UniHeap: Managing Persistent Objects Across Managed Runtimes for Non-Volatile Memory

Daixuan Li  Benjamin Reidys  Jinghan Sun
Thomas Shull  Josep Torrellas  Jian Huang
Non-Volatile Memory: Opportunities & Challenges

- Performance & Persistency
- Byte-Addressable
- Data Durability
- Programmability

System Platform Research Group at UIUC
Programmability Challenge of NVM

Mark all the persistent object updates in the code

Object 1

Object 2

Hierarchical Volatile Cache

CLWB & SFENCE Instructions

Persist Object

NVM Device

Correctness Problem

Performance Bugs
Leveraging Managed Runtime to Manage NVM

Hardware Complexity  | Managed Data Objects  | Popular Programming Models
AutoPersist: An Easy-to-Use NVM Framework

Thomas Shull
University of Illinois at Urbana-Champaign
shull@illinois.edu

Jian Huang
University of Illinois at Urbana-Champaign
jhuang8761@illinois.edu

Josey Terrazas
University of Illinois at Urbana-Champaign
terrazas@illinois.edu

Abstract
Byte-addressable, non-volatile memory (NVM) is emerging as a revolutionary memory technology that provides persistence, near-DRAM performance, and scalable capacity. To facilitate its use, many NVM programming models have been proposed. However, most models require programmers to explicitly specify the data structures or objects that should reside in NVM. Such requirement increases the burden on programmers, complicates software development, and introduces opportunities for correctness and performance bugs.

We believe that requiring programmers to identify the data structures that should reside in NVM is undesirable. Instead, programmers should only be required to identify durable roots—the entry points to the persistent data structures at recovery time. The NVM programming framework should then automatically ensure that all the data structures reachable from these roots are in NVM, and store to these data structures are persistently completed in an intuitive order.

To this end, we present a new NVM programming framework, named AutoPersist, that only requires programmers to identify durable roots. AutoPersist then persists all the data structures that can be reached from the durable roots in an automated and transparent manner. We implement AutoPersist as a threat-aware extension to the Java language and perform experiments with a variety of applications running on Intel Optane DC persistent memory. We demonstrate that AutoPersist requires minimal code modifications, and significantly outperforms expert-marked Java NVM applications.

CCS Concepts: • Hardware → Non-volatile memory; • Software and its engineering → Just-in-time compilers; Source code generation.

1 Introduction
There have recently been significant technological advances towards providing fast, byte-addressable non-volatile memory (NVM), such as Intel 3D XPoint [17], Phase Change Memory (PCM) [12], and ReRAM [11]. These memory technologies promise near-DRAM performance, scalable memory capacity, and data durability, which offer great opportunities for software systems and applications.

To enable applications to take advantage of NVM, many NVM programming frameworks have been proposed, such as Intel PMDK [1], Memorystore [15], NVHalps [11], Espresso [46], and others [20, 21, 24, 36, 46]. While the underlying model to ensure data consistency [14, 15] varies across frameworks, all of these frameworks share a common trait: they require the programmer to explicitly specify the data structures or objects that should reside in NVM. This limitation results in substantial effort from programmers, and introduces opportunities for correctness and performance bugs due to the increased programming complexity [15]. Moreover, it limits the ability of applications to use existing libraries.

We believe that requiring users to identify all the data structures or objects that reside in NVM is undesirable. Instead, the user should only be required to identify the durable roots, which are the named entities into durable data structures at recovery time. Given this input, the NVM framework should then automatically ensure that all the data structures reachable from these durable roots are in NVM.

In this paper, we present a new NVM programming framework named AutoPersist that only requires programmers to identify the set of durable roots. While most NVM frameworks are implemented in C or C++, we chose to implement AutoPersist as an extension to the Java language. As is common for managed languages, Java already provides transparent support for object memory management, as well as high-level semantics for programmers.

Keywords • Java, Non-Volatile Memory, JT Compilation

ACM Reference Format

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or suppress to post to a repository or archive, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. © 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-6725-3/19/06...
https://doi.org/10.1145/3318241.3319468

System Platform Research Group at UIUC
Managing Persistent Object Across Runtimes is Desirable

File System Enable Data Sharing with File Abstraction.
Managing Persistent Object Across Runtimes is Desirable

Enable Data Sharing with **Persistent Object** Abstraction.

Java  
Python

Unified Persistence Layer

Object  
NVM Device

File System Enable Data Sharing with **File** Abstraction.
Mana

Persistent Object Across Runtimes is Desirable

Web Service  Shared Libraries  Data Analytics

Sharing persistent objects across multiple runtimes is Needed.
State-of-the-Art Object Sharing Approaches

- **Thrift/Protobuf:**
  - Java
  - Python
  - RPC Protocol

- **Shared Memory:**
  - Java
  - Python
  - Shared Memory
  - Does not support NVM

**Serialization Overhead**
UniHeap: Managing Persistent Objects Across Runtimes

Java  Python  ..........  JavaScript

Unified Persistence Layer

Shared NVM Heap

NVM Device
Challenges of Persistent Object Management Across Runtimes

Unified Persistence Layer

Shared NVM Heap

NVM Device

Unified Object Model

Persistent and Crash-Safe Implementation

Efficient and Correct GC

System Platform Research Group at UIUC
Challenges of Persistent Object Management Across Runtimes

