Technology Guide

Intel® AVX-512 - Ultra Parallelized Multi-hash Computation for Data Streaming Workloads

Authors

Leyi Rong

Yipeng Wang

Weigang Li

Hongjun Ni

1 Introduction

Sketch-based algorithms[1] are emerging technologies that are broadly used in network measurement and network telemetry workloads, generating approximate estimations of networking flows. It is used to prevent distributed denial-of-service (DDoS) attacks, monitor network usage, and for various Quality of service (QoS) purposes. Compared to hash tables or other lossless algorithms, sketch-based algorithms are designed with a compact and optimized data structure for memory efficiency and computing throughput.

The core data structure, i.e. Sketch, consists of a two-dimensional (2D) array. Each row of the array corresponds to a resulting digest space indexed by an independently computed hash function. With more independent hash functions, more accurate estimation results can be provided. Nevertheless, increasing the number of hash computations increases the amount of CPU consumption, which may prevent the application from processing high-volume networking traffic. Thus, a high throughput multi-hash computation methodology is desired in this domain. Note that the term "multi-hash" in this technology guide does not correspond to the following Intel whitepaper[2], which by contrast proposes extensions to cryptographic hash algorithms.

This technology guide proposes a novel model to accelerate multi-hash computation by leveraging Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions. This proposed innovation achieves an average performance gain of up to 2x for the critical key-add and key-lookup operations, compared with the standard CRC-32 instruction approach for state-of-the-art algorithms. Moreover, as different workloads have diverse requirements for the hashing algorithm (randomness, cryptography, small data velocity, etc.), our proposed model supports algorithm customization for different purposes. The solution provides developers with a robust, flexible foundation to build high-throughput networking measurement and monitoring applications. By leveraging its optimized data plane, implementers can achieve excellent performance across a diverse range of network telemetry workloads.

This document is part of the Network & Edge Platform Experience Kits.

[1] Finding Frequent Items in Data Streams. In Proc. of ICALP.

[2] Multi-Hash: A Family of Cryptographic Hash Algorithm Extensions. Intel White Paper (July 2012)

Table of Contents

1 Introduction 1
1.1 Terminology 3
1.2 Reference Documentation 3
2 Overview 3
3 Technology Description 4
3.1 Background 4
3.2 Motivation 5
3.3 New Proposal Model 5
3.3.1 Parallel Hash Computation with Splitting Input and Vectorized Seeds by Leveraging Intel® AVX-512 6
3.3.2 Accelerate Multiplication and Addition Operations by Leveraging Intel AVX-512 IFMA Instruction in Hash Computation 8
3.3.3 Accelerate Sketch Counter Updates by Leveraging Intel AVX-512 Gather and Scatter Instructions 8
4 Performance Benchmarking 9
4.1 Benchmarking Platform 9
4.2 Benchmarking Results 9
5 Summary 10

Figures

Figure 1. Example Sketch data structure 4
Figure 2. Example Bloom filter algorithm 5
Figure 3. Example Sketch algorithm with different seeds for hash functions 5
Figure 4. Ultra parallelized multi-hash computation workflow 6
Figure 5. Parallelized initializing the multi-hash computation 7
Figure 6. Continuously processing the input data as 8-byte-block 7
Figure 7. Processing the remaining input data less than 8 Bytes 8
Figure 8. Performance benchmarking on Sketch Add compared with CRC-32 instruction 10
Figure 9. Performance benchmarking on Sketch Lookup compared with CRC-32 instruction 10

Tables

Table 1. Terminology 3
Table 2. Reference Documents 3
Table 3. Performance Benchmark Platform Configuration 9

Document Revision History

Revision	Date	Description
001	August 2023	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
CRC-32	32-Bit Cyclic Redundancy Check
DPDK	Data Plane Development Kit (dpdk.org)
IFMA	Integer Fused Multiply Add
Intel® AVX	Intel® Advanced Vector Extensions (Intel® AVX)
ISA	Instruction Set Architecture
MAD	Multiply-Add-Divide

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
DPDK Official Website	https://www.dpdk.org/
Intel® 64 and IA-32 Architectures Software Developer's Manual	https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.htmlhttps://software.intel.com/content/www/us/en/develop/articles/intelsdm.html
Intel® 64 and IA-32 Architectures Software Optimization Reference Manual	https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

2 Overview

To our best knowledge, there is no highly parallel solution for multi-hash computation. We came up with the following three alternative solutions as the initial options:

1. Using specific hardware (for example, network adapter, FPGA) to do the hash computation
2. Using a long-output-size hash function(for example, SHA3-512[3]), then evenly dividing the results into multiple parts for the multi-hash results
3. Using CRC-32 instruction accelerated hash function with different seeds

There are some disadvantages of the aforementioned proposed solutions:

1. **Specific hardware (for example, network adapter):** The hashing algorithm runs in specific hardware that is normally fixed and rigid, therefore, it can not meet the diverse requirements of the hashing algorithm used in various data stream algorithms. Also, data stream analysis applications are usually run at the top layer of the networking stack. For example, many firewalls and networking telemetry applications run after the networking packets are decrypted and decompressed. Sending data stream back to the hardware devices is subject to long device communication latency. It's better to produce the hash result close to the workload that runs on the CPU.
2. **Long-output-size hash function:** Hashing algorithms such as SHA3-512 can generate long hash values that can be used to substitute multiple shorter hashing computations. But there are several caveats. 1) Existing long-output hashing algorithms are not flexible enough to be customized for performance-efficiency trade-offs. 2) The algorithm must have an excellent avalanche effect, i.e., when an input changes slightly, the output should change significantly. This requires more complex arithmetic, which results in lower performance. The performance on Intel® Xeon® Processor E3-1220 v5 of SHA3-512 is **164 cpb**[4] (cycle per byte) with 8 byte of input size. Although the performance benchmarking platforms are not the same, the existing long-output hashing algorithm falls significantly behind in terms of throughput when compared to the proposed model of **4.5 cpb**.
3. **CRC-32 instruction:** Intel has introduced hardware CRC-32 computation ISA to accelerate CRC hashing computation. But the CRC-32 based implementation lacks flexibility and parallelism compared to our proposal. The performance data shows that our proposal achieves 2x throughput by using Intel AVX-512 with a customized hashing algorithm compared to CRC-32 based algorithm.

[3] SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. NIST (August 2015)

[4] https://en.wikipedia.org/wiki/Secure_Hash_Algorithms#Comparison_of_SHA_functions

This technology guide proposes a novel model to process multiple hash functions in parallel by leveraging Intel AVX-512 instruction sets. Various data stream algorithms, such as sketch as well as other classic algorithms like Bloom filter, could benefit from this proposal.

The model consists of three key ideas:

- Parallel hash computation with splitting input and vectorized seeds by leveraging Intel AVX-512
- Accelerate multiplication and addition arithmetic using Intel AVX-512 IFMA instruction in hash computation
- Accelerate sketch counter updates by leveraging Intel AVX-512 gather and scatter instructions

3 Technology Description

3.1 Background

Multi-hash computation (multiple independent hashing functions) has been broadly used in many algorithms of the data stream analysis domain, like sketch-based algorithms, Bloom filter, etc. Figure 1 shows the fundamental data structure of a sketch as an example (a *d* x *m* 2D array). Sketch-based algorithms are popular and effective approaches used in the network measurement and network monitoring domains to estimate the size of networking flows. It is used to prevent DDoS attacks, monitor networking usages, and for various QoS purposes.

In a common scenario, when a data stream (for example, a stream of networking packets) goes through the system, the sketch-based algorithm will process *d* times of independent hash computations for each packet. The corresponding hash digests, i.e., the results generated by each hash function, will be used to index the counter among *m* counters (bins) of each row. For each packet (key) encountered, the corresponding counter will be incremented or decremented. The eventual counter values are summarized from all *d* arrays that can be used to estimate the frequency of the key (i.e., the size of a networking flow). The multi-hash computation will help to improve the accuracy of the frequency estimation results, especially for heavy hitter data streams. In other words, more hashing computations tend to result in better accuracy.

