Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches

Gennady Pekhimenko§ Vivek Seshadri§
Onur Mutlu§ Michael A. Kozuch†
Phillip B. Gibbons† Todd C. Mowry§

§ **Carnegie Mellon University** † **Intel Labs Pittsburgh**

Published at PACT 2012

Presented by Marc-Philippe Bartholomä

Problem & Goal

Large Cache Improves Performance

- Larger capacity ⇒ fewer misses ⇒ better performance
- Larger capacity ⇒ fewer off-chip cache misses
  - Avoids memory bandwidth bottleneck
  - Especially important for multi-core with shared memory

But increasing capacity by scaling the conventional design:

- Slower caches
- More power consumption
- More area required

Large Cache Improves Performance

- Larger capacity ⇒ fewer misses ⇒ better performance
- Larger capacity ⇒ fewer off-chip cache misses
  - Avoids memory bandwidth bottleneck
  - Especially important for multi-core with shared memory

Idea: Compress the data in caches to save on hardware costs

But inc ign:

- Slower caches
- More power consumption
- More area required

Goals of Cache Compression

- Compression/decompression need to be very **fast**
  - Decompression is on the critical path
- **Simple** compression logic avoids large power and area costs
- Must compress the data **effectively**
  - Otherwise there isn’t much gain in capacity

Background

Data Patterns in Applications: Zeroes

0x00000000	**0x00000000**	**0x00000000**	**0x00000000**

16-byte cache line

Data Pattern: Repeated Values

0xCAFE4A11	0xCAFE4A11	0xCAFE4A11	0xCAFE4A11

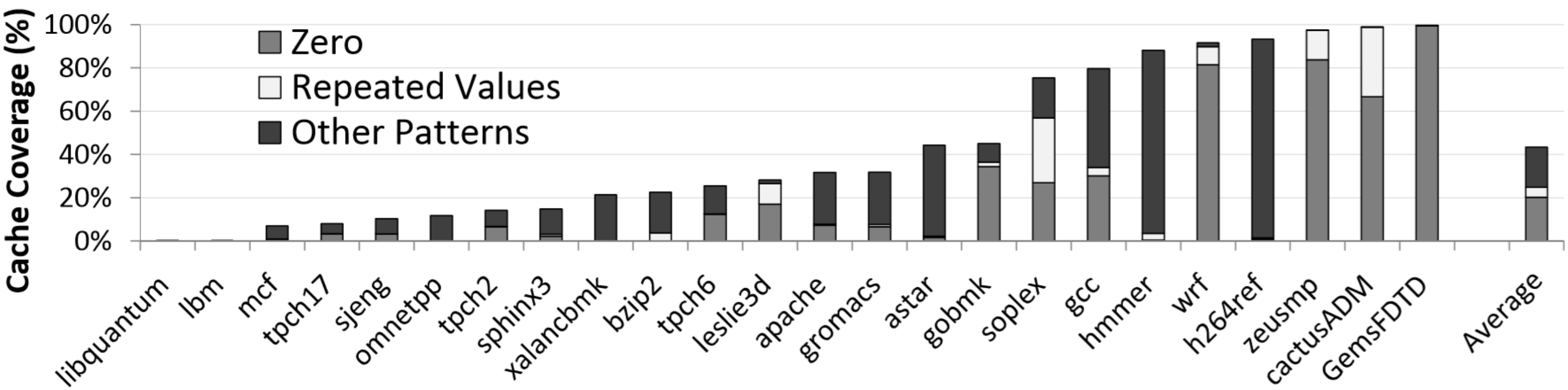
Data Pattern: Narrow Values

Values have more storage allocated than necessary

0x000000CA	0x000000FE	0x0000004A	0x00000011

Data Patterns are Frequent

43% of application cache lines can be compressed on average



Narrow Values are included in Other Patterns

Data Patterns: Low Dynamic Range

The values are larger than the difference between them

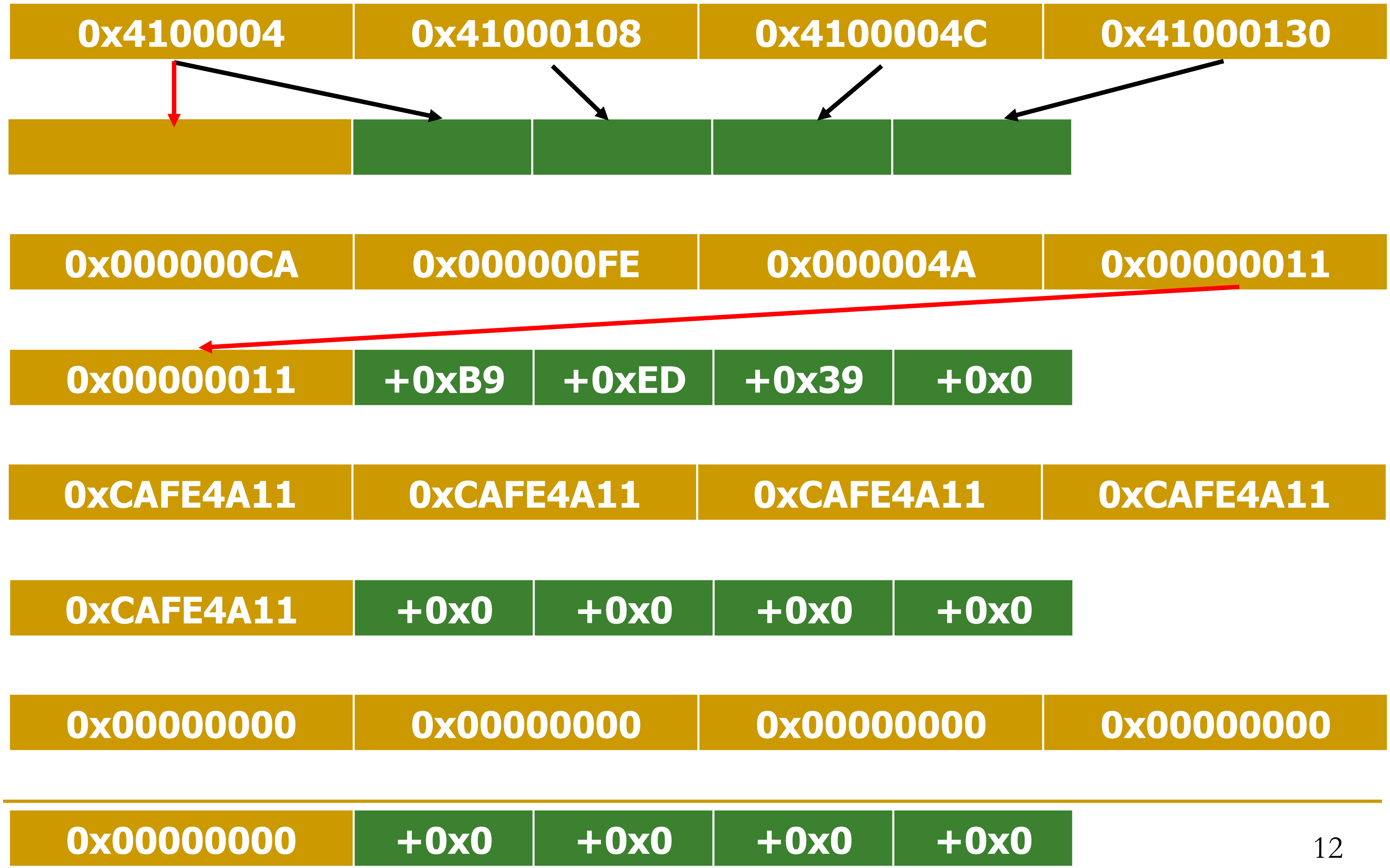
0x4100004	0x41000108	0x4100004C	0x41000130

