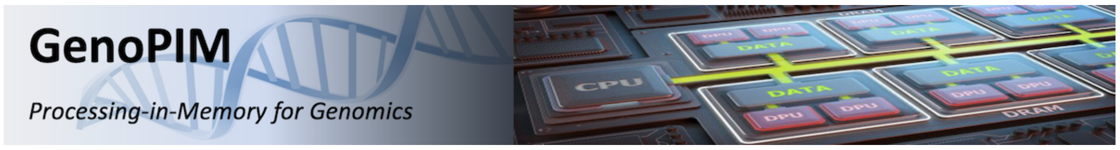

Side effects of Allocating UPMEM Processing Units

Abstract

We consider the Processing-in-Memory technology designed by the UPMEM company and analyze the side effects of allocating Data Processing Units (DPUs). In particular, we study how it impacts the disk speed.

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).

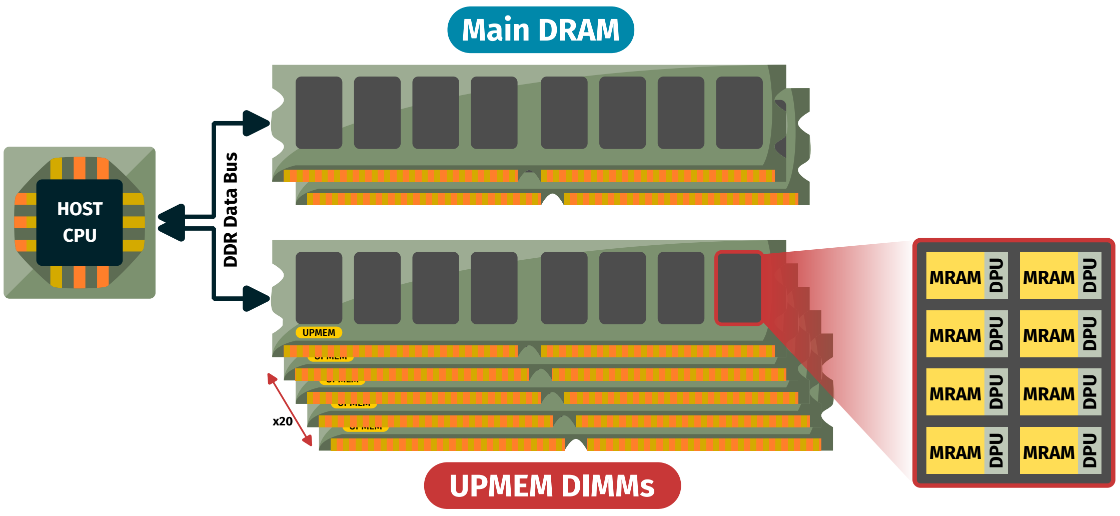

Figure 1: Overview of a server enriched with UPMEM DIMMs

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.

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 measure the execution time of reading one million sequences from a short reads dataset with a single thread, with and without a PIM rank allocated in the background. To experiment a multithreaded context, we also execute the benchmark with several threads that execute this same task in parallel with OpenMP. Measures are done with warmed up cache (i.e. we execute several measures and ignore the first one). Note that the PIM rank, when allocated, is not used at all. The DPUs do not run and no data transfer happens.

3 Analysis Results

3.1 Reading from the disk with warmed up cache

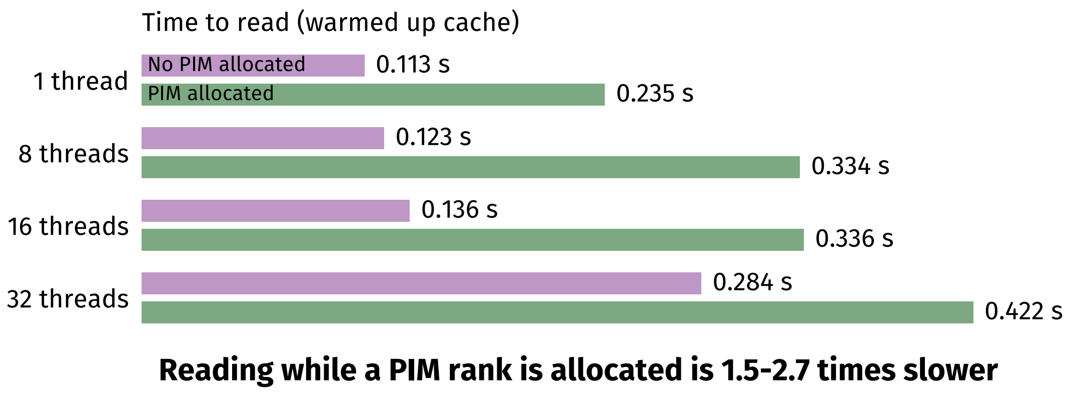

Figure 2: Execution time of reading with and without a PIM rank allocated

We see in Figure 2 a significant difference in the reading speed when a PIM rank is allocated in the background, even if the latter remains inactive. We suppose that this is due to the PIM allocation affecting the system virtual pages. In the end, reading while a PIM rank is allocated is between 1.5 and 2.7 times slower.

We deduce the following recommendation from this observation:

Recommendation 3.1

We recommend being mindful with the scope of the DPUs allocations.
In particular, we advise to free the DPUs as soon as possible to make sure it doesn't interfere with subsequent reading operations.
In cases where the disk reading speed is the main bottleneck, it may also be preferable to first fully load a file into memory before allocating any DPUs and then to iterate the data again to perform some PIM processing.

3.2 Writing to the disk

We repeat our benchmark, but replace reading short reads with writing blocks of *lorem ipsum* text to a file. This time, we see no execution time difference with and without a PIM rank allocated in the background. We conclude that the PIM allocation does not affect the disk writing speed.

References

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