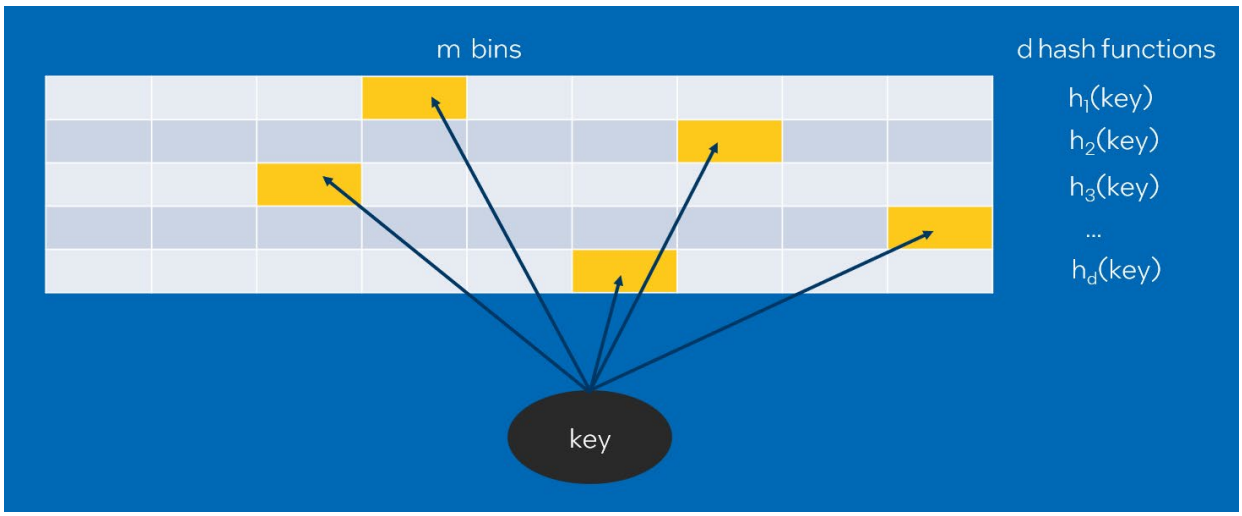

Figure 1. Example Sketch data structure

Besides sketch, Bloom filter based algorithm also leverages multiple independent hash functions to reduce the false positives when querying certain items in a data set. As shown in Figure 2, the example Bloom filter array consists of 12 elements, and a set of four independent hash functions are used to update the array for each key.

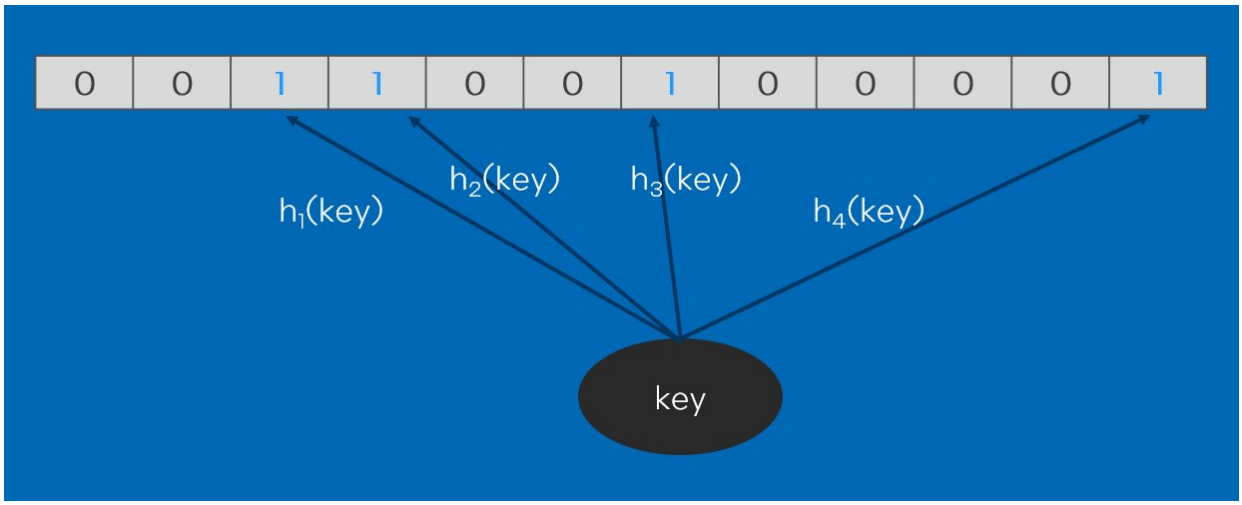

Figure 2. Example Bloom filter algorithm

3.2 Motivation

Based on our analysis and performance profiling of a state-of-the-art sketch-based algorithm, hash computation is the number one performance hotspot of the total CPU consumption. As high throughput networking applications (100Gb – 1Tb) become more and more common in data center use cases, it is critical to develop fast multi-hashing algorithms with minimal CPU consumption.

Our investigation shows that existing CPU-based algorithms are not flexible nor performant enough to meet high throughput requirements. One may argue that specialized hardware (for example, network adapters, FPGA) can be used to compute hash instead of using CPU. However, our investigation found networking profiling tasks tend to require maximum flexibility and could be high in the networking stack. In other words, in many use cases, network adapter-provided computation is either too rigid or not usable. Softwares such as firewalls and networking telemetry are mainly running in CPU with processed data streams. CPU cores are still the primary computing resources used by many of our customers' software applications. More discussions can be found in Section 2.

3.3 New Proposal Model

Multiple independent hash computations require different seeds to be the inputs to each computation. With different seeds, the same hashing function could generate independent (randomized) results. The seeds can be random numbers to get a good mixture to result in a hash digest with good randomness. Figure 3 shows a sketch using various seeds but the same hash function to generate independent hash values.

