Solution Brief

Intel® QuickAssist Technology - Deliver Compression Efficiencies in the Cloud with Intel® QAT and QATzip

QATzip provides an extremely easy programming interface to extremely powerful compression accelerators embedded in the 4th Gen Intel® Xeon® Scalable processor.

Authors

Gordon McFadden

Joel D Schuetze

Gokhan Simsek

Executive Summary

QATzip is a user space library, which builds on top of the Intel® QuickAssist Technology (Intel® QAT) user space library, to provide extended accelerated compression and decompression services by offloading the actual compression and decompression requests to one or more Intel devices. QATzip produces data using the standard gzip format with extended headers or LZ4 blocks with LZ4 frame format. The data can be decompressed with a compliant gzip* or LZ4 implementation. QATzip is designed to take full advantage of the performance provided by Intel® QuickAssist Technology. This paper discusses why QATzip exists, its applications, value for developers, and reviews the performance gains.

This document is part of the Network Transformation Experience Kits.

Introduction

The QATzip library was created to solve a relatively simple problem. A few generations ago, Intel® QAT Hardware was available as a PCIe plugin card, or in a subset of the chipsets. Customers may have different servers; some with the acceleration hardware, some without. A common software stack was required; one that is able to detect the presence of Intel® QAT and use it or make use of a software library if the platform did not have Intel® QAT. The requirement was to have a simple and fast programming model with seamless installation that would be consistent through future generations.

