MetaSys

A Practical Open-Source Metadata Management System to Implement and Evaluate Cross-Layer Optimizations

Nandita Vijaykumar

Ataberk Olgun, Konstantinos Kanellopoulos, F. Nisa Bostanci

Hasan Hassan, Mehrshad Lotfi, Phillip B. Gibbons, Onur Mutlu

SAFARI

UNIVERSITY OF TORONTO

ETH Zürich

TOBB ETÜ
University of Economics & Technology

Executive Summary

Problem

- Cross-layer techniques are challenging to implement because they require full-stack changes
- Existing open-source infrastructures for implementing cross-layer techniques are not designed to provide key features:

Key Idea – Provide:

- Rich dynamic HW/SW interfaces
- Low-overhead metadata management
- Interfaces to key hardware components (e.g., prefetcher)

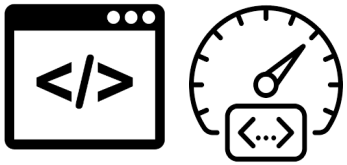
Our **goal** is twofold:

1. Develop an efficient and flexible framework to enable rapid implementation of new cross-layer techniques
2. Perform a detailed limit study to quantify the overheads associated with general metadata systems

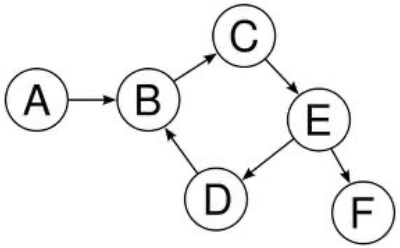
Outline

- **Background on Expressive Memory**
- MetaSys
  - Software Interface
  - Key Structures
- FPGA Implementation
- Evaluation

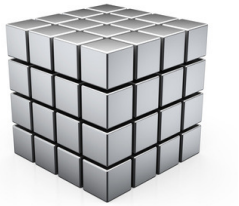
Higher-level information is not visible to HW



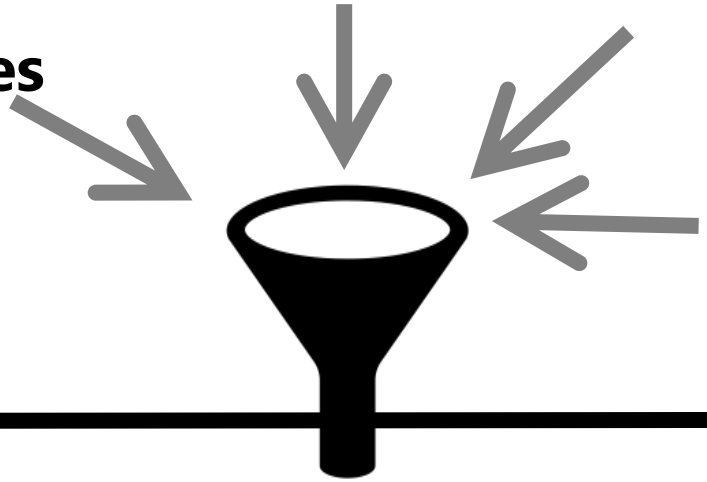
Code Optimizations



Access Patterns



Data Structures



Integer

Float

Data Type

Char

Software

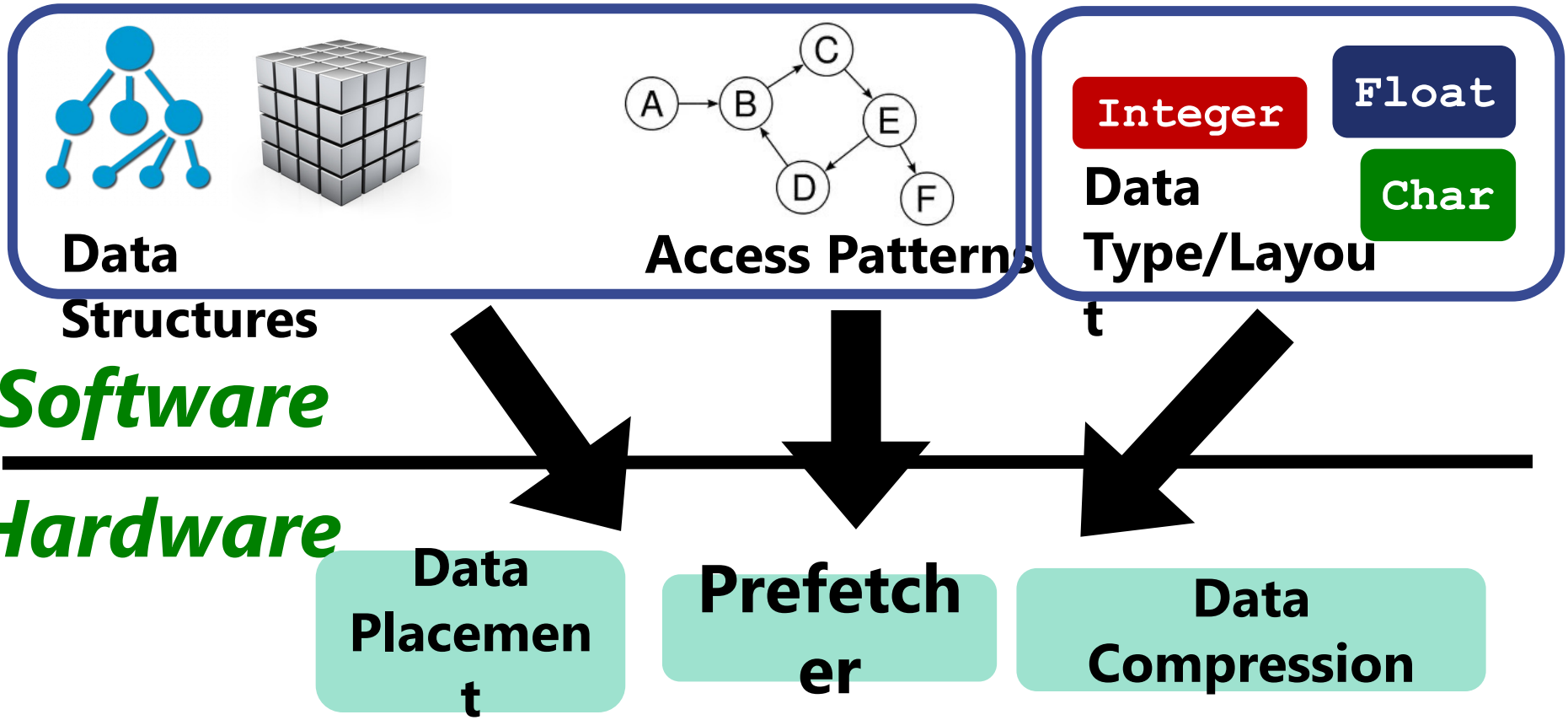
Hardware

100011111…
101010011…

Instructions
Memory Addresses

With a richer abstraction:

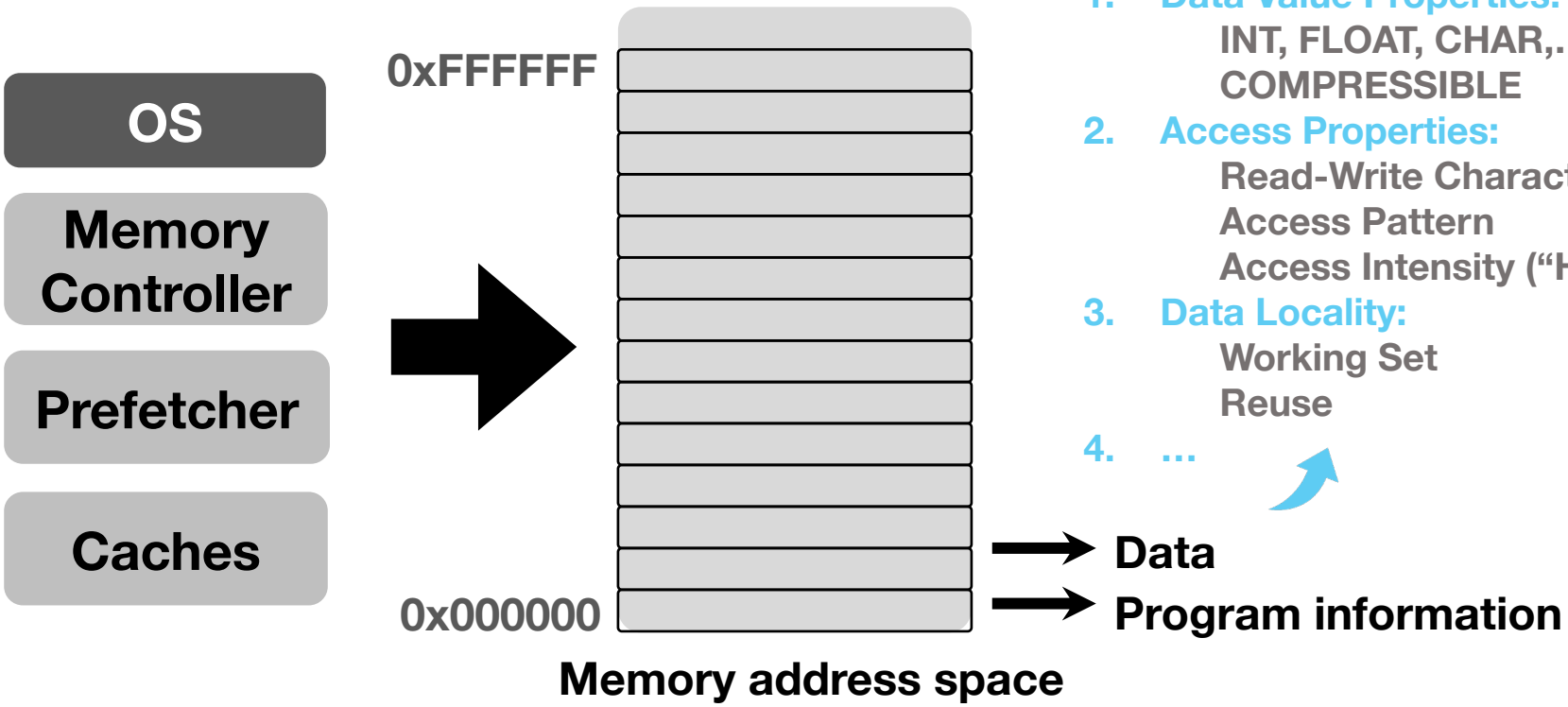
SW can provide program information can significantly help hardware



Outline

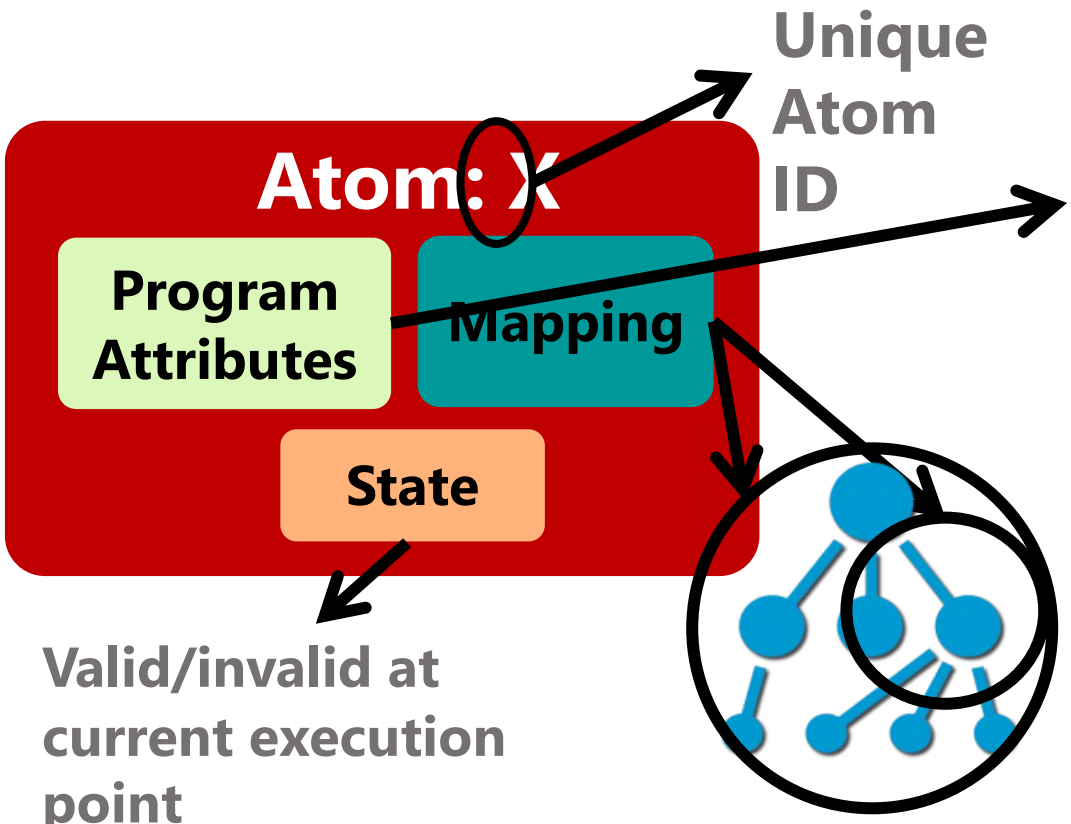
- Background on Expressive Memory
- **MetaSys**
  - **Software Interface**
  - **Key Structures**
- FPGA Implementation
- Evaluation

Metadata: Data Semantics



The ATOM

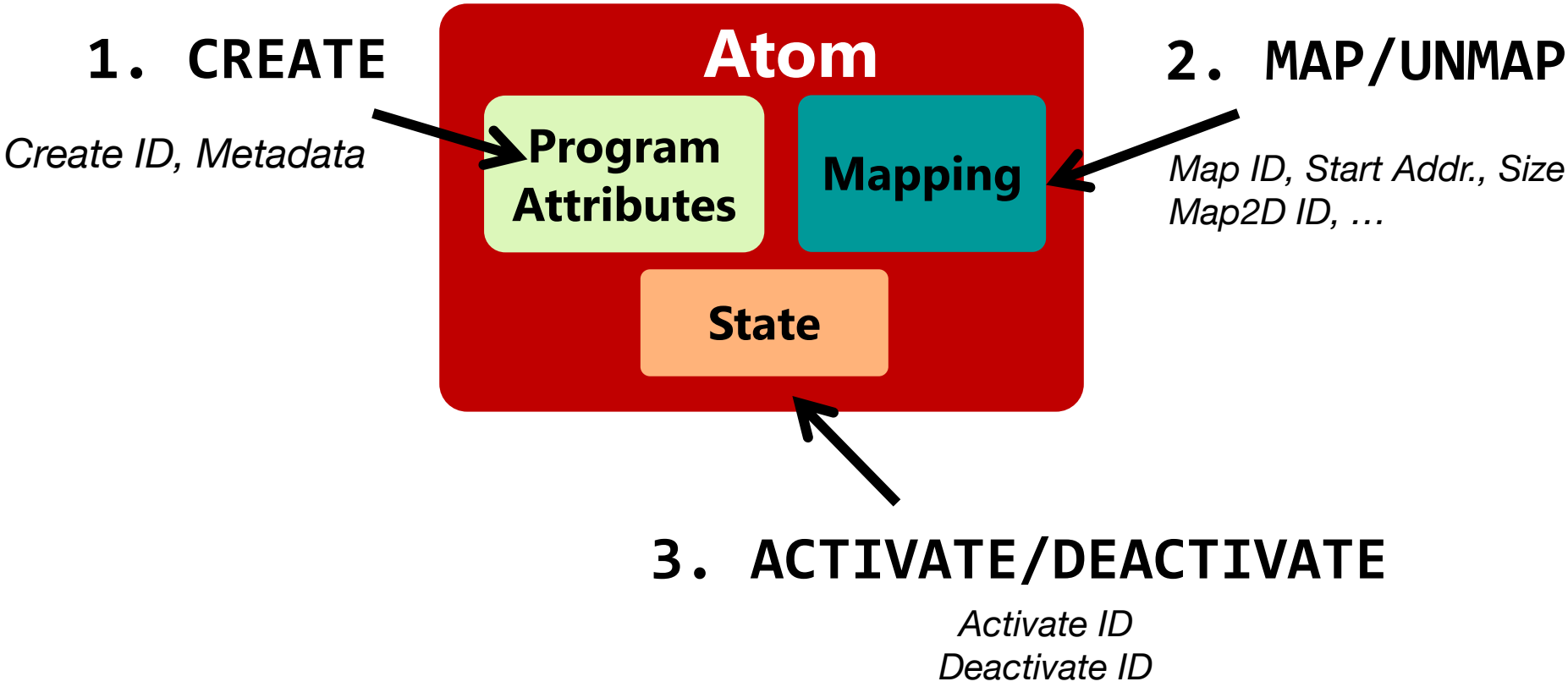
An abstraction to express data semantics



