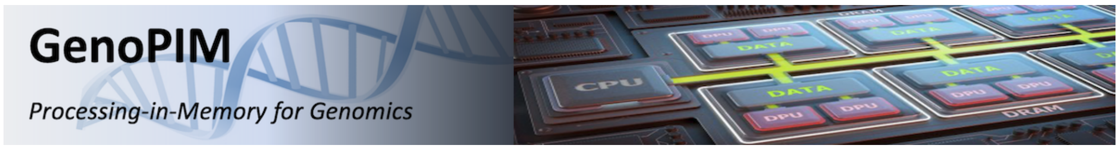

Non-cryptographic Hash Functions on UPMEM Data Processing Units

Abstract

Non-cryptographic hash functions are important for several data structures such as hash tables, Bloom filters or sketches, that do not have strict security requirements and rather need speed and collision resistance.

In this work, we consider the Processing-in-Memory technology designed by the UPMEM company and analyze the performance of various non-cryptographic hash functions on Data Processing Units (DPUs).

Table of Contents

1 UPMEM Data Processing Units

The UPMEM company has designed a Processing-in-Memory (PiM) architecture particularly fitted to accelerate memory-intensive applications [1]. This architecture extends the computational power of a CPU (called the host) by adding dual in line memory modules (DIMM) which embed data processing units (DPU) next to memory banks. These banks contain a 64 MB RAM (called MRAM) and a 64 kB scratchpad (called WRAM). One UPMEM DIMM is composed of two ranks of 64 DPUs, for a total of 8 GB of MRAM memory. A server equipped with UPMEM DIMMs becomes a massively parallel environment as illustrated on Figure 1, with each DPU running up to 24 threads (called tasklets).

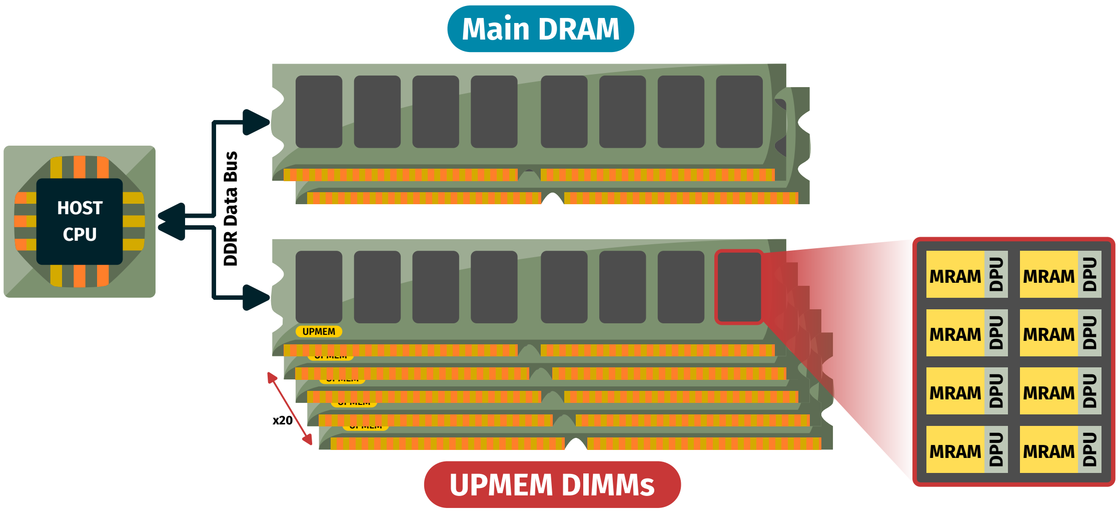

Figure 1: Overview of a server enriched with UPMEM DIMMs

Porting an application on such an architecture requires writing two programs: one for the host, and one for the DPUs. The host is responsible for orchestrating the DPUs, which include sending data to the MRAMs, launching the execution of the ranks, and retrieving potential results back to the host DRAM. This layout means the DPUs need to run enough computation to amortize the time it takes to transfer data back and forth between the host DRAM and the MRAMs.

The DPUs are a 32-bits architecture with no hardware support of multiplications. The latter are provided by software emulation and are thus expensive in terms of clock cycles. The DPUs also do not offer advanced vectorized instructions like recent CPUs do. As a consequence, hash functions that perform well on CPUs may be slow when it comes to running on DPUs. This work aims at highlighting non-cryptographic hash functions that are a good fit for the UPMEM PIM architecture.

