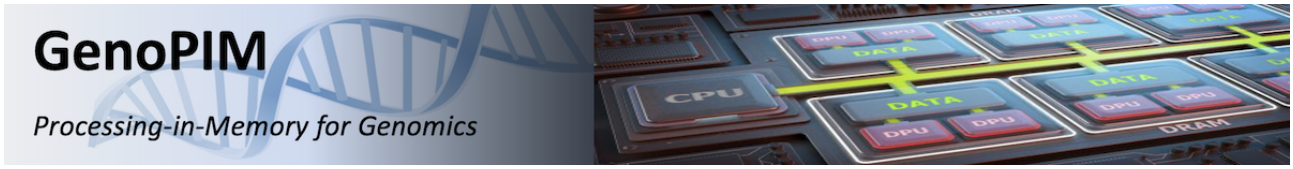

Analysis of Data Transfers with UPMEM Data Processing Units

Abstract

We consider the Processing-in-Memory technology designed by the UPMEM company and analyze the cost of data transfers between the host CPU and the Data Processing Units. We leverage our findings to devise a few practical recommendations for programmers to get the best performances when porting an application to this architecture.

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).

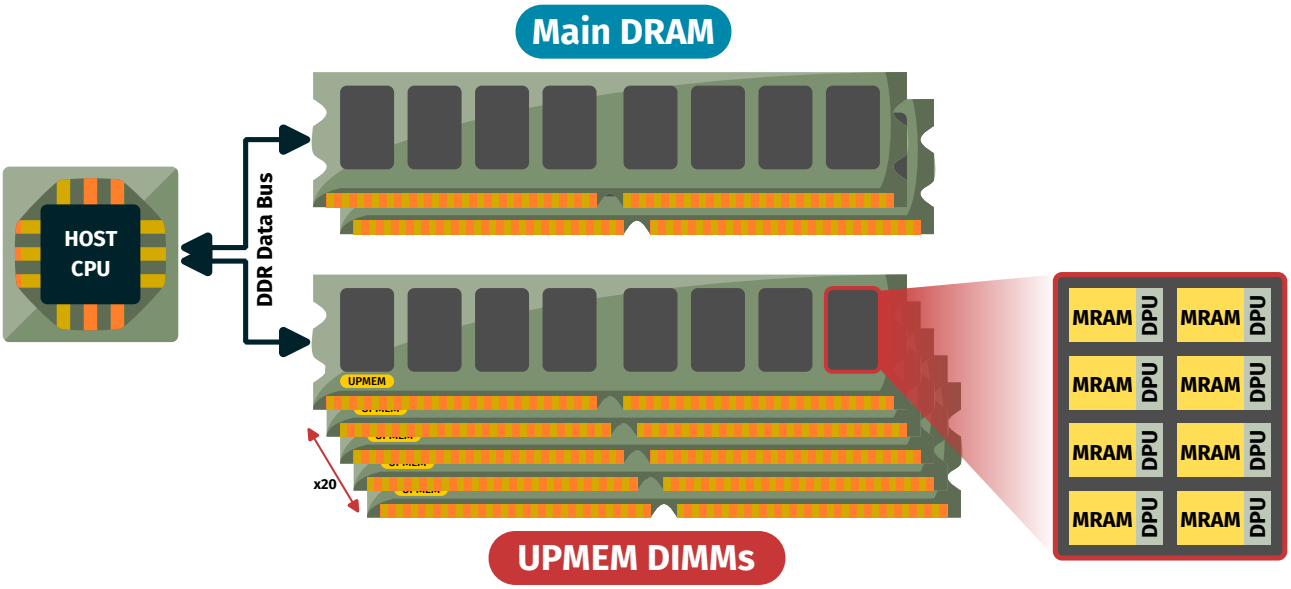

Figure 1: Overview of a server enriched with UPMEM DIMMs

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.

In this report, we analyze experimentally the cost of these transfers.

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.

In our benchmarks, we allocate a given number of ranks, and transfer an array of 32 bits integers. We test several configurations where we broadcast the same data to all ranks, or send and receive different data to and from each DPU.

3 Analysis Results

In Figure 2, we report the time it takes to broadcast various amounts of data. The transfer duration increases with the number of ranks and the array size.

