Advancing Near-Data Processing with Precise Exceptions and Efficient Data Fetching

Sairo Santos[†‡] Tiago R. Kepe[†] Francis B. Moreira[§] Paulo C. Santos[§] Marco A. Z. Alves[†]
[†]Department of Informatics – Federal University of Paraná – Curitiba, Brazil
[‡]Department of Exact Sciences and Information Technology – Federal Rural University of the Semi-arid – Angicos, Brazil
[§]Informatics Institute – Federal University of Rio Grande do Sul – Porto Alegre, Brazil
Email:[†]{trkepe, fbm, mazalves}@inf.ufpr.br [‡]{sairo.santos@ufersa.edu.br} [§]{pcssjunior@inf.ufrgs.br}

***Abstract*—Near-Data Processing (NDP) modifies the traditional computer system design by placing logic near the memory, bringing computation to the data. One NDP approach places such elements on the logic layer of 3D-stacked memories to quickly access data while avoiding reliance on narrow buses and better accessing the parallelism these devices offer. However, NDP architectures often fail to fully leverage available memory resources. In this work, we propose adding an instruction buffer to a common NDP design with large vector instructions. This modification allows the NDP to fetch instruction operands out of program order and delegates some responsibility regarding precise exceptions to the near-data device. Our results show our modifications cause a reduction in execution time of up to 28% while consuming up to 25% less energy.**

***Index Terms*—near-data processing, 3d-stacked memory, computer architecture**

I. Introduction

Big Data applications behave in a data-centric fashion that evades cache memory logic, presenting extensive data streaming and little data reuse [1]–[5]. Thus, most data accesses by such applications require fetching data from the main memory, which consumes excessive time and energy [6]–[8]. The issues caused by this are widely referred to as the memory wall [6].

Near-Data Processing (NDP) is a research field that addresses the memory wall issue by adding processing elements close to the data storage elements of computer systems, thus making them more data-centric, as opposed to traditional computation-centric architectures [7]. One issue NDP faces is the added complexity of maintaining overall system consistency when a processing element separate from the host processor is added to the architecture. This is often achieved by only offloading one instructions per time to the NDP device and handling instructions strictly in order, especially in fine-grain NDP architetures [7], [9], [10], which causes inefficiencies such as forcing the device to be idle between tasks. In this paper, we extend an existing fine-grain NDP architecture by adding an instruction buffer to the device so it is able to fetch data from the main memory more efficiently by loading instruction operands out of order while guaranteeing system consistency. In the remainder of this text we describe how we extend an NDP architecture by adding precise exceptions support and efficient data fetching, and we simulate and evaluate the performance of common kernels on this experimental architecture regarding execution time and energy consumption.

This work was partially supported by the Serrapilheira Institute (grant number Serra-1709-16621), CAPES and CNPq (Brazilian Government).

II. Proposed Modifications

We propose adding two elements to NDP architectures: (i) an instruction buffer and (ii) a memory disambiguation mechanism. With an instruction buffer, a NDP architecture becomes able to pool its specific instructions, which enables loading the operands of multiple instructions in parallel. This modification allows the NDP device to utilize even more of the data throughput the main memory provides, which may convert into further performance gains. Execution and committing of instructions is still done in order, and thus data fetching can safely be performed out of order.

Supporting this ability requires the system to take steps to guarantee system consistency. First, the system must ensure that no instructions between NDP instructions in the program impact the memory hierarchy state, so as to not risk the NDP device fetching outdated data from memory and subsequently committing erroneous results to the memory. Second, the NDP device must be able to maintain the instructions in this buffer, thus becoming responsible for flushing instructions and data should exceptions arise.

To ensure precise exceptions, however, we must guarantee that no data shall be fetched from the memory that will be modified by an instruction that already exists in the instruction buffer. A memory disambiguation mechanism is thus added to keep track of the memory addresses to which instructions will write their results, which are checked to ensure that data won't be fetched out of order if an older near-data instruction is going to modify them.

III. Evaluation Methodology and Results

We chose to use the Vector-In-Memory Architecture (VIMA) [11] NDP architecture for this case study, but believe our results can be replicated on any NDP architecture based on similar principles, e.g. any architecture that supports fine-grain offloading to a NDP co-processor. NDP approaches that consider 3D-stacked memories often vectorize large amounts of data as operands to exploit the wide internal bandwidth of these devices, achieving increased throughput as a result. In this case, we consider that VIMA uses 8 KB vectors

as operands in its instructions, as described in its original reference [11]. Figure 1 illustrates a 3D-memory module with the VIMA architecture.

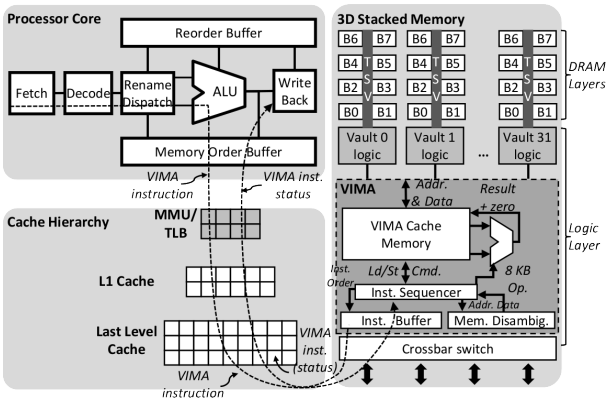

Fig. 1. 3D-memory module with VIMA architecture. Adapted from [11] to include our proposed modifications.