2 Evaluation Setup

We execute our experiments on a server with an Intel® Xeon® Silver 4215 CPU @ 2.5 GHz processor (Skylake architecture), 256 GB of DDR4 @ 2.4 GHz RAM, and 20 UPMEM DIMMs @ 350 MHz (which corresponds to 40 ranks, i.e. 2560 DPUs, and a total of 160 GB of MRAM memory). The server operates on Debian 10 and uses version 2023.2.0 of the UPMEM's SDK.

We consider various hash functions: additive and rotative string hashing algorithms[1,2], Murmur3[3], SuperFastHash[4], FNV-1a[5], Jenkins's one-at-a-time[6] and xxHash[7]. For some hash functions, we indicate in the name which seed was used, as it may impact the performance. During the compiler optimizations, the number of instructions required to transform a multiplication by a constant seed into additions and shifts may indeed differ. We use a 32-bits integer output, and we pipe the result into a modulo operation to clamp the integer into [0, 8191]. Since the interval size is a power of 2, this modulo is done with a bitwise AND.

To evaluate the relevance of these hash functions on the UPMEM PIM architecture, we consider two metrics.

Clustering spread. We compute the hash value for every number from 0 to 8191, and we see the result as a bucket index in the context of a hash table. We count the number of occurrences of each index to track the required size of each bucket if we were to store all the input values in this hash table. In the end, we consider the root-mean-square (rms) of all bucket sizes as an indicator of the clustering spread and collision frequency. A perfect hash function would map each input to a different bucket and give a rms of 1. A spread imbalance would on the contrary leave some buckets empty while others contain many items, which gives a high rms score. So the closer to 1, the better.

Execution time. On one DPU, 16 tasklets compute the hash of every number from 0 to 8191, and repeat this process 1000 times. The work is distributed between tasklets with a static round-robin scheduling.

As a follow-up, we also run the execution time benchmark on single-threaded CPU to compare the rankings of hash functions on both architectures.