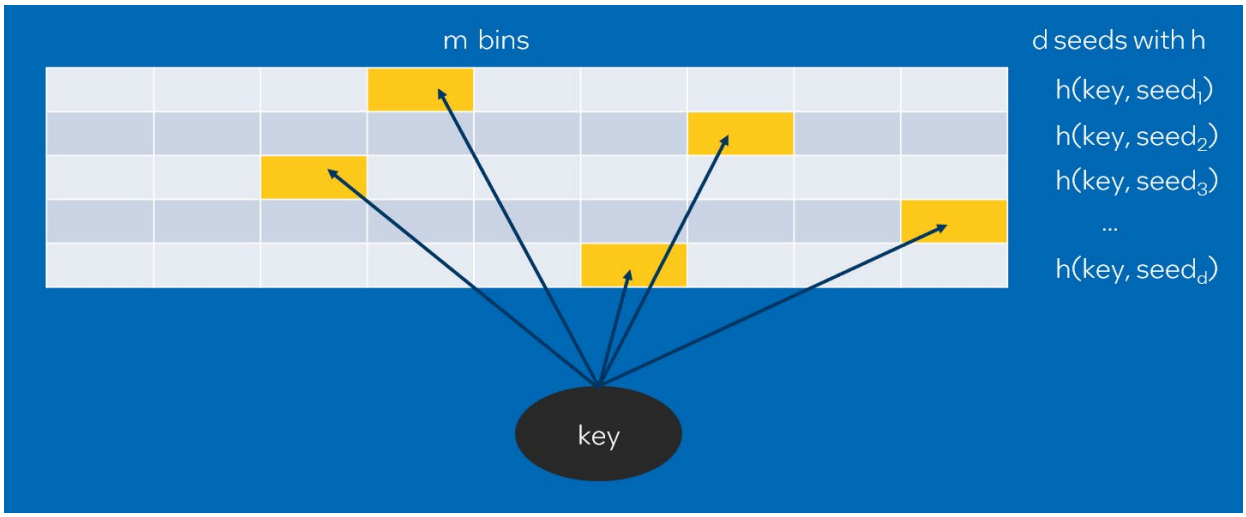

Figure 3. Example Sketch algorithm with different seeds for hash functions

Our proposed approach follows data-level-parallelism by taking multiple seeds as input, and calculating multiple hashes in parallel. The implementation is based on xxHash[5], which is a fast and popular non-cryptographic hash algorithm. Other algorithms can be easily adapted to our model as well for even better throughput. Our key innovations can be summarized as the following.

3.3.1 Parallel Hash Computation with Splitting Input and Vectorized Seeds by Leveraging Intel® AVX-512

To calculate eight independent hash values, we use eight random seeds, then continuously split the input data stream (for example, an input key) into 64-bit width data chunks and then get consumed one by one. To fully take advantage of Intel AVX-512 ISA, the data chunk size multiplies the seed count should be equal to 512. For example, if the seed count is four, then the data stream should be divided into 128-bit chunks. The process is roughly illustrated in Figure 4.

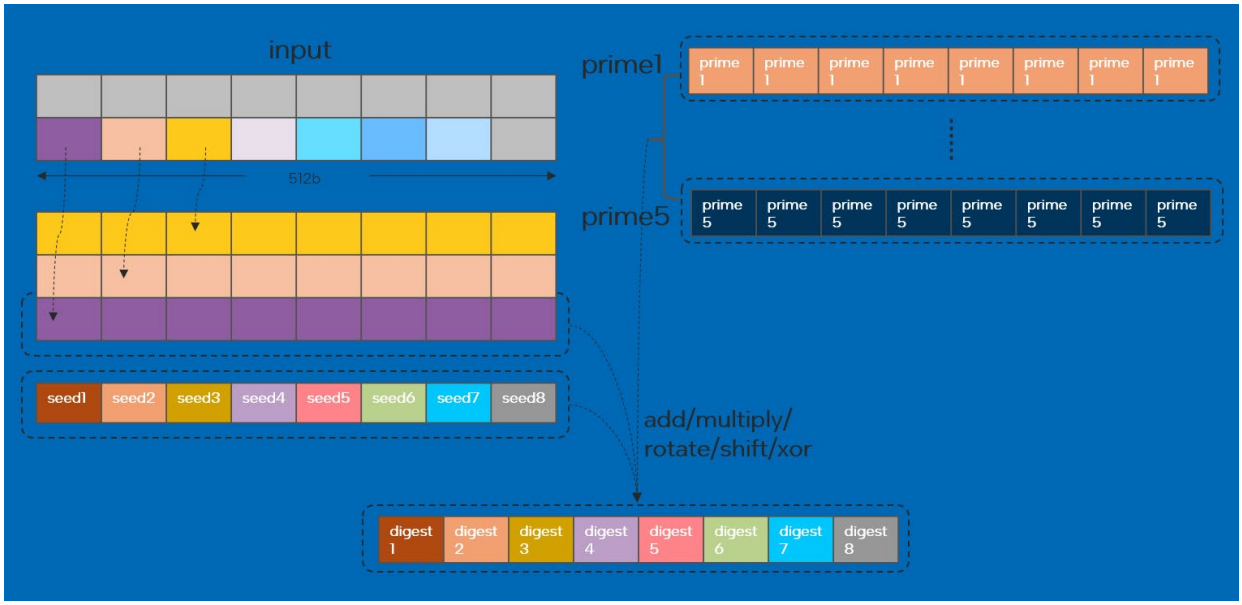

Figure 4. Ultra parallelized multi-hash computation workflow

We have pre-defined five different prime numbers (P1 to P5) and eight different seeds to be used during the computation. The five prime numbers are getting involved in the hash computation later to target a good mixture of hash results. Each prime number is broadcasted into the vector of eight items. The eight different seeds are also put into a 512-bit wide vector register.

