

A Primer on Memory Persistency

Synthesis Lectures on Computer Architecture

Editor

Natalie Enright Jerger, *University of Toronto*

Editor Emerita

Margaret Martonosi, *Princeton University*

Founding Editor Emeritus

Mark D. Hill, *University of Wisconsin, Madison*

Synthesis Lectures on Computer Architecture publishes 50- to 100-page books on topics pertaining to the science and art of designing, analyzing, selecting, and interconnecting hardware components to create computers that meet functional, performance, and cost goals. The scope will largely follow the purview of premier computer architecture conferences, such as ISCA, HPCA, MICRO, and ASPLOS.

[A Primer on Memory Persistency](#)

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

[In-/Near-Memory Computing](#)

Daichi Fujiki, Xiaowei Wang, Arun Subramaniyan, and Reetuparna Das
2021

[Robotic Computing on FPGAs](#)

Shaoshan Liu, Zishen Wan, Bo Yu, and Yu Wang
2021

[AI for Computer Architecture: Principles, Practice, and Prospects](#)

Lizhong Chen, Drew Penney, and Daniel Jiménez
2020

[Deep Learning Systems: Algorithms, Compilers, and Processors for Large-Scale Production](#)

Andres Rodriguez
2020

Parallel Processing, 1980 to 2020

Robert Kuhn and David Padua

2020

Data Orchestration in Deep Learning Accelerators

Tushar Krishna, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, and Ananda Samajdar

2020

Efficient Processing of Deep Neural Networks

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer

2020

Quantum Computer System: Research for Noisy Intermediate-Scale Quantum Computers

Yongshan Ding and Frederic T. Chong

2020

A Primer on Memory Consistency and Cache Coherence, Second Edition

Vijay Nagarajan, Daniel J. Sorin, Mark D. Hill, and David Wood

2020

Innovations in the Memory System

Rajeev Balasubramonian

2019

Cache Replacement Policies

Akanksha Jain and Calvin Lin

2019

The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition

Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan

2018

Principles of Secure Processor Architecture Design

Jakub Szefer

2018

General-Purpose Graphics Processor Architectures

Tor M. Aamodt, Wilson Wai Lun Fung, and Timothy G. Rogers

2018

Compiling Algorithms for Heterogenous Systems

Steven Bell, Jing Pu, James Hegarty, and Mark Horowitz

2018

Architectural and Operating System Support for Virtual Memory

Abhishek Bhattacharjee and Daniel Lustig
2017

Deep Learning for Computer Architects

Brandon Reagen, Robert Adolf, Paul Whatmough, Gu-Yeon Wei, and David Brooks
2017

On-Chip Networks, Second Edition

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh
2017

Space-Time Computing with Temporal Neural Networks

James E. Smith
2017

Hardware and Software Support for Virtualization

Edouard Bugnion, Jason Nieh, and Dan Tsafirir
2017

Datacenter Design and Management: A Computer Architect's Perspective

Benjamin C. Lee
2016

A Primer on Compression in the Memory Hierarchy

Somayeh Sardashti, Angelos Arelakis, Per Stenström, and David A. Wood
2015

Research Infrastructures for Hardware Accelerators

Yakun Sophia Shao and David Brooks
2015

Analyzing Analytics

Rajesh Bordawekar, Bob Blainey, and Ruchir Puri
2015

Customizable Computing

Yu-Ting Chen, Jason Cong, Michael Gill, Glenn Reinman, and Bingjun Xiao
2015

Die-stacking Architecture

Yuan Xie and Jishen Zhao
2015

Single-Instruction Multiple-Data Execution

Christopher J. Hughes
2015

Power-Efficient Computer Architectures: Recent Advances

Magnus Själander, Margaret Martonosi, and Stefanos Kaxiras
2014

FPGA-Accelerated Simulation of Computer Systems

Hari Angepat, Derek Chiou, Eric S. Chung, and James C. Hoe
2014

A Primer on Hardware Prefetching

Babak Falsafi and Thomas F. Wenisch
2014

On-Chip Photonic Interconnects: A Computer Architect's Perspective

Christopher J. Nitta, Matthew K. Farrens, and Venkatesh Akella
2013

Optimization and Mathematical Modeling in Computer Architecture

Tony Nowatzki, Michael Ferris, Karthikeyan Sankaralingam, Cristian Estan, Nilay Vaish, and David Wood
2013

Security Basics for Computer Architects

Ruby B. Lee
2013

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition

Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle
2013

Shared-Memory Synchronization

Michael L. Scott
2013

Resilient Architecture Design for Voltage Variation

Vijay Janapa Reddi and Meeta Sharma Gupta
2013

Multithreading Architecture

Mario Nemirovsky and Dean M. Tullsen
2013

Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU)