0x000000CA	0x000000FE	0x0000004A	0x00000011

0xCAFE4A11	0xCAFE4A11	0xCAFE4A11	0xCAFE4A11

0x00000000	0x00000000	0x00000000	0x00000000

Base+Delta Encoding



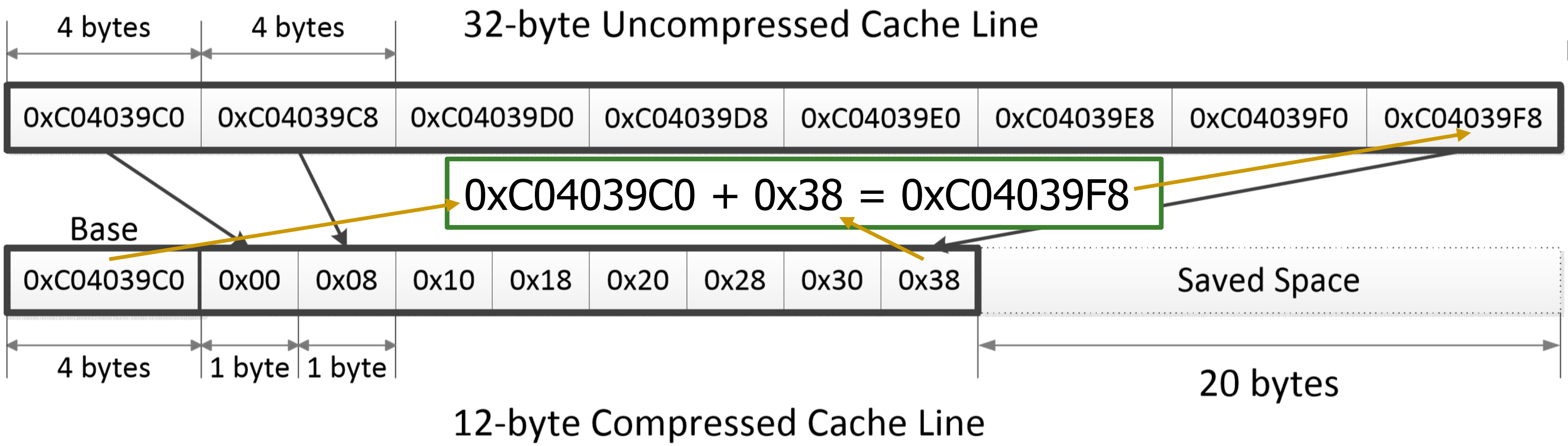
Novelty

Novelty

- Compress on cache line granularity
  - Previous approaches work on individual words
- View data patterns as Low Dynamic Range
- Apply Base+Delta compression to caches
  - Instead of general purpose compression
  - Instead of special case handling for some patterns

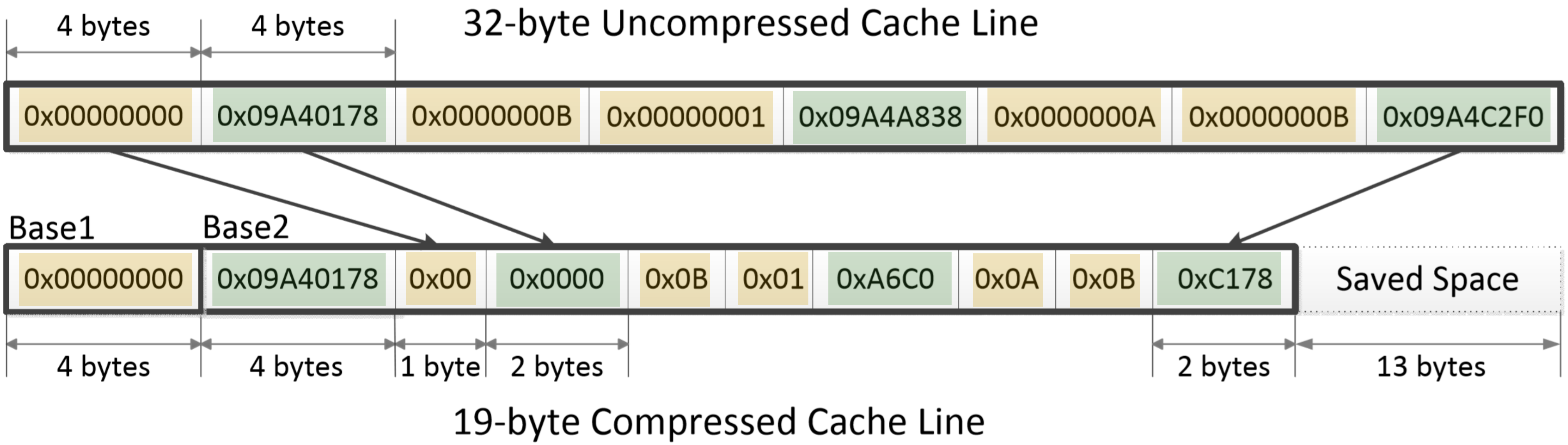
Key Approach and Ideas

Base+Delta Compression



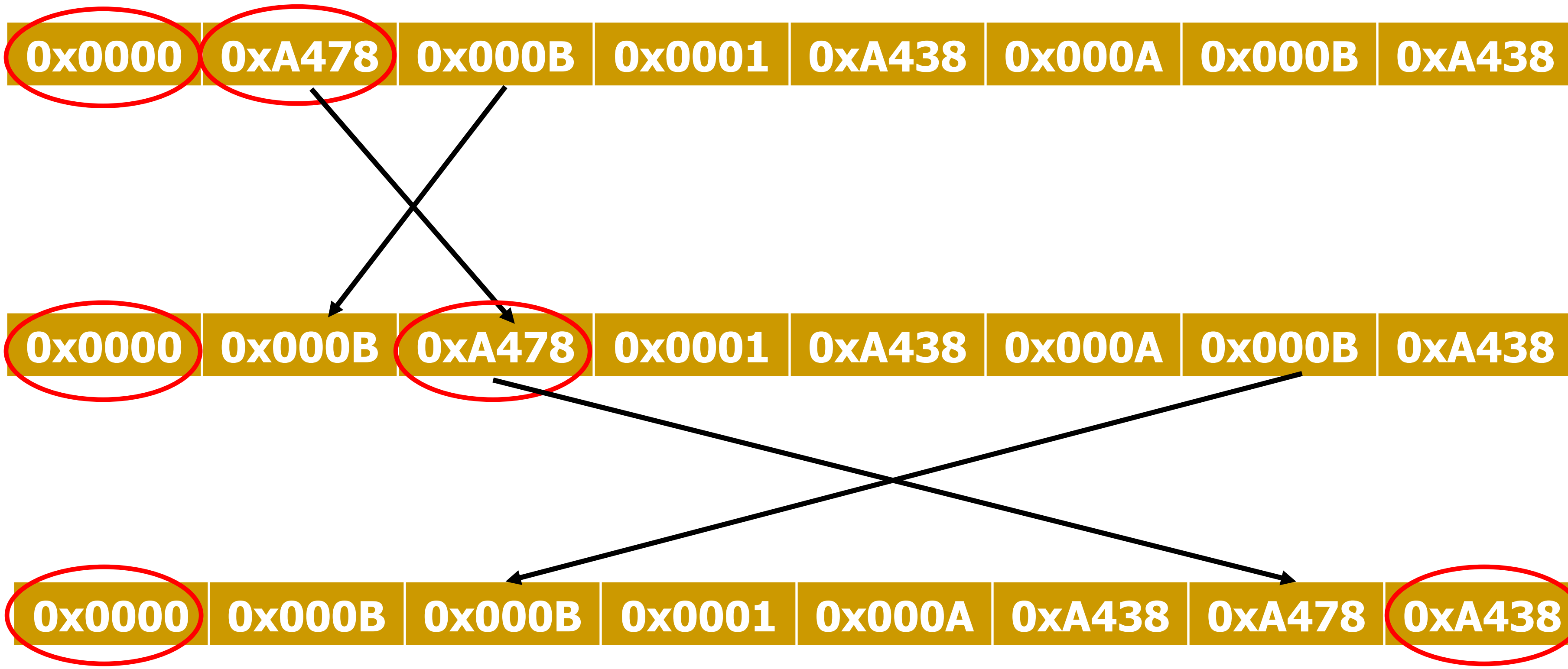
- Fast decompression (vector addition)
- Simple hardware (addition/subtraction and comparison)
- Effectively compresses observed patterns

Room for Improvement



- Multiple bases allow compression of more cache lines
- Need to encode multiple bases in compressed line

Finding Bases?



