JVM Performance Engineering

Inside OpenJDK and the HotSpot Java Virtual Machine

Monica Beckwith

Addison-Wesley

Hoboken, New Jersey

Cover image: Amiak / Shutterstock
Figures 7.8–7.18, 7.22–7.29: The Apache Software Foundation

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2024930211

ISBN-13: 978-0-13-465987-9
ISBN-10: 0-13-465987-2

$PrintCode

To my cherished companions, who have provided endless inspiration and comfort throughout the journey of writing this book:

In loving memory of Perl, Sherekhan, Cami, Mr. Spots, and Ruby. Their memories continue to guide and brighten my days with their lasting legacy of love and warmth.

And to Delphi, Calypso, Ivy, Selene, and little Bash, who continue to fill my life with joy, curiosity, and playful adventures. Their presence brings daily reminders of the beauty and wonder in the world around us.

This book is a tribute to all of them—those who have passed and those who are still by my side—celebrating the unconditional love and irreplaceable companionship they have graciously shared with me.

This page intentionally left blank

Contents

Preface

Welcome to my guide to JVM performance engineering, distilled from more than 20 years of expertise as a Java Champion and performance engineer. Within these pages lies a journey through the evolution of the JVM—a narrative that unfolds Java's robust capabilities and architectural prowess. This book meticulously navigates the intricacies of JVM internals and the art and science of performance engineering, examining everything from the inner workings of the HotSpot VM to the strategic adoption of modular programming. By asserting Java's pivotal role in modern computing—from server environments to the integration with exotic hardware—it stands as a beacon for practitioners and enthusiasts alike, heralding the next frontier in JVM performance engineering.

Intended Audience

This book is primarily written for Java developers and software engineers who are keen to enhance their understanding of JVM internals and performance tuning. It will also greatly benefit system architects and designers, providing them with insights into JVM's impact on system performance. Performance engineers and JVM tuners will find advanced techniques for optimizing JVM performance. Additionally, computer science and engineering students and educators will gain a comprehensive understanding of JVM's complexities and advanced features.

With the hope of furthering education in performance engineering, particularly with a focus on the JVM, this text also aligns with advanced courses on programming languages, algorithms, systems, computer architectures, and software engineering. I am passionate about fostering a deeper understanding of these concepts and excited about contributing to coursework that integrates the principles of JVM performance engineering and prepares the next generation of engineers with the knowledge and skills to excel in this critical area of technology.

Focusing on the intricacies and strengths of the language and runtime, this book offers a thorough dissection of Java's capabilities in concurrency, its strengths in multithreading, and the sophisticated memory management mechanisms that drive peak performance across varied environments.

Book Organization

Chapter 1, "The Performance Evolution of Java: The Language and the Virtual Machine," expertly traces Java's journey from its inception in the mid-1990s to the sophisticated advancements in Java 17. Highlighting Java's groundbreaking runtime environment, complete with the JVM, expansive class libraries, and a formidable set of tools, the chapter sets the stage for Java's innovative advancements, underlying technical excellence, continuous progress, and flexibility.

Key highlights include an examination of the OpenJDK HotSpot VM's transformative garbage collectors (GCs) and streamlined Java bytecode. This section illustrates Java's dedication to

performance, showcasing advanced JIT compilation and avant-garde optimization techniques. Additionally, the chapter explores the synergistic relationship between the HotSpot VM's client and server compilers, and their dynamic optimization capabilities, demonstrating Java's continuous pursuit of agility and efficiency.

Another focal point is the exploration of OpenJDK's memory management with the HotSpot GCs, particularly highlighting the adoption of the "weak generational hypothesis." This concept underpins the efficiency of collectors in HotSpot, employing parallel and concurrent GC threads as needed, ensuring peak memory optimization and application responsiveness.

The chapter maintains a balance between technical depth and accessibility, making it suitable for both seasoned Java developers and those new to the language. Practical examples and code snippets are interspersed to provide a hands-on understanding of the concepts discussed.

Chapter 2, "Performance Implications of Java's Type System Evolution," seamlessly continues from the performance focus of Chapter 1, delving into the heart of Java: its evolving type system. The chapter explores Java's foundational elements—primitive and reference types, interfaces, classes, and arrays—that anchored Java programming prior to Java SE 5.0.

