DRAM-less Accelerator for Energy Efficient Data Processing

Jie Zhang[1], Gyuyoung Park[1], David Donofrio[2], John Shalf[2], Myoungsoo Jung[1]
Computer Architecture and Memory Systems Laboratory,
Korea Advanced Institute of Science and Technology (KAIST)[1], Lawrence Berkeley National Laboratory[2]
http://camelab.org

I. Introduction

General purpose hardware accelerators become a major data processing resource in many computing domains. However, the processing capability of hardware accelerators is often limited by costly software overheads and memory copies to support compulsory data movement between different processors and solid-state drives. This in turn wastes a significant amount of energy in accelerated systems.

Figure 1 shows results of an empirical evaluation that we performed on a real accelerated system. The system employs a high-performance multi-core based accelerator [1] and an advanced solid state drive (SSD) [2] through two different PCIe slots [3]. In this evaluation, we execute representative data-intensive workloads [4] on the accelerated system and compares the results to those of an idealized environment that has enough memory space to accommodate all data within the accelerator. We normalize application-level performance and energy consumption of the accelerated system with those of the ideal system. As shown in the figure, *the performance of such data processing on the accelerated system degrades as much as 74%, while consuming energy more than the ideal system by 9 times, on average.* This is because the SSD access requests, generated by computation kernels operating in the target accelerator, introduce many software interventions at the host side. Unfortunately, the SSD accesses consume most CPU cycles to move target data among multiple PCIe physical interconnections and software interface barriers.

To address these challenges, we propose, `DRAM-less`, a hardware automation approach that integrates many state-of-the-art phase change memory (PRAM) modules into its data processing fabric to dramatically reduce the unnecessary data copies with a minimum of software modifications. We implement a new memory controller that plugs a real 3x nm multi-partition PRAM to 28nm FPGA logic cells and interoperate its design into a PCIe accelerator emulation platform. The evaluation results reveal that `DRAM-less` consuming only 19% of the total energy of traditional accelerated systems.

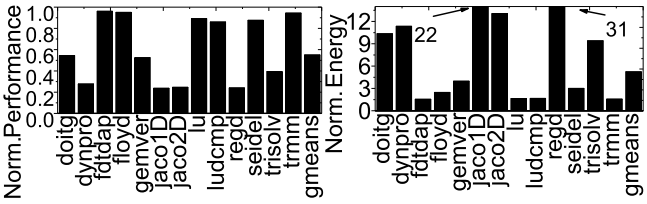
II. DRAM-less

A. Challenges

Figure 2 compares the architecture design and communication protocol between a conventional accelerated system and a system that employs our `DRAM-less` architecture. The accelerated system employs a CPU, an accelerator and an SSD as its external storage (Figure 2a). To prepare data for the accelerator, the conventional hardware acceleration approach requires retrieving a large amount of low-level data (in the form of files) from the SSD and deserializes them as a representation of objects within the host's DRAM. The host then transfers the data to the internal DRAM of the accelerator. Once the accelerator completes its data processing task(s), the results are written back to the SSD in an inverse order of data loading procedure. In this data transfer model, a CPU is required to frequently intervene to move the data among multiple user applications and OS modules. As the hardware accelerator and SSD devices, in practice, employ different software stacks, such interventions introduce many user/kernel mode switches and redundant data copies, which result in the waste of many CPU cycles.

B. High-Level View of DRAM-less

To reduce these software overhead and redundant memory copies, we build a hardware automated PRAM subsystem and tightly integrate it in the multi-core accelerator (cf. Figure 2b). In this design, we replace the accelerator-side DRAM with our proposed PRAM



(a) Performance. (b) Energy consumption.

Fig. 1: Performance degradation in near-data processing due to data movement overheads.

subsystem and allows all processing elements to access PRAM over a set of conventional memory instructions such as loads and stores. Since the large capacity of PRAM can accommodate a whole set of data to process, applications in our `DRAM-less` directly load the input data from the internal PRAMs without an external storage access. Consequently, as shown in Figure 2b, the host needs to prepare only computation kernel(s) without any input preparations (ⓐ); it can simply issue the kernel to the target hardware accelerator for its execution (ⓑ). We also implement an FPGA-based controller to automate all data accesses in PRAM modules, which fully complies with the memory instructions, `DRAM-less` can remove the data management (filesystem and storage stack) from its I/O path.

C. DRAM-less Data Processing Architecture

Processor. Figure 3 illustrates a high-level view of the proposed `DRAM-less`'s internal structure and microarchitecture of *processing element* (PE) therein [5]. The computation core of PE is designed by a multi-way single instruction, multiple data (SIMD) architecture, which is aimed to improve the performance of vector data processing. We choose this multicore architecture for our accelerator prototype implementation as a representative low-power processor.