- Gets more difficult with more bases

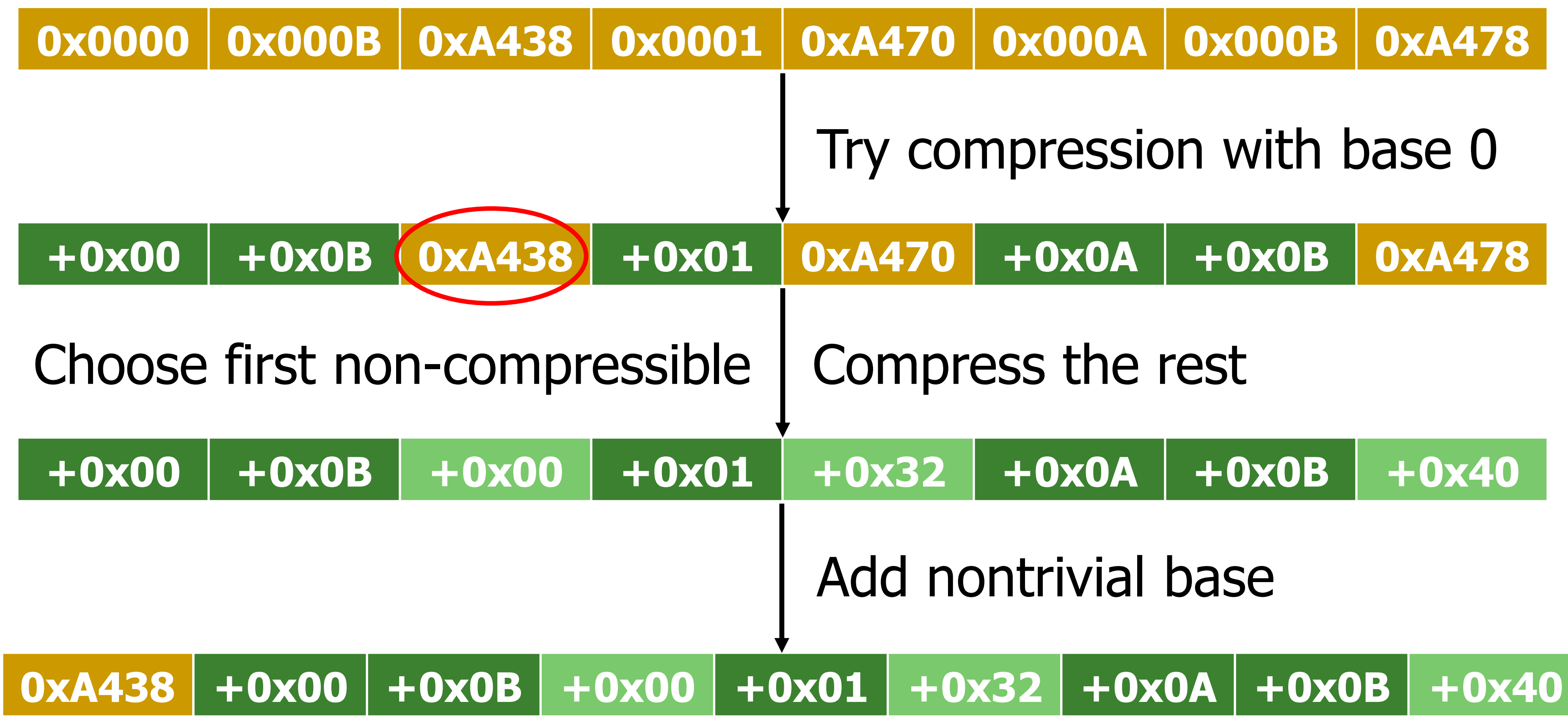
Base Delta Immediate (BΔI)

- 2 bases
  - 1 is always 0x00000000 ⇒ no need to save
  - 1 is arbitrary
  - Values with respect to the zero base are the “immediates”

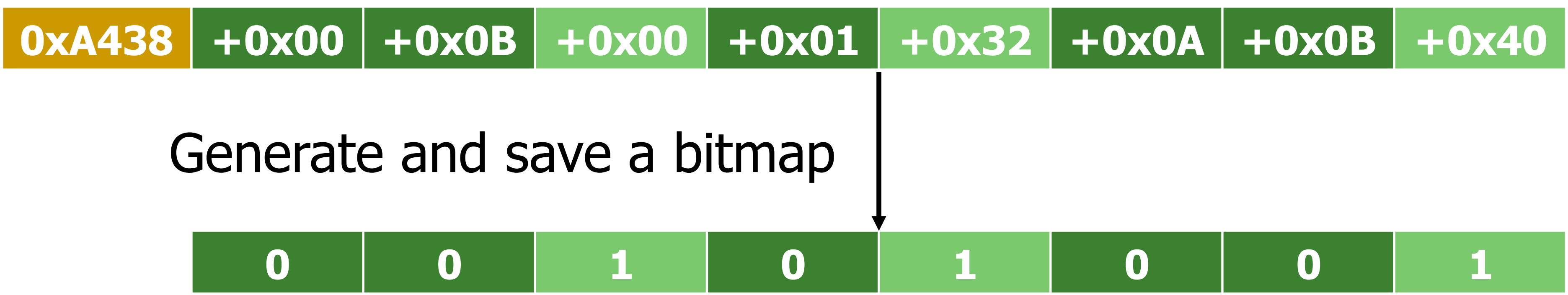
- Slightly better than Base+Delta with 2 arbitrary bases
  - Which in turn compresses better than Base+Delta with other number of bases

