Evaluation of Hyperdimensional Computing on PIM and GPU

John Ramsden

***Abstract*— I perform a case study of energy usage and performance on two hardware accelerators primarily used for parallel compute: a GPU, and a processing in memory (PIM) accelerator. I perform experiments to investigate how energy usage differs between the two platforms. I discover that while PIM can show speedups over sequential CPU workloads, it does not compete with GPUs for both performance and energy efficiency in my workload.**

I. Introduction

In traditional systems, where CPUs are connected to main memory over a limited bandwidth channel, there is a significant obstacle known as the "memory wall", or "Von Neumann bottleneck". Processing-in-memory (PIM) offers a solution to overcome this challenge.

UPMEM DRAM Processing Units (DPUs) [1] represent a novel technology that facilitates PIM. They provide general-purpose processors where computation occurs in close proximity to data, resulting in exceptionally high memory bandwidth and low memory access latency. These advantages make it particularly valuable for applications that involve extensive computational tasks with frequent data access.

Due to the large number of DPUs accessible per UPMEM module (64), DPUs excel when applied to highly parallel computations. I evaluate how DPUs compare with the processing capabilities of modern GPUs in terms of parallel performance. Additionally, I examine the energy usage of these systems to facilitate a comprehensive comparison between the two solutions.

Hyperdimensional Computing (HDC) [2] is used in artificial intelligence to model the behavior of a large number of neurons. It relies on high-dimensional vectors (typically with at least 10,000 dimensions), referred to as "hypervectors". In a previous study [3], I conducted an evaluation of PIM by parallelizing the workload over DPUs. Now, I assess HDC on GPUs using a similar approach to the DPU evaluation.

In my previous work, as well as in this paper, I build upon PULP-HD [4]. The authors of PULP-HD developed an application for classifying electromyography (EMG) signals into hand gestures. Classification entails computing the Hamming distance between previously trained vectors, referred to as associative memory, and encoded hypervectors. Their implementation was designed for a low power RISC accelerator, and used OpenMP for parallelism.

The HDC workload is a very computation-heavy workload. The workload involves taking signal samples (the input of our workload) and encoding them to compute a hypervector. This compute-intensive workload is an extremely good candidate for a GPU, especially because the computation can be done completely in parallel over each sample. The HDC workload involves 32bit logical integer operations to manipulate hypervectors.

To compare GPU parallel performance and benefits to what DPUs achieved, I developed a GPU-based implementation using CUDA based on the same workload used with DPUs. I evaluated the advantages of using DPUs versus GPUs to further investigate the benefits of near-data computing and how it ultimately impacts performance, energy use, and cost.

II. Hardware

The hardware used in my experiments consists of a DPU system equipped with UPMEM DPUs, a GPU system with a GeForce RTX 3070 Ti, and a baseline system for sequential experiments with a Xeon(R) Silver 4216 CPU.

DPU System:

- 160 GiB PIM-enabled UPMEM DRAM
- 2304 DPUs @ 450MHz

GPU System:

- GeForce RTX 3070 Ti
- @ 1.58GHz base, 1.77GHz boost
- 6144 CUDA Cores

CPU Baseline System:

- Xeon(R) Silver 4216 CPU
- @ 2.10GHz

What is immediately apparent with this hardware is the large difference in compute capability of DPU (450MHz, 2304 DPUs) and GPU (1.58GHz, 6144 CUDA cores). Due to the substantial difference, I expected a significant performance difference between the two hardware platforms.

A. DPU

The DPU hardware I use has several relevant technical details I will refer to.

DPUs have a small (64KiB) working memory (WRAM). WRAM is used to hold stack and heap space. A larger 64MiB slice of memory is available to each DPU in the form of MRAM. MRAM is seen as an external peripheral, and transfers between WRAM and MRAM need to be made explicitly.

To hide memory latency, each DPU is equipped with up to 24 hardware threads, referred to as tasklets. While these tasklets cannot advance simultaneously, as the design follows an interleaved multi-threading (IMT) design [3], [5], they significantly enhance performance by enabling progression when one tasklet is in a busy waiting state for DMA.

III. Performance

To compare performance between the two hardware platforms I ran the same workload on GPU, DPU and CPU. I then compared GPU and DPU performance by examining the speed-up present between the baseline sequential CPU implementation and the two hardware platforms under test.

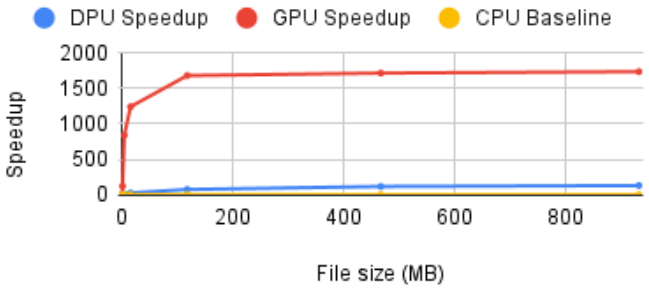

Fig. 1. DPU and GPU speedup multiplier over sequential CPU.

