TECHNOLOGY GUIDE

Intel Corporation

Galois Fields New Instructions - Method for Calculating Toeplitz Hash Using GFNI

Authors

Vladimir Medvedkin

Andrey Chilikin

Konstantin Ananyev

1 Introduction

The Toeplitz hash function is used for hash calculation to distribute packets across the queues with receive side scaling (RSS) in Ethernet devices. The Toeplitz hash function is frequently implemented in hardware of modern network interface cards (NIC) but it can also be used to calculate hash values in software use cases.

Software implementations of the Toeplitz hash function require a lot of CPU cycles, and calculation time is nondeterministic, for example due to the multiple branching involved in different implementations.

The proposed solution outlined here allows the elimination of branches and dramatically reduces the number of CPU cycles needed to calculate the Toeplitz hash signature.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	May 2022	Initial release.

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
GFNI	Galois Fields New Instructions
NIC	Network Interface Cards
RSS	Receive Side Scaling

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
Intel® Intrinsics Guide	https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#text=vgf2p8affineqb
Data Plane Development Kit	https://www.dpdk.org/

2 Overview

Modern NICs support multiple queues to provide network processing scalability for multicore CPUs. Incoming network packets can be distributed across these queues by RSS technology with the Toeplitz algorithm as the default hashing function. It works by parsing ingress packets and generating an n-tuple comprised of specific fields in the packet. Then NIC computes the Toeplitz hash signature using this n-tuple and a predefined Toeplitz RSS hash key.

Pseudocode for a Toeplitz function would look like the following:

```
// For hash-input input[] of length N bytes (8N bits)
// and random secret key K of X bits
ComputeHash(input[], N)
Result = 0;
For each bit b in input[] {
if (b == 1) then Result ^= (left-most 32 bits of K);
shift K left 1 bit position;
}
return Result;
```

In a conventional Toeplitz hash implementation in software, the number of iterations depends on the input value, or to be more precise on the number of non-zero bits in the input value. As such the number of cycles, and hence computation time, can vary significantly depending on input.

2.1 Existing Implementations and Limitations

There are several software implementations of the Toeplitz hash function. Some of them (DPDK*/Linux*/FreeBSD*) calculate hash by viewing input tuple bit by bit. This technique is very computationally intensive, and thus is both slow and indeterminate in terms of the number of cycles required. Another approach uses precalculated tables for every byte in a tuple. While this technique is faster, it is memory bound. Therefore, in real environments it can take a lot of cycles and performance may vary significantly. Some examples of software Toeplitz function implementations are:

- FreeBSD: sys/net/toeplitz.c: toeplitz_hash()
- Linux: include/xen/interface/io/netif.h: xen_netif_toeplitz_hash
- DPDK: lib/hash/rte_thash.h:rte_softrss()

2.2 Overview of Galois Fields New Instructions (GFNI)

Recent Intel® CPUs, from the 3rd Generation Intel® Xeon® Scalable processors onwards, have a new instruction set called the Galois Fields New Instructions (GFNI). One of these instructions is `vgf2p8affineqb[1]`, which computes an affine transformation in the Galois Field (2^8). For this instruction, an affine transformation is defined as "A * x + b" where "A" is an 8x8 bit matrix, and "x" and "b" are 8-bit vectors. One SIMD (single instruction, multiple data) register (operand 1) holds "x" as either 16, 32, or 64 8-bit vectors. A second SIMD register (operand 2) or memory operand contains 2, 4, or 8 "A" values, which are operated upon by the correspondingly aligned 8 "x" values in the first register. The "b" vector is constant for all calculations.

In fact, this instruction is a multiplication of a matrix of size 8x8 (matrix A) and a matrix of size 8x1 (one byte – x) with elements over the field Galois Field of order 2 (GF(2)) to obtain the resulting matrix of size 8x1.

2.3 GFNI-Based Toeplitz Hash Algorithm

Let us express the previously shown ComputeHash function as follows:

- N – size (in bits) of resulting hash value
- M – size (in bits) of input tuple value
- K – size (in bits) of Toeplitz key

Here, K must be ≥ N + M – 1

Let us use the following notation:

- h_n – n^{th} bit of hash value
- t_m – m^{th} bit of input tuple
- k_i – i^{th} bit of Toeplitz key

So, we can express the value of nth bit of result as follows:

```
hn = 0;
for (m = 0; m < M; m++)
        if (tm == 1)
                hn = hn ^ km + n
```

It can be expressed as:

```
hn = 0;
for (m = 0; m < M; m++)
        hn = hn ^ (tm AND km + n)
```

Since h_n, t_m and k_i are bits that is {0, 1} so we can use the GF(2) arithmetic. In GF(2) "AND" is equal to multiplication and ^ is equal to addition, so we can express the value of h_n as follows:

$$h_n = \sum_m t_m * k_{m+n}$$

Where m belongs to [0, M)

So, from a mathematical perspective the Toeplitz hash could be represented as a multiplication of an n x m Henkel matrix (that is, row-reversed Toeplitz matrix), which represents the hash key and an n-bit vector, which in turn represents the input tuple.