Mechanism

Finding base for BΔI

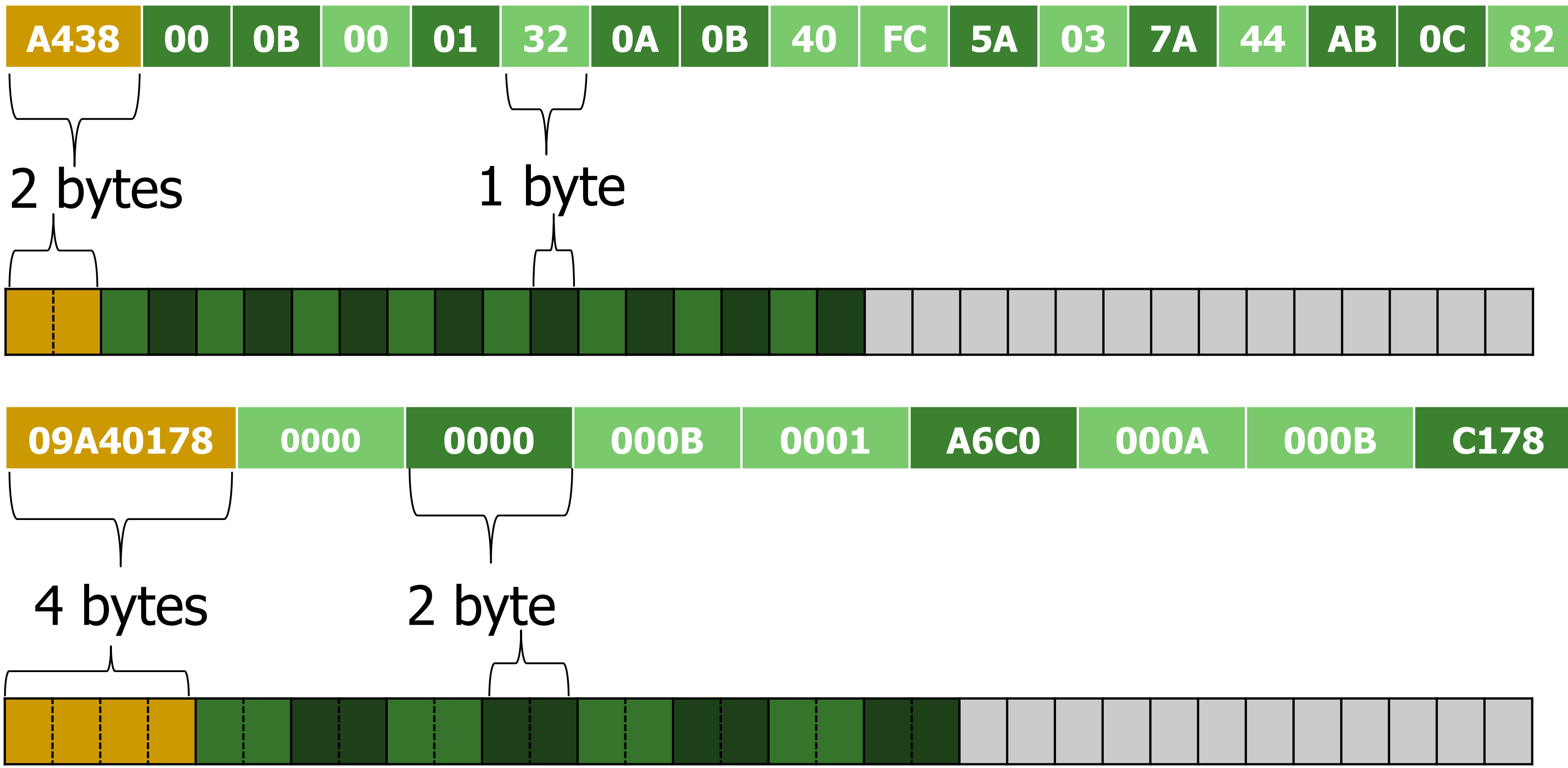


Attribute Deltas to Bases

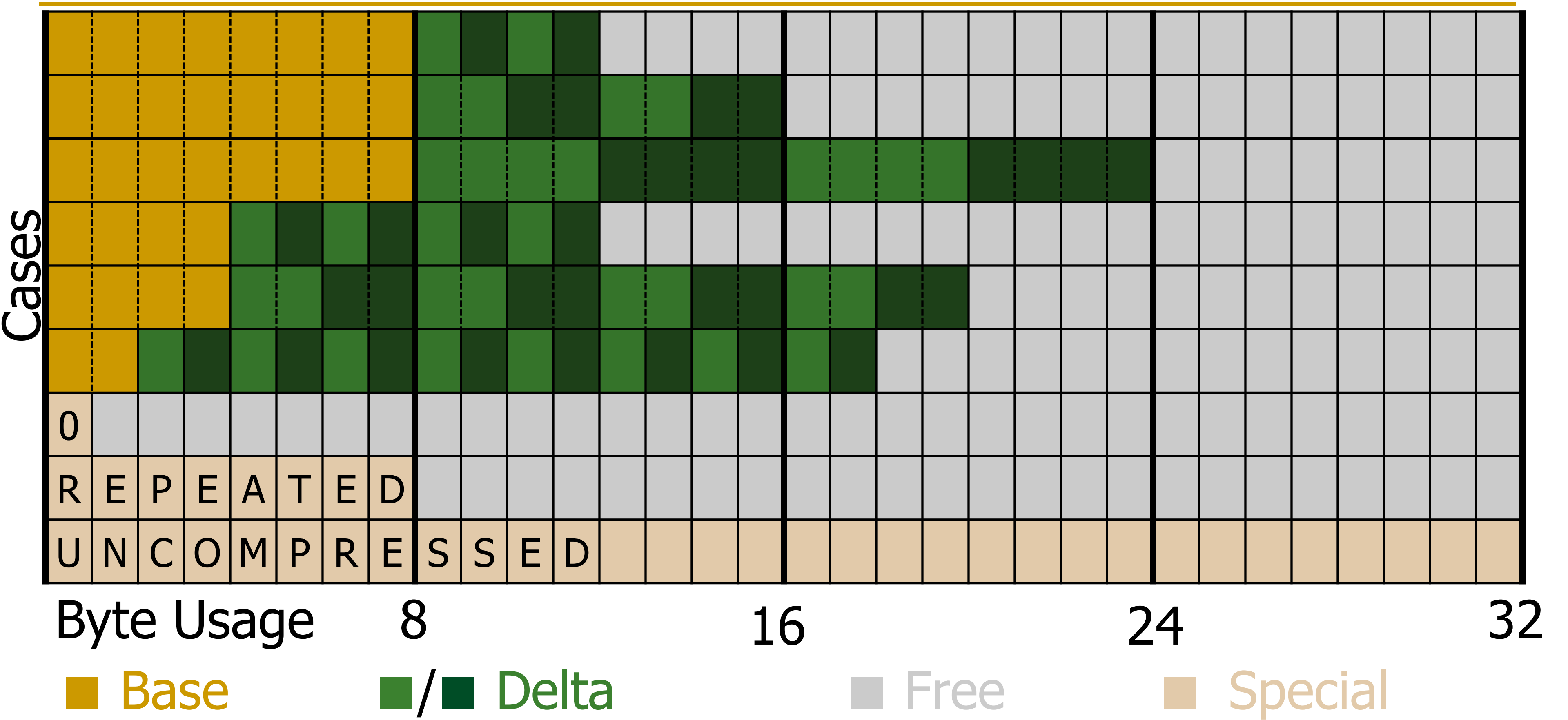


Note: Decompression becomes **masked** vector addition

Determining Base and Delta Sizes



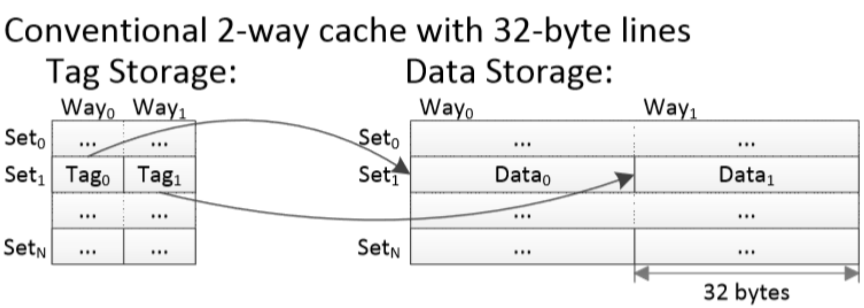
Determining Base and Delta Sizes

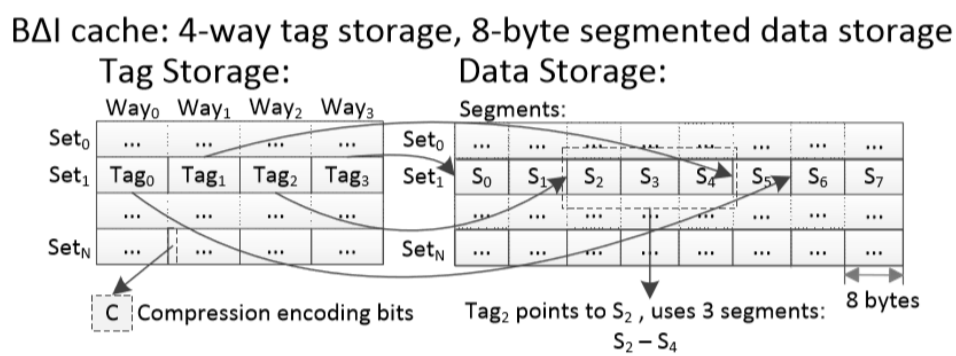


- Zero line and repeated values are special cases
- Everything is attempted in parallel and shortest is chosen

Changes in Cache Organization

- Double the amount of cache tags
- Add encoding bits for cases and bitmask for base determination
- Segment the cache lines and add segment pointers to the tags



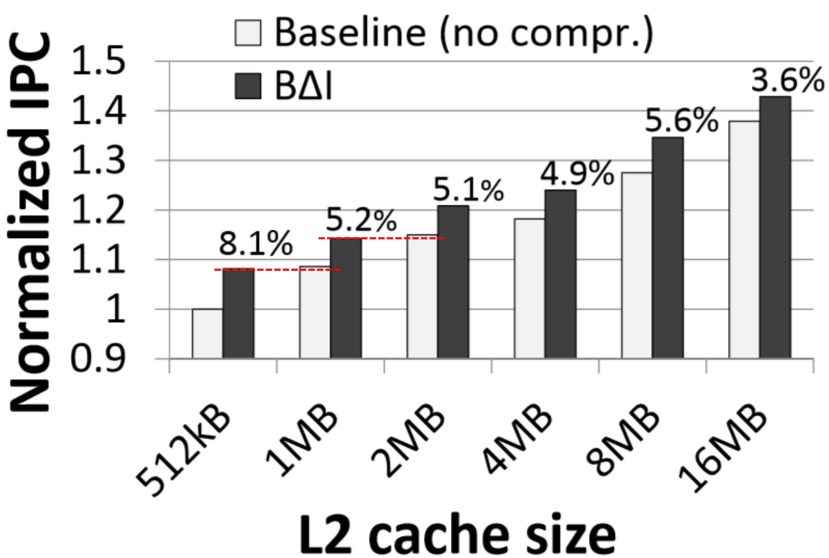


Key Results: Methodology and Evaluation

Methodology

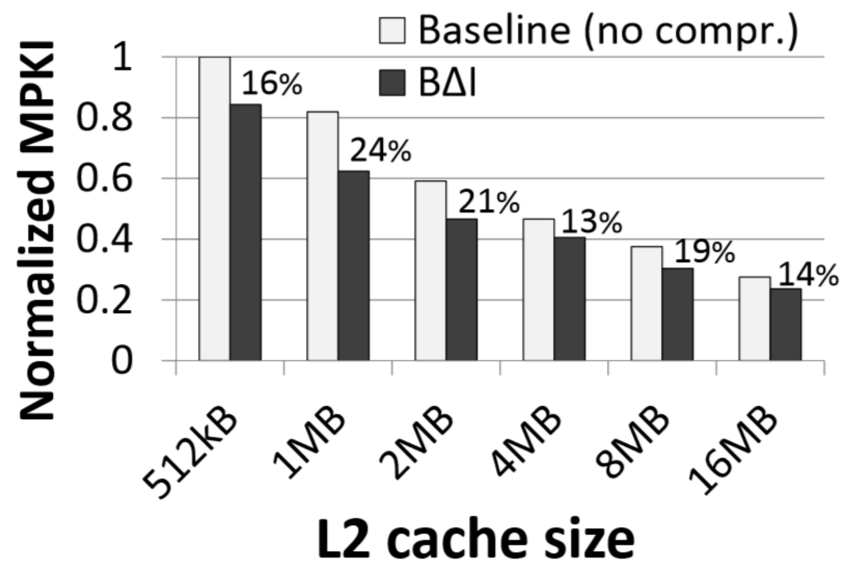
- x86-based Simulation
- 1-4 cores
- SPEC2006, TPC-H and Apache web server workloads
- L1/L2/L3 cache latencies from CACTI [Thoziyoor+, ISCA’08]

