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

Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Todd C. Mowry

Phillip B. Gibbons*, Michael A. Kozuch*

Carnegie Mellon

intel*
Motivation: “Memory Wall”

Main memory latency has significant effect on performance (e.g., 300+ cycles)

Computer Architecture: From Microprocessors to Supercomputers, 2011
Motivation: Caching

Cache capacity is hard to increase due to area and power limitations

Intel Core i7

IBM Power7+
Motivation for Data Compression

Significant redundancy in in-cache data:

| 0x00000000 | 0x0000000B | 0x00000003 | 0x00000004 | ... |

How can we exploit this redundancy?

- **Data compression** helps
- Provides effect of a larger cache without making it physically larger
Background on Cache Compression

• Key requirements:
  – **Fast** (low decompression latency)
  – **Simple** (avoid complex hardware changes)
  – **Effective** (good compression ratio, e.g., 1.5 for 2MB cache means effective size of 3MB)
Outline

• Motivation & Background
• Shortcomings of Prior Work
• Base-Delta-Immediate: Key Idea
• Base-Delta-Immediate: Implementation
• Evaluation
• Conclusion
• Future Work
Why Not Software Compression?

- Lossless data compression
  - Lempel-Ziv
    - DEFLATE (gzip, png)
    - LZW (gif)
  - Huffman coding
- Lossy data compression
  - Multimedia (audio, video)

- Effective, but **slow and complex** in hardware
  - 100+ cycles vs. 10-15 cycles for L2/L3 cache hits
# Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Decompression Latency</th>
<th>Complexity</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequent Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequent Pattern</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Our proposal: Base-Delta-Immediate</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Zero Value Compression

• **Idea**: one bit to encode zero value cache lines

• **Advantages**:
  – Low decompression latency
  – Low complexity

• **Disadvantages**:
  – Low average compression ratio
# Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Decompression Latency</th>
<th>Complexity</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Frequent Value Compression

**Idea:** encode cache lines based on frequently occurring values

Frequent Values:

- **00** - 0x00000000
- **01** - 0x00000001
- **10** - 0x000000FF
- **11** - not frequent

```
0x00000001 0x00000000 0x00000001
0x00000000 0x00000000 0x000000FF
0x000000FF 0x000000FF 0x000000FF
0xABCDEEFF 0xABCDEEFF 0xABCDEEFF
```
### Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Decompression Latency</th>
<th>Complexity</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>✓</td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td>Frequent Value</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
</tbody>
</table>
Frequent Pattern Compression

Idea: encode cache lines based on frequently occurring patterns, e.g., first half of a word is zero

Frequent Patterns:
000 – All zeros
001 – First half zeros
010 – Second half zeros
011 – Repeated bytes
100 – All ones
...
111 – Not a frequent pattern
## Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Decompression Latency</th>
<th>Complexity</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Frequent Value</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Frequent Pattern</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Decompression Latency</th>
<th>Complexity</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Frequent Value</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Frequent Pattern</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Our proposal: Base-Delta-Immediate</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Outline

• Motivation & Background
• Shortcomings of Prior Work
• Base-Delta-Immediate: Key Idea
• Base-Delta-Immediate: Implementation
• Evaluation
• Conclusion
• Future Work
**Key Data Patterns in Real Applications**

**Zero Values**: initialization, sparse matrices, NULL pointers

| 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | ... |

**Repeated Values**: common initial values, adjacent pixels

| 0x000000C0 | 0x000000C0 | 0x000000C0 | 0x000000C0 | ... |

**Narrow Values**: small values stored in a big data type

| 0x000000C0 | 0x000000C8 | 0x000000D0 | 0x000000D8 | ... |

**Other Patterns**: pointers to the same memory region

| 0xC04039C0 | 0xC04039C8 | 0xC04039D0 | 0xC04039D8 | ... |
How Common Are These Patterns?

