White Paper

Intel® Intelligent Storage Acceleration Library

Accelerating Compression Workloads: A zlib Shim Using Intel® Intelligent Storage Acceleration Library (Intel® ISA-L)

Authors

Veronika Karpenko

Pablo De Lara Guarch

Tomasz Kantecki

Intel Corporation

Table of Contents

1. Introduction

The zlib compression library has become ubiquitous across modern computing infrastructure, serving as the backbone for data compression in countless applications. As throughput requirements increase, the need for accelerated compression solutions has become critical. Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) addresses these challenges by providing optimized implementations of DEFLATE compression algorithm that leverage modern CPU instruction sets to deliver measured throughput improvements while intercepting zlib API through a shim layer approach.

The Intel ISA-L shim serves as an invaluable testing tool, enabling developers to easily evaluate their zlib-dependent applications with Intel ISA-L's enhanced performance characteristics without requiring code modifications. This drop-in replacement capability allows development teams to conduct comprehensive performance benchmarking, compatibility testing and validation in their existing environments, providing the data and benefits of migrating to Intel ISA-L.

This whitepaper outlines Intel ISA-L shim implementation, use cases and performance benchmarks, API coverage, and deployment best practices.

1.1 Problem Statement

Zlib-dependent applications cannot leverage Intel ISA-L performance benefits without code modifications and costly integration efforts. Intel ISA-L provides faster compression/decompression through a different API which has created the need for a drop-in shim library.

1.2 Solution Overview

Modern applications increasingly rely on data compression for storage efficiency, network transmission, and overall performance optimization. Zlib currently serves as the default standard compression library for DEFLATE algorithm.

This whitepaper presents a guide for utilizing Intel ISA-L through the Intel ISA-L shim library to intercept zlib API without having to make modifications in zlib-dependent applications. Intel ISA-L optimizes the implementation of DEFLATE algorithm and provides a drop-in replacement that partially replicates zlib's API while internally using Intel ISA-L's optimized compression and decompression functions to deliver performance improvements.

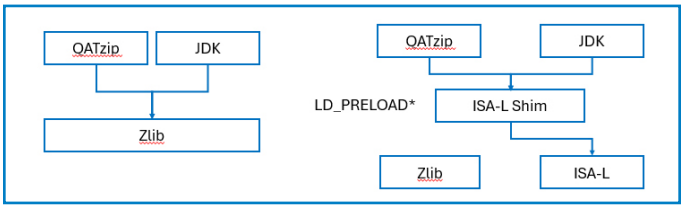

Figure 1. Zlib-dependent workloads can preload Intel® ISA-L shim to intercept zlib without any code modifications.

1.3 Key Benefits of Intel ISA-L Shim Library

- **Compatibility:** Thousands of existing applications that depend on zlib's API can use the shim library by simply preloading it through LD_PRELOAD mechanism for immediate performance testing.
- **Integration:** Zero code changes and minimal integration effort required while also avoiding the need to upstream changes to zlib-dependent applications.
- **Performance Gains:** Improvement in compression/decompression throughput leveraging Intel Instruction Set Architecture optimizations.
- **API Coverage:** Intel ISA-L's structures and functions map to zlib's z_stream interface and functions. Support for deflate, inflate and other zlib functions. More information in 3.2 Intercepted Functions Mapping section.

1.4 Target Applications

Intel ISA-L is designed for applications demanding high-throughput data compression and decompression.

Intel ISA-L supports both compression and decompression workloads, offering compression levels 1 through 3 that deliver performance comparable to zlib's equivalent levels while providing significantly higher throughput. This makes Intel ISA-L particularly well-suited for performance-critical scenarios where data processing speed is critical.

2. Technical Background

2.1 zlib

The zlib compression library, first released in 1995, implements the DEFLATE algorithm as specified in RFC 1951, which combines Lempel-Ziv 1977 (LZ77) sliding window compression with Huffman coding to achieve effective general-purpose data compression.

Zlib delivers excellent compression ratios across diverse data types and maintains broad cross-platform compatibility and adoption in thousands of existing applications. Zlib's design can present performance bottlenecks in modern high-throughput applications.

Zlib-ng attempts to address some of the performance limitations of zlib through optimized implementations, but Intel ISA-L generally outperforms zlib-ng too. The performance testing and comparison in this document will focus on zlib and Intel ISA-L.