Fig. 1 illustrates the comparison between both hardware platforms and the sequential CPU. What is immediately apparent is how strongly the GPU scales, with it achieving up to 1700x speedup over a sequential CPU baseline. When scaling the workload across the GPU I used large enough block sizes and thread count to fully spread my workload across the GPU so that each thread dealt with one sample. For each of my experiments I varied the ratio of block size to thread size until I found the optimal combination. These block and thread sizes can be seen in Table. I.

In order to leverage hardware resources and identify performance plateaus, I tested with varying file sizes (corresponding to input samples). In my previous work [3], I used data sizes large enough to ensure each DPU had work to do, and I further distributed this data across tasklets on each DPU

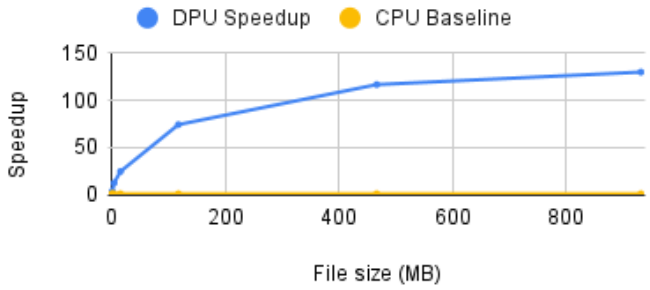

Fig. 2. DPU speedup over sequential CPU.

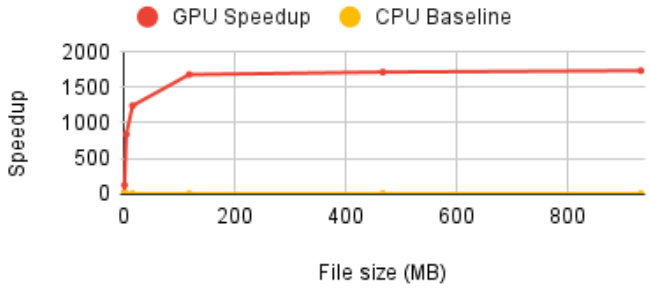

Fig. 3. GPU speedup over sequential CPU.

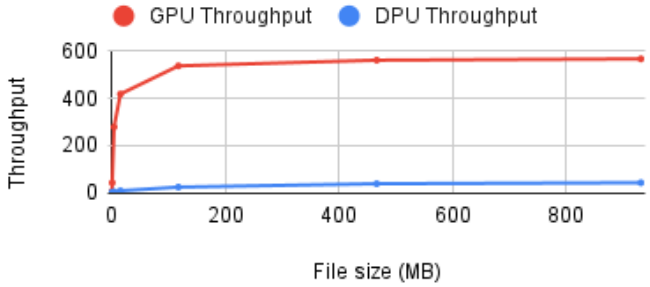

Fig. 4. GPU and DPU throughput.

Size (MB)	Time	Speedup	Blocks	Threads
0.05	0.01	6.13	16	16
0.49	0.01	122.06	128	32
3.7	0.01	838.32	192	128
15	0.04	1,241.90	512	192
117	0.22	1,681.66	2,048	384
466	0.83	1,716.62	4,096	512
931	1.64	1,736.73	4,096	512

TABLE I
GPU WORKLOAD. SPEEDUP COMPARED TO SEQUENTIAL CPU.

Size (MiB)	Time	Speedup	DPUs	Tasklets
0.05	0.16	0.44	12	15
0.49	0.39	3.65	254	18
3.7	0.90	12.39	512	18
15	1.81	24.58	1024	18
117	4.94	74.21	2304	18
466	12.23	116.56	2304	18
931	21.99	129.78	2304	18

TABLE II
DPU WORKLOAD. SPEEDUP COMPARED TO SEQUENTIAL CPU.

by utilizing heap space stored in WRAM. DPU and tasklet distribution can be seen in Table. II. Although the use of heap space made memory allocation conventional and simple, it imposed restrictions on space, limiting the file size to 4MiB due to the available heap space on the DPU. For my new experiments, I modified my original code to use buffers stored in MRAM and to explicitly transfer data between WRAM and MRAM as needed. This adjustment allowed me to better utilize DPU resources (where I was previously limited by WRAM) and consequently scale to larger sample sizes, with my tests involving file sizes as large as 1GiB.

As with the GPU workload, performance and throughput begins to plateau with larger file sizes, as shown in Fig. 2 and Fig. 3. The graphs showing speedup also correspond to throughput, as shown in Fig. 4. From this I deduce that we begin to fully utilize the compute capabilities on the various platforms. With smaller amounts of data performance and throughput is reduced as sufficient data to reach peak performance is not provided to the GPU or DPU.