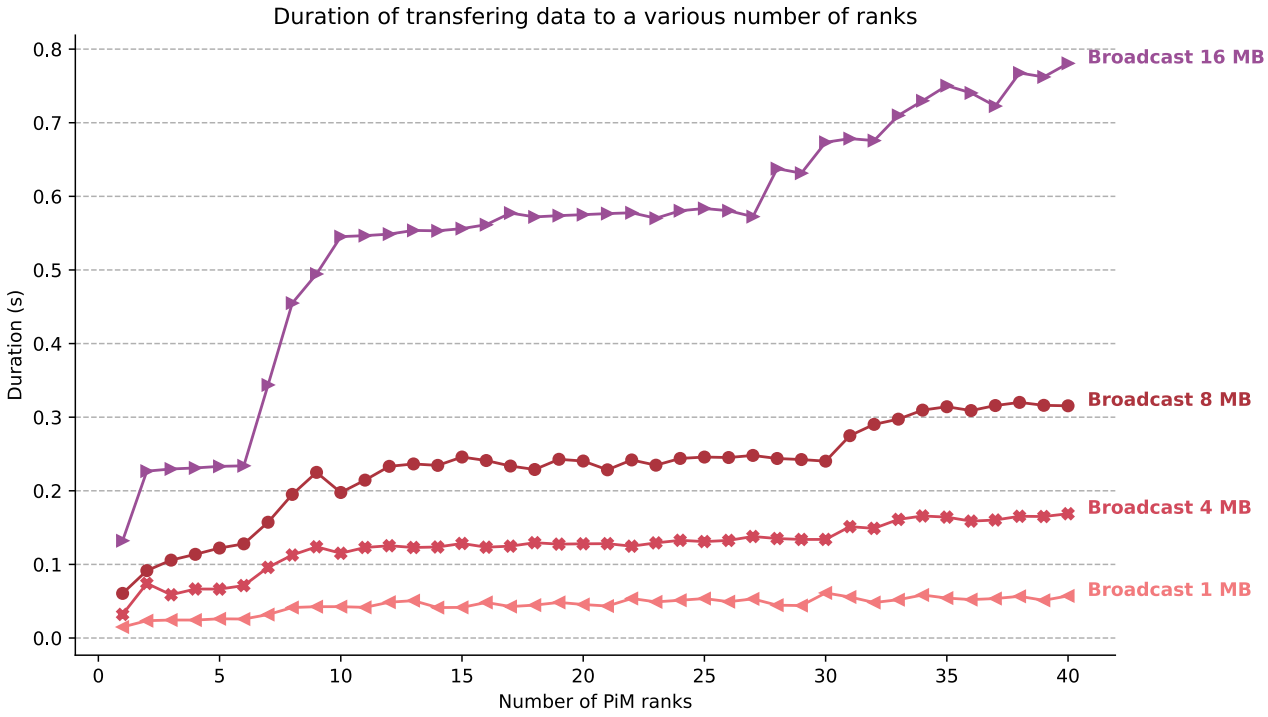

Figure 2: Broadcasting the same data to all ranks

To complement our analysis, we also compare the time it takes to broadcast the data with one operation, or by splitting the transfer in multiple blocks. As we see in Figure 3, when using 10 ranks, it is 30% faster to broadcast 64 MB by executing 8 transfers of 8 MB rather than sending the whole array at once. Interestingly, we notice the same phenomena for other series as well. It appears that the optimal data size is 8 MB. We suspect this happens because our server is equipped with a L3 cache of size 11 MB, which means dealing with larger blocks may become memory-bound.

Recommendation 3.1

We recommend to split transfers in blocks of 8 MB when possible.