SPEC2006, databases, web workloads, 2MB L2 cache
“Other Patterns” include Narrow Values

43% of the cache lines belong to key patterns
### Key Data Patterns in Real Applications

**Zero Values**: initialization, sparse matrices, NULL pointers

| 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 | ... |

**Repeated Values**: common initial values, adjacent pixels

| 0x000000C0 | 0x000000C0 | 0x000000C0 | 0x000000C0 | ... |

**Narrow Values**: small values stored in a big data type

| 0x000000C0 | 0x000000C8 | 0x000000D0 | 0x000000D8 | ... |

**Other Patterns**: pointers to the same memory region

| 0xC04039C0 | 0xC04039C8 | 0xC04039D0 | 0xC04039D8 | ... |
Low Dynamic Range:

Differences between values are significantly smaller than the values themselves
Key Idea: Base+Delta (B+Δ) Encoding

- **Fast Decompression:** vector addition
- **Simple Hardware:** arithmetic and comparison
- **Effective:** good compression ratio

32-byte Uncompressed Cache Line

- 4 bytes
  - Base
- 20 bytes
  - 12-byte Compressed Cache Line
  - Fast Decompression:
  - Simple Hardware:
  - Effective:
Can We Do Better?

• Uncompressible cache line (with a single base):

| 0x09A40178 | 0x00000000 | 0x09A4A838 | 0x0000000B | ... |

• Key idea:
  Use more bases, e.g., two instead of one

• Pro:
  – More cache lines can be compressed

• Cons:
  – Unclear how to find these bases efficiently
  – Higher overhead (due to additional bases)
B+Δ with Multiple Arbitrary Bases

Compression Ratio

# of bases is fixed

2 bases – the best option based on evaluations
How to Find Two Bases Efficiently?

1. **First base** - first element in the cache line
   - ✓ Base+Delta part

2. **Second base** - implicit base of 0
   - ✓ Immediate part

Advantages over 2 arbitrary bases:
- Better compression ratio
- Simpler compression logic

**Base-Delta-Immediate (BΔI) Compression**
Average compression ratio is close, but BΔI is simpler
Outline

• Motivation & Background
• Shortcomings of Prior Work
• Base-Delta-Immediate: Key Idea
• **Base-Delta-Immediate: Implementation**
• Evaluation
• Conclusion
• Future Work
BΔI Implementation

• Decompressor Design
  – Low latency

• Compressor Design
  – Low cost and complexity

• BΔI Cache Organization
  – Modest complexity
BΔI Decompressor Design

Compressed Cache Line

Uncompressed Cache Line

Vector addition
BΔI Compressor Design

32-byte Uncompressed Cache Line

Compression Selection Logic (based on compr. size)

Compressed Cache Line
BΔI Compression Unit: 8-byte $B_0$ 1-byte $\Delta$

32-byte Uncompressed Cache Line

$B_0 = V_0 B_0$

Is every element within 1-byte range?

Yes

No
**BΔI Cache Organization**

**Tag Storage:**
- Conventional 2-way cache with 32-byte cache lines
  - Set 0
  - Tag 0, Tag 1
  - Way 0, Way 1
  - ... ...

**Data Storage:**
- 32 bytes
  - Data 0
  - ... ...
  - Data 1
  - ... ...

**BΔI: 4-way cache with 8-byte segmented data**

**Tag Storage:**
- 8 bytes
  - Set 0
  - Tag 0, Tag 1, Tag 2, Tag 3
  - ... ...

**Data Storage:**
- 8 bytes
  - S0, S1, S2, S3, S4, S5, S6, S7
  - ... ...

- Tags map to multiple adjacent segments
- 2.3% overhead for 2 MB cache

- Twice as many tags
- C - Compressed encoding bits
- Tags and data are 8 bytes wide for 2 MB caches
Outline

• Motivation & Background
• Shortcomings of Prior Work
• Base-Delta-Immediate: Key Idea
• Base-Delta-Immediate: Implementation
• Evaluation
• Conclusion
• Future Work
Methodology

• Simulator
  – x86 event-driven simulator based on Simics
    [Magnusson+, Computer’02]

• Workloads
  – SPEC2006 benchmarks, TPC, Apache web server
  – 1 – 4 core simulations for 1 billion representative instructions

• System Parameters
  – L1/L2/L3 cache latencies from CACTI [Thoziyoor+, ISCA’08]
  – 4GHz, x86 in-order core, cache size (512kB - 16MB), simple memory model (300-cycle latency for row-misses)
Methodology