The algorithm also broadcasts the truncated 64-bit-wide input data into the 512-bit-wide vector with eight copies. Then it mixes the 8-seed vector and the pre-defined five prime numbers by using addition, multiplication, rotation, shift, and xor arithmetic operations in a vectorized style.

The whole process can be broken into the following steps:

1. Initially, the seed vector and the P5 vector are added, and then adds the vectorized stream length in byte count. Then it temporarily stored the result in vector m. as shown in Figure 5.

[5] https://xxhash.com/

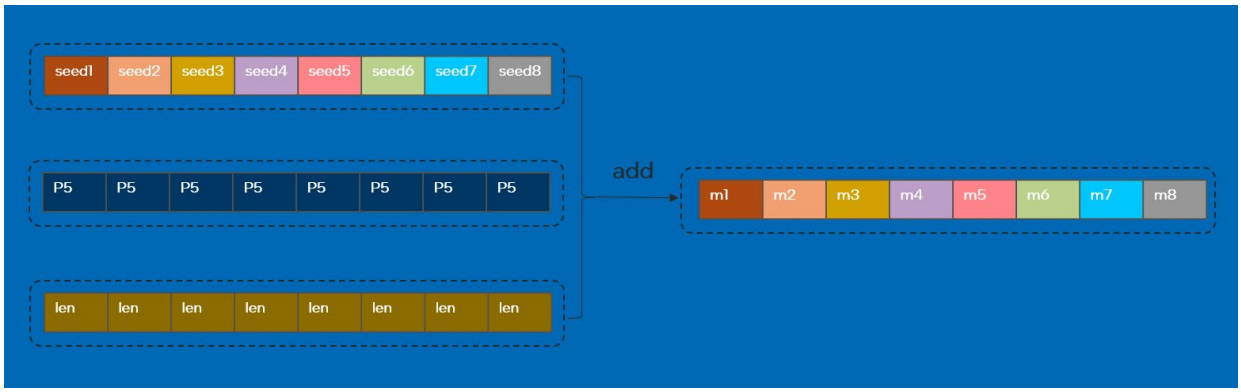

Figure 5. Parallelized initializing the multi-hash computation

2. Read the input data stream (for example, an input key) by 8-byte-stepping iteratively; then broadcast to an Intel AVX-512 vector register and execute the vectorized multiplication and rotation operations with input data vector and P2; then take XOR operation with the step1's result vector m to generate the intermediate result vector m'; then execute a vectorized left-rotation. After that, multiply with vector P1; then take addition with vector P4 to generate the vector m". If the input data stream length is larger than 8 bytes(64-bit), the generated vector m" will be taken to participate in the new round computation with the next 64b length of input data. Until there are no more chunks of 8-byte-block of the input data left, the arithmetic of the vectorized 8-byte-block data input will be finished. This step is shown in Figure 6.