The narrative continues with the transformative enhancements from Java SE 5.0 onward, such as the introduction of generics, annotations, and VarHandle type reference—all further enriching the language. The chapter spotlights recent additions such as switch expressions, sealed classes, and the much-anticipated records.

Special attention is given to Project Valhalla's ongoing work, examining the performance nuances of the existing type system and the potential of future value classes. The section offers insights into Project Valhalla's ongoing endeavors, from refined generics to the conceptualization of classes for basic primitives.

Java's type system is more than just a set of types—it's a reflection of Java's commitment to versatility, efficiency, and innovation. The goal of this chapter is to illuminate the type system's past, present, and promising future, fostering a profound understanding of its intricacies.

Chapter 3, "From Monolithic to Modular Java: A Retrospective and Ongoing Evolution," provides extensive coverage of the Java Platform Module System (JPMS) and its breakthrough impact on modular programming. This chapter marks Java's bold transition into the modular era, beginning with a fundamental exploration of modules. It offers hands-on guidance through the creation, compilation, and execution of modules, making it accessible even to newcomers in this domain.

Highlighting Java's transition from a monolithic JDK to a modular framework, the chapter reflects Java's adaptability to evolving needs and its commitment to innovation. A standout section of this chapter is the practical implementation of modular services using JDK 17, which navigates the intricacies of module interactions, from service providers to consumers, enriched by working examples. The chapter addresses key concepts like encapsulation of implementation details and the challenges of JAR hell, illustrating how Jigsaw layers offer elegant solutions in the modular landscape.

Further enriching this exploration, the chapter draws insightful comparisons with OSGi, spotlighting the parallels and distinctions, to give readers a comprehensive understanding of Java's

modular systems. The introduction of essential tools such as *jdeps*, *jlink*, *jdeprscan*, and *jmod*, integral to the modular ecosystem, is accompanied by thorough explanations and practical examples. This approach empowers readers to effectively utilize these tools in their developmental work.

Concluding with a reflection on the performance nuances of JPMS, the chapter looks forward to the future of Java's modular evolution, inviting readers to contemplate its potential impacts and developments.

Chapter 4, "The Unified Java Virtual Machine Logging Interface," delves into the vital yet often underappreciated world of logs in software development. It begins by underscoring the necessity of a unified logging system in Java, addressing the challenges posed by disparate logging systems and the myriad benefits of a cohesive approach. The chapter not only highlights the unification and infrastructure of the logging system but also emphasizes its role in monitoring performance and optimization.

The narrative explores the vast array of log tags and their specific roles, emphasizing the importance of creating comprehensive and insightful logs. In tackling the challenges of discerning any missing information, the chapter provides a lucid understanding of log levels, outputs, and decorators. The intricacies of these features are meticulously examined, with practical examples illuminating their application in tangible scenarios.

A key aspect of this chapter is the exploration of asynchronous logging, a critical feature for enhancing log performance with minimal impact on application efficiency. This feature is essential for developers seeking to balance comprehensive logging with system performance.

Concluding the chapter, the importance of logs as a diagnostic tool is emphasized, showcasing their role in both proactive system monitoring and reactive problem-solving. Chapter 4 not only highlights the power of effective logging in Java, but also underscores its significance in building and maintaining robust applications. This chapter reinforces the theme of Java's ongoing evolution, showcasing how advancements in logging contribute significantly to the language's capability and versatility in application development.

Chapter 5, "End-to-End Java Performance Optimization: Engineering Techniques and Microbenchmarking with JMH," focuses on the essence of performance engineering within the Java ecosystem. Emphasizing that performance transcends mere speed, this chapter highlights its critical role in crafting an unparalleled user experience. It commences with a formative exploration of performance engineering's pivotal role within the broader software development realm, highlighting its status as a fundamental quality attribute and unraveling its multifaceted layers.

