Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh
Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, Onur Mutlu
Executive Summary

• **Our Goal:** Accelerating pointer chasing inside main memory

• **Challenges:** Parallelism challenge and Address translation challenge

• **Our Solution:** In-Memory Pointer Chasing Accelerator (IMPICA)
  • Address-access decoupling: enabling parallelism in the accelerator with low cost
  • IMPICA page table: low cost page table structure

• **Key Results:**
  • 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  • 6% - 41% reduction in energy consumption
Linked Data Structures

• Linked data structures are widely used in many important applications
Linked Data Structures

• Linked data structures are widely used in many important applications
Linked Data Structures

- Linked data structures are widely used in many important applications

![Database Diagram]

![B-Tree Diagram]
Linked Data Structures

• Linked data structures are widely used in many important applications

Database

Key-value stores

B-Tree
Linked Data Structures

• Linked data structures are widely used in many important applications.
Linked Data Structures

• Linked data structures are widely used in many important applications

Linked data structures are connected by pointers

B-Tree

Hash Table
The Problem: Pointer Chasing

• Traversing linked data structures requires chasing pointers
The Problem: Pointer Chasing

• Traversing linked data structures requires chasing pointers

Find(A)
The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers

Find(A)

- H
- E
- Q
- A
- F
- M

 addr (H)

 Data (H)

 MEM

 CPU
The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers

**Find(A)**

```
H
/   \
E   Q
\   /
A   F M
```

```
MEM
```

```
CPU
```

```
Addr (E)
```

```
Data (E)
```
The Problem: Pointer Chasing

• Traversing linked data structures requires chasing pointers

Find(A)

- H
- E
- Q
- A
- F
- M

MEM

CPU

Addr (A)

Data (A)
The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers

Find(A)

Serialized and irregular access pattern
6X cycles per instruction in real workloads
Our Goal
Our Goal

Accelerating pointer chasing inside main memory
Our Goal

Accelerating pointer chasing inside main memory
Our Goal

Accelerating pointer chasing inside main memory
Our Goal

Accelerating pointer chasing inside main memory
Our Goal

Accelerating pointer chasing inside main memory

Find(A)

H
E
Q
A
F
M

CPU

DRAM layers

Logic layer
Our Goal

Accelerating pointer chasing inside main memory

Find(A)

H
E
Q
A
F
M

DRAM layers

Logic layer

CPU

Find (A)

Data (A)
Outline

• Motivation and Our Approach
• Parallelism Challenge
• IMPICA Core Architecture
• Address Translation Challenge
• IMPICA Page Table
• Evaluation
• Conclusion
Parallelism Challenge
Parallelism Challenge

Time

CPU core
Parallelism Challenge

CPU core → Comp → Memory access → Comp → Time
Parallelism Challenge

CPU core → Memory access → Comp

In-Memory Accelerator
Parallelism Challenge

Time

CPU core

Comp

Memory access

Comp

In-Memory Accelerator

Comp

Memory access

Comp
Parallelism Challenge

Time

CPU core

Comp  Memory access  Comp

In-Memory Accelerator

Comp  Memory access  Comp

Faster for one operation
Parallelism Challenge

- CPU core
  - Compute
  - Memory access
  - Compute

- CPU core

- In-Memory Accelerator
  - Compute
  - Memory access
  - Compute
Parallelism Challenge

Time

CPU core

Comp

Memory access

Comp

CPU core

Comp

Memory access

Comp

In-Memory Accelerator

Comp

Memory access

Comp
Parallelism Challenge

- CPU core
  - Comp
  - Memory access
  - Comp

- CPU core
  - Comp
  - Memory access
  - Comp

- In-Memory Accelerator
  - Comp
  - Memory access
  - Comp
  - Comp
  - Memory access
  - Comp
Parallelism Challenge

In-Memory Accelerator

CPU core

Comp | Memory access | Comp

Comp | Memory access | Comp

Time

Slower for two operations
Parallelism Challenge and Opportunity

• A simple in-memory accelerator can still be slower than multiple CPU cores
Parallelism Challenge and Opportunity

• A simple in-memory accelerator can still be **slower** than multiple CPU cores
Parallelism Challenge and Opportunity