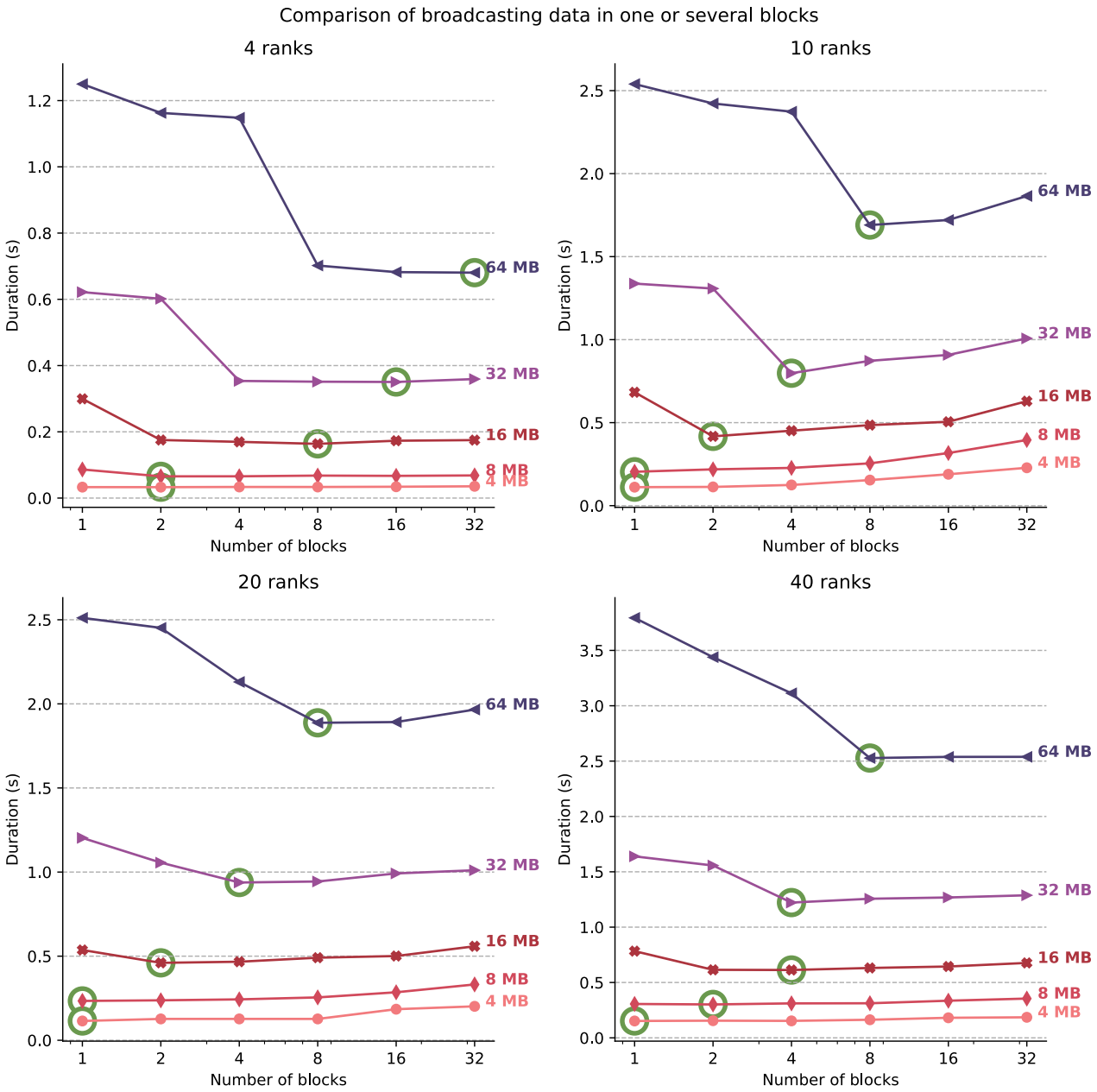

Figure 3: Broadcasting data in one or multiple blocks. Green circles show the minimum for each series

In Figure 4, we compare the time it takes to broadcast data with sending different arrays to each DPU. As we would expect, we see a significant difference. Sending different arrays is indeed more memory-intensive for the host as it requires to constantly fetch new addresses in the DRAM. On the flip side, broadcasting benefits from caching when the data fits in L3.