Solution Description

Usages and Applications

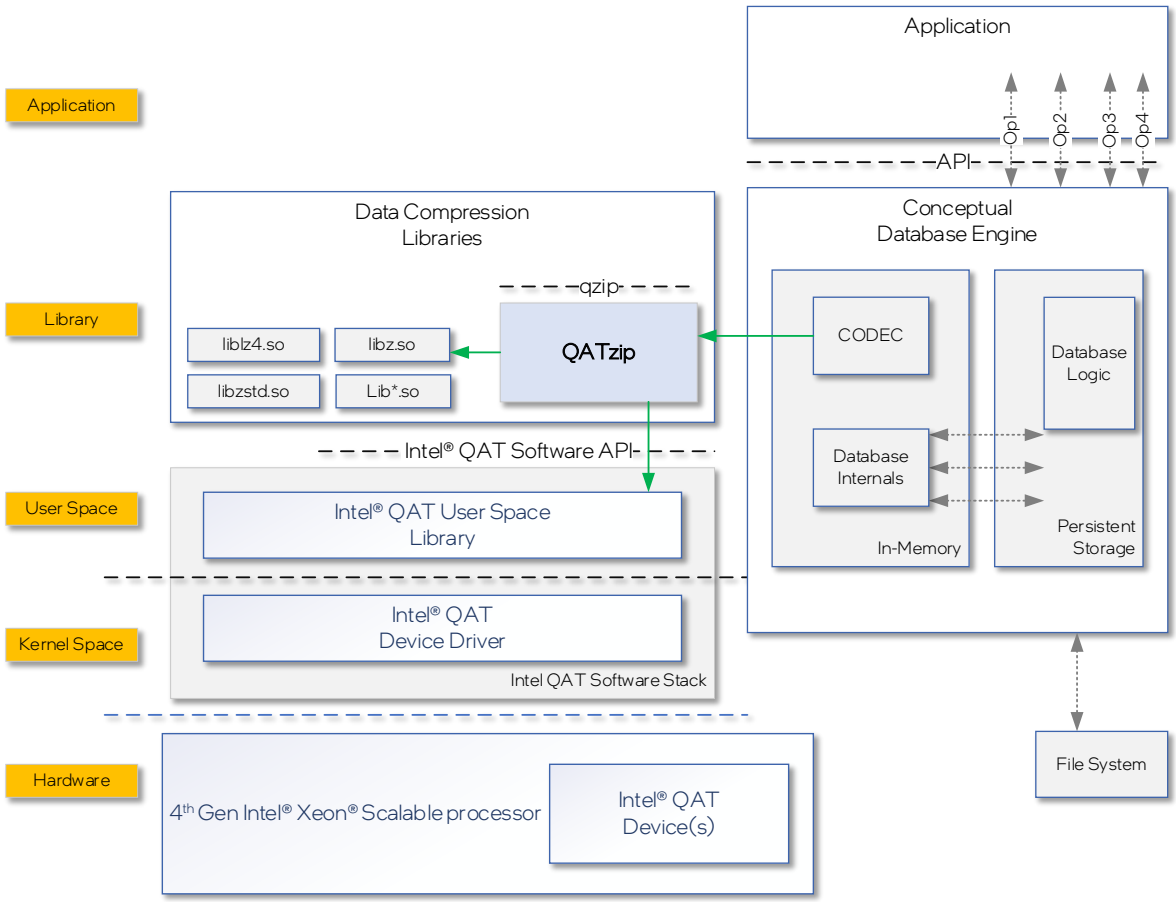


Figure 1. QATzip with a Conceptual Database Engine for a Datastore Application

QATzip is a user-space library built on Intel® QAT Software stack, supplying added features, capabilities, and utilities. It aims to simplify usage of Intel® QAT Software by supplying programming abstractions. For example, QATzip Library initializes the Intel® QAT hardware resources on the first call, thus cutting the need for initialization of the device resources separately. The application/middleware software developer can focus on the needs of the application, minimizing the need to know the Intel® QAT software specific concepts. Similarly, if the software application is deployed on a platform without Intel® QAT capabilities, QATzip seamlessly uses the software equivalent of the data compression libraries.

Typically, QATzip integrates with a software middleware, which has the in-memory compression/decompression functionality. The middleware might be enabling database engines, web compression, and big data applications.

As an example, see Figure 1 that depicts a conceptual integration with a storage/database application. Here, the database engine's Codec (encoder/decoder) wraps the calls to the QATzip, and the database engine's internal logic remains the same. If the Intel® QAT software stack is not available on the platform or the Intel® QAT software device detects an error, QATzip seamlessly falls back to the equivalent software implementation. In this case, storage application's logic does not need to change if the underlying compression implementation changes.

QATzip also provides the qzip command line application, which can be used for file compression to immediately validate and interact with the Intel® QAT software and automate compression command line applications with other standard system utilities.

Features and Benefits

The features and benefits of this solution are as follows:

- Dynamic memory allocation for zero copy by exposing qzMalloc() and qzFree() allowing working buffers to be pinned, contiguous buffers that can be used for DMA (Direct Memory Access) operations to and from the hardware.
- Instance over-subscription, allowing several threads in the same process to seamlessly share a smaller number of hardware instances.
- Conceptual integration with a storage/database application; allocating from huge page memory when there is kernel memory contention. See Figure 1.
- Configurable accelerator device sharing among processes.
- Optional software failover for both compression and decompression services. QATzip may switch to software if there are insufficient system resources including acceleration instances or memory. This feature allows for a common software stack between server platforms that have acceleration devices and non-accelerated platforms.
- Automatic recovery from hardware compression failure.
- Provide streaming interface of compression and decompression to achieve better compression ratio and throughput for data sets that are submitted piecemeal.
- "qzip" utility supports compression from regular files, pipes, and block devices.
- For standard gzip format, try hardware decompression first before switching to software decompression.
- Enable adaptive polling mechanism to save CPU (Central Processing Unit) usage in stress mode.
- 'qzip' utility supports compression files and directories into 7z format.
- Starting with the Intel® QAT Gen 4 device, QATzip supports LZ4/LZ4s (de) compression algorithms as well as Zstd6 compressed data format through software post processing of compressed data blocks with LZ4s format.

Methodology

QATzip was designed from its inception to be a straightforward path to accelerate compression and decompression operations using the Intel® QuickAssist Technology. In its simplest form, all setup APIs (Application Programming Interfaces) can be skipped, and compression requests can be called directly.

Session

QATzip is constructed with the concept of sessions, allowing a software developer to describe the format of the compressed data required, including algorithm and compression level.

The default session compresses using the deflate algorithm, compression level six and wraps the compressed block of data with RFC compliant gzip header and footer. Large submissions will be chunked and submitted in 64 kilobyte blocks.

Buffers

Access to the Intel® QAT accelerator required pinned memory. QATzip uses the User Space DMA-able Memory (USDM) memory driver and sets up a pool of memory buffers that are contiguous pinned memory. The source and destination buffers used for accessing Intel® QAT might be either pinned or traditional buffers. If the buffer is not pinned from USDM, then QATzip library will copy the data to an internal buffer entry from the buffer pool.