With precision, the chapter delineates the metrics pivotal to gauging Java's performance, encompassing aspects from footprint to the nuances of availability, ensuring readers grasp the full spectrum of performance dynamics. Stepping in further. It explores the intricacies of response time and its symbiotic relationship with availability. This inspection provides insights into the mechanics of application timelines, intricately weaving the narrative of response time, throughput, and the inevitable pauses that punctuate them.

Yet, the performance narrative is only complete by acknowledging the profound influence of hardware. This chapter decodes the symbiotic relationship between hardware and software,

emphasizing the harmonious symphony that arises from the confluence of languages, processors, and memory models. From the subtleties of memory models and their bearing on thread dynamics to the foundational principles of Java Memory Model, this chapter journeys through the maze of concurrent hardware, shedding light on the order mechanisms pivotal to concurrent computing.

Moving beyond theoretical discussions, this chapter draws on over two decades of hands-on experience in performance optimization. It introduces a systematic approach to performance diagnostics and analysis, offering insights into methodologies and a detailed investigation of subsystems and approaches to identifying potential performance issues. The methodologies are not only vital for software developers focused on performance optimization but also provide valuable insights into the intricate relationship between underlying hardware, software stacks, and application performance.

The chapter emphasizes the importance of a structured benchmarking regime, encompassing everything from memory management to the assessment of feature releases and system layers. This sets the stage for the Java Micro-Benchmark Suite (JMH), the *pièce de résistance* of JVM benchmarking. From its foundational setup to the intricacies of its myriad features, the journey encompasses the genesis of writing benchmarks, to their execution, enriched with insights into benchmarking modes, profilers, and JMH's pivotal annotations.

Chapter 5 thus serves as a comprehensive guide to end-to-end Java performance optimization and as a launchpad for further chapters. It inspires a fervor for relentless optimization and arms readers with the knowledge and tools required to unlock Java's unparalleled performance potential.

Memory management is the silent guardian of Java applications, often operating behind the scenes but crucial to their success. **Chapter 6**, "Advanced Memory Management and Garbage Collection in OpenJDK," marks a deep dive into specialized JVM improvements, showcasing advanced performance tools and techniques. This chapter offers a leap into the world of garbage collection, unraveling the techniques and innovations that ensure Java applications run efficiently and effectively.

The chapter commences with a foundational overview of garbage collection in Java, setting the stage for the detailed exploration of Thread-Local Allocation Buffers (TLABs) and Promotion Local Allocation Buffers (PLABs), and elucidating their pivotal roles in memory management. As we progress, the chapter sheds light on optimizing memory access, emphasizing the significance of the NUMA-aware garbage collection and its impact on performance.

The highlight of this chapter lies in its exploration of advanced garbage collection techniques. The narrative reviews the G1 Garbage Collector (G1 GC), unraveling its revolutionary approach to heap management. From grasping the advantages of a regionalized heap to optimizing G1 GC parameters for peak performance, this section promises a holistic cognizance of one of Java's most advanced garbage collectors. Additionally, the Z Garbage Collector (ZGC) is presented as a technological marvel with its adaptive optimization techniques, and the advancements that make it a game-changer in real-time applications.

This chapter also offers insights into the emerging trends in garbage collection, setting the stage for what lies ahead. Practicality remains at the forefront, with a dedicated section offering invaluable tips for evaluating GC performance. From sympathizing with various workloads, such as Online Analytical Processing (OLAP) to Online Transaction Processing (OLTP) and Hybrid Transactional/Analytical Processing (HTAP), to synthesizing live data set pressure and data lifespan patterns, the chapter equips readers with the apparatus and knowledge to optimize memory management effectively. This chapter is an accessible guide to advanced garbage collection techniques that Java professionals need to navigate the topography of memory management.

Chapter 7, "Runtime Performance Optimizations: A Focus on Strings, Locks, and Beyond," is dedicated to exploring the critical facets of Java's runtime performance, particularly in the realms of string handling and lock synchronization—two areas essential for efficient application performance.