BΔI vs Baseline ⇒ Capacity Nearly Doubled



(a) IPC

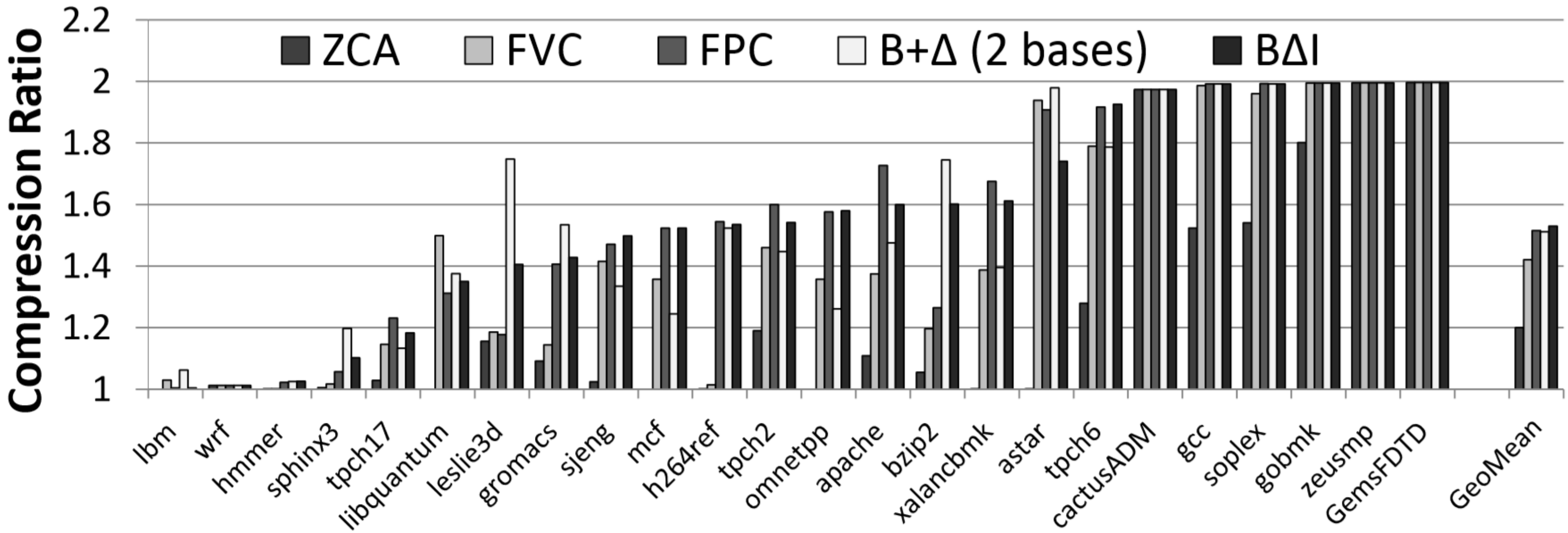
Instructions per Cycle



(b) MPKI

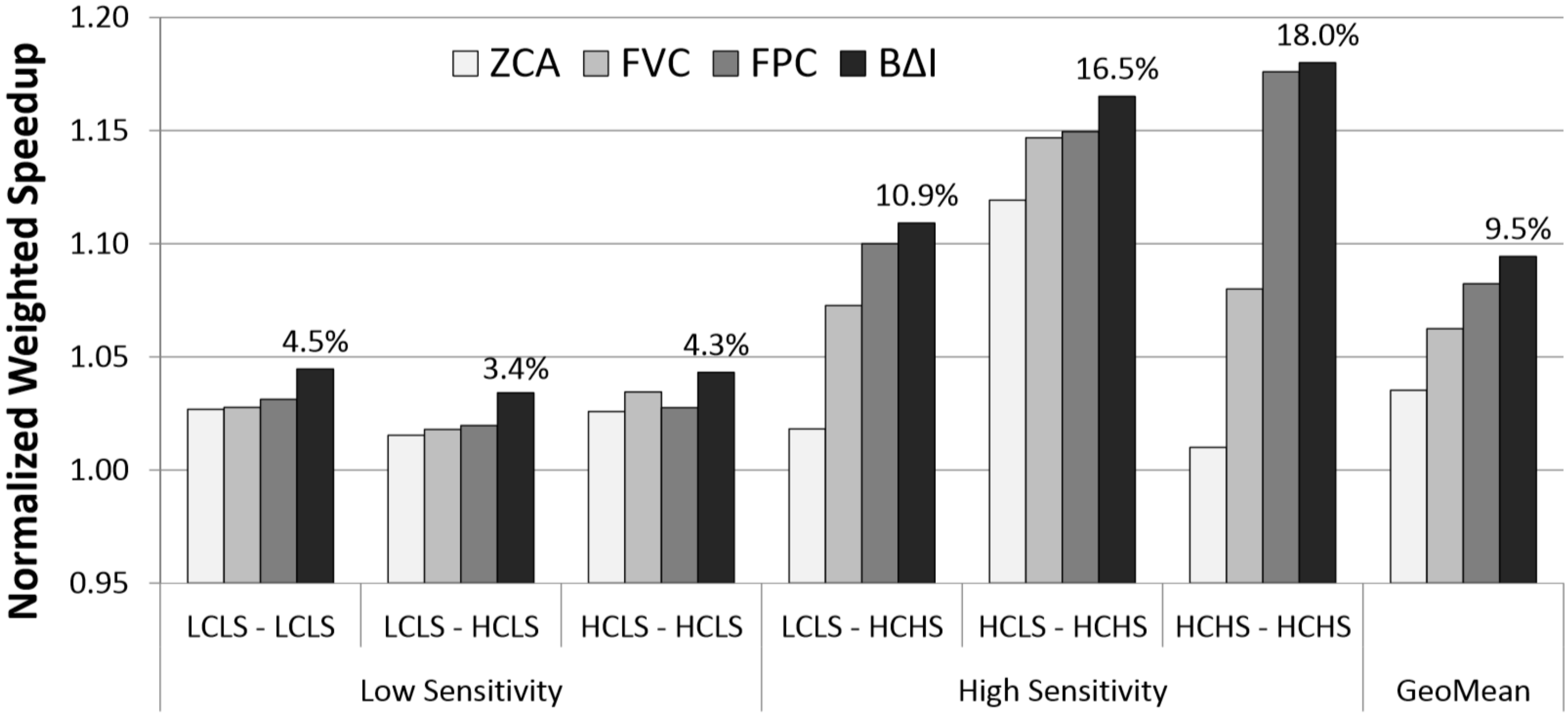
Misses per Kilo Instruction

BΔI vs Other Approaches ⇒ Best Comp. Ratio



- ZCA (Zero-Content Augmented cache): exploits only zeroes
- FVC (Frequent Value Compression): zeroes and common words
- FPC (Frequent Pattern Compression): patterns including repeated values and narrow values

Multi-Core Profits Even More



LC/HC: low/high compressibility, LS/HS low/high cache size sensitivity
(uses 2 cores, 2MB L2 cache)
Missing: LCHS due to absence in sample workloads

Summary

Summary

- **Goal:** Increase cache capacity using data compression at lower cost
- **Key Insight:** A significant fraction (43%) of real-world cache lines can be compressed
- **Key Mechanism:** Base+Delta encoding fits well to exploit low dynamic range patterns
- **Key Results:** BΔI yields nearly the performance gain of a cache with double capacity without the same costs in area and power
  - 5.1% avg. performance increase on single-core over baseline
  - 9.5% avg. performance increase on dual-core over baseline

Strengths

Strengths of the Paper

- Novel approach leading to significant improvement
- Thorough analysis and evaluation of patterns, previous approaches and variants
- Elegant solution and principled design
- Easy-to-understand and well-structured paper
- Transparent to the OS and applications
- Compression mechanism is predictable for the user

Weaknesses

Weaknesses/Limitations of the Paper

- Requires double amount of cache tags
  - Potential bottleneck
- Adds the possibility of eviction when writing with cache-hit
- Because real capacity is unknown, it is harder to optimize applications
- Missing category for multi-core workload
- Analysis of cache size only for Base+Delta (no bitmap)
- Compressed data patterns don’t capture floating point values
- Too much latency for L1 cache

Thoughts and Ideas

Extensions

- Special case with only the 0 base
- Base Finding approach generalizes to 2 arbitrary bases
  - Analyze the benefit of switching between BΔI and Base+Delta with 1 or 2 bases

- To save on cache tags you could load 2 contiguous cache lines

- Include base bitmap in the deltas
- For repeated values of size up to 4 bytes, you could save them using the bitmask for base attribution

Takeaways

Key Takeaways

- Paper is a prime example of principled design
  - Carefully examines the potential
  - Thoroughly analyzes the tradeoffs
  - Picks the best variant

- Data compression is viable for on-chip caches

Questions

Discussion

- Cache Replacement Policy
  - Paper uses slightly modified LRU and leaves detailed study for future work
  - For uncompressed caches: theoretical optimal cache replacement policy (adapted from Computer Systems 2018):

For eviction: Choose entry that will not be referenced again for the longest period of time.

  - Is the shown CRP also optimal for caches with compression? Why or why not?
  - What aspects need to be considered to adapt it?
  - Ideas about what an actual cache replacement policy should do?

Discussion

- Patterns in floating point values? Exploitable with BΔI?