2.2 Intel® Intelligent Storage Acceleration Library (Intel® ISA-L)

Intel ISA-L is a collection of optimized low-level functions specifically designed to accelerate storage and data-intensive applications. Leveraging Intel's advanced instruction sets and architecture-specific optimizations, Intel ISA-L delivers high-performance implementations of common storage operations with minimal CPU overhead.

Intel ISA-L includes:

- Compression - Fast DEFLATE-compatible data compression.
- De-compression - Fast inflate-compatible data decompression.
- igzip - A command line application like gzip, accelerated with Intel ISA-L.

Intel ISA-L represents a significant advancement in accelerated compression by providing an optimized implementation of the DEFLATE algorithm that takes full advantage of modern Intel processors.

2.3 LD_PRELOAD Mechanism

The LD_PRELOAD environment variable provides a powerful mechanism for dynamically replacing shared libraries at runtime without requiring application recompilation or modification. When set, LD_PRELOAD instructs the dynamic linker to load specified libraries before any other shared libraries, effectively allowing symbol interposition where functions in the preloaded library override those in subsequently loaded libraries.

In the context of replacing zlib with Intel ISA-L, this mechanism enables seamless substitution by implementing an Intel ISA-L shim library that exports the same API symbols as zlib (such as deflate(), inflate(), compress(), and uncompress()) while internally leveraging Intel ISA-L's optimized compression algorithms. Applications can benefit from Intel ISA-L's performance improvements transparently by simply setting LD_PRELOAD=./isal-shim. so before execution, making this approach particularly valuable for any zlib-dependent applications, third-party binaries, or environments where code modifications are impractical or impossible.

3. Shim Library Architecture

3.1 Design Principles

API Compatibility and Integration

The Intel ISA-L shim library is a drop-in replacement for zlib, maintaining partial API compatibility with zlib. This design allows applications to immediately benefit from Intel ISA-L acceleration without requiring any source code modifications, recompilation, or configuration changes. Intel ISA-L uses a different API structure (struct isal_zstream) compared to zlib's z_stream interface, requiring translation between them.

Performance Optimization

Intel ISA-L shim focuses on maximizing throughput by leveraging modern CPU instruction sets to deliver substantial performance improvements. These

optimizations make Intel ISA-L particularly valuable for high-throughput applications, data centres, and any workload where compression/decompression has a significant impact.

3.2 Intercepted Functions Mapping

The Intel ISA-L shim intercepts key zlib compression and decompression functions to provide transparent performance acceleration. Functions NOT intercepted use the applications zlib implementation, ensuring compatibility for any zlib functionality not yet implemented in the shim layer.

Deflate/Inflate and related functions

- deflateInit, deflateInit2, deflateSetDictionary, deflate, deflateEnd, deflateSetHeader
- inflateInit, inflateInit2, inflateSetDictionary, inflate, inflateEnd, inflateReset

Utility functions

- compress, uncompress
- compress2, uncompress2

Checksum functions

- crc32, adler32

3.3 Compression Level Mapping

Intel ISA-L has three compression levels vs zlib's 10 levels (0-9), so multiple zlib levels map to each Intel ISA-L level in the shim.

	Zlib	Intel® ISA-L shim
Compression Level	1,3	1
	4-6, -1	2
	7-9	3
	0	Not Supported

This mapping provides the closest reasonable approximation of zlib's 10 compression levels to Intel ISA-L's three levels.

3.4 Compression Ratio

Although Intel ISA-L and zlib both implement the DEFLATE algorithm, their compression-level scales behave differently. At zlib level 1 compared with Intel ISA-L level 1 the resulting compression ratios are generally equivalent. However, as compression levels increase, the mapping between the two libraries becomes less direct. This can lead to observable differences in compression ratio, with zlib achieving higher density at these mid-to-high levels while Intel ISA-L prioritizes throughput over maximum compression. These differences are expected given Intel ISA-L's design goals and its three compression levels, and they should be considered when selecting a level for performance-critical versus size-critical workloads.