Datapath. As shown in Figure 3, multiple PEs have their own L1 and L2 caches, which are connected to the crossbar network via a master port and a slave port. To process data in parallel, most PEs of our `DRAM-less` are allocated in handling the computational *kernel* provided by the host. While these PEs, referred to as *agents*, perform near-data processing, we designate one of PEs as a *server* to schedule all kernel executions on the agents by resuming and suspending them via a "power/sleep controller" (PSC). This server also manages the PRAM traffic requested by other agents. The server employs a memory controller unit (MCU) that takes over the L2 cache misses of an agent and administrates all the associated PRAM accesses by collaborating with the underlying FPGA controllers. The server simply sends a memory read or write message through a bus (cf. Figure 3), and then, the FPGA controllers take over all LPDDR2-NVM transactions (the corresponding messages) upon PRAMs. All these internal components are linked by the crossbar network, which is also bridged to `DRAM-less`'s PCIe module in enabling a host to communicate with the server.

PRAM subsystem. Even though the server and MCU handle all the read/write operations coming from parallel PEs, their memory requests should be converted to LPDDR2-NVM transactions and managed by the three-phase addressing protocol [6]. To mitigate the performance penalty, we design a hardware-automated PRAM subsystem and controller to manage multiple row address and row data buffers (i.e., RAB and RDB [6]) and govern three-phase addressing protocol, respectively. Specifically, our PRAM controller within the FPGA can selectively skip parts of the three addressing phases

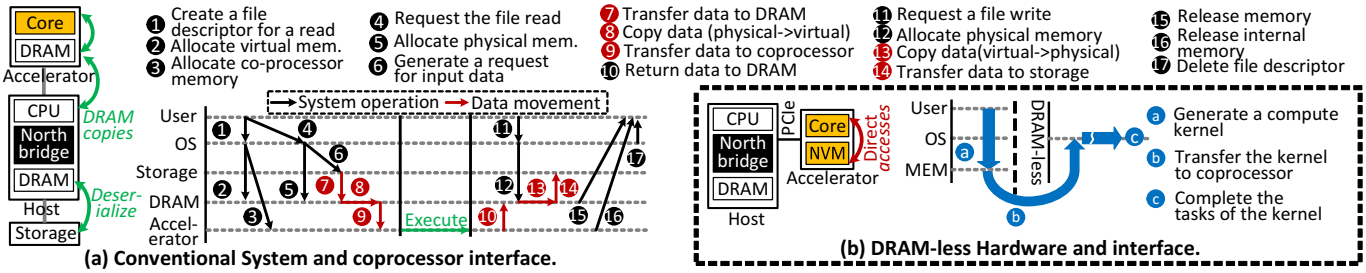


Fig. 2: Comparison between a traditional accelerated approach and our `DRAM-less`.

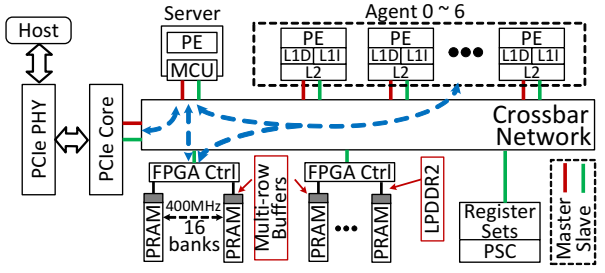
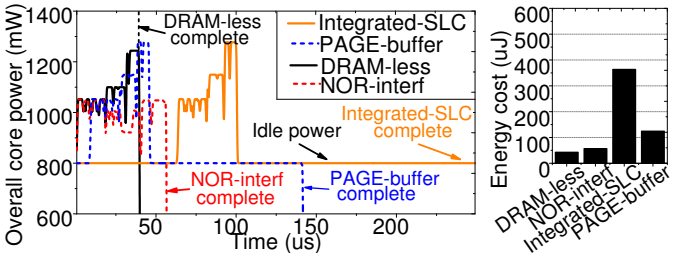


Fig. 3: `DRAM-less` architecture and processing elements.



(a) PE power consumption. (b) PE energy.

Fig. 4: Overall core power and total energy evaluation.

to reduce the I/O latency by being aware of the states of PRAM internal resources (e.g., RAB and RDB). Since the current memory interface generator (MIG) "does not" support PRAM, in addition to the memory controller, we implement our own PRAM physical layer on a 28nm Xilinx FPGA 19K logic cells [7], which manage our multi-partition PRAM modules over LPDDR2-NVM. Our physical layer (PHY) addresses the differences of operating frequency between PRAM and FPGA at 400MHz.