• A simple in-memory accelerator can still be **slower** than multiple CPU cores

• **Opportunity**: a pointer-chasing accelerator spends a long time waiting for memory
Parallelism Challenge and Opportunity

• A simple in-memory accelerator can still be slower than multiple CPU cores

• Opportunity: a pointer-chasing accelerator spends a long time waiting for memory
Our Solution: Address-Access Decoupling
Our Solution: Address-Access Decoupling
Our Solution: Address-Access Decoupling

Time

CPU core

CPU core

Address Engine

Comp Memory access Comp

Comp Memory access Comp
Our Solution: Address-Access Decoupling

Time

CPU core

Comp

Memory access

Comp

CPU core

Comp

Memory access

Comp

Address Engine

Access Engine
Our Solution: Address-Access Decoupling

Time

CPU core

Comp

Memory access

Comp

CPU core

Comp

Memory access

Comp

Address Engine

Comp

Access Engine

Memory access
Our Solution: Address-Access Decoupling

- CPU core
  - Comp
  - Memory access
  - Comp

- Address Engine
  - Comp
  - Comp

- Access Engine
  - Memory access
  - Memory access
Our Solution: Address-Access Decoupling

Time

CPU core

Comp

Memory access

Comp

Comp

CPU core

Comp

Memory access

Comp

Comp

Address Engine

Comp

Comp

Access Engine

Memory access

Memory access

Memory access
Our Solution: Address-Access Decoupling
Our Solution: Address-Access Decoupling
Our Solution: Address-Access Decoupling

Address-access decoupling enables parallelism in both engines with low cost
IMPICA Core Architecture

DRAM Layers

Logic Layer

Memory Controller
IMPICA Core Architecture

DRAM

DRAM Layers

Logic Layer

Address Engine

Memory Controller
IMPICA Core Architecture

DRAM Layers

Logic Layer

DRAM

Memory Controller

Address Engine

Access Engine
IMPICA Core Architecture

DRAM Layers

Logic Layer

DRAM

Memory Controller

Address Engine

Access Queue

Access Engine
IMPICA Core Architecture

DRAM

DRAM Layers

Logic Layer

Memory Controller

Access Queue

Access Engine

Address Engine

Response Queue
IMPICA Core Architecture

DRAM Layers

Logic Layer

DRAM

IMPICA Core Architecture Diagram:
- DRAM
- Access Engine
- Address Engine
- IMPICA Cache
- Access Queue
- Response Queue
- Memory Controller
IMPICA Core Architecture

DRAM

DRAM Layers

Logic Layer

IMPICA Cache

Access Queue

Address Engine

Request Queue

To/From CPU

Access Engine

Response Queue

Memory Controller
IMPICA Core Architecture

DRAM Layers

Logic Layer

DRAM

Memory Controller

IMPICA Cache

Address Engine

Access Queue

Response Queue

Request Queue

To / From CPU

Traversal 1

Traversal 2

Access Engine
Outline

• Motivation and Our Approach
• Parallelism Challenge
• IMPICA Core Architecture
• Address Translation Challenge
• IMPICA Page Table
• Evaluation
• Conclusion
Address Translation Challenge

Page table walk
Address Translation Challenge

No TLB/MMU on the memory side
Duplicating it is costly and creates compatibility issue

Page table walk
The page table walk requires multiple memory accesses
Our Solution: IMPICA Page Table

• Completely decouple the page table of IMPICA from the page table of the CPUs
Our Solution: IMPICA Page Table

• Completely **decouple the page table of IMPICA from the page table of the CPUs**

![Diagram showing CPU Page Table, Virtual Address Space, and Physical Address Space](image_url)
Our Solution: IMPICA Page Table

- Completely **decouple the page table of IMPICA** from the page table of the CPUs
Our Solution: IMPICA Page Table

- Completely **decouple the page table of IMPICA from the page table of the CPUs**

![Diagram showing decoupling of virtual and physical page tables]

Virtual Address Space | Physical Address Space
---|---
Virtual Page | Physical Page
Virtual Page | Physical Page
Our Solution: IMPICA Page Table

• Completely **decouple the page table of IMPICA from the page table of the CPUs**

![IMPICA Page Table Diagram]
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs
Our Solution: IMPICA Page Table

• Completely **decouple the page table of IMPICA** from the page table of the CPUs.
Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs

