**Compress Objects, Not Cache Lines: An Object-Based Compressed Memory Hierarchy**

Po-An Tsai and Daniel Sanchez  
{poantsai, sanchez}@csail.mit.edu

---

**Background and Motivation**

1. Compressed memory hierarchies require uncompressed-to-compressed address translation  
   - Core issues loads/stores to uncompressed address  
   - Data is stored in compressed address

2. Prior compression algorithms focus on compressing fixed-size cache lines and only work well for regular memory layout (e.g., arrays)
   - Compresses much better for out-of-line data types

3. Many programs mainly store objects in main memory and their layout is therefore irregular
   - Object C
   - Object B1
   - Object A2

---

**Objects, not cache lines, are the natural unit of compression!**

**Insight 1:** Object-based applications always follow pointers to access objects

**Insight 2:** There is significant redundancy across objects of the same type

---

**Baseline System: Hotpads, An Object-Based Memory Hierarchy [MICRO’18]**

**Example Hotpads hierarchy**

- RegFile
- L1 Pad
- L2 Pad
- Main Mem
- Objects
- Free space

**Feature 1.** Object-based data movement

**Feature 2.** In-hierarchy object allocation

**Feature 3.** Bulk GC and object eviction process that updates pointers to moved objects

**Zippads: An Object-Based Compressed Memory Hierarchy**

- Point directly to compressed objects to avoid translation

- Uncompressed layout
  - Object A1
  - Object B1
  - Object A2
  - Object C
  - Object B2

- Compressed layout
  - Encode compression info in pointers for fast decompression
  - Compressed size
  - 50 bytes
  - 48 bytes
  - 48-X bytes
  - Compressed object address (48-X bits)
  - Compression encoding bits (X bits)

**COCO: Cross-Object-Compression**

- Exploit redundancy across objects by storing only the bytes that differ from a representative object

- Base object (32B)
- Uncompressed object (32B)

**Evaluation**

**Methodology:**
- Simulate Zippads using Maxsim (ZSim+Maxime JVM)
- 8 Java apps from scientific, DB, graph analytics, KV store
- See our paper for C/C++ apps results

**Zippads significantly reduces memory footprint**
- CMH and Zippads compress well for array-heavy apps
- Zippads compresses much better for object-heavy apps
- COCO adds extra benefits
- Zippads+COCO improves over CMH by 63%

**Zippads reduces memory traffic**
- Achieves the lowest memory traffic (40% lower than CMH)
- Combines benefits of CMH and Hotpads

**Zippads improves performance**
- Outperforms CMH by 24% while reducing footprint much further

See our paper (https://bit.ly/zippads) for more features, details, and evaluation results!