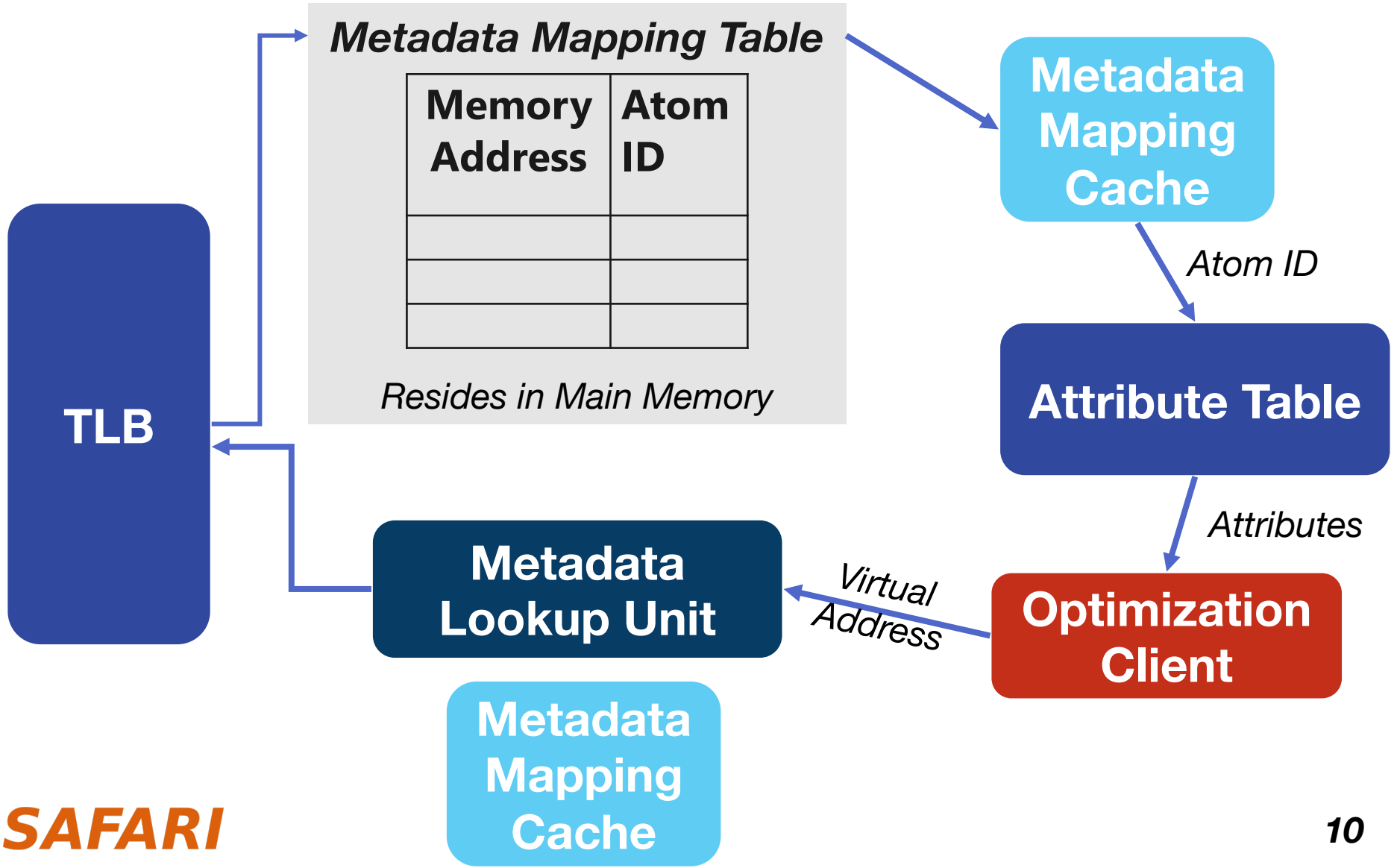
1. **Data Value Properties:**
   - INT, FLOAT, CHAR,…
   - COMPRESSIBLE
2. **Access Properties:**
   - Read-Write Characteristics
   - Access Pattern
   - Access Intensity
3. **Data Locality:**
   - Working Set
   - Reuse
4. **…**

The Software Interface

Three Atom operators



MetaSys Key Structures



Outline

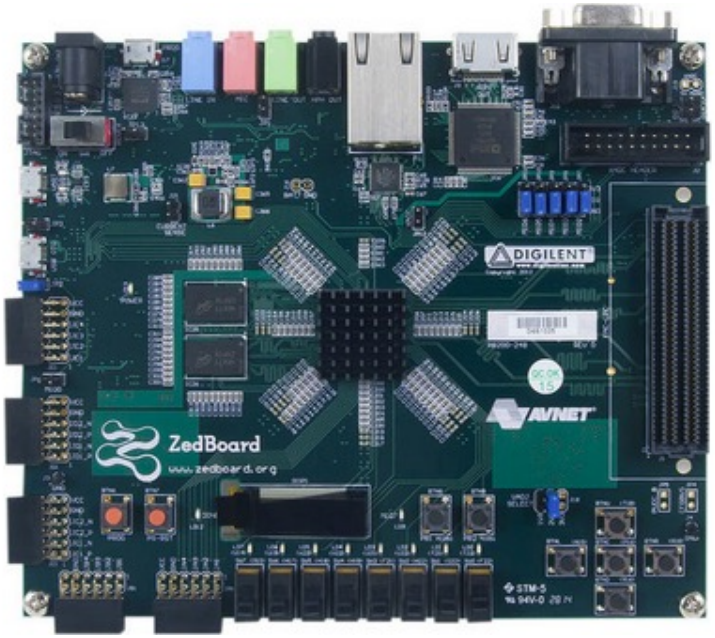
- Background on Expressive Memory
- MetaSys
  - Software Interface
  - Key Structures
- **FPGA Implementation**
- Evaluation

FPGA Prototype

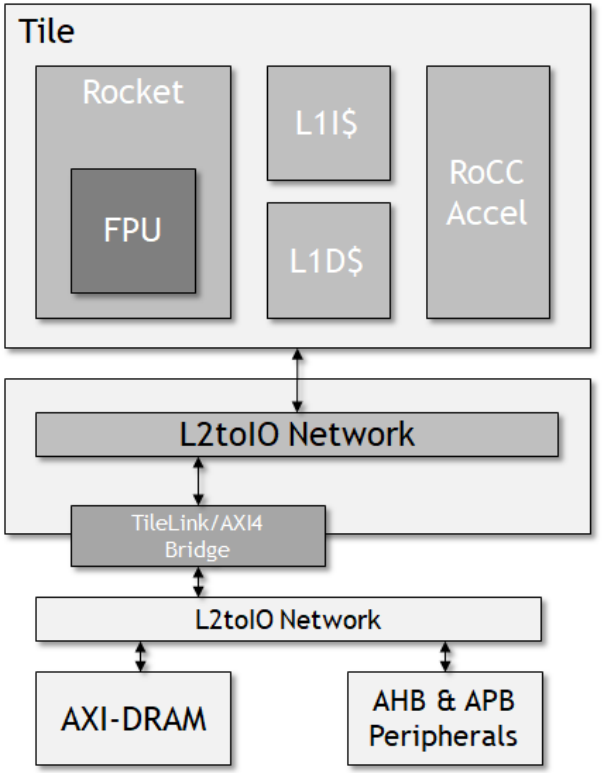
Prototype on Xilinx Zedboard

within a real RISC-V system (Rocket Chip)

