Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency

Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Todd C. Mowry

Phillip B. Gibbons, Michael A. Kozuch

Carnegie Mellon University

SAFARI
Executive Summary

- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Idea**: Compress data in main memory
- **Problem**: How to avoid inefficiency in address computation?
- **Solution**: Linearly Compressed Pages (LCP): fixed-size cache line granularity compression

1. Increases memory capacity (**62%** on average)
2. Decreases memory bandwidth consumption (**24%**)
3. Decreases memory energy consumption (**4.9%**)
4. Improves overall performance (**13.9%**)

Potential for Data Compression

Significant redundancy in in-memory data:

0x00000000 0x0000000B 0x00000003 0x00000004 ...

How can we exploit this redundancy?

• **Main memory compression** helps
• Provides effect of a larger memory without making it physically larger
Challenges in Main Memory Compression

1. Address Computation

2. Mapping and Fragmentation
Challenge 1: Address Computation

Uncompressed Page

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>0</th>
<th>64</th>
<th>128</th>
<th>(N-1)*64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_0$</td>
<td>$L_1$</td>
<td>$L_2$</td>
<td>...</td>
</tr>
</tbody>
</table>

Compressed Page

<table>
<thead>
<tr>
<th>Address Offset</th>
<th>0</th>
<th>?</th>
<th>?</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_0$</td>
<td>$L_1$</td>
<td>$L_2$</td>
<td>...</td>
</tr>
</tbody>
</table>
Challenge 2: Mapping & Fragmentation

Virtual Page
(4KB)

Physical Page
(? KB)

Virtual Address

Physical Address

Fragmentation
Outline

• Motivation & Challenges

• **Shortcomings of Prior Work**

• LCP: Key Idea

• LCP: Implementation

• Evaluation

• Conclusion and Future Work
### Key Parameters in Memory Compression

<table>
<thead>
<tr>
<th>Compression Ratio</th>
<th>Address Comp. Latency</th>
<th>Decompression Latency</th>
<th>Complexity and Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Shortcomings of Prior Work

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Compression Ratio</th>
<th>Address Comp. Latency</th>
<th>Decompression Latency</th>
<th>Complexity and Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM MXT [IBM J.R.D. ’01]</td>
<td>✔ ✔</td>
<td>✗ 2X</td>
<td>✗ 64 cycles</td>
<td>✗</td>
</tr>
</tbody>
</table>

- ✔: Yes, ✔: Yes, ✗: No
# Shortcomings of Prior Work (2)

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Compression Ratio</th>
<th>Address Comp. Latency</th>
<th>Decompression Latency</th>
<th>Complexity And Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM MXT [IBM J.R.D. ’01]</td>
<td>✔️ ✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Robust Main Memory Compression [ISCA’05]</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Shortcomings of Prior Work (3)

<table>
<thead>
<tr>
<th>Compression Mechanisms</th>
<th>Compression Ratio</th>
<th>Address Comp. Latency</th>
<th>Decompression Latency</th>
<th>Complexity And Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM MXT [IBM J.R.D. ’01]</td>
<td>√ √</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Robust Main Memory Compression [ISCA’05]</td>
<td>√</td>
<td>x</td>
<td>√</td>
<td>x</td>
</tr>
<tr>
<td>LCP: Our Proposal</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
Linearily Compressed Pages (LCP): Key Idea

Uncompressed Page (4KB: 64*64B)

LCP effectively solves challenge 1: address computation

- Fixed compressed size
- LCP effectively solves challenge 1: address computation

4:1 Compression

Compressed Data (1KB)

128
LCP: Key Idea (2)

Uncompressed Page (4KB: 64*64B)

4:1 Compression

Compressed Data (1KB)  Metadata (64B)  Exception Storage
But, wait …

Uncompressed Page (4KB: 64 * 64B)

4:1 Compression

Compressed Data (1KB)

How to avoid 2 accesses?

