Java Magazine

JVM Internals

Arm resources show how to optimize your Java applications for AArch64

Alan Zeichick | December 2, 2021
Editor in Chief, Java Magazine

Choosing the best runtime switches can greatly affect the performance of your workload.

Java developers don't need to worry about the deployment hardware as long as a first-class JVM is available, and that's certainly the case for Arm processors. Whether you're writing the code or running the code, it's business as usual: At the end of the day, it's plain old Java bytecode.

Of course, many Java developers and systems administrators want to know more, and there are several excellent resources, especially two posts from Arm.

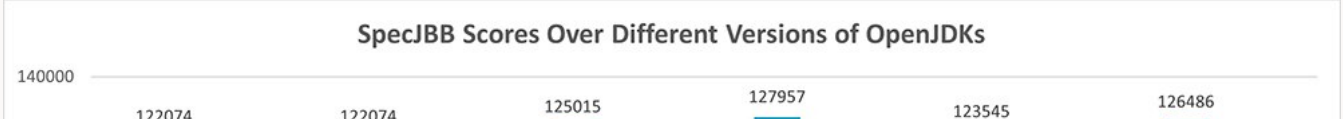
Java performance on Ampere A1 Compute

Arguably the most important post for the deployment of processor-intensive applications is a November 2021 post from Arm senior performance engineer Shiyou (Alan) Huang. His post, "Improving Java performance on OCI Ampere A1 Compute instances," begins with the following:

> Oracle Cloud Infrastructure (OCI) has recently launched the Ampere A1 Compute family of Arm Neoverse N1-based VMs and bare-metal instances. These A1 instances use Ampere Altra CPUs that were designed specifically to deliver performance, scalability, and security for cloud applications. The A1 Flex VM family supports an unmatched number of VM shapes that can be configured with 1-80 cores and 1-512GB of RAM (up to 64GB per core). Ampere A1 compute is also offered in bare-metal configurations with up to 2-sockets and 160-cores per instance.
>
> In this blog, we investigate the performance of Java using SPECjbb2015 on OCI A1 instances. We tuned SPECjbb2015 for best performance by referring to the configurations used by the online SPECjbb submissions. Those Java options may not apply to all Java workloads due to the very large heap size and other unrealistic options. The goal here is to see the best scores we can achieve on A1 using SPECjbb. We compared the performance results of SPECjbb2015 over different versions of OpenJDKs to identify a list of patches that improve the performance. As SPECjbb is a latency-sensitive workload, we also presented the impact of Arm LSE (Large System Extensions) on the performance in this blog.

Huang's paper presents two metrics to evaluate the performance of a JVM: `max-jOPS` for throughput and `critical-jOPS` for critical throughput under service-level agreements (SLAs).

One of Huang's charts, reproduced below as **Figure 1**, shows the SPECjbb scores from OpenJDK 11 to 16 using Arm's tuned configurations.