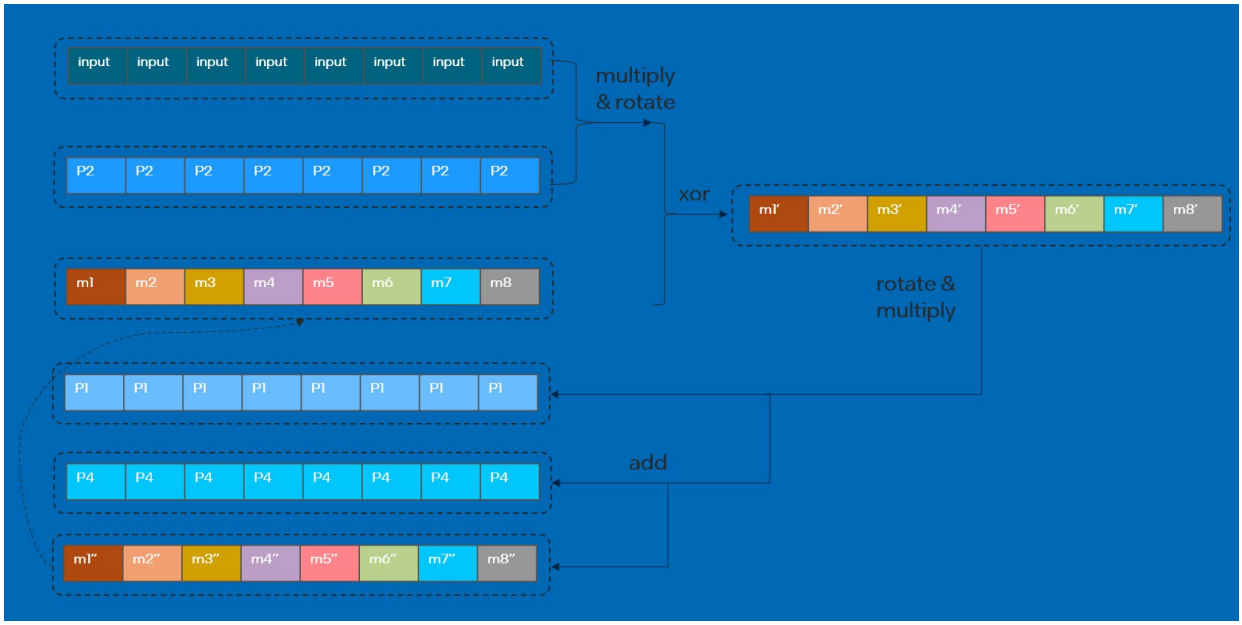

Figure 6. Continuously processing the input data as 8-byte-block

3. As there might be less than eight bytes of input data left that needs to be processed after the previous 8-byte-stepping calculation, it might need to cope with the remaining input data. The process is similar to the previous step 2, with vectorized processing of the remaining input data by leveraging add/multiply/rotate/shift/xor arithmetic operations. Finally, after consuming all input data, apply the last avalanche operation to the previous intermediate result to get the final vector digest d of all eight seeds. The full process is shown in Figure 7.

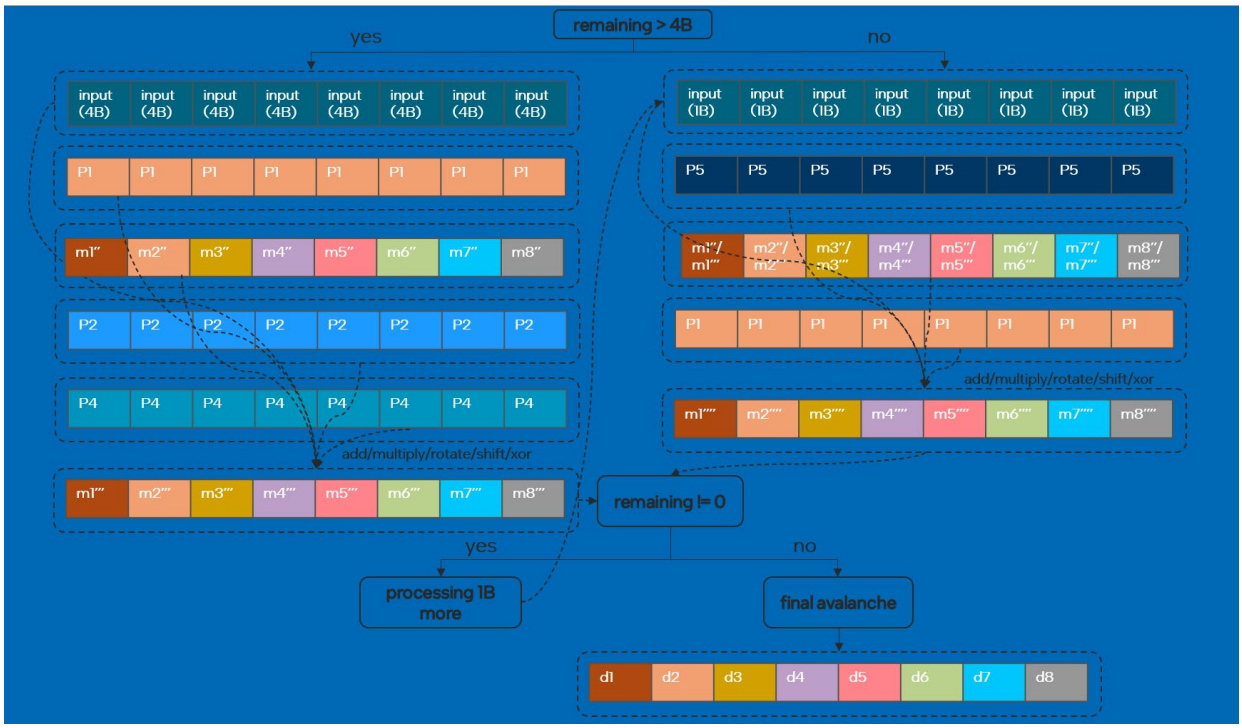

Figure 7. Processing the remaining input data less than 8 Bytes

3.3.2 Accelerate Multiplication and Addition Operations by Leveraging Intel AVX-512 IFMA Instruction in Hash Computation

Intel AVX-512 IFMA instruction is introduced in 8th Gen Intel® Core™ i3 Processors and 3rd Gen Intel® Xeon® Scalable Processors, which can support fused multiply and add operation of integers belonging to Intel AVX-512 instruction sets. Since MAD (Multiply-Add-Divide) operation is essential in many hash functions to spread out the keys into the hash buckets, the vectorized Intel AVX-512 IFMA instruction will speed up the process of multiplication and addition operations. Replacing the vectorized multiplication and addition described in the above section with Intel AVX-512 IFMA instruction can accelerate the computation.