3.5 Shim Code Flow Diagrams

Deflate

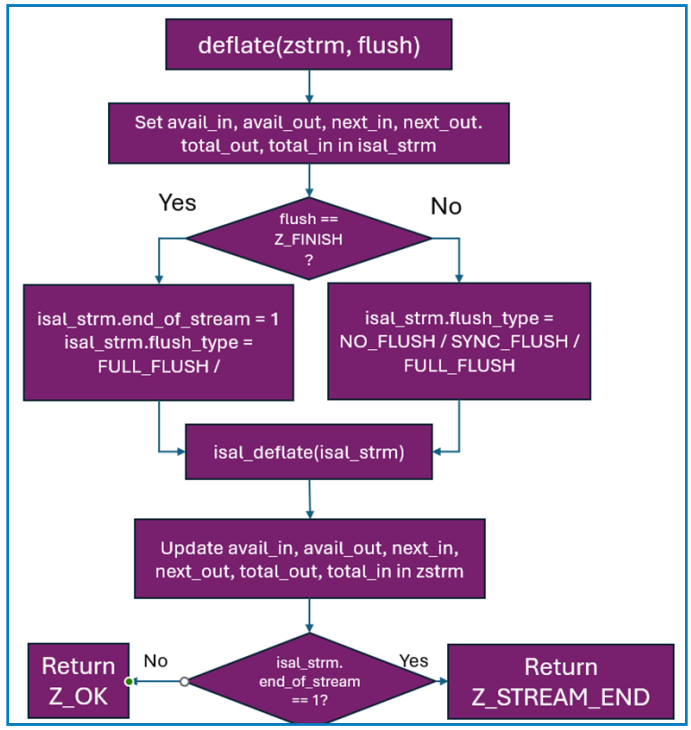

Figure 2. Intel® ISA-L shim deflate() code flow diagram

Inflate

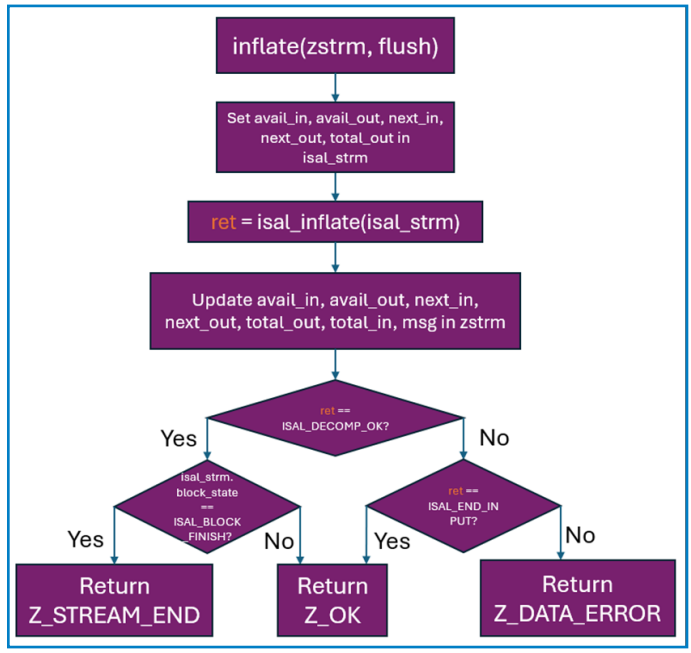

Figure 3. Intel® ISA-L shim inflate() code flow diagram

4. Performance Analysis

Performance evaluation of the Intel ISA-L shim implementation is completed using multiple applications to assess both compression ratio and throughput compared to standard zlib. The two software versions are tested and compared by collecting key performance indicators including compression time, throughput rates, and compression ratios. The evaluation is repeated on two distinct hardware platforms to ensure comprehensive analysis. Throughput measurements focus on both input processing rates (compressed data handling) and output generation rates (decompressed data production). The performance results demonstrate significant throughput improvements over traditional zlib implementations, while maintaining comparable compression ratios across data types and real-world application scenarios.

4.1 QATzip; zlib vs Intel ISA-L (No HW)

It is important to note that the QATzip performance tests in this section do not utilize Intel® QuickAssist Technology (Intel® QAT) hardware acceleration. The comparison is between two software implementations: the default zlib fallback and the Intel ISA-L (loaded via LD_PRELOAD). Any performance gains observed reflect the benefit of using Intel ISA-L over zlib at the software level, not any hardware offload capability.

4.2 Real-World Use Cases

A link to the test commands and scripts used to collect the following benchmarking data can be found in section 7.4 Glossary/Links. Disclaimer: Performance varies by use, configuration and other factors. See Section 7 for configuration details.