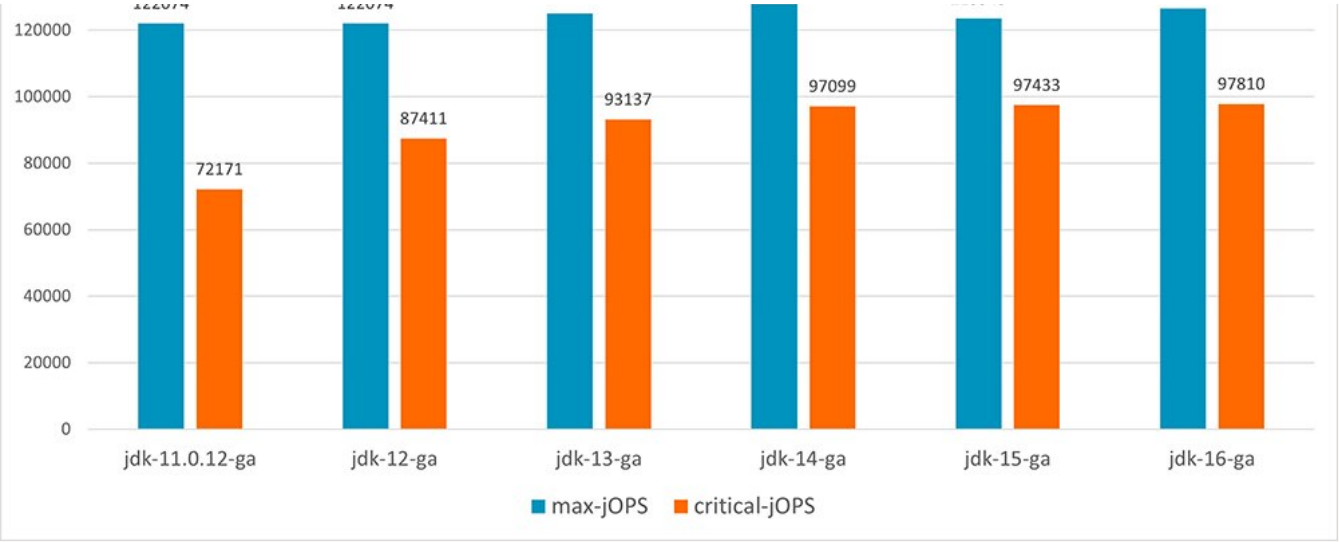


Figure 1. SPECjbb scores from OpenJDK 11 to 16 using Arm's tuned configurations

Java performance on Neoverse N1 systems

Huang wrote another relevant blog post, " Improving Java performance on Neoverse N1 systems," in July 2021.

In this post, Huang writes the following:

> …we investigate Java performance using an enterprise benchmark on Arm's Neoverse N1 based CPU, which has demonstrated huge cost/power/performance improvements in server CPUs. We built OpenJDK from the source code with different compiling flags to test the impact of LSE (Large System Extensions), the new architecture extension introduced in ARMv8.1. The Java enterprise benchmark we tested provides two throughput-based metrics: maximum throughput and throughput under service level agreement (SLA). We tuned performance by tweaking an initial set of Java runtime flags and augmented system parameters using an automated optimization tool.

Huang summarizes the tests as follows: "As we can see, tuning is critical to the performance of Java applications. The throughput is improved by 40% and the SLA score for more than 80% after the tuning configurations are applied" (as shown in **Figure 2**).

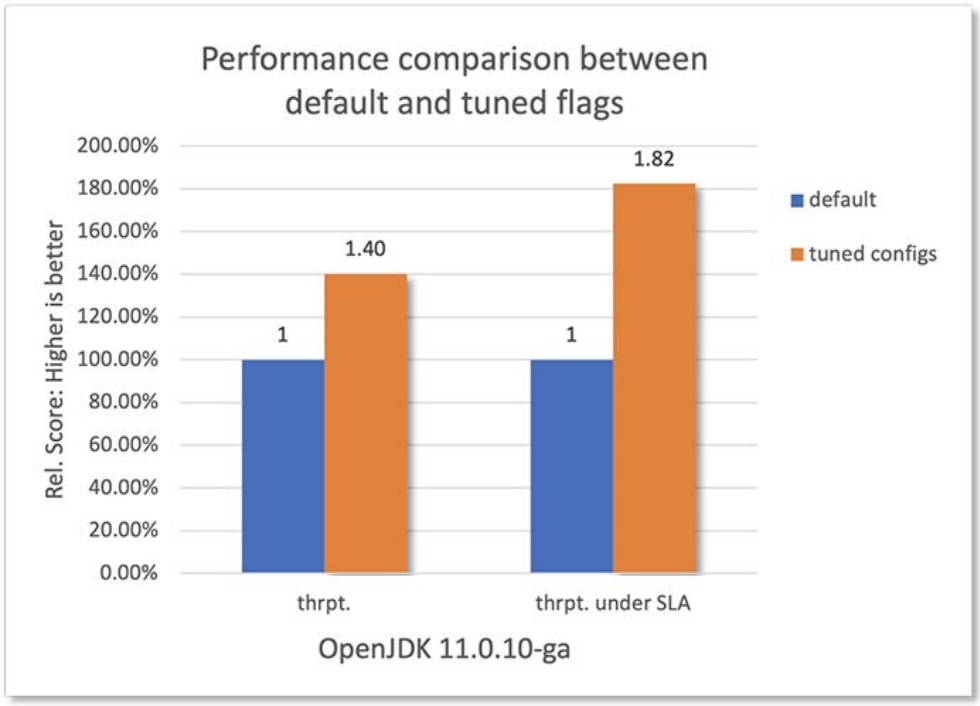


Figure 2. Tuning the JVM's runtime flags can greatly impact performance.