• Simulator
  – x86 event-driven simulator based on Simics
    \[Magnusson+, Computer'02\]

• Workloads
  – SPEC2006 benchmarks, TPC, Apache web server
  – 1 – 4 core simulations for 1 billion representative instructions

• System Parameters
  – L1/L2/L3 cache latencies from CACTI \[Thoziyoor+, ISCA’08\]
  – 4GHz, x86 in-order core, cache size (512kB - 16MB), simple memory model (300-cycle latency for row-misses)
Key Metrics

• **IPC** – instructions per cycle
• **MPKI** – misses per kilo instruction
• **Compression ratio** – effective cache size increase, e.g., 1.5 for 2MB cache means effective size of 3MB

• **Weighted Speedup:**

\[
WS = \sum_{i=1}^{N} \frac{IPC_{i}^{shared}}{IPC_{i}^{alone}}
\]

\[
\begin{align*}
N & \text{ – number of cores} \\
IPC_{i}^{alone} & \text{ IPC of core } i \text{ running alone/shared in the system} \\
IPC_{i}^{shared} & \text{ IPC of core } i \text{ running alone/shared in the system}
\end{align*}
\]
**Compression Ratio: BΔI vs. Prior Work**

SPEC2006, databases, web workloads, 2MB L2

<table>
<thead>
<tr>
<th>Application</th>
<th>ZCA</th>
<th>FVC</th>
<th>FPC</th>
<th>BΔI</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wrf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hmmmer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sphinx3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpch17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lesie3d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gromacs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sjeng</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h264ref</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpch2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>omnetpp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xalancbmk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>astar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tpch6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cactusADM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soplex</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gobmk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zeusmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GemsFDTD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
<td></td>
<td>1.53</td>
</tr>
</tbody>
</table>

**BΔI** achieves the highest compression ratio
**Single-Core: IPC and MPKI**

**Normalized IPC**
- **Baseline (no compr.)**
- **BΔI**

<table>
<thead>
<tr>
<th>L2 cache size</th>
<th>Baseline (no compr.)</th>
<th>BΔI</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>512KB</td>
<td>0.9</td>
<td>0.2</td>
<td>8.1%</td>
<td>5.2%</td>
<td>5.1%</td>
</tr>
<tr>
<td>1MB</td>
<td>1.0</td>
<td>0.4</td>
<td>5.1%</td>
<td>4.9%</td>
<td>5.6%</td>
</tr>
<tr>
<td>2MB</td>
<td>1.1</td>
<td>0.6</td>
<td>4.9%</td>
<td>5.6%</td>
<td>3.6%</td>
</tr>
<tr>
<td>4MB</td>
<td>1.2</td>
<td>0.8</td>
<td>5.6%</td>
<td>3.6%</td>
<td>0.0%</td>
</tr>
<tr>
<td>8MB</td>
<td>1.3</td>
<td>1.0</td>
<td>3.6%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>16MB</td>
<td>1.4</td>
<td>1.2</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

**Normalized MPKI**
- **Baseline (no compr.)**
- **BΔI**

<table>
<thead>
<tr>
<th>L2 cache size</th>
<th>Baseline (no compr.)</th>
<th>BΔI</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>512KB</td>
<td>16%</td>
<td>0.2</td>
<td>16%</td>
<td>24%</td>
<td>21%</td>
</tr>
<tr>
<td>1MB</td>
<td>24%</td>
<td>0.4</td>
<td>24%</td>
<td>21%</td>
<td>13%</td>
</tr>
<tr>
<td>2MB</td>
<td>21%</td>
<td>0.6</td>
<td>21%</td>
<td>13%</td>
<td>19%</td>
</tr>
<tr>
<td>4MB</td>
<td>13%</td>
<td>0.8</td>
<td>13%</td>
<td>19%</td>
<td>14%</td>
</tr>
<tr>
<td>8MB</td>
<td>19%</td>
<td>1.0</td>
<td>19%</td>
<td>14%</td>
<td>0.0%</td>
</tr>
<tr>
<td>16MB</td>
<td>14%</td>
<td>1.2</td>
<td>14%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

