ME-HPTs:
Memory-Efficient Hashed Page Tables

HPCA 2023

Jovan Stojkovic, Namrata Mantri, Dimitrios Skarlatos*, Tianyin Xu, Josep Torrellas

University of Illinois at Urbana-Champaign

*Carnegie Mellon University
Virtual Memory and Page Tables

- Virtual memory is an essential technique in modern computing systems
  - Memory virtualization
  - Process isolation
- Virtual memory performance depends on the page table organization
  - Radix page tables – slow and not scalable
  - Hashed page tables – memory inefficient
Radix Page Tables: Memory-Efficient Multi-Level Trees

L1- PGD  L2- PUD  L3- PMD  L4- PTE
Radix Page Walk: Expensive Pointer Chase

x86-64 Radix Page Tables

Address A

47 … 39  38 … 30  29 … 21  20 … 12  11 … 0

Virtual Address

9-bits  9-bits  9-bits  9-bits  Page Offset

CR3 ➔ pgd ➔ PGD
CR3 ➔ pud ➔ PUD
CR3 ➔ pmd ➔ PMD
CR3 ➔ pte ➔ PTE

TLB Entry

Expensive Pointer Chase
Hashed Page Tables

Page walk requires a single memory access

PGD  PUD  PMD  PTE

HASH

Hashed Page Table
Hashed Page Tables

Hash collisions

PGD ─ PUD ─ PMD ─ PTE-A

HASH

PGD ─ PUD ─ PMD ─ PTE-B

Hashed Page Table
Hashed Page Tables: Recent Advances Make Them Compelling

Elastic Cuckoo Page Tables (ECPTs)
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert E

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hashed Page Table Way 0

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hashed Page Table Way 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>

Hashed Page Table Way 2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G</td>
</tr>
</tbody>
</table>
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2

Insert E
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert E

H₀(E)

- Hashed Page Table Way 0
  - D
  - F

- Hashed Page Table Way 1
  - C
  - A

- Hashed Page Table Way 2
  - B
  - G
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

- Hashed Page Table Way 0
  - D
  - E

- Hashed Page Table Way 1
  - C
  - A

- Hashed Page Table Way 2
  - B
  - G
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert F

Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert F

H2(F)

D
E
Hashed Page Table Way 0

C
A
Hashed Page Table Way 1

B
G
Hashed Page Table Way 2
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert F

H2(F)

D
E

H2 (F)

C
A

B
G

Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Insert G

Hashed Page Table Way 0
D
E

Hashed Page Table Way 1
C
A

Hashed Page Table Way 2
B
G
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

Hashed Page Table Way 0

Hashed Page Table Way 1

Hashed Page Table Way 2

Insert G

H1 (G)
Hashed Page Tables: Recent Advances Make Them Compelling

Cuckoo Hashing

<table>
<thead>
<tr>
<th>Hashed Page Table Way 0</th>
<th>Hashed Page Table Way 1</th>
<th>Hashed Page Table Way 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Outline of this talk

- **Problem: Contiguous Memory Requirements of Hashed Page Tables**
- ME-HPTs: Memory-Efficient Hashed Page Tables
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion
Hashed Page Tables: Large Contiguous Memory Chunks

- With hashed page tables – unity of allocation is one way of the page table

<table>
<thead>
<tr>
<th></th>
<th>Hashed Page Table</th>
<th>Hashed Page Table</th>
<th>Hashed Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Way 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Way 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Way 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hashed Page Tables: Large Contiguous Memory Chunks

- With hashed page tables – unity of allocation is one way of the page table

- With large memory applications, size of a way can be 10s-100s of MBs!

  e.g., GUPS, SysBench 64MB per way
Hashed Page Tables: Contiguity is Expensive!

- Finding large contiguous memory chunks is expensive in busy fragmented servers

---

Linux server
2GHz
0.7 FMFI
Hashed Page Tables: Contiguity is Expensive!

- Finding large contiguous memory chunks is expensive in busy fragmented servers.

Applications need to stall for millions of cycles for allocation!

With higher fragmentation, the system even fails to allocate 64MB chunks!

Linux server
2GHz
0.7 FMFI
Contributions

- Four novel architectural techniques to provide Memory-Efficient Hashed Page Tables (ME-HPTs)
- Reduced memory contiguity requirement by 92%
- Sped-up applications by 9% on average
- Allow large-memory applications to run at high performance on highly fragmented servers
Outline of this talk

- Problem: Contiguous Memory Requirements of Hashed Page Tables
- **ME-HPTs: Memory-Efficient Hashed Page Tables**
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion
Memory-Efficient Hashed Page Tables: ME-HPTs Design Overview

- Memory-Efficient Hashed Page Tables (ME-HPTs): Four novel architectural techniques
  - Directly minimizing contiguity requirements
    - Logical-to-Physical (L2P) Table
    - Dynamically Changing Chunk Size
  - Indirectly minimizing contiguity requirements by minimizing memory consumption
    - In-place Page Table Resizing
    - Per-way Page Table Resizing
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table
Memory Efficient Hashed Page Tables: Logical-to-Physical (L2P) Table
Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes

L2P is fixed in size! What if the application needs more memory?
Memory Efficient Hashed Page Tables: Dynamically Changing Chunk Sizes
Memory Efficient Hashed Page Tables: Design Overview

- ME-HPTs: Four novel architectural techniques
- Directly minimizing contiguity requirements
  - Logical-to-Physical (L2P) Table
  - Dynamically Changing Chunk Size
- Indirectly minimizing contiguity requirements by minimizing memory consumption
  - In-place Page Table Resizing
  - Per-way Page Table Resizing
Baseline Page Table Resizing

Old HPT

Chunk

Chunk

New HPT

Chunk

Chunk

Chunk

Chunk
Baseline Page Table Resizing

Old HPT

New HPT

PPN
PPN
PPN
PPN
PPN
Baseline Page Table Resizing

Old HPT

New HPT

PPN

PPN

PPN

PPN
Baseline Page Table Resizing

Old HPT

New HPT

PPN

PPN

PPN
Baseline Page Table Resizing

<table>
<thead>
<tr>
<th>Old HPT</th>
<th>New HPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPN</td>
<td>PPN</td>
</tr>
<tr>
<td>PPN</td>
<td>PPN</td>
</tr>
<tr>
<td>PPN</td>
<td>PPN</td>
</tr>
</tbody>
</table>
Baseline Page Table Resizing

Old HPT

New HPT

PPN

PPN

PPN

PPN
Baseline Page Table Resizing

Until the old table is deallocated, we keep both tables in memory!
Memory Efficient Hashed Page Tables: In-Place Page Table Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks
Memory Efficient Hashed Page Tables: In-Place Page Table Table Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks
Memory Efficient Hashed Page Tables: In-Place Page Table Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks
Memory Efficient Hashed Page Tables: In-Place Page Table Resizing

- Keep both tables in shared memory space
- Same hash function for both tables
- On rehash, some entries stay in the same chunk, others move to new chunks
Memory Efficient Hashed Page Tables: In-Place Page Table Table Resizing

Old HPT

New HPT

Old + New

Max(Old, New)
Always consume max(old, new) instead of sum(old, new)!

Save energy by moving only half of the entries!
Baseline Page Table Resizing

Way 0

Way 1

Way 2
Baseline Page Table Resizing

Often memory underutilized!
Memory Efficient Hashed Page Tables: Per Way Page Table Resizing

Memory much better utilized!

Need to care: where to insert new element and which way to upsize!
ME-HPTs Implementation: Hiding the L2P Table Access Latency

- Elastic Cuckoo Page Tables (ECPTs) use
  - Cuckoo walk caches (CWCs) to prune the number of parallel requests
  - Rehash Pointers to decide if a new or old HPT needs to accessed

- **Access L2P table in parallel and later choose the needed address**
Outline of this talk

- Page Table Organizations
- Hashed Page Tables Memory Requirements
- **ME-HPTs: Memory-Efficient Hashed Page Tables**
  - ME-HPTs Design
  - ME-HPTs Key Results
- Conclusion
Significant Memory Contiguity Savings

Required Contiguous Memory [MB]

- Elastic Cuckoo Page Tables
- Memory-Efficient Hashed Page Tables

64X reduction

BC BFS CC DC DFS GUPS MUMmer PR SSSP SysBench TC Average
Significant Memory Contiguity Savings

ME-HPTs reduce contiguity of ECPTs by up to 64X, with 92% average!
Improved Application Performance

9% speedup
Improved Application Performance

ME-HPTs outperform ECPTs by 3-18% with an average of 9%
Conclusion

- Four novel architectural techniques to provide Memory-Efficient Hashed Page Tables
  - L2P Table
  - Dynamically Changing Chunk Sizes
  - In-Place Page Table Resizing
  - Per-Way Page Table Resizing
- Reduced memory contiguity requirement by 92%
- Sped-up applications by 9% on average
- Allow large-memory applications to run at high performance on highly fragmented servers
ME-HPTs: Memory-Efficient Hashed Page Tables

HPCA 2023

Jovan Stojkovic, Namrata Mantri, Dimitrios Skarlatos*, Tianyin Xu, Josep Torrellas

University of Illinois at Urbana-Champaign

*Carnegie Mellon University
ME-HPTs Key Results:
Significant Memory Contiguity Savings

92% reduction without THP
ME-HPTs Key Results: Significant Memory Contiguity Savings

89% reduction with THP
ME-HPTs Key Results: Memory Consumption Reduction

Fig. 10: Reduction in page table memory attained by ME-HPT over the ECPT baseline. The number on top of each bar is the absolute reduction in Mbytes.
ME-HPTs Key Results:
Number of L2P Table Entries Used per App

Fig. 14: Number of L2P table entries used per application.
ME-HPTs Other Use Cases

- Techniques applicable to various hash table designs beyond HPTs

- **Scalable Secure Directories**
  - Directories as set-associative structures
  - Efficient resizing required

- **Memory Indexing**
  - Hash tables commonly used to implement memory indices of databases, file systems...
  - Dynamic resizing key operation: in-place resizing useful

- **Key-value Stores**
  - Dynamic structures whose size is unknown ahead of time