Zedboard



Rocket Chip

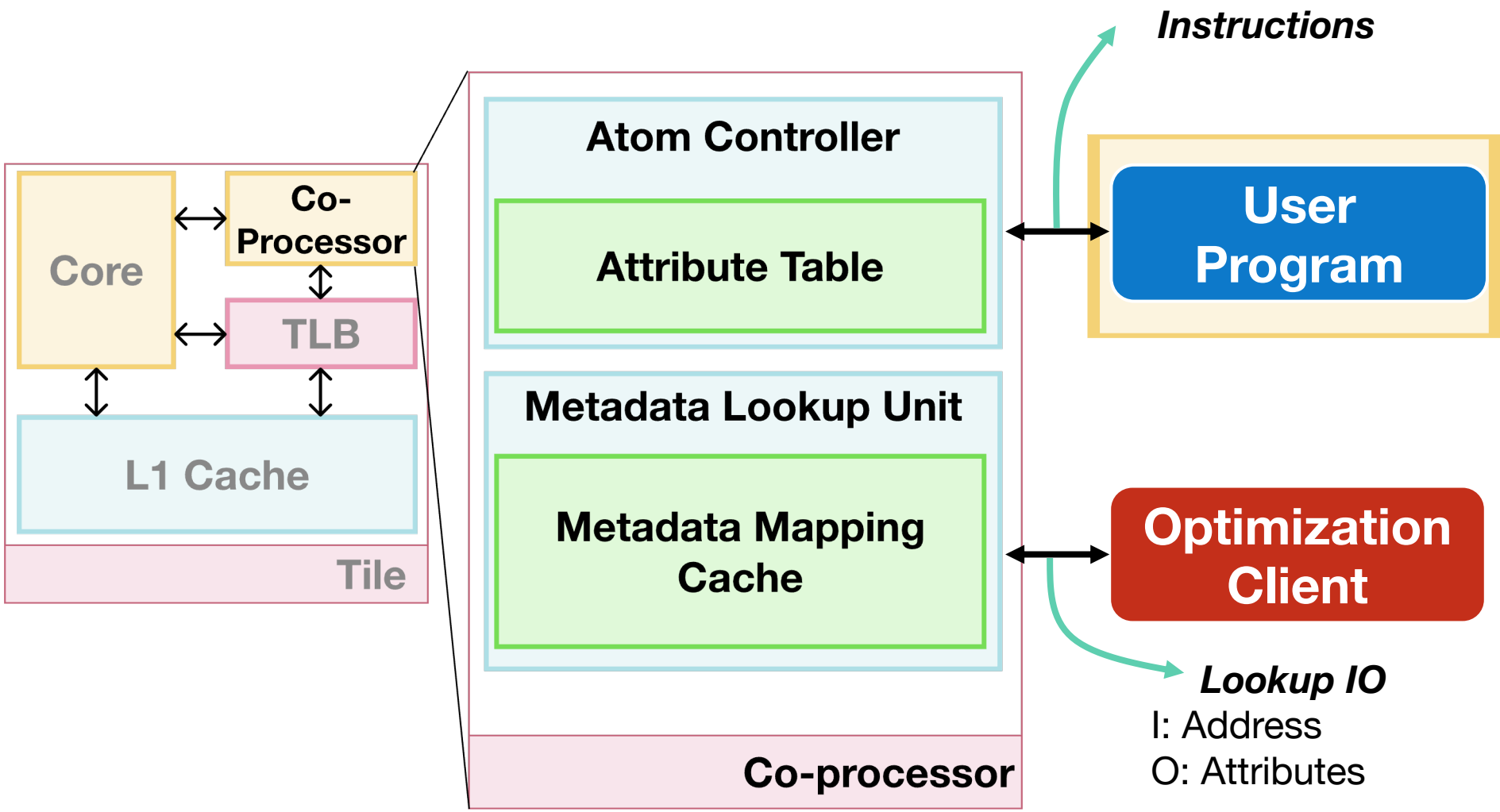


MetaSys in Rocket Chip

Implement two main components:

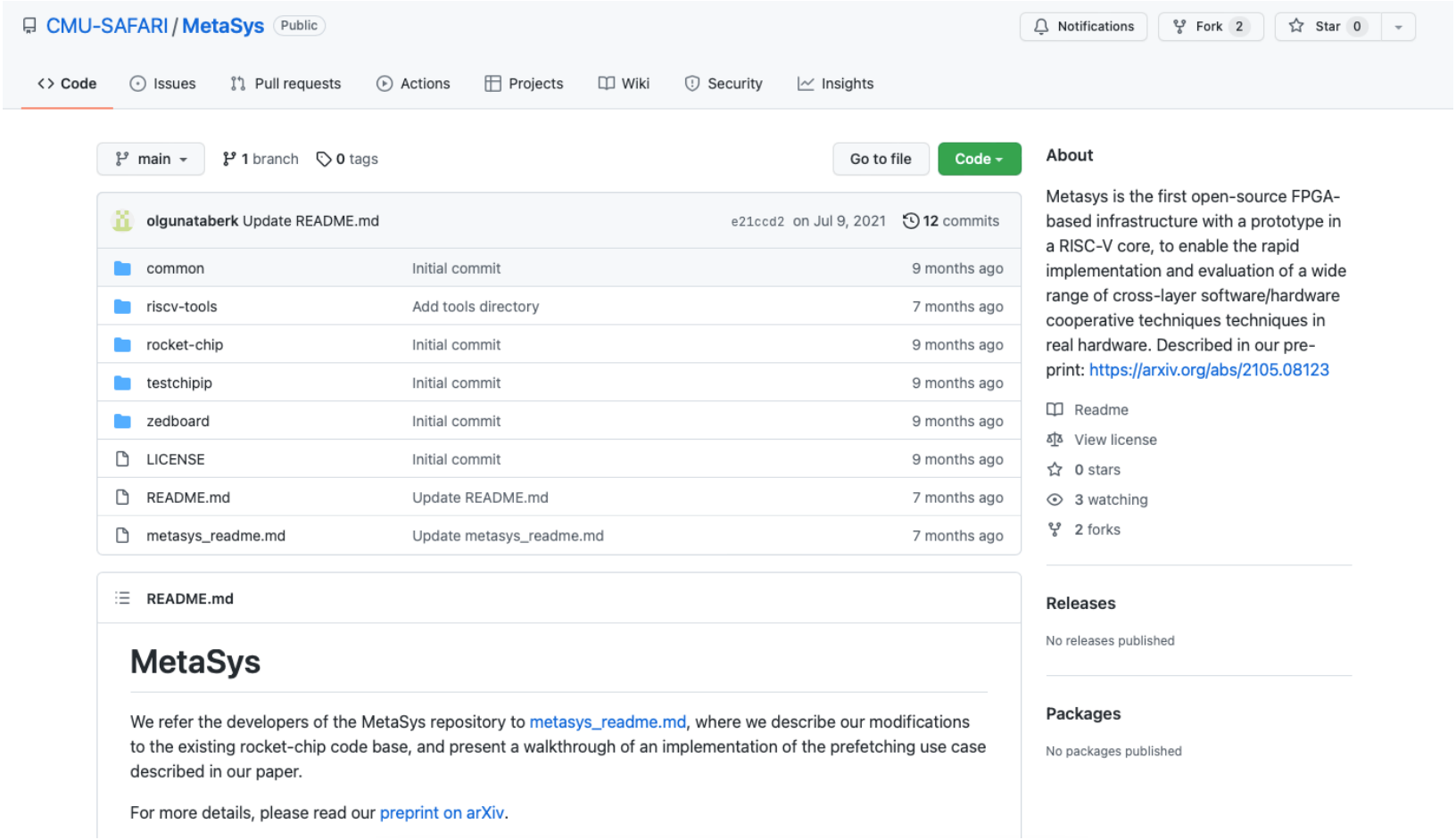
1. Atom Controller
   - Manages the attribute table (CREATE – (DE)ACTIVATE)
   - Performs atom mapping (MAP/UNMAP)
     - Physical address → Atom ID
2. Metadata Lookup Unit
   - Responds to clients:
     - Provides atom attributes
   - Contains the metadata mapping cache

Changes in Rocket Chip



Source on Github

https://github.com/CMU-SAFARI/MetaSys



Outline

- Background on Expressive Memory
- MetaSys
  - Software Interface
  - Key Structures
- FPGA Implementation
- **Evaluation**

Characterizing Metadata Management

Our **goal** is twofold:

1. ...
2. Perform a detailed limit study to quantify the overheads associated with general metadata systems

