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
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

- Software
  - Data Structures
  - Code Optimizations
- Hardware
  - Instructions
  - Memory Addresses

100011111...
101010011...

Data Types:
- Integer
- Float
- Char
With a richer abstraction:
SW can provide program information can significantly help hardware

Data Structures

Access Patterns

Software

Hardware

Data Placement

Prefetcher

Data Compression

Integer

Float

Char

Data Type/Layout
Outline

• Background on Expressive Memory

• MetaSys
  • Software Interface
  • Key Structures

• FPGA Implementation

• Evaluation
Metadata: Data Semantics

1. Data Value Properties: INT, FLOAT, CHAR, … COMPRESSIBLE
2. Access Properties:
   - Read-Write Characteristics
   - Access Pattern
   - Access Intensity (“Hotness”)
3. Data Locality:
   - Working Set
   - Reuse

SAFARI
The ATOM
An abstraction to express data semantics

1. Data Value Properties: INT, FLOAT, CHAR,… COMPRESSIBLE
3. Data Locality: Working Set Reuse
4. …
The Software Interface

Three Atom operators

1. CREATE
   Create ID, Metadata

2. MAP/UNMAP
   Map ID, Start Addr., Size Map2D ID, ...

3. ACTIVATE/DEACTIVATE
   Activate ID
   Deactivate ID
MetaSys Key Structures

Metadata Mapping Table

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Atom ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Resides in Main Memory

TLB

Metadata Lookup Unit

Attribute Table

Optimization Client

Metadata Mapping Cache

Virtual Address

Metadata Mapping Cache

Atom ID

Attributes
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)
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 Optimization

- User Program
- Optimization Client
- Instructions
- Lookup IO
  - I: Address
  - O: Attributes

Diagram:
- Core
- Co-Processor
- TLB
- L1 Cache
- Atom Controller
- Attribute Table
- Metadata Lookup Unit
- Metadata Mapping Cache
- Co-processor
Source on Github

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

**MetaSys**

We refer the developers of the MetaSys repository to `metasys_readme.md`, where we describe our modifications to the existing rocket-chip code base, and present a walkthrough of an implementation of the prefetching use case described in our paper.

For more details, please read our preprint on arXiv.
Outline

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

 MMC can cover ~81% of all memory requests on 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., prefether)

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
Backup

MAP(Virtual address, Mapping range)

Mapping Management Unit ①

Physical Address  ID

Metadata Mapping Cache ⑤

Main Memory

Metadata Table

CREATE(TagID, ClientID, Properties)

Private Metadata Table

Optimization Client ②

Trigger ⑨

TLB

Metadata Lookup Unit ③

④
## Table 1. MetaSys instructions.

<table>
<thead>
<tr>
<th>MetaSys Operator</th>
<th>MetaSys ISA Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>CREATE(ClientID, TagID, Metadata)</td>
</tr>
<tr>
<td>(UN)MAP</td>
<td>(UN)MAP TagID, start_addr, size</td>
</tr>
<tr>
<td></td>
<td>(UN)MAP2D TagID, start_addr, lenX, sizeX, sizeY</td>
</tr>
<tr>
<td></td>
<td>(UN)MAP3D TagID, start_addr, lenX, lenY, sizeX, sizeY, sizeZ ;</td>
</tr>
</tbody>
</table>

## Table 2. The MetaSys software library function calls.

<table>
<thead>
<tr>
<th>Library Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE(ClientID, TagID, *meta)</td>
<td>ClientID -&gt; PMT[TagID] = *metadata</td>
</tr>
<tr>
<td>MAP(start*, end*, TagID)</td>
<td>MMT[start...end] = TagID</td>
</tr>
<tr>
<td>UNMAP(start*, end*)</td>
<td>MMT[start...end] = 0</td>
</tr>
</tbody>
</table>
Table 3. Comparison between MetaSys and XMem interfaces.

<table>
<thead>
<tr>
<th>Operator</th>
<th>XMem [164]</th>
<th>MetaSys</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>Compiler pragma to communicate static metadata at program load time.</td>
<td>Selects a hardware optimization, dynamically associates metadata with an ID, and communicates both to hardware at runtime (implemented as a new instruction).</td>
</tr>
<tr>
<td>(UN)MAP</td>
<td>Associate memory ranges with tag IDs (implemented as new instructions).</td>
<td>Same semantics and implementation as XMem.</td>
</tr>
<tr>
<td>(DE)ACTIVATE</td>
<td>Enable/disable optimizations associated with a tag ID (implemented as new instructions).</td>
<td>Does not exist as the same functionality can now be done with CREATE.</td>
</tr>
</tbody>
</table>
Fig. 3. Data-dependent accesses in vertex-centric graph processing model (left), speedup with the MetaSys prefetcher (right).
Listing 1. Metasys-based Graph Prefetcher. Available online [57].
/* Example bounds checking software */
metadata_map((void*) (array1), mapSize, 1); // Map TagID 1 to array "array1"
metadata_map((void*) (array2), mapSize, 2); // Map TagID 2 to array "array2"
metadata_map((void*) (array3), mapSize, 3); // Map TagID 3 to array "array3"

// Access every element of each array with a stride of one element
for(i = 0; i < array_size; i += 1)
{
    metadata_create(0,1,1); // Create metadata with TagID=1 and Metadata=1 to ClientID=0
    int elem1 = array1[i];
    metadata_create(0,2,2); // Create metadata with TagID=2 and Metadata=2 to ClientID=0
    int elem2 = array2[i];
    int result = elem1 + elem2;
    metadata_create(0,3,3); // Create metadata with TagID=3 and Metadata=3 to ClientID=0
    array3[i] = result;
}

/* Hardware Bounds Checker Functionality */
HardwareBoundsChecker(CreateTagID, Address):
    TagIDRegister <= CreateTagID // Software communicates TagID using CREATE
    MetadataTagID <= PerformMetadataLookup(Address) // Bounds checker client performs a metadata lookup to find the TagID of address
    if MetadataTagID != TagIDRegister: // Interrupt rocket core if TagIDs do not match (i.e., access is out of bounds)
        InterruptRocketCore()

Listing 2. MetaSys-based bounds checking example. Full source code is available online [57].
Fig. 4. Performance overheads for (left) bounds checking, (right) return address protection.
Fig. 6. Additional memory accesses introduced by MetaSys metadata lookups.
Fig. 8. MetaSys performance overhead on systems with varying amounts of memory bandwidth.
Fig. 12. Performance overhead with no address translation overhead for metadata.
Backup

Fig. 14. Alleviating MMC contention in microbenchmarks.
Fig. 15. Performance overhead of MAP/CREATE instructions.