Unified Object Model

Persistent and Crash-Safe Implementation

Efficient and Correct GC

Unified Persistence Layer

Shared NVM Heap

NVM Device

Java
Python
JavaScript
## Unified Object Model and Type System

<table>
<thead>
<tr>
<th>Uniheap</th>
<th>char</th>
<th>short</th>
<th>int</th>
<th>long</th>
<th>float</th>
<th>double</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Java</strong></td>
<td>boolean, byte</td>
<td>char</td>
<td>int</td>
<td>long</td>
<td>float</td>
<td>double</td>
<td>reference, array</td>
</tr>
<tr>
<td><strong>Python</strong></td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>long</td>
<td>float</td>
<td>-</td>
<td>list, dict, tuple</td>
</tr>
<tr>
<td><strong>JavaScript</strong></td>
<td>boolean</td>
<td>-</td>
<td>num</td>
<td>num</td>
<td>num</td>
<td>num</td>
<td>array</td>
</tr>
</tbody>
</table>

Two built-in types: **numeral type** and **reference type**
Challenges of Persistent Object Management Across Runtimes

Unified Persistence Layer

- Java
- Python
- JavaScript

Shared NVM Heap

NVM Device

Unified Object Model

Persistent and Crash-Safe Implementation

Efficient and Correct GC
Compatible with Automated Data Persistence Approach

Durable Root

Volatile Memory

Non-Volatile Memory

A

B

R

set_root
Compatible with Automated Data Persistence Approach

Durable Root

- set_root

Volatile Memory

Non-Volatile Memory

Atomic Region

- atomic_begin
- Persist Objects
- atomic_end

- Crash Consistency
- Failure Atomic
The Persistent Overhead of Managing Persistent Objects

Redo and undo logs bring duplicate write overhead

Reduce Logging Overhead with Atomic Update and Out-of-Place Update
Managing Persistent Objects in A Log-Structured Manner

**Object Field Index Table**

```
fid1  fid2  fid3  ...  fidn
```

**DRAM**

**Log Region**

```
log  log  Tx2  log  log  log
```

```
log  log  log  log
```

**Heap Header**  **Plass Region**  **Root Table**  **Valid Bitmap**  **Object Region**  **Log Region**

**NVM**
Managing Persistent Objects in A Log-Structured Manner

- Heap Header
- Plass Region
- Root Table
- Valid Bitmap
- Object Region
- Log Region

NVM

Shared NVM Heap
Managing Persistent Objects in A Log-Structured Manner

Decoupling the Data and Metadata of Objects
Managing Persistent Objects in A Log-Structured Manner

Object Field Index Table

| fid1 | fid2 | fid3 | ... | fidn |

DRAM

Log Region

- Tx1
  - log
  - log
  - log

- Tx2
  - log
  - log

- Tx3
  - log
  - log
  - log

Heap Header
Plass Region
Root Table
Valid Bitmap
Object Region
Log Region

NVM
Managing Persistent Objects in A Log-Structured Manner

Transaction-based Out-of-place Object updates

- Crash Safety
- Concurrent Access
- Garbage Collection
Managing Persistent Objects in A Log-Structured Manner

Object Field Index Table

| fid1 | fid2 | fid3 | ... | fidn |

DRAM

Log Region

Address Remapping with Cached Index Table
Managing Persistent Objects in A Log-Structured Manner

Efficient and crash-safe persistent object management
Challenges of Persistent Object Management Across Runtimes

Unified Object Model

Persistent and Crash-Safe Implementation

Efficient and Correct GC

Java

Python

JavaScript

Unified Persistence Layer

Shared NVM Heap

NVM Device

System Platform Research Group at UIUC
Garbage Collection of UniHeap

- Marking phase
- Relocation phase
- Compaction phase
- Clean-up phase
Garbage Collection of UniHeap

• Marking phase  ✔ Naturally Crash-Safe
• Relocation phase
• Compaction phase  ✔ Keep old data until Clean up Phases
• Cleanup phase

Crash Safety of GC
Coordinated GC Across Managed Runtimes

Uniheap

Root Table

vroot

droot

HotSpot JVM

CPython

Spidermonkey

Mapping Set

Mapping Set

Mapping Set
Put It All Together

UniHeap Shared Library

Shared NVM Heap

Direct Memory Access

Unified Interface

Runtime

Java Program

Python Program

……..

JavaScript Program

Java Module

Python Module

JavaScript Module

Java Module

Python Module

JavaScript Module

System Platform Research Group at UIUC
**Experiment Setup**

- **CPU:** 24-core Intel 2nd Xeon
- **NVM:** 8 * 128GB Intel Optane DC

**Benchmarks**

- **Java:** YCSB over QuickCached and H2
- **Python:** Python Performance Benchmark Suite
- **JavaScript:** JetStream2
Performance of Persistent Object Sharing

UniHeap outperforms existing approach by 1.2x - 3.4x
UniHeap can scale to support multiple managed runtimes.
UniHeap Summary

Java

Python

Unified Persistence Layer

Shared NVM Heap

NVM Device

JavaScript
Thanks!

Daixuan Li
daixuan2@illinois.edu

Benjamin Reidys  Jinghan Sun  Thomas Shull
Josep Torrellas  Jian Huang