IMPICA Page Table

Map linked data structure into IMPICA regions

IMPICA page table is a partial-to-any mapping
## IMPICA Page Table: Mechanism

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Region Table</td>
<td>Flat Page Table (2MB)</td>
<td>Small Page Table (4KB)</td>
<td>Physical Address</td>
</tr>
</tbody>
</table>

The diagram illustrates the process of converting a virtual address to a physical address through the IMPICA page table mechanism. The virtual address is divided into segments, each of which is associated with a different page table. The final physical address is obtained by concatenating the outputs of these tables.
IMPICA Page Table: Mechanism

Virtual Address

- Bit [47:41]
- Bit [40:21]
- Bit [20:12]
- Bit [11:0]

Region Table

- Flat Page Table (2MB)

- Tiny region table is almost always in the cache

- Small Page Table (4KB)

Physical Address
IMPICA Page Table: Mechanism

Virtual Address

Bit [47:41]

Region Table

Flat Page Table (2MB)

Flat page table saves one memory access

Bit [40:21]

[420x390] Bit [40:21]

Bit [20:12]

Bit [11:0]

Small Page Table (4KB)

Physical Address
Outline

• Motivation and Our Approach
• Parallelism Challenge
• IMPICA Core Architecture
• Address Translation Challenge
• IMPICA Page Table

• Evaluation
• Conclusion
Evaluated Workloads

• Microbenchmarks
  • Linked list (from Olden benchmark)
  • Hash table (from Memcached)
  • B-tree (from DBx1000)

• Application
  • DBx1000 (with TPC-C benchmark)
Evaluation Methodology

• Simulator: gem5

• System Configuration
  • CPU
    • 4 OoO cores, 2GHz
    • Cache: 32KB L1, 1MB L2
  • IMPICA
    • 1 core, 500MHz, 32KB Cache

• Memory Bandwidth
  • 12.8 GB/s for CPU, 51.2 GB/s for IMPICA

• Our simulator code will be released in Dec.
  • [https://github.com/CMU-SAFARI](https://github.com/CMU-SAFARI)
Result – Microbenchmark Performance

Baseline + extra 128KB L2

IMPICA

Linked List
Hash Table
B-Tree

Speedup
Result – Microbenchmark Performance

- Linked List
- Hash Table
- B-Tree

Baseline + extra 128KB L2
IMPICA

Speedup
Result – Microbenchmark Performance

- Linked List: Baseline + extra 128KB L2: 1.9X
- Hash Table: Baseline + extra 128KB L2: 1.3X
- B-Tree: Baseline + extra 128KB L2: 1.2X
Result – Database Performance
Result – Database Performance

![Database Performance Diagram]

Baseline + extra 128KB L2
Baseline + extra 1MB L2
IMPICA
Result – Database Performance

Database Throughput

Baseline + extra 128KB L2
Baseline + extra 1MB L2
IMPICA

+2%
Result – Database Performance

- Baseline + extra 128KB L2: +2%
- Baseline + extra 1MB L2: +5%
- IMPICA
Result – Database Performance

- Baseline + extra 128KB L2: +2%
- Baseline + extra 1MB L2: +5%
- IMPICA: +16%

Database Throughput

0.90 1.00 1.10 1.20

Baseline + extra 128KB L2  Baseline + extra 1MB L2  IMPICA
Result – Database Performance

**Database Throughput**
- Baseline + extra 128KB L2: +2%
- Baseline + extra 1MB L2: +5%
- IMPICA: +16%

**Database Latency**
- Baseline + extra 128KB L2
- Baseline + extra 1MB L2
- IMPICA
Result – Database Performance

Database Throughput

Baseline + extra 128KB L2
Baseline + extra 1MB L2
IMPICA

$+2\%$
$+5\%$
$+16\%$

Database Latency

Baseline + extra 128KB L2
Baseline + extra 1MB L2
IMPICA

$-0\%$
$-4\%$
$-13\%$
Energy Consumption
Energy Consumption

- Baseline + extra 128KB L2
- IMPICA

Normalized Energy

Linked List  Hash Table  B-Tree  DBx1000
Energy Consumption

Normalized Energy

Baseline + extra 128KB L2

IMPICA

Linked List
Hash Table
B-Tree
DBx1000

-41%
-24%
-10%
-6%
More in the Paper

• Interface and design considerations
  • CPU interface and programming model
  • Page table management
  • Cache coherence