✔ Metadata (MD) cache
Key Ideas: Summary

✓ Fixed compressed size per cache line

✓ Metadata (MD) cache
Outline

• Motivation & Challenges
• Shortcomings of Prior Work
• LCP: Key Idea
  • LCP: Implementation
• Evaluation
• Conclusion and Future Work
LCP Overview

- Page Table entry extension
  - compression type and size (fixed encoding)
- OS support for multiple page sizes
  - 4 memory pools (512B, 1KB, 2KB, 4KB)
- Handling uncompressible data
- Hardware support
  - memory controller logic
  - metadata (MD) cache
Page Table Entry Extension

Page Table Entry

- **c-bit (1b)** – compressed or uncompressed page
- **c-type (3b)** – compression encoding used
- **c-size (2b)** – LCP size (e.g., 1KB)
- **c-base (3b)** – offset within a page
Physical Memory Layout

Page Table

$PA_0$

$PA_1$

$PA_2$

...
Memory Request Flow

1. *Initial Page Compression*

2. *Cache Line Read*

3. *Cache Line Writeback*
1. Initial Page Compression

2. Cache Line Read

3. Cache Line Writeback

Processor

Memory Controller

Last-Level Cache

Core ↔ TLB

Disk

4KB

1KB

2KB

DRAM

Compress/Decompress

MD Cache

$Line

$Line

$Line

LD

LD
Handling Page Overflows

- Happens after writebacks, when all slots in the exception storage are already taken

- Two possible scenarios:
  - Type-1 overflow: requires larger physical page size (e.g., 2KB instead of 1KB)
  - Type-2 overflow: requires decompression and full uncompressed physical page (e.g., 4KB)

Compressed Data

Exception Storage

Happens infrequently - once per ~2M instructions
Compression Algorithms

• Key requirements:
  • Low hardware complexity
  • Low decompression latency
  • High effective compression ratio

• Frequent Pattern Compression \cite{ISCA'04}
  • Uses simplified dictionary-based compression

• Base-Delta-Immediate Compression \cite{PACT'12}
  • Uses low-dynamic range in the data
Base-Delta Encoding \textit{[PACT’12]}

32-byte Uncompressed Cache Line

0xC04039C0 0xC04039C8 0xC04039D0 ... 0xC04039F8

0xC04039C0

Base

12-byte Compressed Cache Line

0x00 0x08 0x10 ... 0x38

BDI \textit{[PACT’12]} has two bases:

1. zero base (for narrow values)
2. arbitrary base (first non-zero value in the cache line)

Fast Decompression:
- vector addition

Simple Hardware:
- arithmetic and comparison

Effective:
- good compression ratio
LCP-Enabled Optimizations

- Memory bandwidth reduction:
  - 1 transfer instead of 4

- Zero pages and zero cache lines
  - Handled separately in TLB (1-bit) and in metadata (1-bit per cache line)
Outline

• Motivation & Challenges
• Shortcomings of Prior Work
• LCP: Key Idea
• LCP: Implementation

• Evaluation

• Conclusion and Future Work
Methodology

• Simulator: x86 event-driven based on Simics

• Workloads (32 applications)
  • SPEC2006 benchmarks, TPC, Apache web server

• System Parameters
  • L1/L2/L3 cache latencies from CACTI [Thoziyoor+, ISCA’08]
  • 512kB - 16MB L2 caches
  • DDR3-1066, 1 memory channel

• Metrics
  • Performance: Instructions per cycle, weighted speedup
  • Capacity: Effective compression ratio
  • Bandwidth: Bytes per kilo-instruction (BPKI)
  • Energy: Memory subsystem energy
## Evaluated Designs

<table>
<thead>
<tr>
<th>Design</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>Baseline (no compression)</td>
</tr>
<tr>
<td>RMC</td>
<td>Robust main memory compression (RMC) and FPC&lt;sup&gt;[ISCA’05]&lt;/sup&gt;</td>
</tr>
<tr>
<td>LCP-FPC</td>
<td>LCP framework with FPC</td>
</tr>
<tr>
<td>LCP-BDI</td>
<td>LCP framework with BDI&lt;sup&gt;[PACT’12]&lt;/sup&gt;</td>
</tr>
<tr>
<td>LZ</td>
<td>Lempel-Ziv compression (per page)</td>
</tr>
</tbody>
</table>
Effect on Memory Capacity
32 SPEC2006, databases, web workloads, 2MB L2 cache

LCP-based designs achieve competitive average compression ratios with prior work
Effect on Bus Bandwidth

32 SPEC2006, databases, web workloads, 2MB L2 cache

LCP-based designs significantly reduce bandwidth (24%) (due to data compression)
Effect on Performance

LCP-based designs significantly improve performance over RMC.
Effect on Memory Subsystem Energy

