A Primer on Memory Persistency

A Primer on Memory Persistency

Vaibhav Gogte, Aasheesh Kolli, and homas F. Wenisch

ISBN: 978-3-031-79193-2 paperback
ISBN: 978-3-031-79205-2 PDF

DOI 10.1007/978-3-031-79205-2

A Publication in the Springer series
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Lecture #58
Series Editor: Natalie Enright Jerger, *University of Toronto*
Editor Emerita: Margaret Martonosi, *Princeton University*
Founding Editor Emeritus: Mark D. Hill, *University of Wisconsin, Madison*
Series ISSN
Print 1935-3235 Electronic 1935-3243

A Primer on Memory Persistency

Vaibhav Gogte
University of Michigan

Aasheesh Kolli
Pennsylvania State University

Thomas F. Wenisch
University of Michigan

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #58

ABSTRACT

This book introduces readers to emerging persistent memory (PM) technologies that promise the performance of dynamic random-access memory (DRAM) with the durability of traditional storage media, such as hard disks and solid-state drives (SSDs). Persistent memories (PMs), such as Intel's Optane DC persistent memories, are commercially available today. Unlike traditional storage devices, PMs can be accessed over a byte-addressable load-store interface with access latency that is comparable to DRAM. Unfortunately, existing hardware and software systems are ill-equipped to fully avail the potential of these byte-addressable memory technologies as they have been designed to access traditional storage media over a block-based interface. Several mechanisms have been explored in the research literature over the past decade to design hardware and software systems that provide high-performance access to PMs.

Because PMs are durable, they can retain data across failures, such as power failures and program crashes. Upon a failure, recovery mechanisms may inspect PM data, reconstruct state and resume program execution. Correct recovery of data requires that operations to the PM are properly ordered during normal program execution. *Memory persistency models* define the order in which memory operations are performed at the PM. Much like memory consistency models, memory persistency models may be relaxed to improve application performance. Several proposals have emerged recently to design memory persistency models for hardware and software systems and for high-level programming languages. These proposals differ in several key aspects; they relax PM ordering constraints, introduce varying programmability burden, and introduce differing granularity of failure atomicity for PM operations.

This primer provides a detailed overview of the various classes of the memory persistency models, their implementations in hardware, programming languages and software systems proposed in the recent research literature, and the PM ordering techniques employed by modern processors.

KEYWORDS

persistent memory, non-volatile memory, storage-class memory, memory persistency models, strict persistency model, epoch persistency model, strand persistency model, failure atomicity, logging mechanisms

Contents

Preface

Upcoming persistent memory (PM) technologies aim to revolutionize the landscape of future storage systems. These memory technologies promise to deliver near-DRAM performance coupled with the non-volatility of traditional storage media, such as hard disks and solid-state drives (SSDs). PMs provide a byte-addressable load-store interface with access latency similar to DRAM, unlike hard disks and SSDs, which can only be accessed over a block-based interface. Unfortunately, existing hardware, software, and programming systems are ill-equipped to utilize the complete potential of these byte-addressable storage technologies, as they have been designed over generations to access storage over a block-based interface. Existing systems require an expensive software layer to access hard disks and SSDs. While the software layer introduces negligible overheads relative to the inherent latency of accessing a hard disk or SSD, the overheads are prohibitive relative to much faster PMs. Several mechanisms have emerged in the research literature over the past decade and are recently being employed by CPU vendors to design hardware and software systems that provide high-performance access to PMs.

PMs, such as Intel's Optane DC persistent memories, are commercially available today. Because PMs are durable, they can retain data across failures, such as power failures and program crashes. Upon a failure, recovery mechanisms can inspect the data stored in PM, use it to reconstruct the application state, and resume program execution. Correct data recovery requires that the operations to the PM are properly ordered during normal program execution. Unfortunately, ordering PM operations is complicated in modern processor systems by the volatile cache hierarchy and various buffers throughout the memory system. These hardware mechanisms reorder, coalesce, and elide memory operations, which complicate their ordering at the PM. For example, write-back caches may lazily drain updates to the PM (e.g., when cacheline conflicts occur), thereby reordering the updates relative to the original program order. On a failure, the volatile state in the caches is lost. Recovery mechanisms may then observe unintentional reordering of memory operations in the PM post-failure.