D. Hardware Automation Details

Multi-resource aware interleaving. To reduce the latency of data movements between PRAMs and L2 caches, we schedule memory requests by being aware of PRAM's multiple partitions and row-buffers [6]. Specifically, each PRAM module in the proposed `DRAM-less` can sense data out from a partition to a RDB, while transferring data out from other RDB(s) to target cache(s), in parallel. Thus, one can fully overlap the times to transfer data with PRAM accesses.

Selective erasing. A program is in practice composed by RESET and SET, which introduces a long write latency [8]. Similar to other NVMs (e.g., flash), PRAM also supports erase operations that reset a large number of cells (greater than cells in a program unit) to pristine state. Overwrites on erased cells only require SET operations, which can reduce the write latency. We investigate a simple but effective optimization for overwrites, referred to as *selective erasing*, which can reduce the overwrite latency without the penalty of erase operations (i.e., 60 *ms*). Specifically, while the server loads the target kernel, the PRAM subsystem can selectively program the all-zero data word for only the addresses that will be overwritten soon (but before completing the corresponding computation).

III. Evaluations and Conclusion

Experiments. We implement and evaluate four accelerated systems. (1) **Integrated-SLC**: We input SLC [9] flash based SSDs into the hardware accelerator, similar to active SSD approaches [10], [11]. (2) **DRAM-less**: We tightly integrate PRAMs into multi-core coprocessors and directly accesses PRAM-based subsystems over the hardware automation solution. (3) **PAGE-buffer**: We leverage the 3x nm memory sample that we used for `DRAM-less`, but performs all I/Os via a page-based interface with an assistance of its internal DRAM. (4) **NOR-intf**: We use 9x nm parallel PRAM [12] that employs a serial peripheral NOR flash interface. Like our PRAM, it allows a host to access the memory at byte-granule.

Power consumption and energy cost. To dig deeper power and total energy costs, we capture first 16KB data processing, and their results are illustrated in Figure 4. `NOR-interf` consumes, on average, 14% lower agent PE power compared to other approaches. This is because `NOR-interf` access the underlying storage over byte granularity without any help of DRAM. Thus, the load/store unit of DRAM-less processor is stalled and makes the whole processor idle. Note that, although `Integrated-SLC` and `PAGE-buffer` exhibit the actual time and overall core power for data processing similar to those of `DRAM-less`, their actual completion time is delayed and suffers from the long latency of data transfers, which increases their energy costs by 7 times and 1.9 times compared to `DRAM-less`.

In conclusion, `DRAM-less` can remove the overheads imposed by the host-side interventions and costly data movement. `DRAM-less` achieves lower energy consumption than advanced accelerated systems and active storage approaches.

IV. Original Publication

J. Zhang, G. Park, D. Donofrio, J. Shalf and M. Jung. 2019. DRAM-less: Hardware Acceleration of Data Processing with New Memory. IEEE HPCA. *https://github.com/jiezhang-camel/jiezhang-camel.github.io/blob/master/files/paper4-DRAMless.pdf*

V. Acknowledgement

This research is mainly supported by NRF 2021R1A2C4001773, MemRay grant (G01190170), DOE DEAC02-05CH11231, NRF 2015M3C4A7065645, and KAIST start-up package (G01190015).

References

[1] T. Anderson *et al.*, "A 1.5 ghz vliw dsp cpu with integrated floating point and fixed point instructions in 40 nm cmos," in *ARITH*, IEEE, 2011.
[2] Intel, "Intel ssd 750 series," *http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-750-series.html*, 2015.
[3] J. Zhang and M. Jung, "Flashabacus: a self-governing flash-based accelerator for low-power systems," in *EuroSys*, ACM, 2018.
[4] L.-N. Pouchet, "Polybench: the polyhedral benchmark suite," *http://www.cs.ucla.edu/~{}pouchet/software/polybench/*, 2012.
[5] T. Instruments, "Tms320c66x dsp corepac user guide," 2011.
[6] G. Park *et al.*, "{BIBIM}: A prototype multi-partition aware heterogeneous new memory," in *HotStorage 18*, 2018.
[7] Xilinx, "Xilinx zynq-7000 all programmable soc zc706 evaluation kit." https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf, 2018.
[8] M. K. Qureshi *et al.*, "Preset: improving performance of phase change memories by exploiting asymmetry in write times," *Computer Architecture News*, 2012.
[9] Micron, "Mt29f2g08aabwp/mt29f2g16aabwp nand flash datasheet," 2004.
[10] B. Gu *et al.*, "Biscuit: A framework for near-data processing of big data workloads," in *ISCA*, IEEE, 2016.
[11] S. Cho *et al.*, "Active disk meets flash: a case for intelligent ssds," in *ICS*, ACM, 2013.
[12] Numonyx, "Omneo, p8p pcm, 128-mbit parallel phase change memory datasheet,"