32 SPEC2006, databases, web workloads, 2MB L2 cache

LCP framework is more energy efficient than RMC
Effect on Page Faults

32 SPEC2006, databases, web workloads, 2MB L2 cache

LCP framework significantly decreases the number of page faults (up to 23% on average for 768MB)
Other Results and Analyses in the Paper

• Analysis of page overflows
• Compressed page size distribution
• Compression ratio over time
• Number of exceptions (per page)
• Detailed single-/multicore evaluation
• Comparison with stride prefetching
  • performance and bandwidth
Conclusion

- **Old Idea:** Compress data in main memory
- **Problem:** How to avoid inefficiency in address computation?
- **Solution:** A new main memory compression framework called LCP (Linearly Compressed Pages)
  - **Key idea:** fixed-size for compressed cache lines within a page
- **Evaluation:**
  1. Increases memory capacity (62% on average)
  2. Decreases bandwidth consumption (24%)
  3. Decreases memory energy consumption (4.9%)
  4. Improves overall performance (13.9%)
Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency

Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Todd C. Mowry

Phillip B. Gibbons, Michael A. Kozuch

Carnegie Mellon University SAFARI
Backup Slides
Large Pages (e.g., 2MB or 1GB)

- Splitting large pages into smaller 4KB sub-pages (compressed individually)
- 64-byte metadata chunks for every sub-page

<table>
<thead>
<tr>
<th>2KB</th>
<th>2KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2KB</td>
<td>M</td>
</tr>
</tbody>
</table>
Physically Tagged Caches

Critical Path

Virtual Address

TLB

Address Translation

Physical Address

L2 Cache Lines

Core

tag  |  data
---|---
tag  |  data
>tag  |  data
Changes to Cache Tagging Logic

Before:
- **p-base** – physical page base address
- **c-idx** – cache line index within the page

After:
- **p-base**
- **c-idx**
Analysis of Page Overflows

Type-1 Overflows per instr. (log-scale)
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
GPGPU Evaluation

- Gpgpu-sim v3.x
- Card: NVIDIA GeForce GTX 480 (Fermi)
- Caches:
  - DL1: 16 KB with 128B lines
  - L2: 786 KB with 128B lines
- Memory: GDDR5
Effect on Bandwidth Consumption

Normalized BPKI

<table>
<thead>
<tr>
<th></th>
<th>BDI</th>
<th>LCP-BDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPEG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sprv</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>backprop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotspot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InvIdx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bfs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dmr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sssp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CUDA Parboil Rodinia Mars Lonestar
Effect on Throughput

Normalized Performance

Baseline

BDI

BFS | MUM | JPEG | NN | LPS | STO | CONS | SCP

CUDA | Parboil | Rodinia | Mars | Lonestar | GeoMean

GeoMean
Physical Memory Layout

Page Table

\[ PA_0 \]
\[ PA_1 \]
\[ PA_2 \]

\[ c\text{-base} \]

\[ PA_1 + 512\times 4 \]

\[ PA_2 + 512\times 1 \]
Page Size Distribution
Compression Ratio Over Time

[Graph showing compression ratio over time for different methods: zeusmp, cactusADM, astar, h264ref]
IPC (1-core)
Weighted Speedup

![Bar chart showing performance improvement with different numbers of cores. The chart indicates that LCP-BDI performs better as the number of cores increases.](image)
Bandwidth Consumption

![Normalized BPKI Graph](image)
Page Overflows

Type-1 Overflows per instr. (log-scale)

- apache
- astar
- bzip2
- cactusADM
- calculix
- dealii
- gamess
- gcc
- GmresFDTD
- gohmkl
- gromacs
- h264ref
- hammer
- ibm
- leslie3d
- libquantum
- mcf
- milc
- namd
- omnetpp
- perlbench
- povray
- sjeng
- soplex
- sphinx3
- tpcce
- tpcche17
- tpcche2
- tpcche6
- wrf
- xalancbmk
- zeusmp
- GeoMean
Stride Prefetching - IPC

![Graph showing normalized IPC for various benchmarks with different prefetching techniques.](image-url)
Stride Prefetching - Bandwidth

Normalized BPKI

- Stride Prefetching
- LCP-BDI
- LCP-BDI + Prefetching hints

[Bar chart showing normalized BPKI for various benchmarks with different prefetching methods]