Memory persistency models guarantee ordering of PM operations. Memory persistency models are analogous to memory consistency models, which define the visibility order of memory operations to shared memory. Similarly, memory persistency models define the allowable order in which memory operations are performed at the PM. Several memory persistency models have been introduced in the literature. Some proposals have been defined as extensions to the hardware ISA and others to the semantics of high-level programming languages. The proposals differ in several key aspects: they relax PM ordering constraints in different ways, introduce varying programmability burden, and introduce differing granularity of failure atomicity for PM operations. This primer provides a detailed overview of the various classes of memory persistency models, their implementations in hardware, programming languages, and software systems proposed in the recent research literature, and the PM ordering techniques employed by modern processor systems. We organize this primer in six chapters, with each chapter detailing different aspects of PM programming.

Chapter 1 provides a brief overview of different technologies that are considered key contenders for designing PMs. It briefly covers Phase Change Memory, Spin Torque Transfer RAM, and Ferroelectric RAM that might be used to construct high-density, low-access-latency PMs. The chapter also provides a performance characterization of the commercially available Intel Optane DC Persistent Memories. PMs can be used in different configurations by storage systems. Modern file systems may access them over a block-based interface by directly replacing hard disks or SSDs with PMs, or file systems may be developed from the ground up and optimized to directly access byte-addressable storage. We cover the trade-offs for these alternatives in Chapter 1.

Chapter 2 discusses the mechanisms proposed by hardware vendors, such as Intel and ARM, to provide PM ordering guarantees. We discuss instruction set extensions introduced by the vendors to durably store and order PM accesses. As discussed earlier, correct recovery requires that memory operations are ordered to the PM. Chapter 2 provides examples to show why ordering is required for correct recovery.

Chapter 3 details different memory persistency models. Like memory consistency models, memory persistency models may be relaxed, albeit at a higher programmability burden, to improve performance. We cover strict and relaxed persistency models, which offer varying ordering constraints on the PM operations. We define these persistency models formally, and also discuss the hardware implementations proposed in the literature that build each of these models.

The persistency models discussed in Chapter 3 assume atomicity for individual updates (e.g., 8-byte updates in Intel x86). That is, in the case of a failure, either the entirety of the update is applied to the PM or the memory location retains its original value. Recovery mechanisms build from this foundation to provide failure atomicity for multiple updates. They may construct logging mechanisms to ensure that either all of the updates or none of the updates within a failure-atomic program region are durable across a failure. Chapter 4 describes these logging mechanisms. It also focuses on mechanisms proposed in the research literature to construct failure atomicity in hardware.

Chapter 5 discusses the software mechanisms designed to program persistent memory systems. It describes file systems, software transactions, and programming frameworks designed for PM programming. The testing frameworks aimed at finding memory ordering bugs in the recovery systems designed for PMs are also covered in Chapter 5. Chapter 6 concludes the primer.

This primer assumes that the reader is familiar with the basics of computer architecture, with a basic understanding of hardware caching and the memory hierarchy. We cite relevant works that readers can use to obtain a complete understanding of the architecture details that are orthogonal to the topics discussed in this book. Wherever possible, we show quantitative comparison between different proposals discussed in the research literature.

Vaibhav Gogte, Aasheesh Kolli, and Thomas F. Wenisch
February 2022

Acknowledgments

We would like to thank our reviewer, Jishen Zhao, for her valuable feedback on this book. We would also like to acknowledge our colleagues at Google, notably David Culler, Rama Govindaraju, Jack Humphries, Samira Khan, and Parthasarathy Ranganathan, for their comments on the early draft of this synthesis lecture. Special thanks to Natalie Enright Jerger and Michael Morgan for their endless patience and constant encouragement that helped us finish this manuscript.

Vaibhav Gogte, Aasheesh Kolli, and Thomas F. Wenisch
February 2022