Note: Compression ratio results may differ between zlib and Intel ISA-L because the two libraries use different compression level scales. Some workloads default to **zlib level 6**, which maps to Intel **ISA-L level 2**, so differences in compression ratio are expected. See 3.4 Compression Ratio for more information.

Intel® Xeon® 6780E Processor (Platform 1)

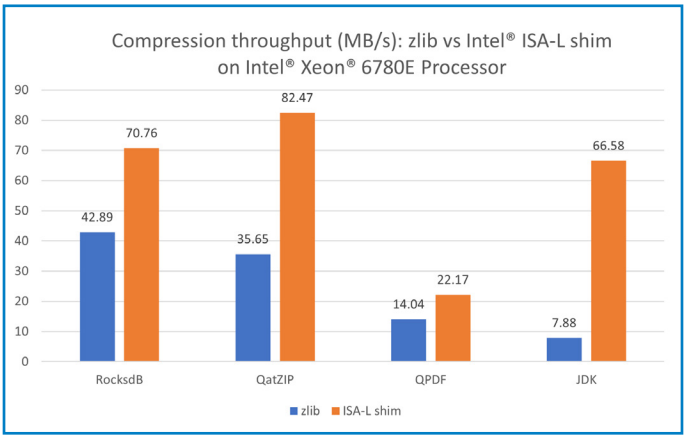

Figure 4. Compression Throughput Comparison of zlib & Intel® ISA-L on specific workloads on Intel® Xeon® 6780E processor (higher is better). Zlib levels mapped to Intel ISA L levels per Section 3.3.

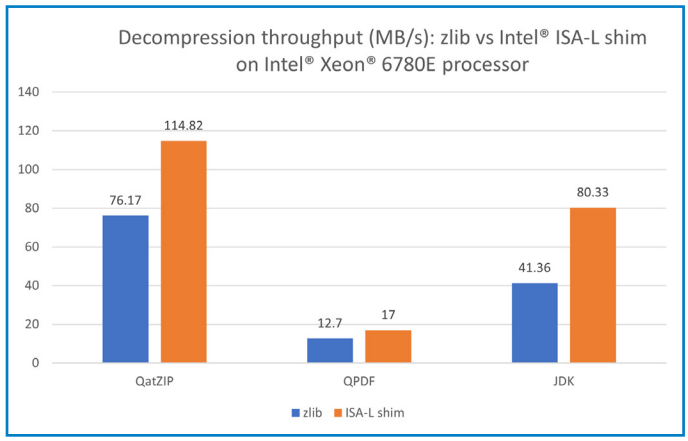

Figure 5. Decompression Throughput Comparison of zlib & Intel® ISA-L on specific workloads on Intel® Xeon® 6780E processor (higher is better). Zlib levels mapped to Intel ISA L levels per Section 3.3.

- RocksDB using db_bench

database	RocksDB original	ISA-L shim
Compression ratio %	80.29	80.30
Original file size (B)	228,098,961	228,098,961
Compressed file size (B)	183,148,401	183,166,177
Compression time (s)	5.07	3.07
Comp Throughput (MB/s)	42.89	70.76

Compression throughput improvement: 1.6x

- zlib (QATzip SW fallback) using qzip

Dickens	QATzip original	ISA-L shim
Compression ratio %	46.19	44.94
Original file size (B)	10,192,446	10,192,446
Compressed file size (B)	4,708,171	4,581,059
Compression time (s)	0.28	0.12
Comp Throughput (MB/s)	35.65	82.47
Decompression time (s)	0.133	0.088
Dec Throughput (MB/s)	76.17	114.82

Compression throughput improvement: 2.31x

Decompression throughput improvement: 1.5x

- QPDF using qpdf

Reymont	QPDF original	ISA-L shim
Compression ratio %	34.00	38.00
Original file size (B)	6,627,202	6,627,202
Compressed file size (B)	2,263,357	2,571,039
Compression time (s)	0.45	0.28
Comp Throughput (MB/s)	14.04	22.17
Decompression time (s)	0.17	0.12
Dec Throughput (MB/s)	12.7	17.0

Compression throughput improvement: 1.6x

Decompression throughput improvement: 1.33x

- JDK using GzipFileCompressor/GzipFileDecompressor

Dickens	JDK original	ISA-L shim
Compression ratio %	37.99	42.99
Original file size (B)	10,192,446	10,192,446
Compressed file size (B)	3,871,646	4,381,777
Compression time (s)	1.23	0.14
Comp Throughput (MB/s)	7.88	66.58
Decompression time (s)	0.24	0.12
Dec Throughput (MB/s)	41.36	80.33