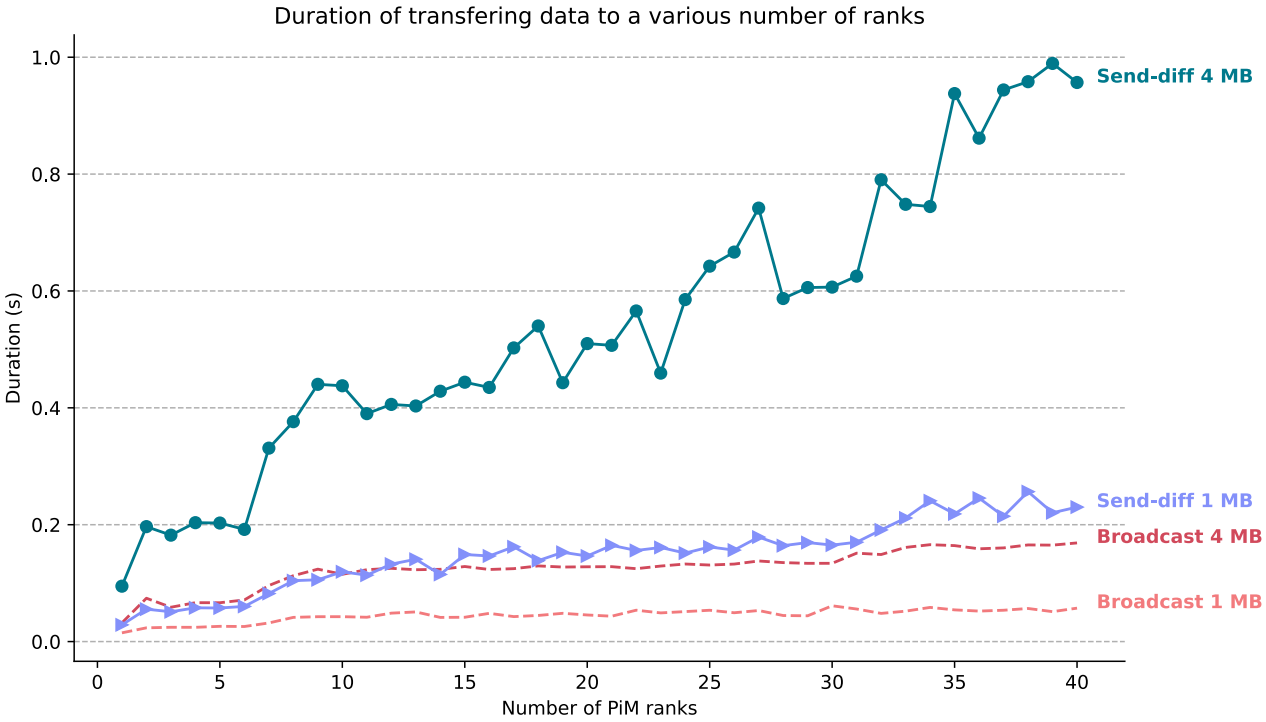

Figure 4: Broadcasting is faster than sending different data to each DPU

We also measure the time it takes to get results from the DPUs back to the host, and show results in Figure 5. It appears retrieving data is around twice slower than sending.

Recommendation 3.2

To obtain better speed-ups, we recommend porting applications with larger data transfers from the host to the DPUs rather than the opposite.