Large System Extensions

A key section of both of Huang's blog posts is his discussion of Arm's Large System Extensions (LSE), which first appeared in the Arm 8.1 architecture. He describes LSE as "a set of atomic operations such as compare-and-swap (CAS), atomic load and increment (LDADD). Most of the new operations have load-acquire or store-release semantics which are low-cost atomic operations comparing to the legacy way to implement atomic operations through pairs of load/store exclusives."

Huang adds that in his experiments, "We see that LSE benefits the workloads that are highly concurrent and use heavy synchronizations," as seen in **Figure 3**.

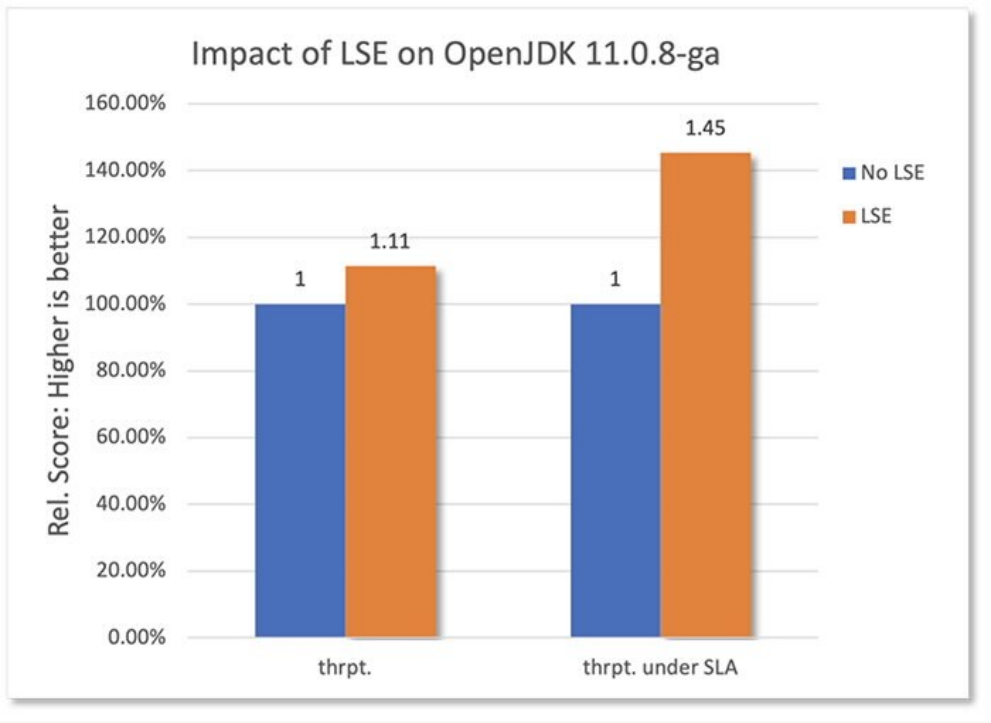


Figure 3. The impact of Arm's LSE

Relevant reading beyond the Arm posts

- Java on Arm processors: Understanding AArch64 vs. x86
- Java on Arm: The AArch64 hardware, software, cloud, and JDK
- Update on 64-bit ARM Support for Oracle OpenJDK and Oracle JDK
- Moving to Ampere A1 Compute instances on Oracle Cloud Infrastructure (OCI)
- Deploy Java applications on Ampere A1 on Oracle Cloud Infrastructure
- Arm-based cloud computing is the next big thing: Introducing Arm on Oracle Cloud Infrastructure
- Enabling Java for Windows on Arm64
- Oracle's Ampere A1 Arm VMs: First impressions

Alan Zeichick
Editor in Chief, Java Magazine

Alan Zeichick is editor in chief of *Java Magazine* and editor at large of Oracle's Content Central group. A former mainframe software developer and technology analyst, Alan has previously been the editor of *AI Expert, Network Magazine*, *Software Development Times*, *Eclipse Review*, and *Software Test & Performance*. Follow him on Twitter @zeichick.

Previous Post

Quiz yourself: Functional interfaces and error handling in Java

Mikalai Zaikin | 5 min read

Next Post

Fight ambiguity and improve your code with Java 17's sealed classes

Mala Gupta | 14 min read