Example Code

QATzip can be invoked with minimal setup code. Following is an example of how to compress a buffer:

```
 1 #include <stdio.h>
 2 #include <qatzip.h>
 3
 4 int main( int argc, char *argv[] )
 5 {
 6    unsigned char src[1024] = { "1" };
 7    unsigned char dest[1024];
 8    unsigned int sl, dl, rc;
 9    QzSession_T sess = { 0 };
10
11    sl = 1024; dl = 1024;
12
13    rc = qzCompress( &sess, src, &sl, dest, &dl, 1 );
14
15    printf( "rc = %d sl = %u\n", rc, sl, dl );
16
17 return rc;
18 }
```

This level of simplicity allows developers to focus on their application, and what data needs to be compressed and decompressed, and not on how to compress or decompress. This fast access to fact compression is a hallmark of QATzip.

Important Session Parameters

There are many session options. Some of the more commonly used ones are as follows:

- Compression format to select - deflate, LZ4, LZ4s
- Compression level
- Whether to use software for the operation if no Intel® QAT hardware is available
- The size to submit to Intel® QAT hardware when large buffers are supplied

The following example shows how to set compression level six as a session parameter:

```
 1 #include <stdio.h>
 2 #include <qatzip.h>
 3
 4 int main( int argc, char *argv[] )
 5 {
 6    QzSession_T sess = {0};
 7    QzSessionParams_T params;
 8
 9    char src[1024] = {" "};
10    char dest[1024];
11    unsigned int i,j;
12    int rc;
13
14    i = 1024; j = 1024;
15
16    rc = qzGetDefaults(&params);
17    params.comp_lvl = 6;
18    rc = qzInit( &sess, 1 );
19    rc = qzSetupSession( &sess, &params );
20    rc = qzCompress( &sess, src, &i, dest, &j, 1 );
21
22    printf( "rc = %d j = %d\n", rc, j );
23
24    return 0;
25
26 }
```

Available session parameters are defined in qatzip.h.

Overloading to Cores

There is an interesting artifact of two features of QATzip. The first feature, called sw_backup, allows the usage of cores to compress if there are no Intel® QAT instances available to the process. The second feature is instance sharing. In a threaded application it may not be necessary to assign an Intel® QAT instance to each thread. QATzip will seamlessly share any number of instances to any number of threads. For example, if ten threads require compress services 25% percent of the time, then the ten threads could be used four or five instances with no contention.

If both features are enabled, and the compression requests exceed the capacity of the Intel® QAT device, then QATzip will use the cores for some compression services. This can have the effect of sustaining compression or decompression rates that exceed the capacity of the Intel® QAT device by using cores for some operations.

System Configuration

Table 1 lists the system configuration.

Table 1. System Configuration

Hardware	CPU	2S pre-production 4th Gen Intel® Xeon® Scalable processor
	Platform	pre-production Intel® reference platform
	Memory	1,024 GB (16*64 GB DDR5)
	Hard drive	1x 1.92 TB Intel® SSD SC2KG01
	Microcode	0xf000380
	HT	On
	Intel® QAT Hardware	Gen 4
	Turbo	Disabled
	SNC	Off
Software	Operating System	Ubuntu 22.04.1 LTS
	Kernel	5.15.0-47-generic
	Intel® QAT driver	QAT v20.l.0.9.1
	Intel® ISA-L	ISA-L v2.3.0
	QATzip	QATzip v1.0.9
	Test date	September 2022
	Test by	Intel

Test Setup

To showcase the performance that can be achieved with Intel® QAT devices included on the 4th Gen Intel® Xeon® Scalable processor with QATzip, we created a set of performance scripts that utilize the test application included with the QATzip package.

The test application is named "test" and is located in the test folder. The script makes many simultaneous invocations like the following:

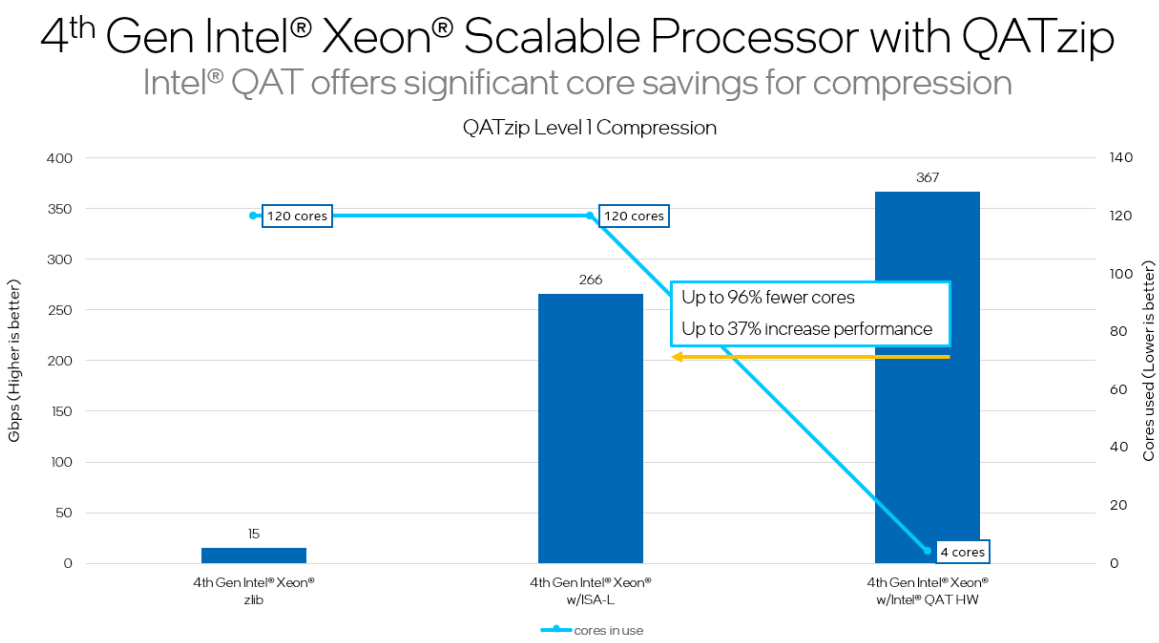
```
numactl -C 0 /usr/local/src/QATzip/test/test -m 4 -l 500 -t 1 -D comp -B 0 -L 1 -C 65536 -S
1 -i ./test/performance_tests/calgary_1mb
```

The command breaks down as follows:

- Run the test application on core 0
- Execute test mode 4, which corresponds to test comp/decomp by configurable parameters
- Execute the test case for 500 iterations
- Use one thread
- Compression direction test
- Disable software fallback so cores are not used
- Compression level 1
- Chunk size of 64K
- Set sleep thread interval to 1 ms
- Selected input file "calgary_1mb", which is the first megabyte of the Calgary corpus files concatenated into one file

When testing with Intel® QAT, the script invoked 24 simultaneous invocations of the test application, with multiple invocations of the test app tied to the same core. Individual results were then aggregated to produce the total throughput.

To obtain the software numbers using the default zlib library, the Intel® QAT devices were shut down and the invocation of the test application was modified to enable software fallback (-B 1). The script invoked 120 simultaneous invocations of the test app, each instance tied to a specific core, and aggregated the individual results to produce total throughput.



QATzip library uses PCIe Add-in Intel® QAT Accelerator, Intel® PCH w/Intel® QAT chipset, or 4th Gen Intel® Xeon® Processor built-in Intel® QAT accelerator.
ISA-L: Intel® Intelligent Storage Acceleration Library

Figure 2. QATzip Performance Results

Results

Figure 2 highlights the value of offloading compression workloads to the Intel® QAT device. Performance is shown for standard zlib library, the optimized compression library Intel® ISA-L, and with offloading to the Intel® QAT devices. With zlib and Intel® ISA-L we fully saturate all 60 cores on both sockets to achieve the specified level of performance. Offloading that same compression workload to Intel QAT allowed for a higher level of performance all while utilizing just four cores.

Summary

This document shows that it is simple to program the QATzip API. To compress data with the default algorithm–deflate – at the default compression level–1- and encapsulate the compressed in a gzip header with RFC compliant extensions that include the source and destination buffers, then no explicit session set is required.

The paper also shows that using QATzip software to access Intel® QAT hardware provides incredible acceleration, freeing cores for other tasks.

Terminology

Table 2. Terminology

Abbreviation	Description
API	Application Programming Interfaces
CPU	Central Processing Unit
DMA	Direct Memory Access
RFC	Request for Comment
USDM	User Space DMA-able Memory

References

Table 3. References

Reference	Source
GZIP file format specification	https://www.ietf.org/rfc/rfc1952.txt
LZ4 Block Format Description	https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md
LZ4 Frame Format Description	https://github.com/lz4/lz4/blob/dev/doc/lz4_Frame_format.md
QATzip package	https://github.com/intel/QATzip
Getting Started Guide	https://www.intel.com/content/www/us/en/content-details/632506/intel-quickassist-technology-intel-qat-software-for-linux-getting-started-guide-hardware-version-2-0.html
Zstandard compression algorithm	https://facebook.github.io/zstd/

Document Revision History

Revision	Date	Description
001	January 2023	Initial release.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

0123/DN/WIT/PDF 767068-001US