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

- Address Offset: 0, 64, 128, (N-1)*64
- Cache Line (64B)

Compressed Page

- Address Offset: 0, ?, ?, ?
Challenge 2: Mapping & Fragmentation

Virtual Page (4KB) → Physical Page (? KB) → Physical Address

Virtual 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>
<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>

IBM MXT [IBM J.R.D. ’01]
## 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>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>
</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>☒</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Robust Main Memory Compression [ISCA’05]</td>
<td>☑</td>
<td>☒</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>LCP: Our Proposal</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>
Linearly Compressed Pages (LCP): Key Idea

Uncompressed Page (4KB: 64*64B)

- Compressed Data (1KB)
- 4:1 Compression
- Fixed compressed size
- LCP effectively solves challenge 1: address computation
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)

64B  64B  64B  64B  ...  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

*PTE*
Page Table Entry Extension

- **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

\[ PA_0 \]

\[ PA_1 \]

\[ PA_2 \]

\[ PA_1 + 512\times 4 \]

\[ PA_2 + 512\times 1 \]
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
Handling Page Overflows

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

<table>
<thead>
<tr>
<th>...</th>
<th>M</th>
<th>E₀</th>
<th>E₁</th>
<th>E₂</th>
</tr>
</thead>
</table>

Compressed Data

Exception Storage

• 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)

Happens infrequently - once per ~2M instructions
Compression Algorithms

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

• Frequent Pattern Compression [ISCA’04]
  • Uses simplified dictionary-based compression

• Base-Delta-Immediate Compression [PACT’12]
  • Uses low-dynamic range in the data
Base-Delta Encoding \([PACT'12]\)

- 32-byte Uncompressed Cache Line
  - \(0x\text{C04039C0} \rightarrow 0x\text{C04039C8} \rightarrow 0x\text{C04039D0} \rightarrow \ldots \rightarrow 0x\text{C04039F8}\)

- 12-byte Compressed Cache Line
  - \(0x\text{00} \rightarrow 0x\text{08} \rightarrow 0x\text{10} \rightarrow \ldots \rightarrow 0x\text{38}\)

BDI \([PACT'12]\) 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:
  
  
  64B   64B   64B   64B

  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 \cite{ISCA05} (RMC) and FPC \cite{ISCA04}</td>
</tr>
<tr>
<td>LCP-FPC</td>
<td>LCP framework with FPC</td>
</tr>
<tr>
<td>LCP-BDI</td>
<td>LCP framework with BDI \cite{PACT12}</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)
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>
<th>2KB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2KB</td>
<td>M</td>
<td>2KB</td>
</tr>
</tbody>
</table>
Physically Tagged Caches

Critical Path

Virtual Address

Address Translation

Physical Address

Core

TLB

L2 Cache Lines

tag data
tag data
tag data
Changes to Cache Tagging Logic

- **Before:**
  - **tag**
  - **data**
  - **tag**
  - **data**
  - **tag**
  - **data**

- **After:**
  - **p-base**
  - **c-idx**

- **p-base** – physical page base address
- **c-idx** – cache line index within the page
Analysis of Page Overflows

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

- apache
- astar
- bzip2
- cactusADM
- gcc
- GemsFDTD
- gromacs
- h264ref
- lbm
- lesie3d
- libquantum
- mcf
- omnetpp
- perlbench
- sjeng
- soplex
- sphinx3
- tpcc
- tpch6
- xalancbmk
- zeusmp
- GeoMean

1.0E-08
1.0E-07
1.0E-06
1.0E-05
1.0E-04
1.0E-03
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

![Bandwidth Consumption Graph](image)

- **BDI**: Blue bars
- **LCP-BDI**: Red bars

### Normalized BPKI

<table>
<thead>
<tr>
<th>CUDA</th>
<th>Parboil</th>
<th>Rodinia</th>
<th>Mars</th>
<th>Lonestar</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>sprv</td>
<td>backprop</td>
<td>PVC</td>
<td>bfs</td>
</tr>
<tr>
<td>MUM</td>
<td>sad</td>
<td>hotspot</td>
<td>PVR</td>
<td>bh</td>
</tr>
<tr>
<td>JPEG</td>
<td></td>
<td>streamcluster</td>
<td>lnvdx</td>
<td>dmr</td>
</tr>
<tr>
<td>NN</td>
<td></td>
<td></td>
<td>SS</td>
<td>mst</td>
</tr>
<tr>
<td>LPS</td>
<td></td>
<td></td>
<td></td>
<td>sp</td>
</tr>
<tr>
<td>STO</td>
<td></td>
<td></td>
<td></td>
<td>sssp</td>
</tr>
<tr>
<td>CONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spmv</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hotspot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

44
Effect on Throughput

Normalized Performance

Baseline  BDI

BFS  MUM  JPEG  NN  LPS  STO  CONS  SCP
spmv  sad  backprop  hotspot  streamcluster
PVC  PVR  Invidx  SS
bfs  bh  dmr  mst  sp  sssp  GeoMean

Baseline
BDI

CUDA  Parboil  Rodinia  Mars  Lonestar
Physical Memory Layout

Page Table

- $PA_0$
- $PA_1$ + 512*4
- $PA_2$ + 512*1
Page Size Distribution
Compression Ratio Over Time

![Graph showing compression ratio over time with different markers for zeusmp, cactusADM, astar, and h264ref, indicating how compression ratio changes with the number of instructions.]
IPC (1-core)
Weighted Speedup

![Performance Improvement Graph]

- **Performance Improvement, %**
  - 0%
  - 5%
  - 10%
  - 15%

- **Number of Cores**
  - 1
  - 2
  - 4

- **Legend**: LCP-BDI
Bandwidth Consumption

- RMC-FPC
- LCP-FPC
- LCP-BDI

Normalized BPKI

- apache
- astar
- cactusADM
- calculix
- dealii
- gamess
- gcc
- Gems-DTB
- gromacs
- h264ref
- hmmer
- ibm
- libquantum
- mcf
- milc
- namd
- omnetpp
- perlbench
- povray
- soplex
- sphinx3
- tpc-c
- tpc-c17
- tpc-c2
- tpc-c6
- wrf
- xalanbmk
- zeusmp
- GeoMean
Page Overflows
Stride Prefetching - IPC

Normalized IPC

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

Samples include: apache, astar, bzip2, cactusADM, calculix, dealii, games, gcc, GemsFDTD, gromacs, h264ref, himmer, ibm, leslie3d, libquantum, mcf, milc, namd, omnetpp, perlbench, povray, sjeng, soplex, sphinx3, tpc, tpcch17, tpcch2, tpcch5, wrf, xalanbmk, zeusmp, GeoMean

Values:
- Stride Prefetching: 1.57, 1.54
- LCP-BDI: 1.91
- LCP-BDI + Prefetching hints: 1.43, 1.68
Stride Prefetching - Bandwidth

[Bar chart showing normalized BPKI for various benchmarks, comparing Stride Prefetching, LCP-BDI, and LCP-BDI + Prefetching hints.]