**BΔI** achieves the performance of a 2X-size cache
Performance improves due to the decrease in MPKI
Multi-Core Workloads

• Application classification based on
  
  **Compressibility**: effective cache size increase  
  (Low Compr. \((LC) < 1.40\), High Compr. \((HC) \geq 1.40\))

  **Sensitivity**: performance (IPC) gain with more cache  
  (Low Sens. \((LS) < 1.10\), High Sens. \((HS) \geq 1.10\); 512kB -> 2MB)

• Three classes of applications:
  – LCLS, HCLS, HCHS, **no LCHS** applications

• For 2-core - **random** mixes of each possible class pairs  
  (20 each, 120 total workloads)
Multi-Core: Weighted Speedup

If at least one application is sensitive, then the performance improvement is the highest (9.5%).
Future Work

• New compression-aware replacement policies
  – Minimal-LRU
  – Cost analysis based on compressed size

• Hardware evaluation of the proposed changes
  – Verilog model

• Main memory (DRAM) compression
  – Increases capacity
  – Improves performance
  – Decreases bandwidth and energy consumption
Conclusion

• **Base-Delta-Immediate** compression - a new compression mechanism

• **Key insight**: 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** of both single-core and multi-core workloads
  - Outperforms state-of-the-art cache compression techniques: FVC and FPC
Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches

Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Todd C. Mowry

Phillip B. Gibbons*, Michael A. Kozuch*

Carnegie Mellon

*
Single-Core: Effect on Cache Capacity

**BΔI** achieves performance close to the upper bound
Good average compression ratio (1.40)
But some benchmarks have low compression ratio
## Multiprogrammed Workloads - I

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LCLS</td>
<td>gromacs</td>
<td>1.43 / L</td>
<td>L</td>
<td>hmer</td>
<td>1.03 / L</td>
<td>L</td>
<td>lbm</td>
<td>1.00 / L</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>sphinx</td>
<td>1.10 / L</td>
<td>L</td>
<td>tpch17</td>
<td>1.18 / L</td>
<td>L</td>
<td>wrf</td>
<td>1.01 / L</td>
<td>L</td>
</tr>
<tr>
<td>HCLS</td>
<td>apache</td>
<td>1.60 / H</td>
<td>L</td>
<td>zeusmp</td>
<td>1.99 / H</td>
<td>L</td>
<td>gcc</td>
<td>1.99 / H</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>sjeng</td>
<td>1.50 / H</td>
<td>L</td>
<td>tpch2</td>
<td>1.54 / H</td>
<td>L</td>
<td>tpch6</td>
<td>1.93 / H</td>
<td>L</td>
</tr>
<tr>
<td>HCHS</td>
<td>astar</td>
<td>1.74 / H</td>
<td>H</td>
<td>bzip2</td>
<td>1.60 / H</td>
<td>H</td>
<td>mcf</td>
<td>1.52 / H</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>soplex</td>
<td>1.99 / H</td>
<td>H</td>
<td>h264ref</td>
<td>1.52 / H</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>libquantum</td>
<td>1.25 / L</td>
<td>L</td>
<td>leslie3d</td>
<td>1.41 / L</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HCLS</td>
<td>GsFDTD</td>
<td>1.99 / H</td>
<td>L</td>
<td>gobmk</td>
<td>1.99 / H</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cactusADM</td>
<td>1.97 / H</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HCHS</td>
<td>xalanckm</td>
<td>1.61 / H</td>
<td>H</td>
<td>omnetpp</td>
<td>1.58 / H</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cache Compression Flow

Hit L1

Writeback Compress

Writeback Decompress

CPU

L1 Data Cache
Uncompressed

L2 Cache
Compressed

Memory
Uncompressed

Hit L2
Decompress

Compress

Miss

Miss

Miss
Example of Base+Delta Compression

- Narrow values (taken from h264ref):

```
0x00000000 0x0000000B 0x00000003 0x00000001 0x00000004 0x00000000 0x00000003 0x00000004
```

32-byte Uncompressed Cache Line

```
0x00000000 0x00 0x0B 0x03 0x01 0x04 0x00 0x03 0x04
```

12-byte Compressed Cache Line

Saved Space: 20 bytes