Hyesoon Kim, Richard Vuduc, Sara Baghsorkhi, Jee Choi, and Wen-mei Hwu
2012

Automatic Parallelization: An Overview of Fundamental Compiler Techniques

Samuel P. Midkiff

2012

Phase Change Memory: From Devices to Systems

Moinuddin K. Qureshi, Sudhanva Gurusurthi, and Bipin Rajendran

2011

Multi-Core Cache Hierarchies

Rajeev Balasubramonian, Norman P. Jouppi, and Naveen Muralimanohar

2011

A Primer on Memory Consistency and Cache Coherence

Daniel J. Sorin, Mark D. Hill, and David A. Wood

2011

Dynamic Binary Modification: Tools, Techniques, and Applications

Kim Hazelwood

2011

Quantum Computing for Computer Architects, Second Edition

Tzvetan S. Metodji, Arvin I. Faruque, and Frederic T. Chong

2011

High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities

Dennis Abts and John Kim

2011

Processor Microarchitecture: An Implementation Perspective

Antonio González, Fernando Latorre, and Grigorios Magklis

2010

Transactional Memory, Second Edition

Tim Harris, James Larus, and Ravi Rajwar

2010

Computer Architecture Performance Evaluation Methods

Lieven Eeckhout

2010

Introduction to Reconfigurable Supercomputing

Marco Lanzagorta, Stephen Bique, and Robert Rosenberg

2009

On-Chip Networks

Natalie Enright Jerger and Li-Shiuan Peh

2009

The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It

Bruce Jacob

2009

Fault Tolerant Computer Architecture

Daniel J. Sorin

2009

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines

Luiz André Barroso and Urs Hölzle

2009

Computer Architecture Techniques for Power-Efficiency

Stefanos Kaxiras and Margaret Martonosi

2008

Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency

Kunle Olukotun, Lance Hammond, and James Laudon

2007

Transactional Memory

James R. Larus and Ravi Rajwar

2006

Quantum Computing for Computer Architects

Tzvetan S. Metodi and Frederic T. Chong

2006

© Springer Nature Switzerland AG 2022
Reprint of original edition © Morgan & Claypool 2022

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

A Primer on Memory Persistency

Vaibhav Gogte, Aasheesh Kolli, and Thomas 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	xv
	Acknowledgments	xix
1	Persistent Memories	1
1.1	Introduction	1
1.2	Persistent Memory Technologies	2
1.3	Intel Optane DC Persistent Memory	4
1.3.1	Performance of Optane DC Persistent Memory	4
1.3.2	Modeling Persistent Memory Systems	5
1.4	System Configurations	6
1.4.1	Traditional Storage Systems	6
1.4.2	Persistent Memory System	8
1.4.3	Persistent Memory Support in Existing File Systems	9
1.4.4	Persistent Memory File Systems	9
1.4.5	Direct Access to Persistent Memories	10
2	Data Persistence	13
2.1	Ensuring Data Durability	13
2.1.1	Persistence at CPU Caches	14
2.1.2	Persistence at PM Controller	15
2.1.3	Persistence at Persistent Memory Device	16
2.1.4	ARMv8.2 ISA Extensions	17
2.2	Ordering Memory Operations to PM	17
2.2.1	Why is Ordering Required?	17
2.2.2	Sources of PM Operation Reordering	18
2.2.3	Applying Correct Memory Order to PM	20
3	Memory Persistency Models	21
3.1	Persistency Models	21
3.2	Recovery Observer	22
3.3	Strict Persistency	23

3.3.1	Buffered Strict Persistency Model	24
3.3.2	Formalizing the Strict Persistency Model	25
3.3.3	Persist Ordering Examples	25
3.4	Hardware Implementations of Strict Persistency	26
3.4.1	Naive Implementation	27
3.4.2	Bulk Persistence	27
3.4.3	Delegated Persistence	28
3.5	Drawbacks of Strict Persistency	30
3.6	Epoch Persistency Model	30
3.6.1	Strong Persist Atomicity	31
3.6.2	Buffered Epoch Persistency Model	32
3.6.3	Formalizing the Epoch Persistency Model	33
3.6.4	Persist Ordering Examples	34
3.7	Hardware Implementations of Epoch Persistency	35
3.7.1	Naive Implementation	36
3.7.2	Buffered Epoch Persistency in Hardware	37
3.7.3	Offline Conflict Resolution	38
3.7.4	Separate Ordering and Durability Barriers	41
3.8	Drawbacks of Epoch Persistency	43
3.9	Strand Persistency Model	43
3.9.1	Strong Persist Atomicity	44
3.9.2	Formalizing the Strand Persistency Model	44
3.9.3	Persist Ordering Examples	45
3.10	Hardware Implementations of Strand Persistency	47
4	Hardware Mechanisms for Atomic Durability	51
4.1	Failure Atomicity	51
4.1.1	Write-Ahead Logging	52
4.1.2	Log Structuring	55
4.1.3	Shadow Paging	55
4.2	Failure-Atomic Mechanisms	55
4.3	Hardware Undo Logging	56
4.3.1	Naive Implementation	56
4.3.2	Optimized Undo Logging	57
4.3.3	Software-Assisted Hardware Logging	59
4.4	Hardware Redo Logging	60
4.4.1	Optimized Redo Logging	61

4.4.2	Durable Hardware Transactions	62
4.5	Hardware Checkpointing Mechanisms	63
4.5.1	Coarse-Grained Checkpointing	63
4.5.2	Checkpointing on Power Failure	64
5	Programming Persistent Memory Systems	65
5.1	File Systems	65
5.1.1	Ext4 DAX	65
5.1.2	NOVA File System	67
5.1.3	SplitFS	68
5.1.4	Strata	69
5.2	Programming PM Systems	70
5.2.1	Transactional Failure Atomicity	70
5.2.2	Intel's PMDK libraries	71
5.2.3	Deferred Commit Transactions	72
5.2.4	Failure Atomicity for Outer-Critical Section	72
5.2.5	Failure Atomicity for Synchronization Free Region	73
5.3	Testing PM Applications	75
6	Conclusion	77
	Bibliography	79
	Authors' Biographies	95

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. Wensch
February 2022