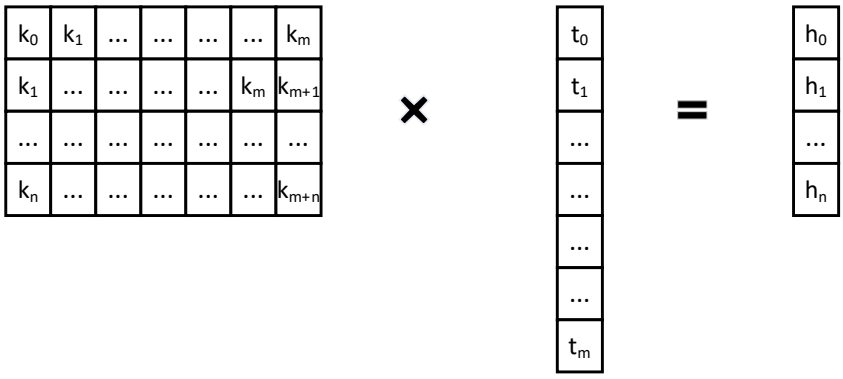

Figure 1. Toeplitz hash representation using matrix multiplication

The problem arises when we want to express multiplication of matrixes with sizes not equal to (8 x 8) and (8 x 1) with GFNI instructions as shown on Figure 2. In fact, in real world, hash values are usually 32 bit values, and input tuples have different sizes depending on their type (for example, ipv4 or ipv6 tuples have very different tuple sizes).

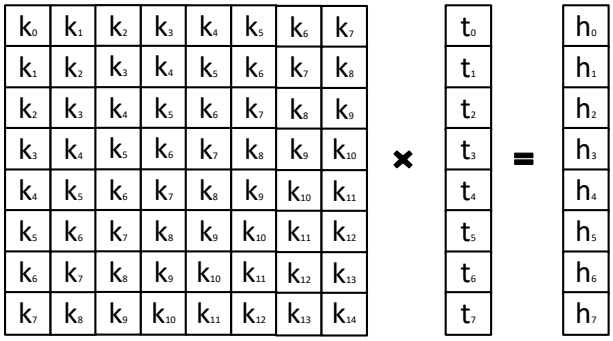

Figure 2. GFNI matrix multiplication

To overcome that problem we split the key matrix and the value matrix into 8 bit chunks so that it can be expressed via GFNI. In this case the key matrix could be represented as shown in Figure 3.

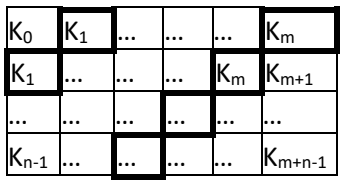

Figure 3. Large hash key matrix representation with 8x8 blocks

In addition, we can represent the hash key as a matrix with m/8 x n/8 elements, where each element itself is a 8x8 bit matrix. In the same way the input value can be represented as a vector of m/8 elements, where each element is an 8-bit vector.

So, H_i byte could be expressed as a multiplication of corresponding row by value bytes:

$$H_i = \sum_j K_{i+j} * T_j$$

Where j is in the range [0, M/8). When arithmetic is defined over GF(2), the addition operation is expressed as an XOR. Multiplication in turn is expressed as an AND operation.

3 Implementation

To apply the GFNI technique to some real-life use case, here is an example where we can accelerate Toeplitz hash calculation for RSS in case of the IPv4 traffic.

For example, for the most commonly used IPv4 4-tuple, which consists of {src_ip, dst_ip, src port, dst port} and is 12 bytes long, so m is 12 (where M = 12 * 8 bit) and n will be equal to N/8, which means 32/8 (because we are expecting to have 32-bit hash value). So, there will be 4 matrixes (8x8 bits each) in each column and 12 matrixes in a row.

At first sight we need to have 48 matrixes, but most matrixes repeat multiple times (up to n). So, in total we only need to have m+n-1 or 15 unique matrices, which are arranged as shown in Figure 4.