The chapter excels at taking a comprehensive approach to demystifying these JVM optimizations through detailed under-the-hood analysis—utilizing a range of profiling techniques, from bytecode analysis to memory and sample-based profiling to gathering call stack views of profiled methods—to enrich the reader's understanding. Additionally, the chapter leverages JMH benchmarking to highlight the tangible improvements such optimizations bring. The practical use of *async-profiler* for method-level insights and NetBeans memory profiler further enhances the reader's granular understanding of the JVM enhancements. This chapter aims to test and illuminate the optimizations, equipping readers with a comprehensive approach to using these tools effectively, thereby building on the performance engineering methodologies and processes discussed in Chapter 5.

The journey continues with an extensive review of the string optimizations in Java, highlighting major advancements across various Java versions, and then shifts focus onto enhanced multithreading performance, highlighting Java's thread synchronization mechanisms.

Further, the chapter helps navigate the world of concurrency, with discussion of the transition from the thread-per-task model to the scalable thread-per-request model. The examination of Java's Executor Service, ThreadPools, ForkJoinPool framework, and CompletableFuture ensures a robust comprehension of Java's concurrency mechanisms.

The chapter concludes with a glimpse into the future of concurrency in Java with virtual threads. From understanding virtual threads and their carriers to discussing parallelism and integration with existing APIs, this chapter is a practical guide to advanced concurrency mechanisms and string optimizations in Java.

Chapter 8, "Accelerating Time to Steady State with OpenJDK HotSpot VM," is dedicated to optimizing start-up to steady-state performance, crucial for transient applications such as containerized environments, serverless architectures, and microservices. The chapter emphasizes the importance of minimizing JVM start-up and warm-up time to enhance efficient execution, incorporating a pivotal exploration into GraalVM's revolutionary role in this domain.

The narrative dissects the phases of JVM start-up and the journey to an application's steady-state, highlighting the significance of managing state during these phases across various

architectures. An in-depth look at Class Data Sharing (CDS) sheds light on shared archive files and memory mapping, underscoring the advantages in multi-instance setups. The narrative then shifts to ahead-of-time (AOT) compilation, contrasting it with just-in-time (JIT) compilation and detailing the transformative impact of HotSpot VM's Project Leyden and its forecasted ability to manage states via CDS and AOT. This sets the stage for GraalVM and its revolutionary impact on Java's performance landscape. By harnessing advanced optimization techniques, including static images and dynamic compilation, GraalVM enhances performance for a wide array of applications. The exploration of cutting-edge technologies like GraalVM alongside a holistic survey of OpenJDK projects such as CRIU and CraC, which introduce groundbreaking checkpoint/restore functionality, adds depth to the discussion. This comprehensive coverage provides insights into the evolving strategies for optimizing Java applications, making this chapter an invaluable resource for developers looking to navigate today's cloud native environments.

The final chapter, **Chapter 9**, "Harnessing Exotic Hardware: The Future of JVM Performance Engineering," focuses on the fascinating intersection of exotic hardware and the JVM, illuminating its galvanizing impact on performance engineering. This chapter begins with an introduction to the increasingly prominent world of exotic hardware, particularly within cloud environments. It explores the integration of this hardware with the JVM, underscoring the pivotal role of language design and toolchains in this process.

Through a series of carefully detailed case studies, the chapter showcases the real-world applications and challenges of integrating such hardware accelerators. From the Lightweight Java Game Library (LWJGL), to the innovative Aparapi, which bridges Java and OpenCL, each study offers valuable insights into the complexities and triumphs of these integrations. The chapter also examines Project Sumatra's significant contributions to this realm and introduces TornadoVM, a specialized JVM tailored for hardware accelerators.

Through these case studies, the symbiotic potential of integrating exotic hardware with the JVM becomes increasingly evident, leading up to an overview of Project Panama, heralding a new horizon in JVM performance engineering. At the heart of Project Panama lies the Vector API, a symbol of innovation designed for vector computations. This API is not just about computations—it's about ensuring they are efficiently vectorized and tailored for hardware that thrives on vector operations. This ensures that developers have the tools to express parallel computations optimized for diverse hardware architectures. But Panama isn't just about vectors. The Foreign Function and Memory API emerges as a pivotal tool, a bridge that allows Java to converse seamlessly with native libraries. This is Java's answer to the age-old challenge of interoperability, ensuring Java applications can interface effortlessly with native code, breaking language barriers.

