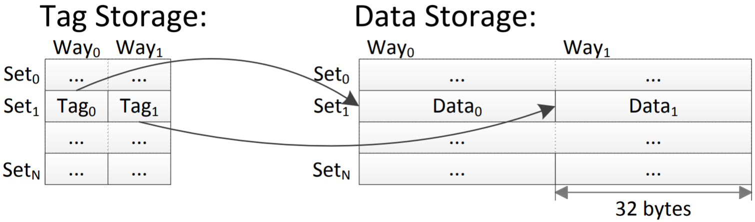
Is it all?

No :(

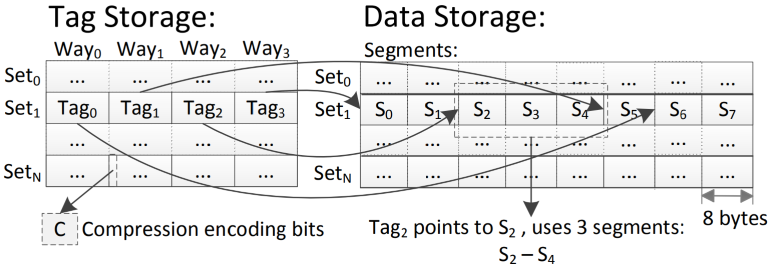
Problems: Unused space in compressed cache lines. We need to address cache lines. Cache eviction policy.

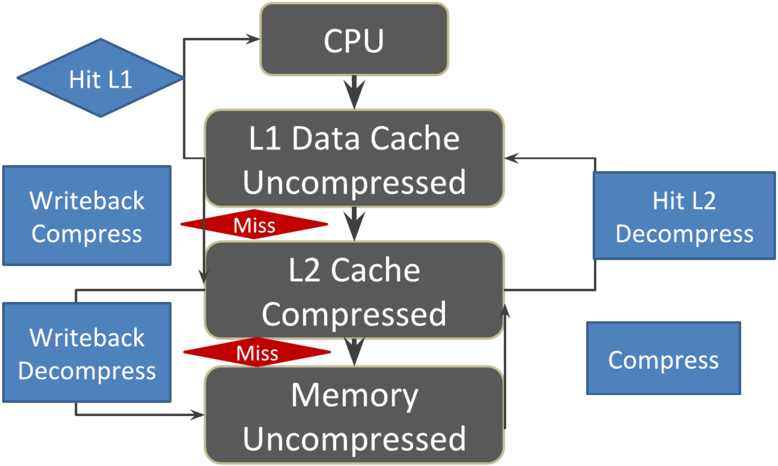
Ideas: Put data in unused space. Change cache organisation. Evicts multiple LRU cache lines.

Conventional 2-way cache with 32-byte lines



BΔI cache: 4-way tag storage, 8-byte segmented data storage




CPU
Hit L1
L1 Data Cache
Uncompressed
Writeback
Compress
Miss
Hit L2
Decompress
L2 Cache
Compressed
Writeback
Decompress
Miss
Compress
Memory
Uncompressed

Conclusion:

- A new Base-Δ-Immediate compression mechanism
- Many cache lines can be efficiently represented using base+delta encoding
- Key properties
  - Low latency decompression
  - Simple hardware implementation
  - High compression ratio with high coverage
- Improves cache hit ratio and performance
- Outperforms state-of-the-art cache compression techniques: FVC and FPC

(standing ovation)