Quantify the overheads of performing lookups in MetaSys

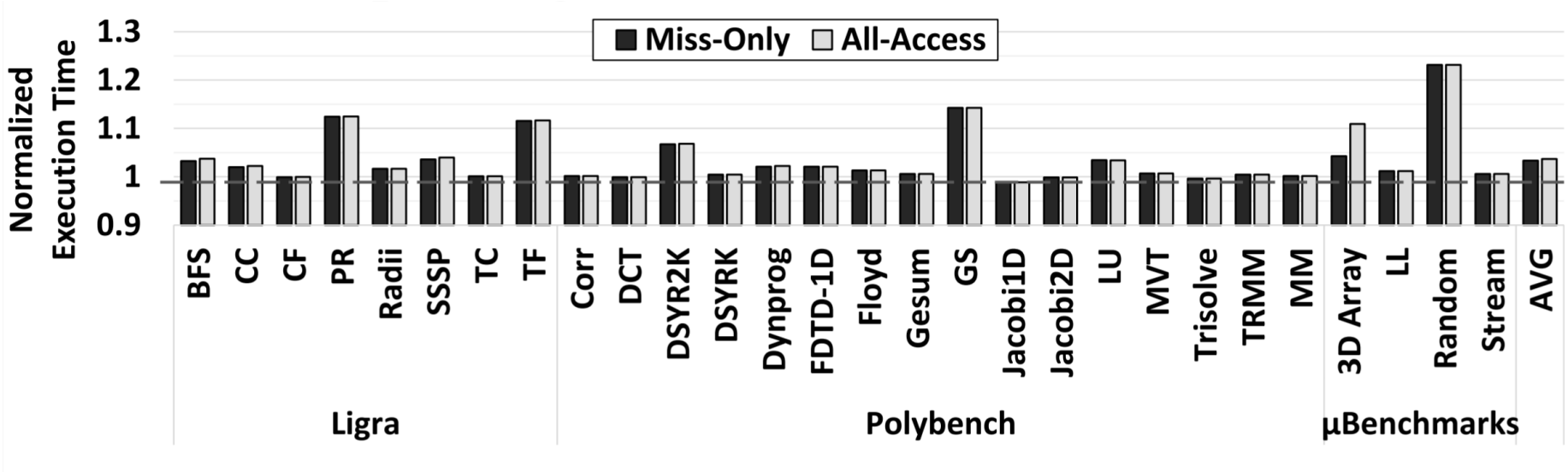
Evaluation Methodology

Run workloads on MetaSys prototype (Zedboard):

Microbenchmarks: Represent a variety of memory access patterns
Polybench: Scientific computation kernels
Ligra: Graph workloads

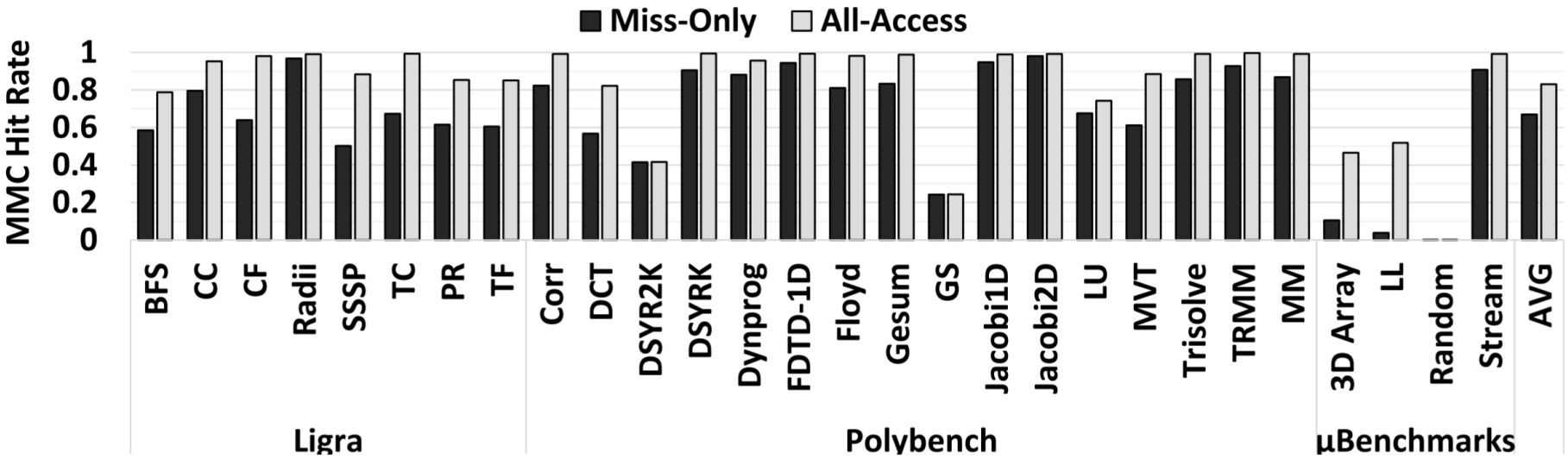
CPU: 25 MHz; in-order Rocket core [21]; **TLB** 16 entries DTLB; LRU policy;
L1 Data + Inst. Cache: 16 KB, 4-way; 4-cycle; 64 B line; LRU policy; MSHR size: 2
MMC: NMRU Policy; 128 entries; 38bits/entry; **Tagging Granularity:** 512B;
Private Metadata Table: 256 entries; 64B/entry; **DRAM:** 533MHz; V_{dd}: 1.5V;
Workloads: Ligra [36]: PageRank(PR), Shortest Path (SSSP), Collaborative Filtering (CF) Teenage Follower (TF), Triangle Counting (TC), Breadth-First Search (BFS) Radius Estimation (Radii), Connected Components (CC); **Polybench** [37]; **μBenchmarks**

Performance Overhead



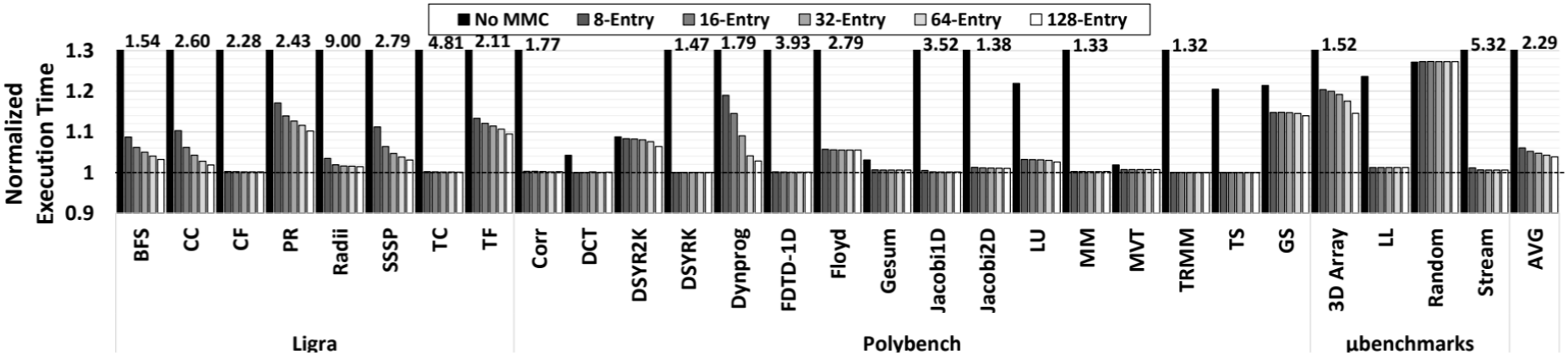
Metadata lookups occur low performance overheads 2.7% on average

MMC hit rate



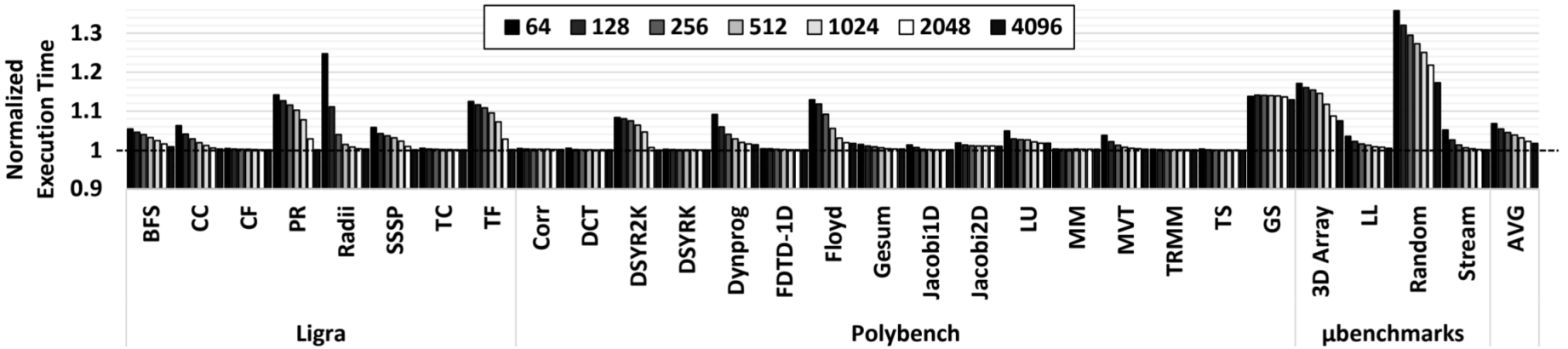
average
MMC hit rate correlates with locality of application