Yet, the integration is no walk in the park. From managing intricate memory access patterns to deciphering hardware-specific behaviors, the path to optimization is laden with complexities. But these challenges drive innovation, pushing the boundaries of what's possible. Looking to the future, the chapter showcases my vision of Project Panama as the gold standard for JVM interoperability. The horizon looks promising, with Panama poised to redefine performance and efficiency for Java applications.

This isn't just about the present or the imminent future. The world of JVM performance engineering is on the cusp of a revolution. Innovations are knocking at our door, waiting to be embraced—with Tornado VM's Hybrid APIs, and with HAT toolkit and Project Babylon on the horizon.

How to Use This Book

1. *Sequential Reading for Comprehensive Understanding:* This book is designed to be read from beginning to end, as each chapter builds upon the knowledge of the previous ones. This approach is especially recommended for readers new to JVM performance engineering.
2. *Modular Approach for Specific Topics:* Experienced readers may prefer to jump directly to chapters that address their specific interests or challenges. The table of contents and index can guide you to relevant sections.
3. *Practical Examples and Code:* Throughout the book, practical examples and code snippets are provided to illustrate key concepts. To get the most out of these examples, readers are encouraged to build on and run the code themselves. (See item 5.)
4. *Visual Aids for Enhanced Understanding:* In addition to written explanations, this book employs a variety of textual and visual aids to deepen your understanding.
    a. *Case Studies:* Real-world scenarios that demonstrate the application of JVM performance techniques.
    b. *Screenshots:* Visual outputs depicting profiling results as well as various GC plots, which are essential for understanding the GC process and phases.
    c. *Use-Case Diagrams:* Visual representations that map out the system's functional requirements, showing how different entities interact with each other.
    d. *Block Diagrams:* Illustrations that outline the architecture of a particular JVM or system component, highlighting performance features.
    e. *Class Diagrams:* Detailed object-oriented designs of various code examples, showing relationships and hierarchies.
    f. *Process Flowcharts:* Step-by-step diagrams that walk you through various performance optimization processes and components.
    g. *Timelines:* Visual representations of the different phases or state changes in an activity and the sequence of actions that are taken.
5. *Utilizing the Companion GitHub Repository:* A significant portion of the book's value lies in its practical application. To facilitate this, I have created JVM Performance Engineering GitHub Repository (https://github.com/mo-beck/JVM-Performance-Engineering). Here, you will find
    a. *Complete Code Listings:* All the code snippets and scripts mentioned in the book are available. This allows you to see the code and experiment with it. Use it as a launchpad for your projects and fork and improve it.
    b. *Additional Resources and Updates:* The field of JVM Performance Engineering is ever evolving. The repository will be periodically updated with new scripts, resources, and information to keep you abreast of the latest developments.
    c. *Interactive Learning:* Engage with the material by cloning the repository, running the GC scripts against your GC log files, and modifying them to see how outcomes better suit your GC learning and understanding journey.

6. *Engage with the Community:* I encourage readers to engage with the wider community. Use the GitHub repository to contribute your ideas, ask questions, and share your insights. This collaborative approach enriches the learning experience for everyone involved.
7. *Feedback and Suggestions:* Your feedback is invaluable. If you have suggestions, corrections, or insights, I warmly invite you to share them. You can provide feedback via the GitHub repository, via email (jvmbook@codekaram.com), or via social media platforms (https://www.linkedin.com/in/monicabeckwith/ or https://twitter.com/JVMPerfEngineer).

In Java's vast realm, my tale takes wing,
A narrative so vivid, of wonders I sing.
Distributed systems, both near and afar,
With JVM shining—the brightest star!

Its rise through the ages, a saga profound,
With each chronicle, inquiries resound.
"Where lies the wisdom, the legends so grand?"
They ask with a fervor, eager to understand.

This book is a beacon for all who pursue,
A tapestry of insights, both aged and new.
In chapters that flow, like streams to the seas,
I share my heart's journey, my tech odyssey.

—Monica Beckwith

Acknowledgments

Reflecting on the journey of creating this book, my heart is full of gratitude for the many individuals whose support, expertise, and encouragement have been the wind beneath my wings.