[1] https://www.partow.net/programming/hashfunctions/#AvailableHashFunctions

[2] https://stackoverflow.com/a/2351171

[3] https://gist.github.com/kevinmoran/471480b1e20a19b0687d81b75fd801c8

[4] https://www.azillionmonkeys.com/qed/hash.html

[5] https://gist.github.com/ctmatthews/c1cbd1b2e68c29a88b236475a0b26cd0

[6] https://en.wikipedia.org/wiki/Jenkins_hash_function

[7] https://github.com/Cyan4973/xxHash

3 Analysis Results

3.1 Clustering spread and execution time on DPUs

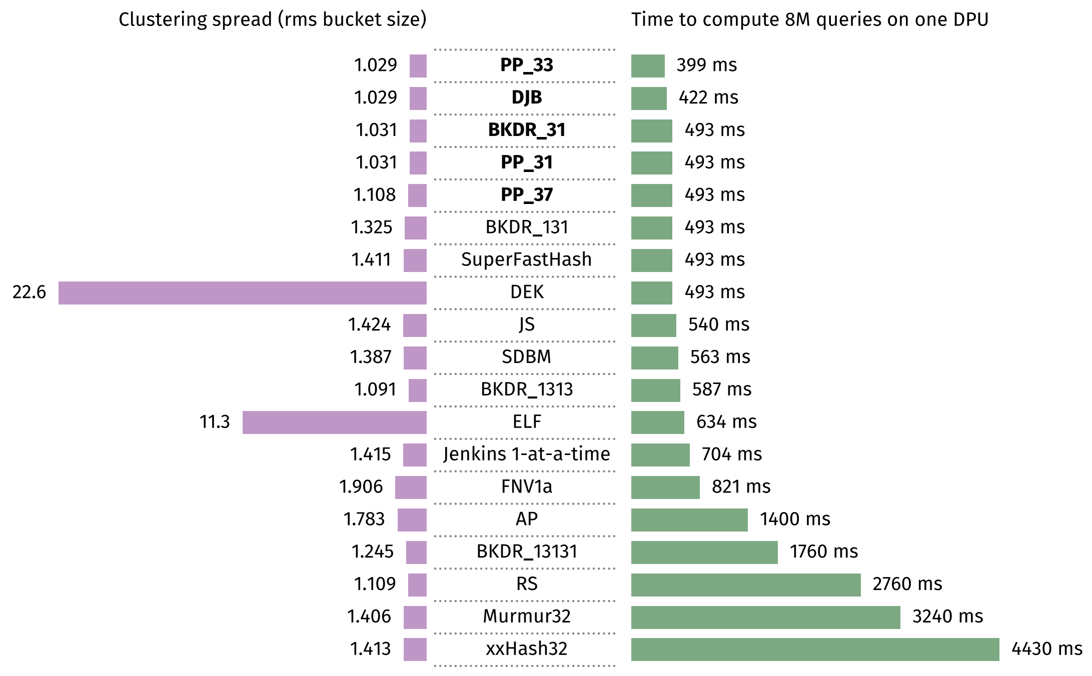

Figure 2: Performance of various non-cryptographic hash functions on DPUs

We give the results in Figure 2. The functions which are slower are those that use a lot of non-constant multiplications, like Murmur for instance. We highlight the top 5 functions, PP_33, DJB, BKDR_31, PP_31 and PP_37 that run fast, exhibit a good clustering spread, and are simple to implement. We recommend using those when needing a non-cryptographic hash on the DPUs.

3.2 Comparison of rankings on CPU and DPU

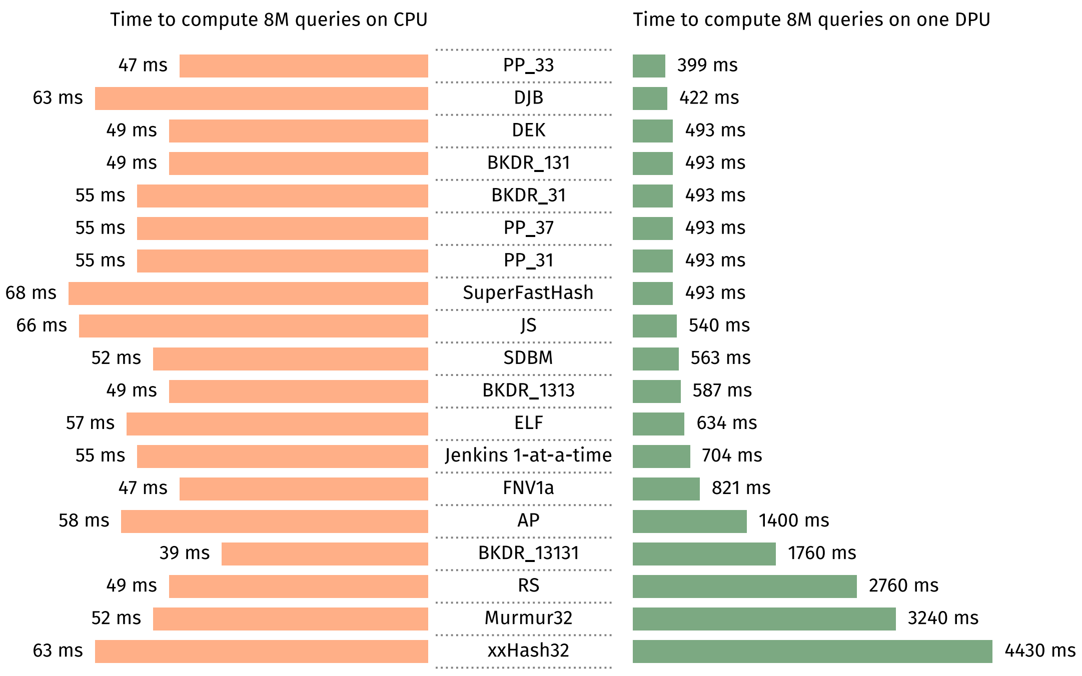

Figure 3: Performance of various non-cryptographic hash functions on DPUs

We give in Figure 3 the execution time on CPU. We see that the performances are much more uniform on CPU, hash functions with non-constant multiplications performing just as well as the others. These results stress the relevance of this study: porting an application to PIM requires taking into account the specificities of this architecture and selecting different hash functions that are more efficient on DPUs.

References

[1] Fabrice Devaux. The true Processing In Memory accelerator. pages 1–24. IEEE Computer Society, August 2019.