Impact of MMC size



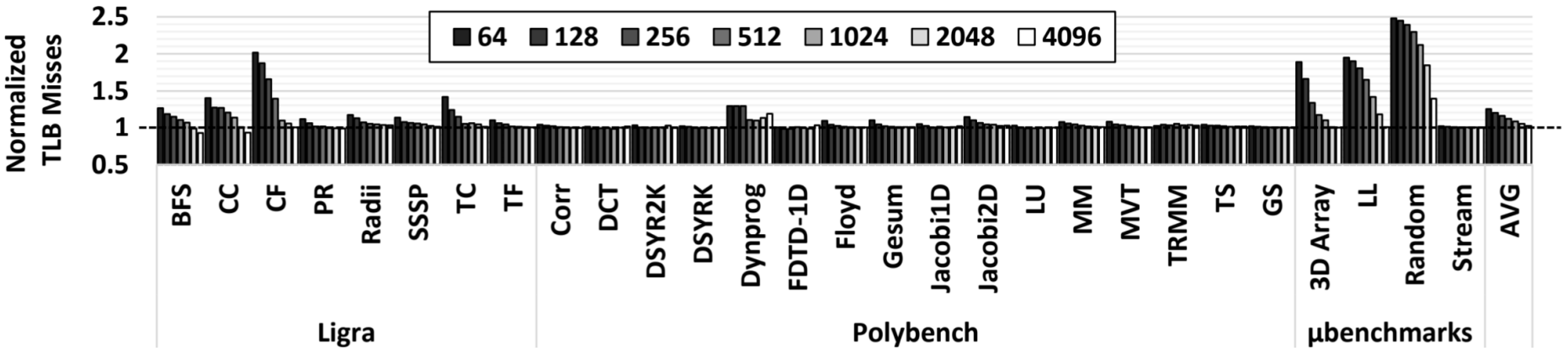
Workloads with low temporal and spatial locality are not sensitive to MMC size

Impact of Tagging Granularity



Performance impact increases with finer granularity

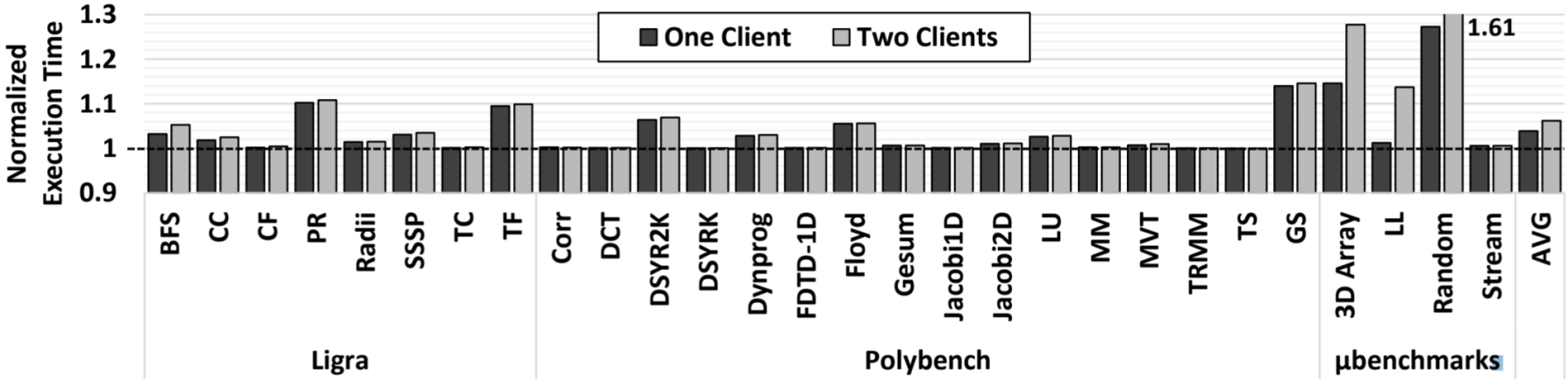
Impact of Tagging Granularity on TLB misses



Fine tagging granularities increase TLB misses

Effect of Contention

One Client: All memory requests originating from rocket core
Two Clients: One client + all memory requests originating from the page table walker



Multiple clients do not significantly affect performance
(0.3% overhead on average)

Executive Summary

Problem

- Cross-layer techniques are challenging to implement because they require full-stack changes
- Existing open-source infrastructures for implementing cross-layer techniques are not designed to provide key features:

Key Idea – Provide:

- Rich dynamic HW/SW interfaces
- Low-overhead metadata management
- Interfaces to key hardware components (e.g., prefetcher)

Our **goal** is twofold:

1. Develop an efficient and flexible framework to enable rapid implementation of new cross-layer techniques
2. Perform a detailed limit study to quantify the overheads associated with general metadata systems

MetaSys: A Practical Open-Source Metadata Management System to Implement and Evaluate Cross-Layer Optimizations

Nandita Vijaykumar

Ataberk Olgun, Konstantinos Kanellopoulos, F. Nisa Bostanci

Hasan Hassan, Mehrshad Lotfi, Phillip B. Gibbons, Onur Mutlu

SAFARI

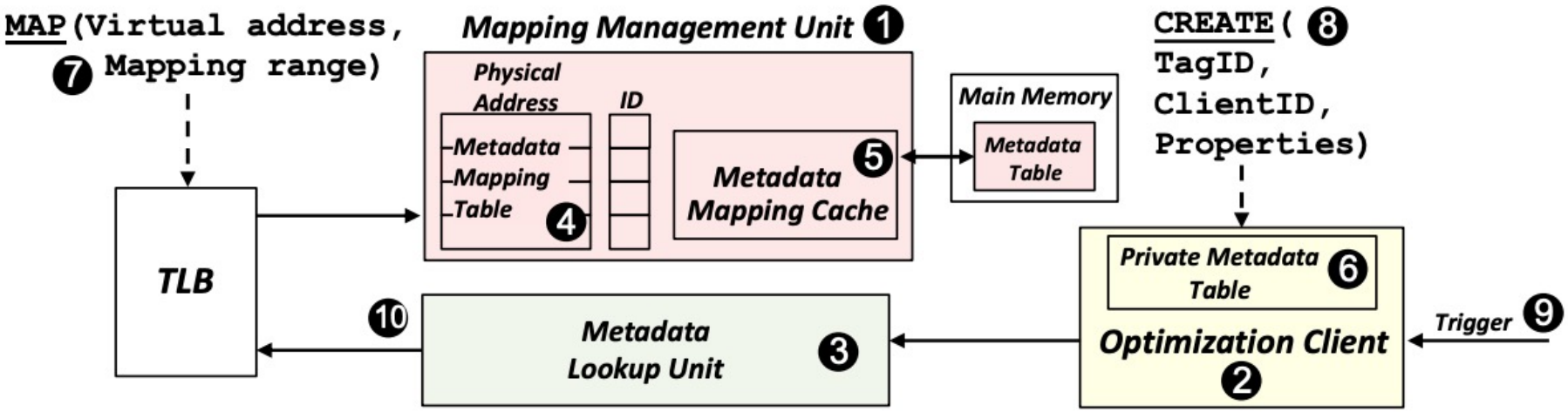
UNIVERSITY OF TORONTO

ETH Zürich

TOBB ETÜ

University of Economics & Technology

Backup


MAP(Virtual address,
7 Mapping range)
Mapping Management Unit 1
Physical Address
ID
Metadata Mapping Table
4
5
Metadata Mapping Cache
Main Memory
Metadata Table
CREATE(8
TagID,
ClientID,
Properties)
TLB
10
Metadata Lookup Unit
3
Private Metadata Table
6
Optimization Client
2
Trigger 9

Backup

Table 1. MetaSys instructions.