At the forefront of my gratitude is my family—the unwavering pillars of support. To my husband, Ben: Your understanding and belief in my work, coupled with your boundless love and care, have been the bedrock of my perseverance.

To my children, Annika and Bodin: Your patience and resilience have been my inspiration. Balancing the demands of a teen's life with the years it took to bring this book to fruition, you have shown a maturity and understanding well beyond your years. Your support, whether it be a kind word at just the right moment or understanding my need for quiet as I wrestled with complex ideas, has meant more to me than words can express. Your unwavering faith, even when my work required sacrifices from us all, has been a source of strength and motivation. I am incredibly proud of the kind and supportive individuals you are becoming, and I hope this book reflects the values we cherish as a family.

Editorial Guidance

A special word of thanks goes to my executive editor at Pearson, Greg Doench, whose patience has been nothing short of saintly. Over the years, through health challenges, the dynamic nature of JVM release cycles and project developments, and the unprecedented times of COVID, Greg has been a beacon of encouragement. His unwavering support and absence of frustration in the face of my tardiness have been nothing less than extraordinary. Greg, your steadfast presence and guidance have not only helped shape this manuscript but have also been a personal comfort.

Chapter Contributions

The richness of this book's content is a culmination of my extensive work, research, and insights in the field, enriched further by the invaluable contributions of various experts, colleagues, collaborators, and friends. Their collective knowledge, feedback, and support have been instrumental in adding depth and clarity to each topic discussed, reflecting years of dedicated expertise in this domain.

- In Chapter 3, Nikita Lipski's deep experience in Java modularity added compelling depth, particularly on the topics of the JAR hell versioning issues, layers, and his remarkable insights on OSGi.
- Stefano Doni's enriched field expertise in Quality of Service (QoS), performance stack, and theoretical expertise in operational laws and queueing, significantly enhanced Chapter 5, bringing a blend of theoretical and practical perspectives.
- The insights and collaborative interactions with Per Liden and Stefan Karlsson were crucial in refining my exploration of the Z Garbage Collector (ZGC) in Chapter 6. Per's

numerous talks and blog posts have also been instrumental in helping the community understand the intricacies of ZGC in greater detail.

- Chapter 7 benefitted from the combined insights of Alan Bateman and Heinz Kabutz. Alan was instrumental in helping me refine this chapter's coverage of Java's locking mechanisms and virtual threads. His insights helped clarify complex concepts, added depth to the discussion of monitor locks, and provided valuable perspective on the evolution of Java's concurrency model. Heinz's thorough review ensured the relevance and accuracy of the content.
- For Chapter 8, Ludovic Henry's insistence on clarity with respect to the various terminologies and persistence to include advanced topics and Alina Yurenko's insights into GraalVM and its future developments provided depth and foresight and reshaped the chapter to its glorious state today.

  Alina has also influenced me to track the developments in GraalVM—especially the introduction of layered native images, which promises to reduce build times and enable sharing of base images.
- Last but not the least, for Chapter 9, I am grateful to Gary Frost for his thorough review of Aparapi, Project Sumatra, and insights on leveraging the latest JDK (early access) version for developing projects like Project Panama. Dr. Juan Fumero's leadership in the development of TornadoVM and insights into parallel programming challenges have been instrumental in providing relevant insights for my chapter, deepening its clarity, and enhancing its narrative.

  It was a revelation to see our visions converge and witness these industry stalwarts drive the enhancements in the integration of Java with modern hardware accelerators.

Mentors, Influencers, and Friends

Several mentors, leaders, and friends have significantly influenced my broader understanding of technology:

- Charlie Hunt's guidance in my GC performance engineering journey has been foundational. His groundbreaking work on String density has inspired many of my own approaches with methodologies and process. His seminal work *Java Performance* is an essential resource for all performance enthusiasts and is highly recommended for its depth and insight.
- Gil Tene's work on the C4 Garbage Collector and his educational contributions have deeply influenced my perspective on low pause collectors and their interactive nature. I value our check-ins, which I took as mentorship opportunities to learn from one of the brightest minds.
- Thomas Schatzl's generous insights on the G1 Garbage Collector have added depth and context to this area of study, enriching my understanding following my earlier work on G1 GC. Thomas is a GC performance expert whose work, including on Parallel GC, continues to inspire me.

