Exploiting Managed Language Semantics to Mitigate Wear-out in Persistent Memory

Shoaib Akram
Ghent University, Belgium
Main memory capacity expansion

Charge storage in DRAM a scaling limitation

Manufacturing complexity makes DRAM pricing volatile

Source: WSTS, IC Insights
Phase change memory (PCM)

Scalable → More Gb for the same price
Byte addressable like DRAM
Latency closer to DRAM
😊 Low write endurance
Why PCM has low write endurance?

Electric pulses to program PCM cells wear them out over time

reset to amorphous

set to crystalline
**PCM** alone as a **DRAM** replacement wears out in a few months for popular Java applications
Mitigating PCM wear-out

Wear-leveling to spread writes across PCM

This talk → Use DRAM to limit PCM writes
OS to limit PCM writes

Coarse-grained page migrations hurt application performance and PCM lifetime
Managed runtimes

Platform independence
Abstract hardware/OS
→ Aka Virtual Machine

Ease programmer’s burden
Garbage collection
Security
GC to limit PCM writes

**GC understands memory semantics**

**GC approaches are pro-active and fine-grained**
Write Distribution in GC heap

nursery  GC  mature

70% of writes
Write Distribution in GC heap

- nursery
  - 70% of writes

- mature
  - 22% to 2% of objects
Write-Rationing Garbage Collection

Limit PCM writes by discovering highly written objects

Kingsguard dynamically monitor writes
Kingsguard-Nursery (KG-N)

nursery

mature

large

DRAM

PCM
Kingsguard-Writers (KG-W)

DRAM

PCM
KG-W drawbacks

Overhead of dynamic monitoring

Limited time window to predict write intensity

Excessive & fixed DRAM consumption
Write-Rationing Garbage Collection

Limit PCM writes by discovering highly written objects

Kingsguard dynamically monitor writes

Crystal Gazer statically profiles objects
Allocation site as a write predictor

Uniform distribution 😞
Skewed distribution 😊
Write distribution by allocation site

Few sites capture majority of writes

% mature objects

Sites sorted by writes

Writes
Volume

Pjbb2005
Crystal Gazer operation

Application Profiling

```
  a = new Object()
  ...
  b = new Object()
```

Advice Generation

```
  a = new Object()
  ...
  b = new_dram Object()
```
Advice generation

Goal: Generate <alloc-site, advice> pairs
advice → DRAM or PCM
input is a write-intensity trace

Two heuristics to classify allocation sites as DRAM or PCM
Classification heuristic (1)

Freq: A threshold % of objects from a site get more than a threshold # writes → DRAM

😊 Aggressively limits PCM writes

😢 No distinction based on object size
Classification heuristic (2)

Write density $\rightarrow$ Ratio of # writes to object size

**Dens:** A *threshold* % of objects from a site have more than a *threshold* write density $\rightarrow$ DRAM
Classification thresholds

Homogeneity threshold → 1%
Frequency threshold → 1
Density threshold → 1
Classification examples

Frequency threshold = 1
PCM writes = ?, DRAM bytes = ?

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Classification examples

Frequency threshold = 1

**PCM writes = ?, DRAM bytes = ?**

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Classification examples

Frequency threshold = 1
PCM writes = 0/256, DRAM bytes = 5008

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Classification examples

Density threshold = 1

**PCM writes = ?**, **DRAM bytes = ?**

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Classification examples

Density threshold = 1
PCM writes = ?, DRAM bytes = ?

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>

→ 32
Classification examples

Density threshold = 1
PCM writes = ?, DRAM bytes = ?

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Classification examples

Density threshold = 1
**PCM writes = 128/256, DRAM bytes = 12**

<table>
<thead>
<tr>
<th>Object Identifier</th>
<th># Writes</th>
<th># Bytes</th>
<th>Allocation site</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O3</td>
<td>128</td>
<td>4</td>
<td>A() + 10</td>
</tr>
<tr>
<td>O4</td>
<td>128</td>
<td>4096</td>
<td>B() + 4</td>
</tr>
</tbody>
</table>
Object placement in Crystal Gazer

$new\_dram() \rightarrow$ Set a bit in the object header

GC $\rightarrow$ Inspect the bit on nursery collection to copy object in DRAM or PCM
Object placement in Crystal Gazer

- **nursery**
- **mature**
- **large**

DRAM

- **mature**
- **large**

PCM

Is marked highly written? ✓
Key features of Crystal Gazer

Eliminates overhead of dynamic monitoring

Less mispredictions due to pro-active nature

Pareto optimal trade-offs b/w capacity and lifetime
Evaluation methodology

15 Applications → DaCapo, GraphChi, SpecJBB

Medium-end server platform

Different inputs for production and advice

Jikes RVM
Emulation platform

App

Jikes RVM

OS

CPU

CPU
PCM write rates

PCM-Only write rate is above 1 GB/s on average

Safe operation is 200 MB/s
PCM write rates

- KG-N
- KG-W
- Dens
- Freq

Write rate in MB/s

- Hsqldb
- Xalan
- Pmd
- Eclipse
- Pjbb
- Page Rank
- Conn Comp
- ALS Fact
- Average
Execution time

<table>
<thead>
<tr>
<th></th>
<th>KG-W</th>
<th>Dens</th>
<th>Freq</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hsqldb</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Xalan</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Pmd</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Eclipse</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Pjbb</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Page Rank</td>
<td>1.30%</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>Conn Comp</td>
<td>1.30%</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>ALS Fact</td>
<td>1.30%</td>
<td>0.9</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Exec. time norm. to KG-N
Crystal Gazer opens up Pareto-optimal trade-offs.

PCM writes relative to KG-N

Crystal Gazer

KG-W

Pjbb2005

DRAM capacity in MB

0.3 0.4 0.5 0.6 0.7 0.8

0.3 0.4 0.5 0.6 0.7 0.8
Write-rationing garbage collection

Hybrid memory is inevitable

All layers can contribute to manage hybrid memory

Write-rationing GC is pro-active and fine-grained