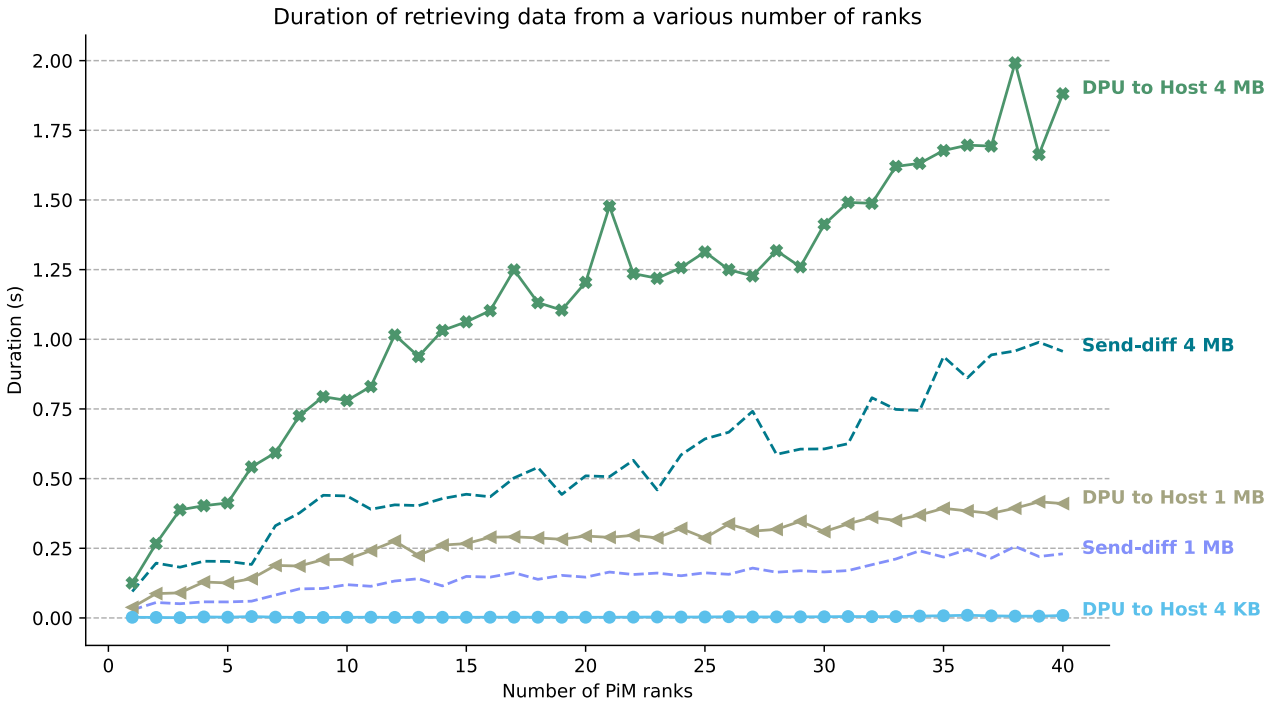

Figure 5: Fig

Finally, we compare in Figure 6 the performance of broadcasting data in MRAM or directly in the WRAM scratchpad. We observe that it is around 3 times faster to write data into the MRAM.

Recommendation 3.3

We recommend writing data from the host into the MRAM rather than the WRAM.

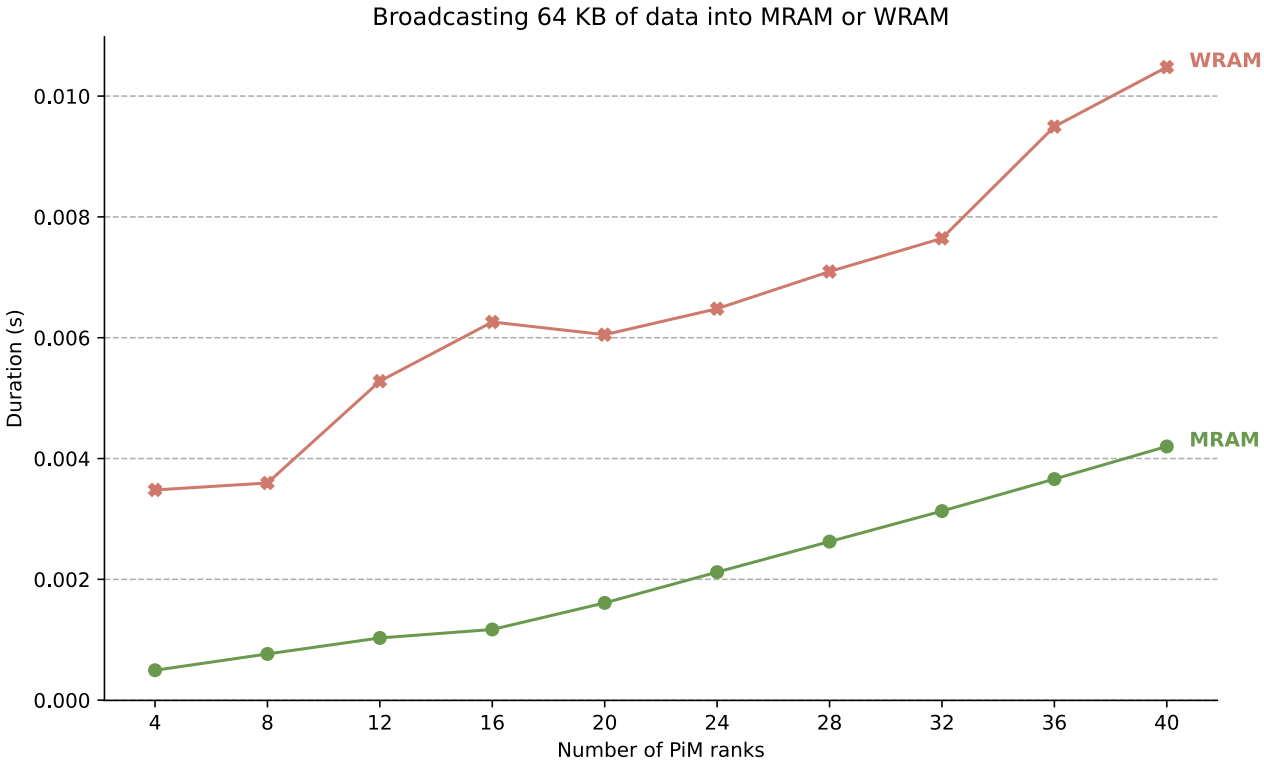

Figure 6: Broadcasting data into MRAM is faster than in WRAM

REFERENCES

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