MetaSys Operator	MetaSys ISA Instructions
CREATE	CREATEClientID, TagID, Metadata
(UN)MAP	(UN)MAP TagID, start_addr, size (UN)MAP2D TagID, start_addr, lenX, sizeX, sizeY (UN)MAP3D TagID, start_addr, lenX, lenY, sizeX, sizeY, sizeZ ;

Table 2. The MetaSys software library function calls.

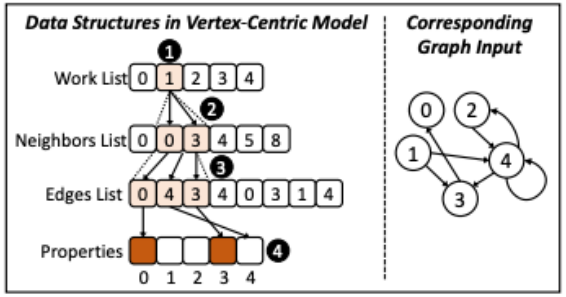
Library Function Call	Description
CREATE(*ClientID*, *TagID*, **meta*)	*ClientID* -> PMT[*TagID*] = **metadata*
MAP(*start**, *end**, *TagID*)	MMT[*start...end*] = *TagID*
UNMAP(*start**, *end**)	MMT[*start...end*] = 0

Backup

Table 3. Comparison between MetaSys and XMem interfaces.

Operator	XMem [164]	MetaSys
CREATE	Compiler pragma to communicate static metadata at program load time.	Selects a hardware optimization, dynamically associates metadata with an ID, and communicates both to hardware at runtime (implemented as a new instruction).
(UN)MAP	Associate memory ranges with tag IDs (implemented as new instructions).	Same semantics and implementation as XMem.
(DE)ACTIVATE	Enable/disable optimizations associated with a tag ID (implemented as new instructions).	**Does not exist** as the same functionality can now be done with CREATE.

Backup



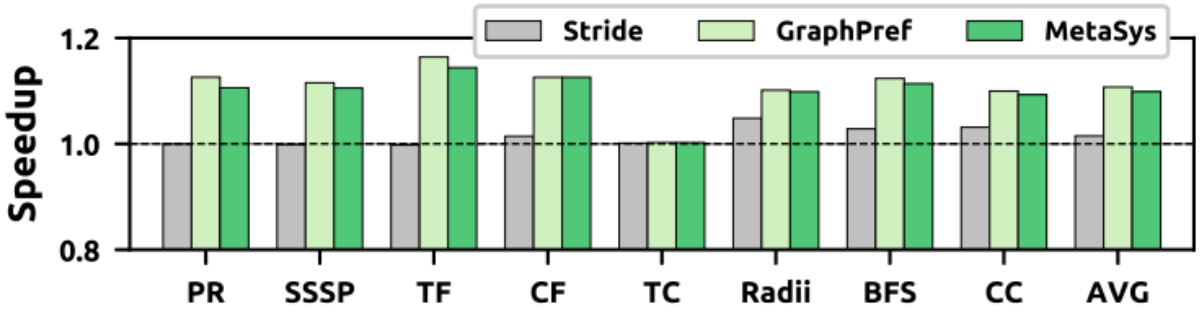


Fig. 3. Data-dependent accesses in vertex-centric graph processing model (left), speedup with the MetaSys prefetcher (right).

Backup

```
1   /* Additional Code in BFS */
2   metadata_create(0, 1, WorkList, { sizeof(Worklist), VertexList, Stride }); // Create metadata (arguments: ClientID=0, tag ID, metadata)
3   metadata_create(0, 2,VertexList, { sizeof(VertexList), EdgeList, Stride});
4   metadata_create(0, 3,EdgeList, { sizeof(Worklist), VertexList, Stride});
5   metadata_create(0, 4,Property, {sizeof(Property), NULL, Stride});
6
7   metadata_map((void*) (WorkList), mapSize, 1); // Map tag 1 to Worklist
8   metadata_map((void*) (VertexList), mapSize, 2);
9   metadata_map((void*) (EdgeList), mapSize, 3);
10  metadata_map((void*) (Property), mapSize, 4);
11
12  /* Hardware Prefetcher Functionality */
13  void snoop_mem_request ( address ): // snoop every memory request
14          (Valid,Base,Bounds,PointerToNextDS,Stride) = metadata_lookup (address); // Access MetaSys using the address
15
16           while (Valid && PointerToNext != NULL) // While the data structure traversal is not complete
17            if (Base < address && address > Bounds) // If the memory request comes from a tracked data structure
18                 initiate_prefetch(address+stride); // Initiate a stride prefetch request
19                 Value = wait_for_value(address+stride); // Wait for the prefetch request to return data
20                 address = &PointerToNextDS[value]; // Discover the address of the next data structure (DS)
21                 (Valid,Base,Bounds,PointerToNextDS,Stride) = metadata_lookup (PointerToNextDS[value]); // Look up metadata for the next DS
```

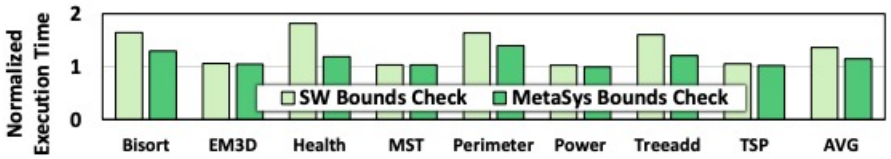
Listing 1. Metasys-based Graph Prefetcher. Available online [57].

Backup

```
1   /* Example bounds checking software */
2   metadata_map((void*) (array1), mapSize, 1); // Map TagID 1 to array "array1"
3   metadata_map((void*) (array2), mapSize, 2); // Map TagID 2 to array "array2"
4   metadata_map((void*) (array3), mapSize, 3); // Map TagID 3 to array "array3"
5
6   // Access every element of each array with a stride of one element
7   for(i = 0 ; i < array_size ; i += 1)
8   {
9       metadata_create(0,1,1); // Create metadata with TagID=1 and Metadata=1 to ClientID=0
10      int elem1 = array1[i];
11      metadata_create(0,2,2); // Create metadata with TagID=2 and Metadata=2 to ClientID=0
12      int elem2 = array2[i];
13      int result = elem1 + elem2;
14      metadata_create(0,3,3); // Create metadata with TagID=3 and Metadata=3 to ClientID=0
15      array3[i] = result;
16   }
17
18  /* Hardware Bounds Checker Functionality */
19  HardwareBoundsChecker(CreateTagID, Address):
20      TagIDRegister <= CreateTagID // Software communicates TagID using CREATE
21      MetadataTagID <= PerformMetadataLookup(Address) // Bounds checker client performs a metadata lookup to find the TagID of address
22      if MetadataTagID != TagIDRegister: // Interrupt rocket core if TagIDs do not match (i.e., access is out of bounds)
23          InterruptRocketCore()
```

Listing 2. MetaSys-based bounds checking example. Full source code is available online [57].

Backup



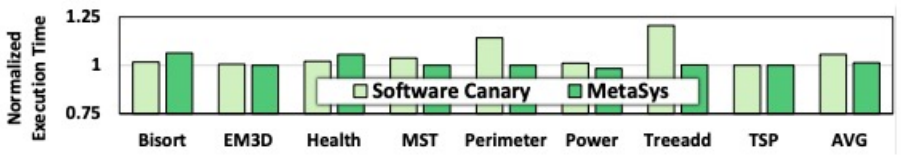


Fig. 4. Performance overheads for (left) bounds checking, (right) return address protection.

Backup

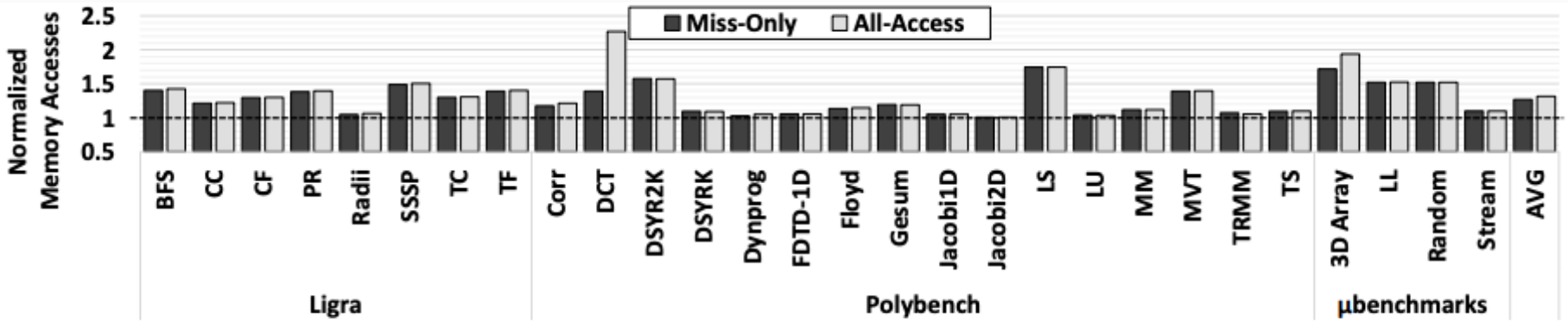


Fig. 6. Additional memory accesses introduced by MetaSys metadata lookups.