Systems Programming and Computer Architecture 2017

- Numerical form:

$$(-1)^s \; M \; 2^E$$

  - **Sign bit** *s* determines whether number is negative or positive
  - **Significand** *M* normally a fractional value in range [1.0,2.0).
  - **Exponent** *E* weights value by power of two

- Encoding
  - MSB (Most Significant Bit) s is sign bit *s*
  - exp field encodes *E* (but is not equal to *E*)
  - frac field encodes *M* (but is not equal to *M*)

s1	exp 8 bits	frac 23 bits

Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches

Gennady Pekhimenko [§] Vivek Seshadri [§]
Onur Mutlu [§] Michael A. Kozuch [†]
Phillip B. Gibbons [†] Todd C. Mowry [§]

[§] Carnegie Mellon University [†] Intel Labs Pittsburgh

Published at PACT 2012

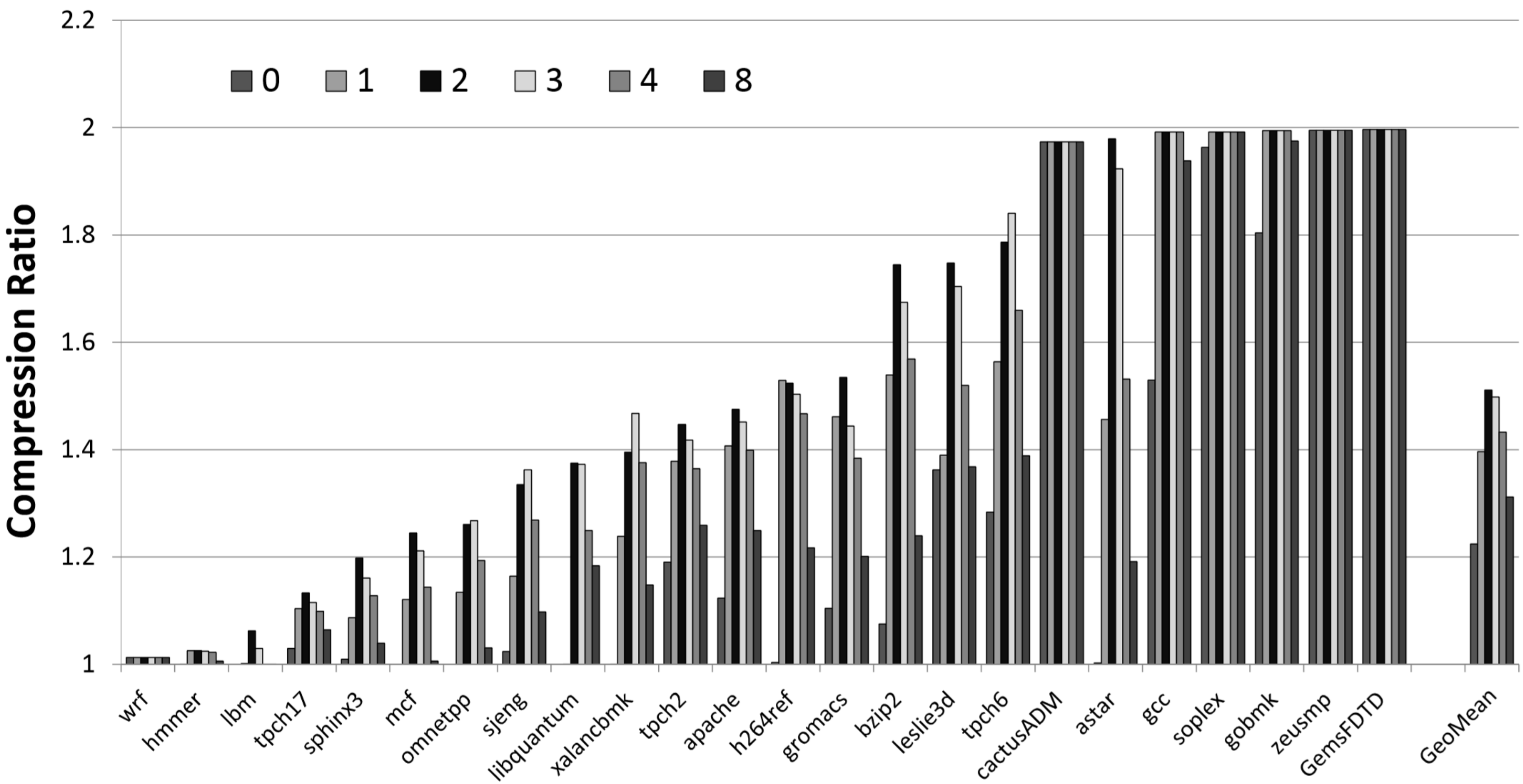
Presented by Marc-Philippe Bartholomä

Backup Slides

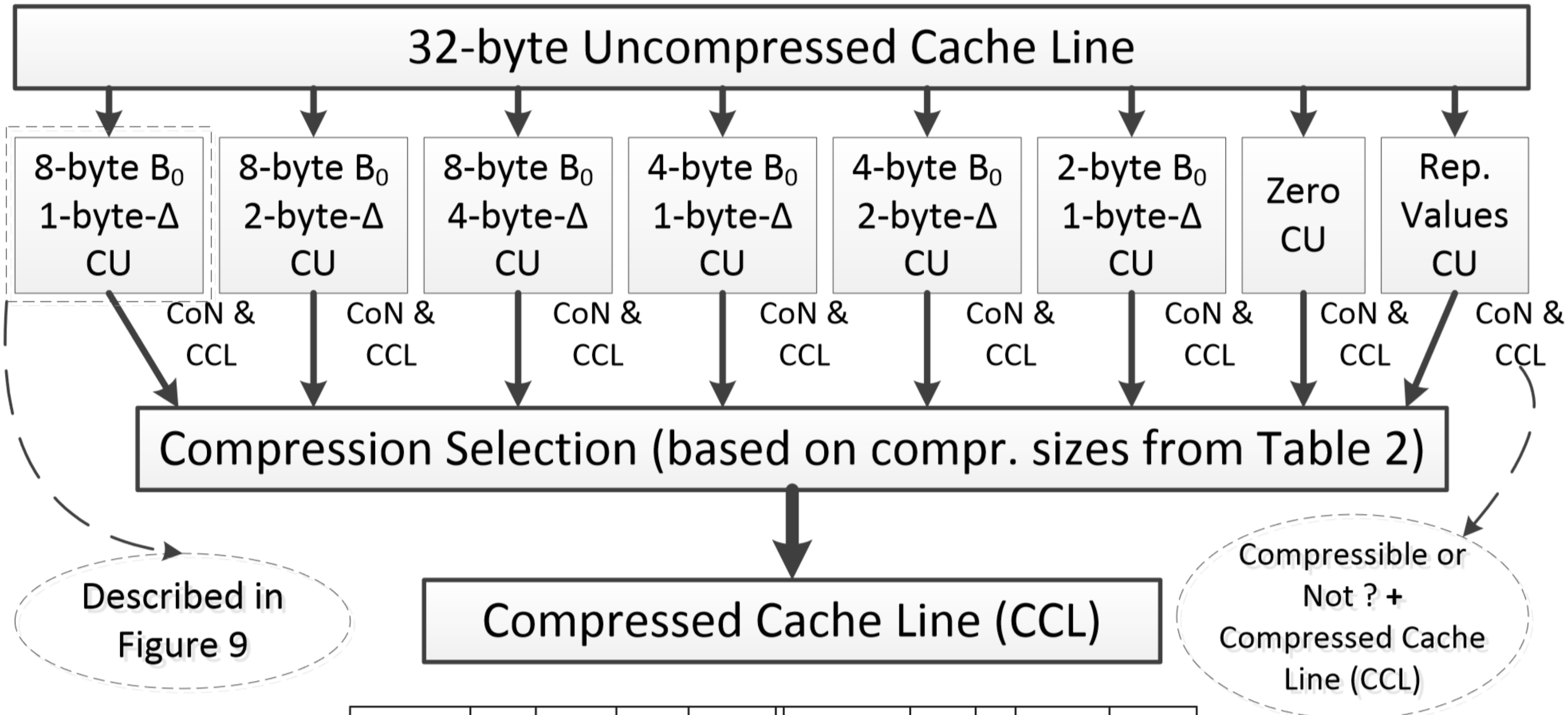
Quantitative Comparison with Prior Work

	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**	✔	✔	✔	✕
BΔI	**Low**	Modest	**High**	✔	✔	✔	✔