• Area and power overhead analysis

• Sensitivity to IMPICA page table design
Conclusion

• Performing pointer-chasing inside main memory can greatly speed up the traversal of linked data structures

• Challenges: Parallelism challenge and Address translation challenge

• Our Solution: In-Memory Pointer Chasing Accelerator
  • Address-access decoupling: enabling parallelism with low cost
  • IMPICA page table: low cost page table structure

• Key Results:
  • 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  • 6% - 41% reduction in energy consumption

• Our solution can be applied to a broad class of in-memory accelerators
Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh
Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, Onur Mutlu

Carnegie Mellon University
University of Virginia
ETH Zürich
SAFARI
Microarchitecture Metrics
Microarchitecture Metrics

![Graph showing TLB MPKI for Linked List, Hash Table, and B-Tree]
Microarchitecture Metrics

![Graph showing TLB MPKI for Linked List, Hash Table, and B-Tree]
Microarchitecture Metrics

TLB MPKI

<table>
<thead>
<tr>
<th>Structure</th>
<th>Linked List</th>
<th>Hash Table</th>
<th>B-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPKI</td>
<td>150</td>
<td>50</td>
<td>0</td>
</tr>
</tbody>
</table>

Normalized cache miss latency

<table>
<thead>
<tr>
<th>Structure</th>
<th>Linked List</th>
<th>Hash Table</th>
<th>B-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>1.00</td>
<td>0.75</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Microarchitecture Metrics

TLB MPKI

Linked List  | Hash Table  | B-Tree
---|---|---

Normalized cache miss latency

Linked List  | Hash Table  | B-Tree
---|---|---

0.00  | 0.25  | 0.50  | 0.75  | 1.00
Sensitivity to IMPICA TLB size & Page Table Design

![Graph showing Address Translation Speedup for different TLB sizes and page table designs. The x-axis represents different data structures: Linked List, Hash Table, B-Tree, DBx1000. The y-axis represents Speedup. The graph compares 32 TLB, 64 TLB, 32 TLB + RPT, and 64 TLB + RPT.]
CPU Interface

• We use packet-based interface between CPU and IMPICA

• Execution steps
  • CPU sends function call and parameter to IMPICA
  • The packet is written to IMPICA data RAM
  • IMPICA loads the function into inst RAM
  • IMPICA writes results to the data RAM, from which the CPU polls the results.
Programming Model

• An IMPICA program is written as a function in the application code with a compiler directive

• The compiler compiles these functions into IMPICA instructions and wraps the function calls with communication codes
Page Table Management

• The application allocates the memory for its linked data structures with a special API

• The OS reserves a portion of the virtual address space as IMPICA regions

• The OS maintains the coherence between CPU page table and IMPICA page table in the page fault handler
IMPICA Page Table Size

• Region Table
  • 4 entries (covers a 2TB memory range)
  • 68 B

• Flat page table (each)
  • $2^{20}$ entries
  • 8 MB

• Small page table (each)
  • $2^9$ entries
  • 4 KB
Handling of Multiple Memory Stacks

• The OS knows the IMPICA region because of our page table management

• The OS always maps the IMPICA region of the same application into the same memory stack, including the corresponding IMPICA page table
Cache Coherence

• We execute every function that operates on the IMPICA regions in the accelerator

• It can be extended with more advanced cache coherence mechanism.
Limit of Parallelism

• The parallelism of IMPICA is limited by
  • Data RAM size (for call stacks)
  • Memory access time vs. address computation time
  • The size of the queues

• Each IMPICA core can easily parallelize 10 – 15 pointer chasing requests.
# Area and Power Overhead

<table>
<thead>
<tr>
<th>Component</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (Cortex-A57)</td>
<td>5.85 mm(^2) per core</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>5 mm(^2) per MB</td>
</tr>
<tr>
<td>Memory Controller</td>
<td>10 mm(^2)</td>
</tr>
<tr>
<td>IMPICA (+32KB cache)</td>
<td>0.45 mm(^2)</td>
</tr>
</tbody>
</table>

- **Power overhead**: average power increases by 5.6%