- Vladimir Kozlov's leadership and work in various aspects of the HotSpot JVM have been crucial in pushing the boundaries of Java's performance capabilities. I cherish our work together on prefetching, tiered thresholds, various code generation, and JVM optimizations, and I appreciate his dedication to HotSpot VM.
- Kirk Pepperdine, for our ongoing collaborations that span from the early days of developing G1 GC parser scripts for our joint hands-on lab sessions at JavaOne, to our recent methodologies, processes, and benchmarking endeavors at Microsoft, continuously pushes the envelope in performance engineering.
- Sergey Kuksenko and Alexey Shipilev, along with my fellow JVM performance engineering experts, have been my comrades in relentless pursuit of Java performance optimizations.
- Erik Österlund's development of generational ZGC represents an exciting and forward-looking aspect of garbage collection technology.
- John Rose, for his unparalleled expertise in JVM internals and his pivotal role in the evolution of Java as a language and platform. His vision and deep technical knowledge have not only propelled the field forward but also provided me with invaluable insights throughout my career.

Each of these individuals has not only contributed to the technical depth and richness of this book but also played a vital role in my personal and professional growth. Their collective wisdom, expertise, and support have been instrumental in shaping both the content and the journey of this book, reflecting the collaborative spirit of the Java community.

About the Author

Monica Beckwith is a leading figure in Java Virtual Machine (JVM) performance tuning and optimizations. With a strong Electrical and Computer Engineering academic foundation, Monica has carved out an illustrious, impactful, and inspiring professional journey.

At Advanced Micro Devices (AMD), Monica refined her expertise in Java, JVM, and systems performance engineering. Her work brought critical insights to NUMA's architectural enhancements, improving both hardware and JVM performance through optimized code generation, improved footprint and advanced JVM techniques, and memory management. She continued her professional growth at Sun Microsystems, contributing significantly to JVM performance enhancements across Sun SPARC, Solaris, and Linux, aiding in the evolution of a scalable Java ecosystem.

Monica's role as a Java Champion and coauthor of *Java Performance Companion*, as well as authoring this current book, highlight her steadfast commitment to the Java community. Notably, her work in the optimization of G1 Garbage Collector went beyond optimization; she delved into diagnosing pain points, fine-tuning processes, and identifying critical areas for enhancement, thereby setting new precedents in JVM performance. Her expertise not only elevated the efficiency of the G1 GC but also showcased her intricate knowledge of JVM's complexities. At Arm, as a managed runtimes performance architect, Monica played a key role in shaping a unified strategy for the Arm ecosystem, fostering a competitive edge for performance on Arm-based servers.

Monica's significant contributions and thought leadership have enriched the broader tech community. Monica serves on the program committee for various prestigious conferences and hosts JVM and performance-themed tracks, further emphasizing her commitment to knowledge sharing and community building.

At Microsoft, Monica's expertise shines brightly as she optimizes JVM-based workloads, applications, and key services, across a diverse range of deployment scenarios, from bare metal to sophisticated Azure VMs. Her deep-seated understanding of hardware and software engineering, combined with her adeptness in systems engineering and benchmarking principles, uniquely positions her at the critical juncture of the hardware and software. This position enables her to significantly contribute to the performance, scalability and power efficiency characterization, evaluation, and analysis of both current and emerging hardware systems within the Azure Compute infrastructure.

Beyond her technical prowess, Monica embodies values that resonate deeply with those around her. She is a beacon of integrity, authenticity, and continuous learning. Her belief in the transformative power of actions, the sanctity of reflection, and the profound impact of empathy defines her interactions and approach. A passionate speaker, Monica's commitment to lifelong learning is evident in her zeal for delivering talks and disseminating knowledge.

Outside the confines of the tech world, Monica's dedication extends to nurturing young minds as a First Lego League coach. This multifaceted persona, combined with her roles as a Java Champion, author, and performance engineer at Microsoft, cements her reputation as a respected figure in the tech community and a source of inspiration for many.