Different Number of Bases in Base+Delta



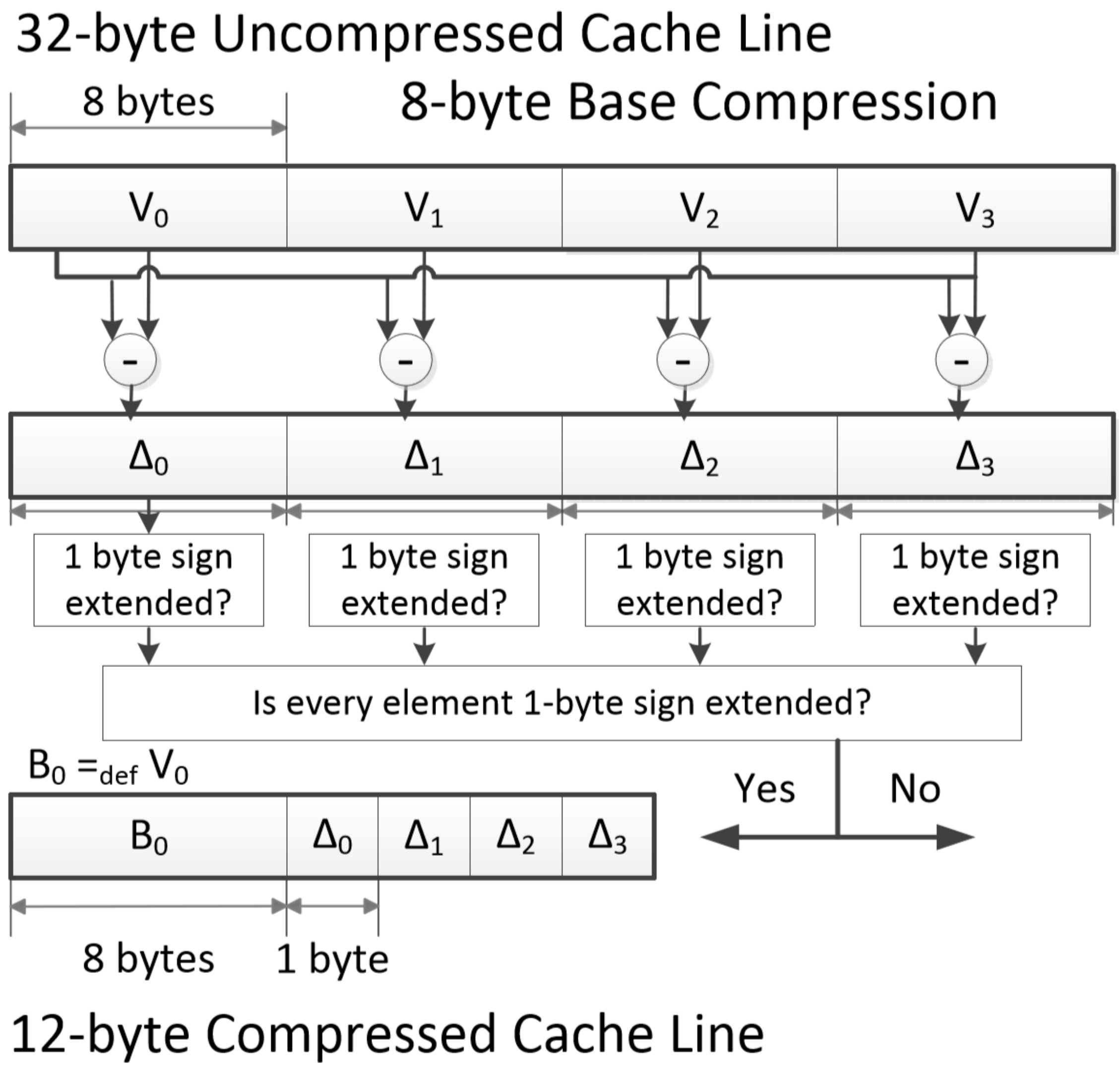
Compression Mechanism: Cases



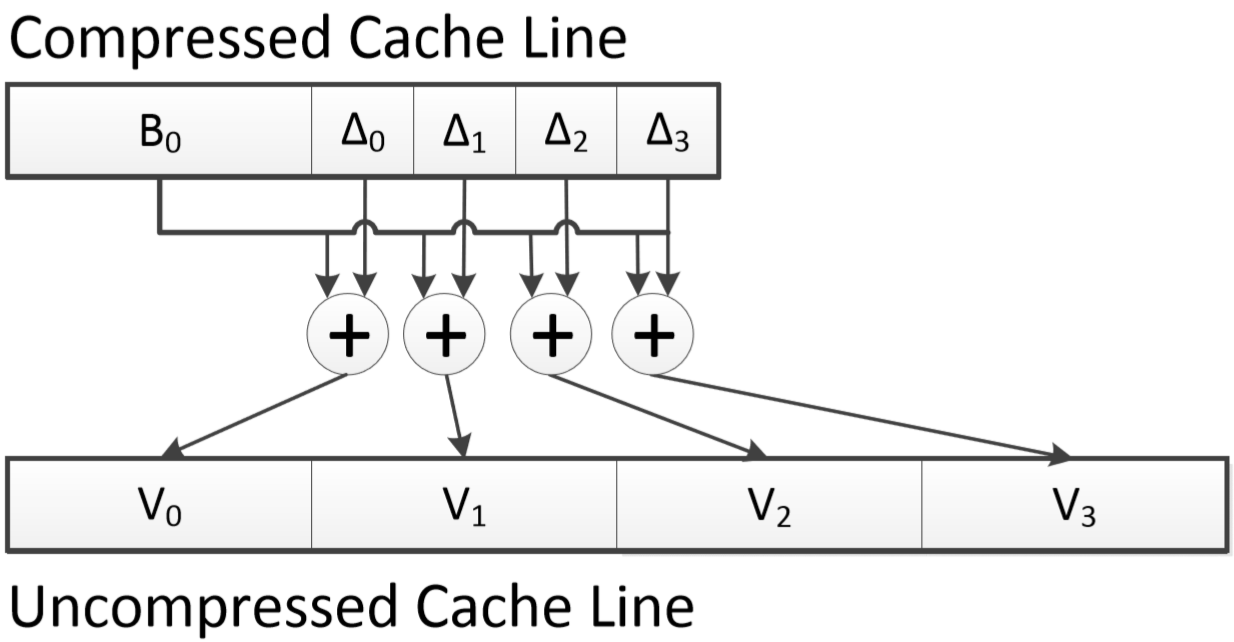
Name	Base	Δ	Size	Enc.	Name	Base	Δ	Size	Enc.
Zeros	1	0	1/1	0000	Rep.Values	8	0	8/8	0001
Base8-Δ1	8	1	12/16	0010	Base8-Δ2	8	2	16/24	0011
Base8-Δ4	8	4	24/40	0100	Base4-Δ1	4	1	12/20	0101
Base4-Δ2	4	2	20/36	0110	Base2-Δ1	2	1	18/34	0111
NoCompr.	N/A	N/A	32/64	1111					

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

Compression Mechanism: Single Case



Decompression Mechanism



Cache Size Analysis

	Baseline	**BΔI**
Size of tag-store entry	21 bits	32 bits (+4–encoding, +7–segment pointer)
Size of data-store entry	512 bits	512 bits
Number of tag-store entries	32768	65536
Number of data-store entries	32768	32768
Tag-store size	84kB	256kB
Total (data-store+tag-store) size	2132kB	2294kB

Table 3: Storage cost analysis for 2MB 16-way L2 cache, assuming 64-byte cache lines, 8-byte segments, and 36 bits for address space.

Multicore Workload Categories

Cat.	Name	Comp. Ratio	Sens.	Name	Comp. Ratio	Sens.	Name	Comp. Ratio	Sens.	Name	Comp. Ratio	Sens.
LCLS	gromacs	1.43 / L	L	hmmer	1.03 / L	L	lbm	1.00 / L	L	libquantum	1.25 / L	L
	leslie3d	1.41 / L	L	sphinx	1.10 / L	L	tpch17	1.18 / L	L	wrf	1.01 / L	L
HCLS	apache	1.60 / H	L	zeusmp	1.99 / H	L	gcc	1.99 / H	L	GemsFDTD	1.99 / H	L
	gobmk	1.99 / H	L	sjeng	1.50 / H	L	tpch2	1.54 / H	L	cactusADM	1.97 / H	L
	tpch6	1.93 / H	L									
HCHS	astar	1.74 / H	H	bzip2	1.60 / H	H	mcf	1.52 / H	H	xalancbmk	1.61 / H	H
	omnetpp	1.58 / H	H	soplex	1.99 / H	H	h264ref	1.52 / H	H			