Backup

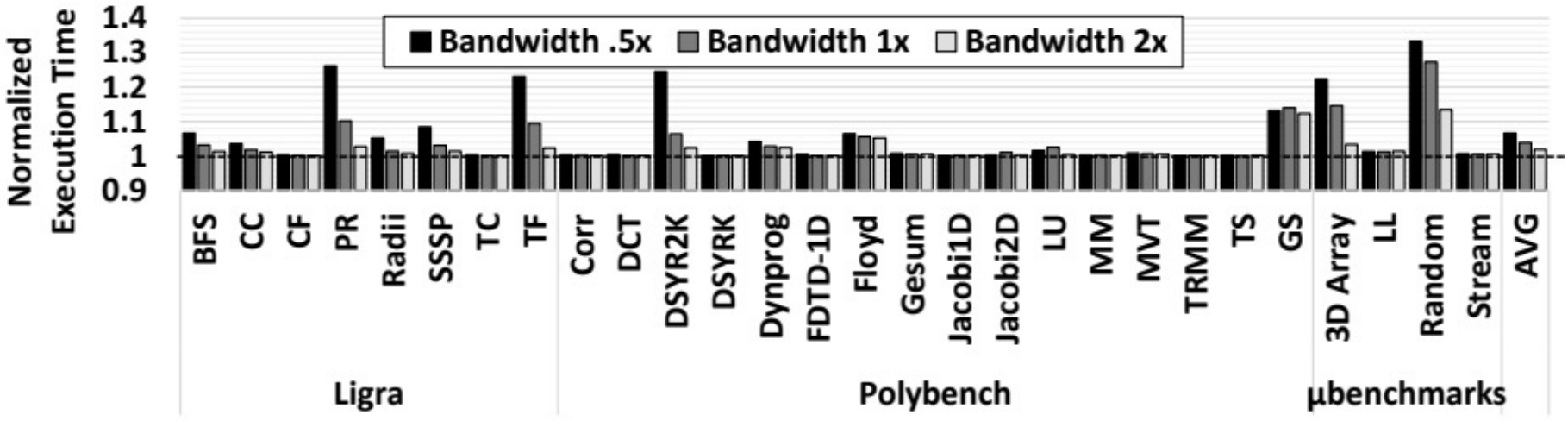


Fig. 8. MetaSys performance overhead on systems with varying amounts of memory bandwidth.

Backup

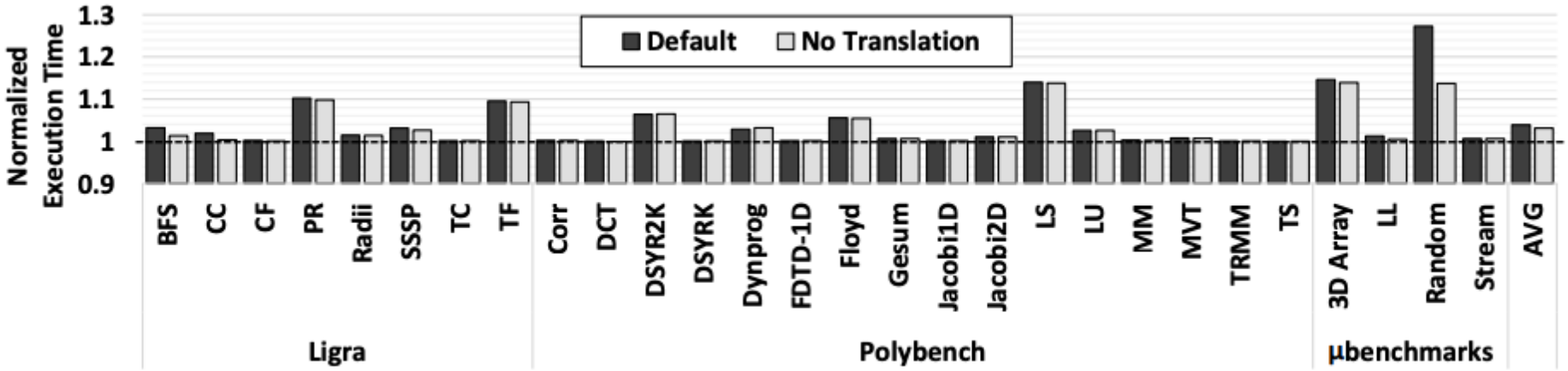


Fig. 12. Performance overhead with no address translation overhead for metadata.

Backup

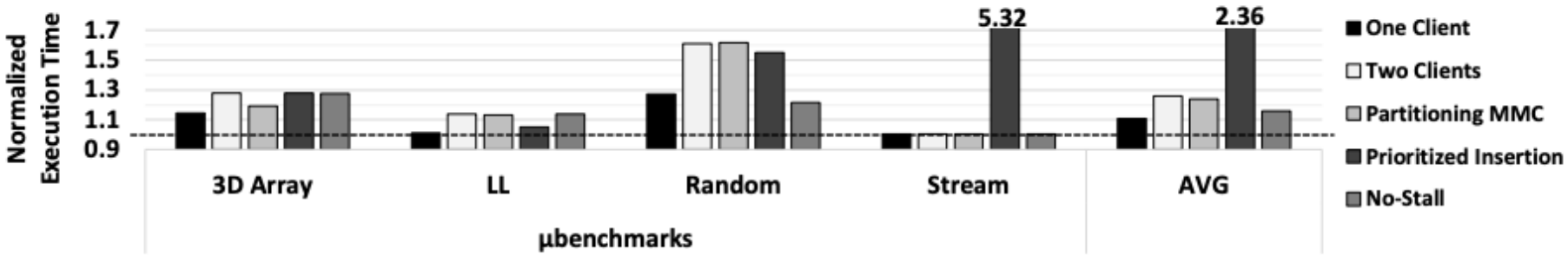


Fig. 14. Alleviating MMC contention in microbenchmarks.

Backup

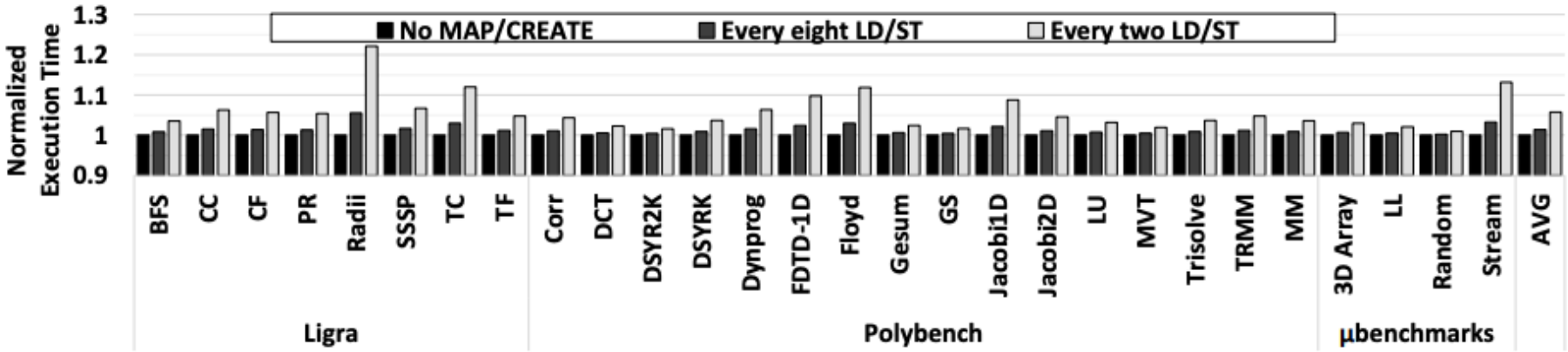


Fig. 15. Performance overhead of MAP/CREATE instructions.