Compression throughput improvement: 8.4x

Decompression throughput improvement: 1.9x

Intel® Xeon® 6768P-B Processor (Platform 2)

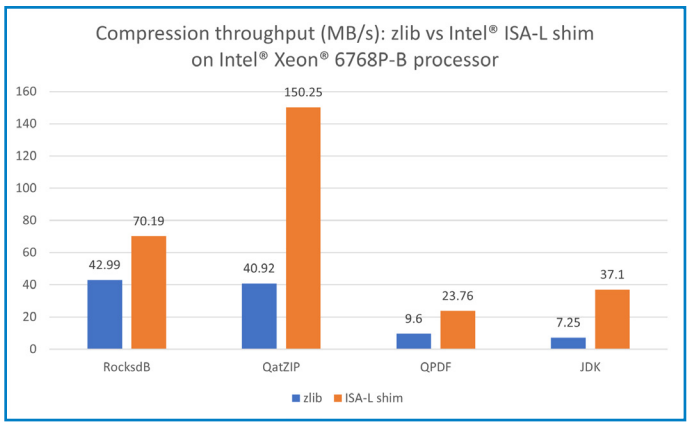

Figure 6. Compression Throughput Comparison of zlib & Intel® ISA-L on specific workloads on Intel® Xeon® 6768P-B processor (higher is better). Zlib levels mapped to Intel ISA L levels per Section 3.3.

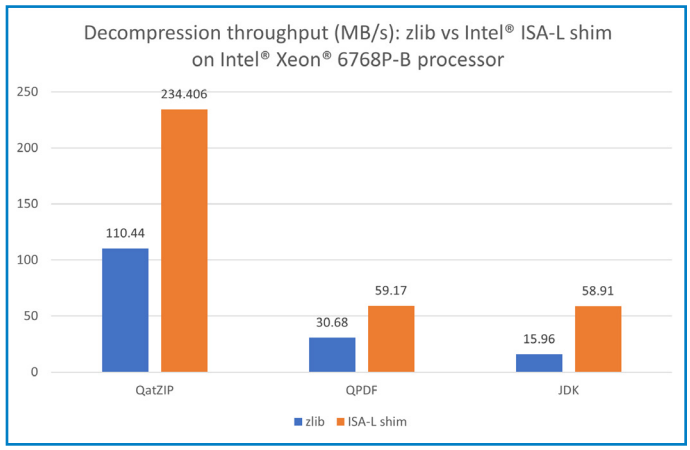

Figure 7. Decompression Throughput Comparison of zlib & Intel® ISA-L on specific workloads on Intel® Xeon® 6768P-B processor (higher is better). Zlib levels mapped to Intel ISA L levels per Section 3.3.

- RocksDB using db_bench

database	RocksDB original	ISA-L shim
Compression ratio %	80.29	80.31
Original file size (B)	228,087,812	228,087,812
Compressed file size (B)	183,147,608	183,165,337
Compression time (s)	5.06	3.09
Comp Throughput (MB/s)	42.99	70.19

Compression throughput improvement: 1.6x

- zlib (QATzip SW fallback) using qzip

Dickens	QATzip original	ISA-L shim
Compression ratio %	46.20	44.10
Original file size (B)	10,192,446	10,192,446
Compressed file size (B)	4,708,171	4,581,059
Compression time (s)	0.24	0.06
Comp Throughput (MB/s)	40.92	150.25
Decompression time (s)	0.09	0.04
Dec Throughput (MB/s)	110.44	234.406

Compression throughput improvement: 3.61x

Decompression throughput improvement: 2.13x

- QPDF using qpdf

Reymont	QPDF original	ISA-L shim
Compression ratio %	34.00	38.00
Original file size (B)	6,627,202	6,627,202
Compressed file size (B)	2,263,357	2,571,039
Compression time (s)	0.65	0.26
Comp Throughput (MB/s)	9.60	23.76
Decompression time (s)	0.21	0.11
Dec Throughput (MB/s)	30.68	59.17

Compression throughput improvement: 2.4x

Decompression throughput improvement: 1.9x

- JDK using GzipFileCompressor/GzipFileDecompressor

Dickens	JDK original	ISA-L shim
Compression ratio %	38.00	43.00
Original file size (B)	10,192,446	10,192,446
Compressed file size (B)	3,871,646	4,381,777
Compression time (s)	1.34	0.262
Comp Throughput (MB/s)	7.25	37.10
Decompression time (s)	0.61	0.17
Dec Throughput (MB/s)	15.96	58.91

Compression throughput improvement: 5.1x

Decompression throughput improvement: 3.69x

5. Deployment Guide

5.1 Installation and Configuration

Intel ISA-L is available across multiple operating systems, making it accessible for a wide range of deployment environments. The following installation and usage examples are specific to Ubuntu Linux.