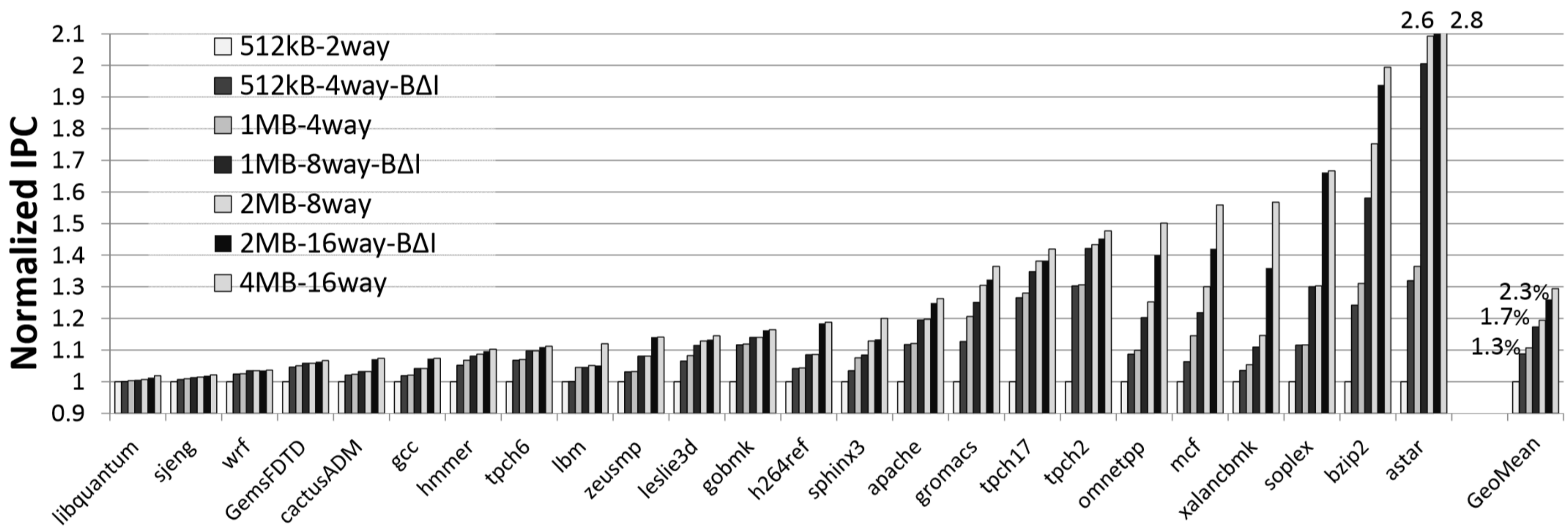
Table 6: Benchmark characteristics and categories: Comp. Ratio (effective compression ratio for 2MB BΔI L2) and Sens. (cache size sensitivity). Sensitivity is the ratio of improvement in performance by going from 512kB to 2MB L2 (L - low ($\leq$ 1.10) , H - high ($>$ 1.10)). For compression ratio: L - low ($\leq$ 1.50), H - high ($>$ 1.50). Cat. means category based on compression ratio and sensitivity.

Quad-Core Results

Cores	No Compression	ZCA	FVC	FPC
1	5.1%	4.1%	2.1%	1.0%
2	9.5%	5.7%	3.1%	1.2%
4	11.2%	5.6%	3.2%	1.3%

Table 7: Average performance improvement of BΔI over other mechanisms: No Compression, ZCA, FVC, and FPC.

Performance "Basically" Doubles



Number of Cache Tags

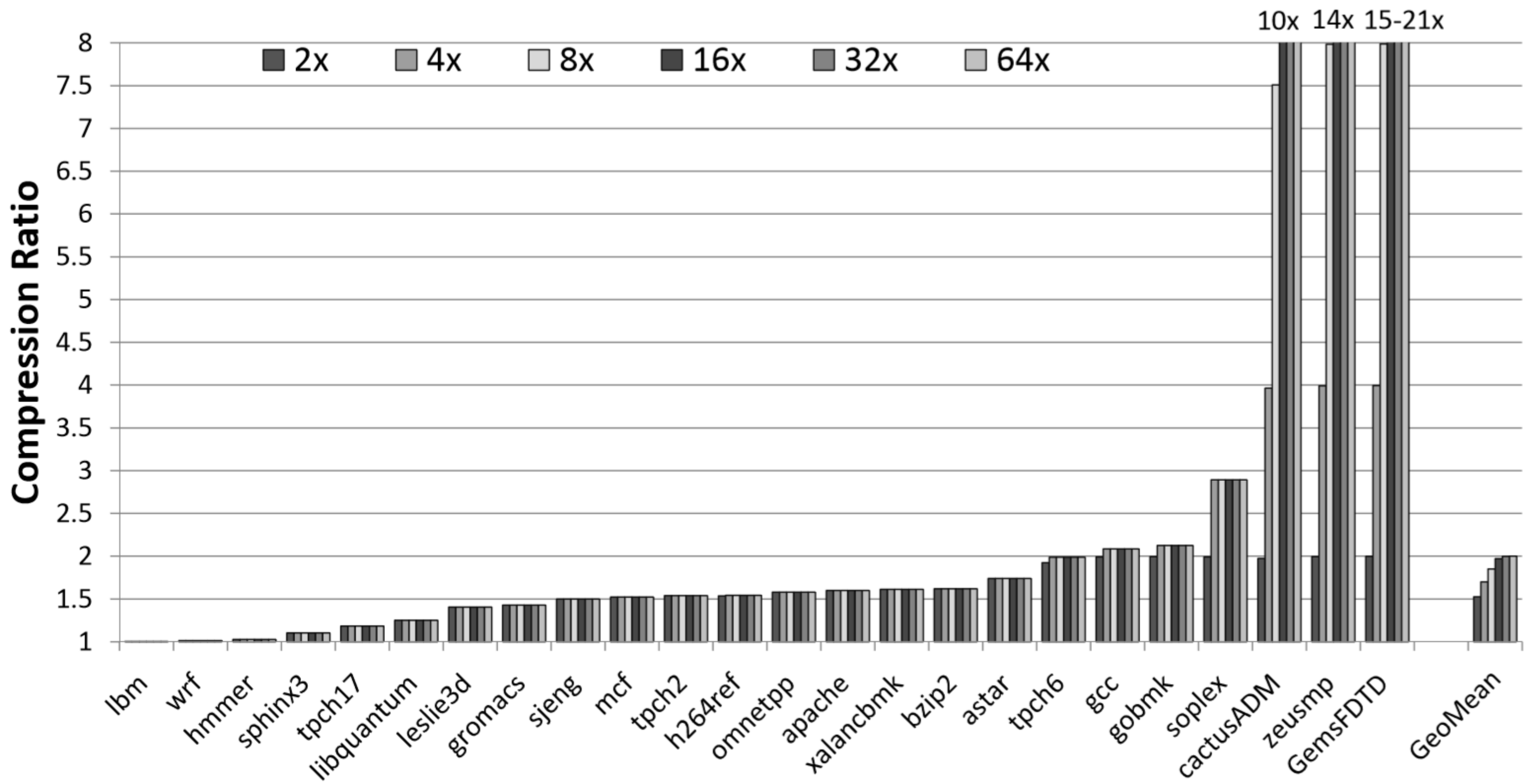
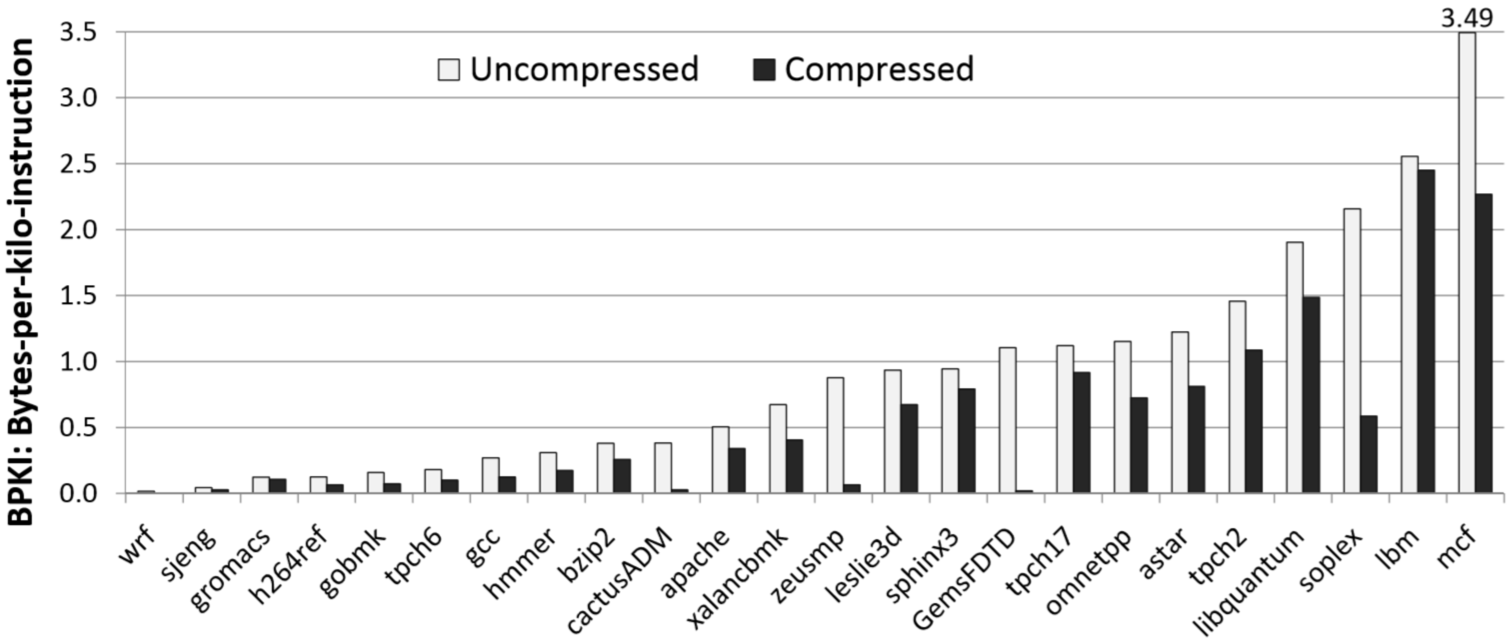


Figure 15: Effective compression ratio vs. number of tags

Effect on Bandwidth



Performance comparison with prior work