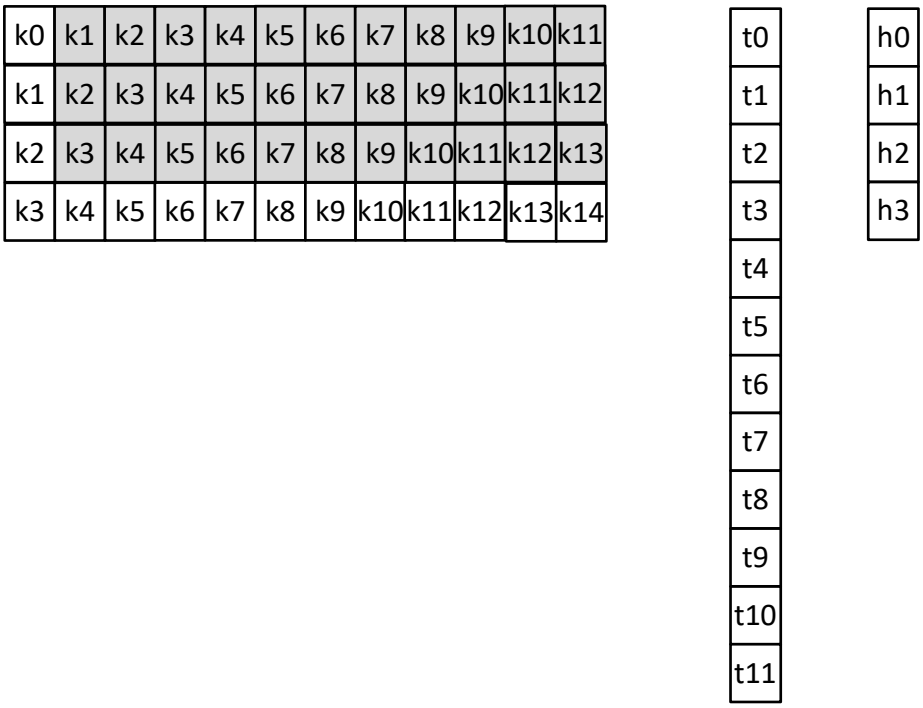

Figure 4. Example for 12-byte tuple

These 15 matrixes can be packed into two 512-bit registers as shown in Figure 5.

Figure 5. zmm registers representation

4 Performance

To test the performance of the GFNI enhanced function, we looked at the implementation in Data Plane Development Kit (DPDK). DPDK is a popular open source, set of libraries for high-speed packet processing in user space.

For 12-byte input tuples, which are commonly used for IPv4/TCP, the current Toeplitz hash implementation in DPDK takes ~178 cycles (varies depending on tuple content). Other tuple sizes used for benchmark are: IPv4 2-tuple (8 bytes), IPv6 2-tuple (32 bytes), IPv6 4-tuple (36 bytes).

The prototype using proposed GFNI implementation takes ~11.8 cycles, demonstrating up to26x times performance boost on the same platform.

Table 3 Performance benchmarks for scalar and GFNI Toeplitz hash calculation for different tuple sizes

	IPv4 2-tuple	IPv4 4-tuple	IPv6 2-tuple	IPv6 4-tuple
Tuple size (bytes)	8	12	32	36
rte_softrss_be (cycles)	178.8	314.1	775.2	872.4
rte_thash_gfni (cycles)	11.8	11.8	19.4	19.4
Ratio (higher is better)	15.1	26.6	39.9	45.0

Table 4 System configuration

ITEM	DESCRIPTION
Time	Thursday Feb 17 01:01:54 PM MST 2022
Board Manufacturer	Supermicro
Product Name	Intel® Xeon® Platinum 8352Y CPU @ 2.20GHz
BIOS Version	Supermicro 1.1
OS	Ubuntu 20.04.3 LTS (Focal Fossa)
Kernel	5.4.0-67-generic
Microcode	0xd000280
CPU Model	Intel® Xeon® Platinum 8352Y CPU @ 2.20GHz
Base Frequency	2.2GHz
Maximum Frequency	N/A
All-core Maximum Frequency	N/A
CPU(s)	32
Thread(s) per Core	2
Core(s) per Socket	32
Socket(s)	1
NUMA Node(s)	1
Prefetchers	All enabled
Turbo	Disabled
PPIN(s)	d884619cb203e802
Power & Perf Policy	Default (Balanced Performance)
TDP	205W
Frequency Driver	intel_pstate
Frequency Governer	Performance
Frequency (MHz)	2.1
Max C-State	6

Installed Memory	64GB (4x16GB DDR4 3200MT/s [2933MT/s])
Huge Pages Size	1GB
Transparent Huge Pages	madvise
Automatic NUMA Balancing	Disabled
Drive Summary	SanDisk SD8SBAT2 240GB

Table 5 Software configuration

ITEM	DESCRIPTION
Workload and version	DPDK 21.11 thash_perf_autotest
Compiler	gcc 9.3
Config	force-max-simd-bitwidth=512

5 Data Plane Development Kit (DPDK) API

To prove the concept, the following DPDK APIs were used:

```
static inline uint32_t
rte_thash_gfni(uint64_t *m, uint8_t *tuple, int len);
void
rte_thash_complete_matrix(uint64_t *matrixes, uint8_t *rss_key, int size);
rte_thash_complete_matrix() – prepares matrices from the given RSS hash
key
rte_thash_gfni() – calculates Toeplitz hash value
```

6 Summary

Toeplitz hash is used in a wide area of network applications and this proposed method can improve overall performance and execution predictability, which can be very important especially for real-time or time-critical environment.

In this guide, we demonstrated two main benefits of using Galois Fields New Instructions for Toeplitz hash implementations. The first one is the fast computation speed with up to 40x speedup compared to existing implementations depending on the input size and the second benefit is the predictable constant time of computation due to removing dependencies of distribution of ‘0’ and ‘1’ bits in the input tuple.

0522/DN/WIPRO/PDF 723537-001US