3.3.3 Accelerate Sketch Counter Updates by Leveraging Intel AVX-512 Gather and Scatter Instructions

Sketch-based algorithms require a counter-update after the proposed hashing computation. Instead of sequentially updating the sketch counter arrays one by one, the Intel AVX-512 gather and scatter instructions can be used as the multiple hash digests are already generated in the previous vectorized hash computation. By using Intel AVX-512 gather and scatter instructions, the sketch counter array can be updated in parallel.

4 Performance Benchmarking

4.1 Benchmarking Platform

As Intel AVX-512 instructions consume wider data per instruction and use more power, in some Intel processors, the CPU frequency may be lower when Intel AVX-512 instructions are executed. As the potential frequency reduction when using Intel AVX-512 is reduced in more recent Intel Xeon Scalable processor families, we use the 4th Gen Intel® Xeon® Scalable processor as the performance benchmarking platform. Table 3 describes the performance benchmarking platform details[6].

Table 3. Performance Benchmark Platform Configuration

Parameter	Description
Architecture	x86_64
CPU Model	Intel® Xeon® Gold 6454S CPU
# Socket(s)	2
# NUMA node(s)	2
Core(s) per socket	32
BIOS version	EGSDREL1.SYS.0091.D05.2210161328
Microcode	0x2b000181
SpeedStep, TurboBoost	Disabled
Core Frequency	2.2 GHz
OS	Ubuntu 22.04 (Jammy Jellyfish)
Kernel	5.15.0
DPDK Version	V23.07
Compiler	GCC 11.3.0
Grub Cmdline	hugepagesz=1G hugepages=40 default_hugepagesz=1G isolcpus=1-15,65-79,33-47,97-111 intel_iommu=on iommu=pt nohz_full=1-15,65-79,33-47,97-111 rcu_nocbs=1-15,65-79,33-47,97-111 nmi_watchdog=0 audit=0 nosoftlockup processor.max_cstate=0 intel_idle.max_cstate=0 hpet=disable mce=off tsc=reliable numa_balancing=disable intel_pstate=disable
Test Command	# DPDK_TEST=member_perf_autotest ./build/app/test/dpdk-test -l 6 --force-max-simd-bitwidth=512 --no-pci

4.2 Benchmarking Results

The optimal implementation solution based on the Count-Min Sketch algorithm is upstreamed on DPDK 22.11 release. The key size for the performance benchmarking is in the range of 4, 8, 9, 13, 16, 32, 37, 40, 48, and 64 bytes, which are the typical key size of the representative value of a data stream when using a sketch-based algorithm. As the performance benchmarking comparison reference, the CRC hash implementation, which is accelerated by CRC-32 instruction, is selected. As the CRC instruction hash implementation is proved as a high-performance hash function and the seed number is eight, in other words, it can be considered as eight hash functions. The test data set consists of 10 M packets of each key size in the above key size range. As the Add and Lookup operations are the most common operations in such sketch-based algorithms, both of the operations will invoke the multiple hash computation as the basic step. Here, the performance benchmarking comparison of Add and Lookup operations are illustrated in Figure 8 and Figure 9. It shows that the accelerated solution achieves an average **2.7x** performance gain on the Add operation and **2.4x** on the Lookup operation. Concerning the hash-collision ratio, it depends on the underlying hash algorithm. Taking our implementation underlying hash algorithm, xxHash64, for example, the collision test result looks good[7].

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

[7] https://github.com/Cyan4973/xxHash/wiki/Collision-ratio-comparison#testing-64-bit-hashes-on-large-inputs-