Build Requirements and Dependencies

Build Intel ISA-L shim by default with CMake:

```
git clone https://github.com/intel/isa-l
cd isa-l/
mkdir build
cd build
cmake ..
make
```

Find isal-shim.so in isa-l/build/igzip/shim/.

Usage and Configuration Pptions

Deploying the Intel ISA-L shim library requires minimal system modifications due to its drop-in replacement design. The recommended approach involves building Intel ISA-L shim from source and using LD_PRELOAD environment variable for non-invasive testing.

Use LD_PRELOAD:

```
LD_PRELOAD=/path/to/isa-l/build/igzip/shim/isal-shim.so <./your_application>
LD_PRELOAD=/path/to/isa-l/build/igzip/shim/isal-shim.so python <./your_application.py>
```

Use LD_PRELOAD environment variable:

```
export LD_PRELOAD=/path/to/isa-l/build/igzip/shim/isal-shim.so./your_application
```

5.2 Troubleshooting

Debugging Tips

- Enable verbose logging: Show how to enable debug output for the shim layer. Building isa-shim.so with debug prints enabled:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
make
```

LD_PRELOAD Troubleshooting Tips

- Set-UID/Set-GID is ignored for security.
- Static binary is not supported.
- Missing symbols - verify with nm.
- Restricted environment - check containers, sudo, systemd.
- Any problems debug with LD_DEBUG=libs.

6. Conclusion

Summary of Benefits of Intel ISA-L Shim:
Performance, compatibility, ease of adoption

The transition from standard zlib to the Intel ISA-L shim library represents a compelling opportunity for applications to achieve substantial performance improvements without the complexity typically associated with manual code changes and integration. Our analysis demonstrates that Intel ISA-L delivers significant compression and decompression throughput improvements depending on workload characteristics, while intercepting the existing zlib implementations via LD_PRELOAD. The drop-in replacement design eliminates traditional migration efforts, enabling immediate benefits without application modifications.

Recommended Use Cases:
Intel ISA-L shim proves most valuable in scenarios where data throughput requirements are the highest priority.

The fastest way to evaluate the performance benefits of Intel ISA-L is to use the Intel ISA-L shim with LD_PRELOAD. This approach requires no code changes and allows applications to immediately leverage Intel ISA-L's optimized compression capabilities, making it ideal for rapid performance assessment and benchmarking.

However, while the shim provides an excellent starting point for evaluation, **production environments should migrate to using Intel ISA-L directly**. Direct integration eliminates the overhead of dynamic library interception, provides access to the full Intel ISA-L API and feature set, and ensures optimal long-term performance and maintainability.

7. Appendices

7.1 Platforms

Platform 1

System Summary	
System	Quanta Cloud Technology Inc. QuantaGrid D55Q-2U, ---
Baseboard	Quanta Cloud Technology Inc. S7Q-MB-MPS-MDP, To be filled by O.E.M.
Chassis	Quanta Cloud Technology Inc. Rack Mount Chassis, 1S7QU9Z000B
CPU Model	Intel®Xeon® 6780E
Architecture	x86_64
Microarchitecture	SRF_SP
L3 Cache (instance/total)	108M/216M
Cores per Socket	144
Sockets	2
Hyperthreading	N/A
CPUs	288
Intel Turbo Boost	Enabled (in BIOS), disabled through SW.
