Managing Hybrid Memories by Predicting Object Write Intensity

Shoaib Akram, Kathryn S. Mckinley, Jennifer B. Sartor, Lieven Eeckhout
Ghent University, Belgium
Shoaib.Akram@UGent.be
DRAM as main memory is facing multiple challenges

Cost high when scaling to 100s of GB
Reliability a concern as stored charge very small
Opportunity for new memory technologies to replace DRAM

PCM cells have limited write endurance, shortening its lifetime.

- Reset to amorphous at 610°C
- Set to crystalline at 350°C
- Read
Hybrid memory is the best of DRAM and PCM

DRAM
- Speed ✔
- Endurance ✔
- Energy
- Density

PCM
- Speed
- Endurance
- Energy ✔
- Density ✔
Future of main memory: limited DRAM, lots of PCM

This work uses DRAM for frequently written data.
Garbage collection: key advantage of using a managed language

Memory automatically reclaimed for reuse
More than just reclaim, stuff better organized
Use GC to keep frequently written objects in DRAM

Reactive approach

- Monitors writes to objects
- More fine-grained compared to hardware and OS approaches
- No page migrations

Write-rationing garbage collection for hybrid memories, PLDI 2018
Use GC to keep frequently written objects in DRAM

Proactive approach
  Use a profile-guided predictor (this work)
Three offline steps in building a write intensity predictor

1. Profiling
   - Profile: <Size, Type, Site, #writes>
2. Feature Selection
   - Site, #writes
3. Classification
   - Site, advice
Profiling methodology

- Java Virtual Machine
  - Jikes RVM (version 3.1.2)
  - 4 MB nursery
  - 2 GB Mark Sweep mature

- Java applications
  - 9 from DaCapo
  - PsuedoJBB 2005
  - Default inputs
The outcome of profiling is a write intensity trace

For each unique object \( X \)

1. Size
2. Type
3. Allocation site \(<\text{method-name, bytecode index}>\)
4. \# Writes
Measuring entropy of different features

<table>
<thead>
<tr>
<th>Object</th>
<th>Size</th>
<th># Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>12 B</td>
<td>1000</td>
</tr>
<tr>
<td>O2</td>
<td>12 B</td>
<td>1000</td>
</tr>
<tr>
<td>O3</td>
<td>64 KB</td>
<td>1000</td>
</tr>
<tr>
<td>O4</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>O5</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

Each size has an entropy of 0
Measuring entropy of different features

<table>
<thead>
<tr>
<th>Object</th>
<th>Size</th>
<th># Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>12 B</td>
<td>1000</td>
</tr>
<tr>
<td>O2</td>
<td>12 B</td>
<td>1000</td>
</tr>
<tr>
<td>O3</td>
<td>64 KB</td>
<td>1000</td>
</tr>
<tr>
<td>O4</td>
<td>32</td>
<td>1000</td>
</tr>
<tr>
<td>O5</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

Size 32 has an entropy of 1
Homogeneity curves compare size vs. type vs. allocation site

Write intensity threshold = 1 K
Heuristics to classify allocation sites as write-intensive or not

• Goals
  1. Minimize DRAM utilization
  2. Minimize PCM writes

• Parameters
  1. Criteria to determine write intensive objects
  2. Homogeneity threshold
Criteria # 1: write frequency

Write frequency threshold = 1 K

<table>
<thead>
<tr>
<th>Object</th>
<th>Site</th>
<th>Size</th>
<th># Writes</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O2</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O3</td>
<td>A</td>
<td>65536</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O4</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
<tr>
<td>O5</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
</tbody>
</table>
Criteria # 2: write density

<table>
<thead>
<tr>
<th>Object</th>
<th>Site</th>
<th>Size</th>
<th># Writes</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O2</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O3</td>
<td>A</td>
<td>65536</td>
<td>1000</td>
<td>✗</td>
</tr>
<tr>
<td>O4</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
<tr>
<td>O5</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
</tbody>
</table>

Write density threshold = 1
### Criteria #1: write frequency

Write frequency threshold = 1 K  
Homogeneity threshold = 50%

<table>
<thead>
<tr>
<th>Object</th>
<th>Site</th>
<th>Size</th>
<th># Writes</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✓</td>
</tr>
<tr>
<td>O2</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✓</td>
</tr>
<tr>
<td>O3</td>
<td>A</td>
<td>65536</td>
<td>1000</td>
<td>✓</td>
</tr>
<tr>
<td>O4</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
<tr>
<td>O5</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗</td>
</tr>
</tbody>
</table>

Site A is write-intensive
## Criteria # 2: write density

Write density threshold = 1  
Homogeneity threshold = 50%

<table>
<thead>
<tr>
<th>Object</th>
<th>Site</th>
<th>Size</th>
<th># Writes</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O2</td>
<td>A</td>
<td>12</td>
<td>1000</td>
<td>✔️</td>
</tr>
<tr>
<td>O3</td>
<td>A</td>
<td>65536</td>
<td>1000</td>
<td>✗️</td>
</tr>
<tr>
<td>O4</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗️</td>
</tr>
<tr>
<td>O5</td>
<td>A</td>
<td>32</td>
<td>0</td>
<td>✗️</td>
</tr>
</tbody>
</table>

Site A is NOT write-intensive
Baseline generational heap organization

mutator nursery GC mature mutator

DRAM
Distribution of writes to objects

Empirical observations

1. Nursery is highly mutated
2. 2% of mature objects get 80% of writes
Generational heap organization in hybrid memory
PCM Writes vs. DRAM Utilization

Homogeneity threshold = 1%
Allocation site predictor yields better tradeoffs than size and type

Homogeneity threshold = 1%, Write-Density (50)
Profile-guided predictor is more effective compared to existing work.

The graph shows normalized writes to PCM for various applications, comparing Kingsguard-Writers (red) and Write-Density (blue). The applications include Lusearch, Pjbb, Lu.Fix, Avrora, Luindex, Hsqlb, Xalan, Sunflow, Pmd, Jython, Pmd.S, Fop, Antlr, and Bloat.
What is missing in the workshop paper?

- Implementation details
  - Compiler sets a bit in the object header
  - GC chooses the correct allocator
- Big data benchmarks
- Emulation on a real NUMA machine
- Performance results
Conclusions

• Exploit GC for improving the lifetime of emerging memories
• Allocation sites correctly predict write intensity
• Use an allocation site predictor to eliminate a large number of writes to PCM
Challenge: limit # writes to PCM

Solution: Use DRAM for frequently written data
Online monitoring introduces mutator and GC overheads
Online monitoring introduces mutator and GC overheads.