IV. POWER

Although performance is significantly better on GPU, a primary concern is power usage and efficiency. DPUs have the potential to achieve better efficiency than GPUs in completing a similar workload over an extended period while saving on power. There are various scenarios where this might be acceptable or preferable. Users with throughput requirements below that of both accelerators, but above that of CPUs could be interested in pursuing this trade-off of losing performance but gaining energy efficiency. Therefore, asking whether we can achieve better efficiency with DPUs is a relevant question.

To measure DPU power usage, I measured the total draw from the baseboard management controller (BMC) connected to my server housing the DPUs. This gives me real-time power usage in watts. The measurement I obtain of DPU power usage shows both power draw from the regular DRAM modules the system is using and the DPUs. To determine how much power only the DPUs were using I also measured an identical server's power usage as a baseline. The difference in power usage between these two servers gives me the baseline power usage of the non-DPU DRAM modules, measured to be 2W. Then, to determine power usage of DPUs when under load I measured total power usage again during my workload, measured to be 50W. Therefore, subtracting non-DPU power, DPUs are using an average of 48W during load.

Due to a limitation of sufficient access to hardware on my test GPU system, I could not access a BMC. To measure GPU power usage, I utilized nvidia-smi to record the power draw in watts during my workload. While this is not the ideal solution, which would be measuring direct power usage at the hardware level, I was unable to get hardware access to my GPU. The GPU uses an average of 192W during execution of a workload.

Fig. 5 shows, in joules, the amount of energy used by the different accelerators to compute our results. In my experiments, energy usage on DPUs ranged from as low as three times more, up to 17 times more energy than the GPU. On average DPUs use on average 7.5 times more energy to compute the same results as a GPU.

So although the DPUs use less power, if we consider the fact that their workload will run longer, as observed in the amount of time taken for the same workload in Fig. 1, they consume significantly more energy overall in my use case.

V. FUTURE WORK

If this project were to be continued, several improvements and future work could be implemented.

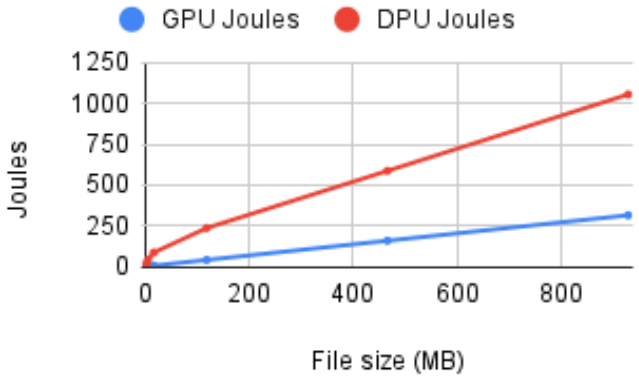

Fig. 5. Joules used by DPU and GPU.

Firstly, achieving a more detailed and precise determination of the actual power usage of components could be pursued. This would involve directly measuring the power draw of both GPU and DPUs at the hardware level, providing a better understanding of the difference in power usage between the two components. Although I measured DPU power usage at the hardware level, employing an actual hardware meter exclusively measuring the DPU power used during the workload would improve accuracy.

Secondly, experiments could be done with very large file sizes and spread across clusters of DPUs. Doing this would make it possible to see how many DPUs would be required to match the performance of a GPU.

Third, while my workload plateaus, I should conduct further investigation to ensure that I am fully utilizing the compute capabilities of the DPUs. I should analyze to determine the bottleneck in my workload and identify where the limiting factor lies in achieving better performance.

Lastly, this case study could be broadened to include other workloads with different performance characteristics to determine if the results found here are common across all workloads.

VI. Conclusion

PIM offers a new way of accomplishing parallel computing, and DPUs can provide a way to improve on performance over that of a sequential CPU implementation. However, their slower execution engine means that they may run longer to accomplish work similar to that of a GPU. This translates to increased energy usage. Previous experiments [6] have shown that DPUs *can* achieve better efficiency than GPUs given the right workloads. Workloads that PIM excel at are memory-bound workloads. Workloads like the one examined in this paper where they are computation-heavy, may not be able to take advantage of increased energy efficiency that other workloads can.

Acknowledgment

I thank Professor Alexandra (Sasha) Fedorova, and Joel Nider (Department of Electrical and Computer Engineering), for their advice and guidance in this project.

Availability

All source code for the projects described in the paper can be found at `https://github.com/UBC-ECE-Sasha/PIM-HDC`.

References

[1] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–24, IEEE Computer Society, 2019.

[2] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[3] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, *et al.*, "A case study of {Processing-in-Memory} in {off-the-Shelf} systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 117–130, 2021.

[4] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "Pulp-hd: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[5] W. Weber and A. Gupta, "Exploring the benefits of multiple hardware contexts in a multiprocessor architecture: Preliminary results," in *The 16th Annual International Symposium on Computer Architecture*, pp. 273–280, May 1989.

[6] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *2021 12th International Green and Sustainable Computing Conference (IGSC)*, pp. 1–7, IEEE, 2021.