Base Frequency	2.2GHz
Maximum Frequency	2.2GHz
All-core Maximum Frequency	2.2GHz
NUMA Nodes	2
Prefetchers	L2 HW: Enabled, L2 Adj: Enabled, DCU HW: Enabled, DCU IP: Enabled, DCU NP: Enabled, LLC Stream: Disabled
PPINs	5d4ddcc00daf61c0, 5d4dc8bec74caff4
Accelerators Available [used]	DLB 0 [0], DSA 0 [0], IAA 0 [0], QAT 0 [0], vRAN Boost 0 [0]
Installed Memory	512GB (16x32GB DDR5 6400MT/s [6400MT/s])
Hugepagesize	2048 kB
Transparent Huge Pages	madvise
Automatic NUMA Balancing	Enabled
NIC	2x BCM57416 NetXtreme-E Dual-Media 10G RDMA Ethernet Controller, 1x Linux-USB Ethernet/RNDIS Gadget
Disk	1x 1.7T SAMSUNG MZ7L31T9
BIOS	2A05.QCT001
Microcode	0x13000131
OS	Ubuntu 24.04.2 LTS
Kernel	6.8.0-85-generic
TDP	330W
Energy Performance Bias	Balanced Performance (6)
Scaling Governor	performance
Scaling Driver	intel_pstate
C-states	POLL: Enabled, C1: Enabled, C1E: Enabled, C6S: Enabled, C6SP: Enabled
Efficiency Latency Control	Custom Mode
CVEs	19 OK, 0 Vulnerable

Platform 2

System Summary	
System	Intel Corporation KVL XRP
Baseboard	Intel Corporation KVL XRP, 0.01
Chassis	Intel Corporation Rack Mount Chassis
CPU Model	Intel®Xeon® 6768P-B
Architecture	x86_64
Microarchitecture	GNR-D
L3 Cache (instance/total)	256M/256M
Cores per Socket	64
Sockets	1
Hyperthreading	Disabled
CPUs	128
Intel Turbo Boost	Disabled
Base Frequency	2.2GHz
Maximum Frequency	3.5GHz
All-core Maximum Frequency	3.3GHz
NUMA Nodes	1
Prefetchers	L2 HW: Enabled, L2 Adj: Enabled, DCU HW: Enabled, DCU IP: Enabled, DCU NP: Enabled, AMP: Enabled, LLCPP: Enabled, AOP: Enabled, Homeless: Enabled, LLC: Disabled
PPINs	52b337caf25a0256
Accelerators Available [used]	DLB 0 [0], DSA 0 [0], IAA 0 [0], QAT 0 [0], vRAN Boost 0 [0]
Installed Memory	32GB (2x16GB DDR5 6400MT/s [5200MT/s])
Hugepagesize	1048576 kB
Transparent Huge Pages	madvise
Automatic NUMA Balancing	Disabled
NIC	1x I210 Gigabit Network Connection, 8x Intel® Ethernet Connection E825-C for QSFP
Disk	1x 953.9G SAMSUNG MZVL21T0HDLU-00B07
BIOS	KVLDCRB1.IPC.3038.P19.2511041029
Microcode	0x10002f3
OS	Ubuntu 24.04.2 LTS
Kernel	6.8.0-100-generic
TDP	325W
Energy Performance Bias	Balanced Performance (6)
Scaling Governor	performance
Scaling Driver	acpi-cpufreq
C-states	POLL: Enabled, C1: Enabled
Efficiency Latency Control	Optimized Power Mode (OPM)
CVEs	19 OK, 0 Vulnerable

7.2 Applications

Application	Version	Link
RocksDB	988357696dc7961789b32092a2e66effb7c2528e	https://github.com/facebook/rocksdb
QATzip	10dbadbab384ab3a0cb514163ff1aa47c2bf42f2 10dbadbab384ab3a0cb514163ff1aa47c2bf42f2	https://github.com/intel/QATzip
QPDF	bf615b8c9c9100152f7dd9861fc3f19716f09405	https://github.com/qpdf/qpdf
JDK	21.0.8 2025-07-15	https://github.com/openjdk/jdk
Intel® ISA-L	2.32.0build1	https://github.com/intel/isa-l
Zlib	1:1.3.dfsg-3.1ubuntu2.1	https://github.com/madler/zlib

7.3 Benchmarking Scripts/Commands

The benchmarking scripts and commands for validating and measuring the performance of the zlib shim with Intel ISA-L across some applications — including RocksDB, QATzip, QPDF, and JDK — are available on the Intel ISA-L GitHub Wiki.

7.4 Glossary/Links

Silesia Corpus files (Dickens, Reymont)	https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia
Intel ISA-L GitHub Wiki (scripts & commands)	https://github.com/intel/isa-l/wiki/Zlib-shim-with-ISA_L

7.5 Figures

0326/VK/PDF 367310-001US