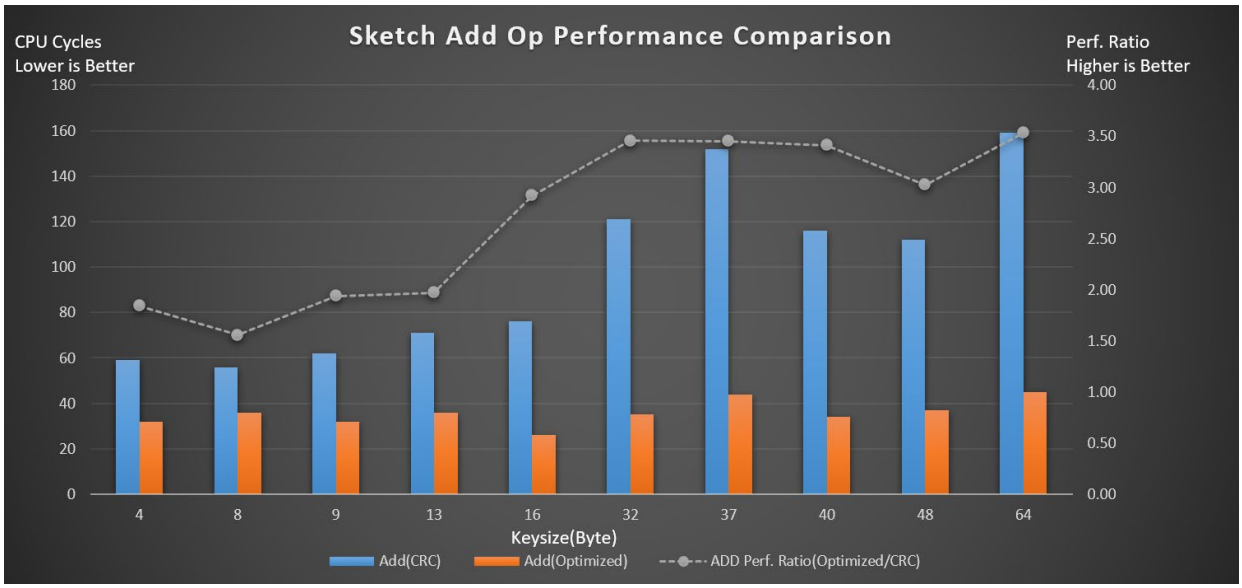

Figure 8. Performance benchmarking on Sketch Add compared with CRC-32 instruction

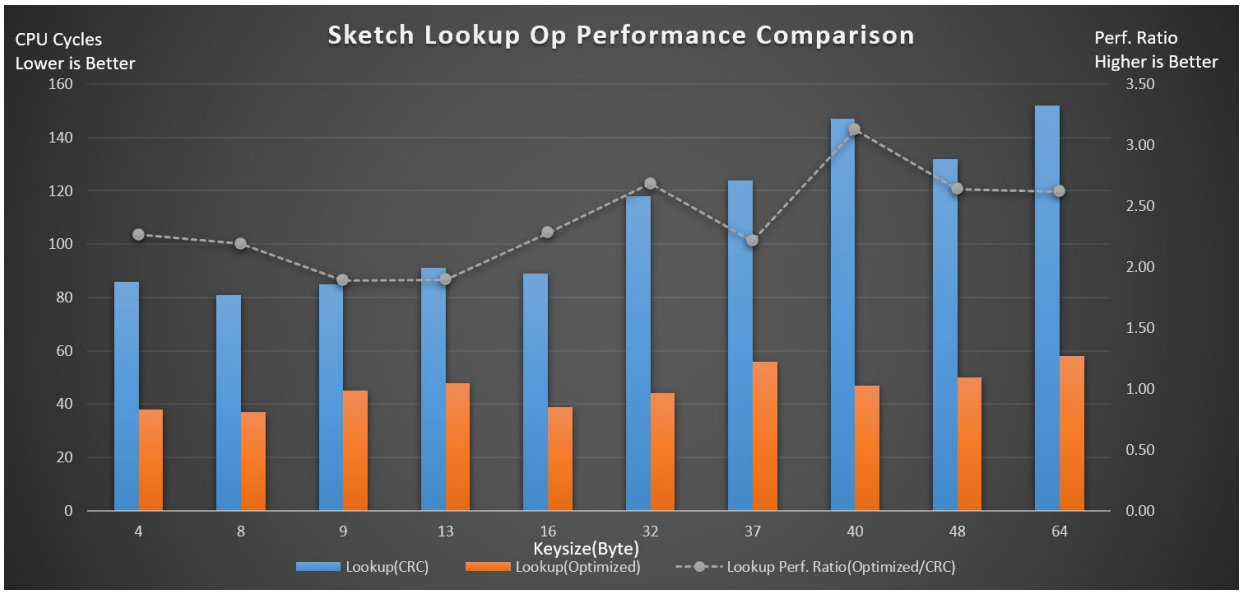

Figure 9. Performance benchmarking on Sketch Lookup compared with CRC-32 instruction

5 Summary

This technology guide demonstrates a brand new ultra-parallelized multi-hash computation model for data streaming workloads. The model offers several advantages, which have been detailed through the recently unveiled technique:

1. High-performance: This technology guide proposes a novel model by leveraging Intel AVX-512 instruction sets to get data-level parallelism when processing multi-hash computations. Based on xxHash, we achieve a performance gain of up to 2x on key-add and key-lookup operations compared to the standard CRC-32 instruction.
2. Flexibility: Although the algorithm implementation is evaluated based on xxHash, other simpler hashing algorithms can also be used with this model to gain even more performance, given the flexibility of the model.
3. Scalability: The model can be scaled to support keys of various lengths and various numbers of hashes as required, based on the level of parallelism supported by Intel AVX-512 instructions.

0823/DN/WIT/PDF 785248 -001US