All simulation experiments were done using SiNUCA [12], an open-source cycle-accurate simulator. We use data streaming kernels *Memory Set*, *Memory Copy* and *Vector Sum* to assess how efficiently the design can exploit the internal bandwidth and throughput of a 3D-stacked memory. Since VIMA has reuse capabilities, we use data reuse kernels *Stencil* and *Bloom Filter* to evaluate the usefulness of this type of feature. We used datasets of 8MB, 16MB, 32MB, and 64MB in size for each experiment and the baseline architecture we considered mirrors Intel's Skylake microarchitecture.

Figure 2 shows speedup results of the original VIMA design against a 16-thread x86 baseline for all the benchmarks and input sizes. These experiment results showcase the expected benefits of a NDP architecture when compared with a traditional computation-centric system when tasked with running a data-centric application. With data-streaming kernels such as *MemSet*, *MemCopy* and *VecSum*, the parallelism intrinsic to NDP is largely leveraged to achieve superior performance. The data reuse kernels, *Stencil* and *Bloom Filter*, see improved performance from the combination of extensive data-streaming and some reuse of data fetched from the memory. Namely, *Bloom* presents superior behavior to the data-streaming kernel benchmarks since the kernel repeatedly accesses several auxiliary data structures, thus making extensive use of the dedicated cache memory, achieving a speedup of over $9\times$ when considering the 16 MB dataset.

Figure 3 shows results of our modified design under the same comparison. While results for *MemCopy*, *VecSum* and *Stencil* stay largely the same regarding both execution time and energy consumption, the *MemSet* and *Bloom* kernels display significant improvement on both fronts. Regarding *MemSet*, this happens mainly due to improved usage of the data throughput offered by the memory device, as hypothesized. The kernel is based on single operand instructions and thus greatly benefits from the ability to load operands of multiple instructions in parallel, accessing more of the throughput

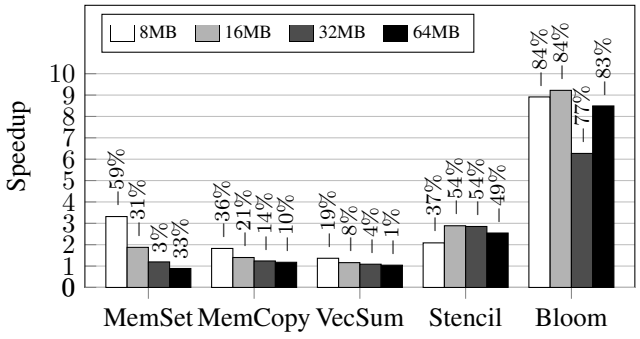

Fig. 2. Speedup of VIMA over baseline x86 with 16 threads (higher is better). Figures over 1 indicate improvement regarding execution time. Numbers over bars refer to energy consumption relative to baseline (lower is better).

capabilities of the memory device. Execution time is reduced by 28% for the largest dataset considered, while energy consumption is reduced by 25% compared to the original NDP design. Meanwhile, the *Bloom* kernel benefits from improved usage of the architecture's data reuse capabilities through the pooling of instructions, which in turn allows it to use the memory throughput more efficiently. Execution time is reduced by 11% compared to the original NDP design while energy consumption is reduced by about 10% compared to the original NDP design.

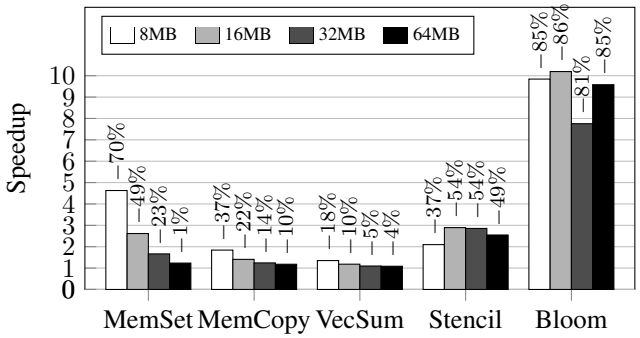

Fig. 3. Speedup of VIMA with proposed modifications normalized to baseline x86 with 16 threads (higher is better). Figures over 1 indicate performance improvement regarding execution time. Numbers over bars refer to energy consumption relative to baseline (lower is better).

IV. Conclusions and Future Work

In this work, we modify an existing 3D-stacked memory-based NDP architecture [11], to advance discussion within this class of NDP device. We analyze its performance by simulating kernels that highlight features of the architecture and the possibilities our modifications enable, which yielded significant execution time and energy consumption improvements.

As future work we plan to explore further possibilities enabled by these modifications. Namely, out-of-order data fetching may benefit near-data multi-threaded processing and also improved performance with smaller vector widths than the one considered in our experiments.

REFERENCES

[1] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *interactions*, vol. 19, no. 3, pp. 50–59, 2012.

[2] P. Xie, G. Sun, F. Wang, and G. Luo, "V-pim: An analytical overhead model for processing-in-memory architectures," in *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE, 2018, pp. 107–108.

[3] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, 2007.

[4] M. K. Qureshi, M. A. Suleman, and Y. N. Patt, "Line distillation: Increasing cache capacity by filtering unused words in cache lines," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 2007, pp. 250–259.

[5] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 316–331.

[6] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, 1995.

[7] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, 2014.

[8] M. Hashemi, E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Accelerating dependent cache misses with an enhanced memory controller," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 444–455.

[9] M. A. Alves, M. Diener, P. C. Santos, and L. Carro, "Large vector extensions inside the hmc," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1249–1254.

[10] P. C. Santos, G. F. Oliveira, D. G. Tomé, M. A. Alves, E. C. Almeida, and L. Carro, "Operand size reconfiguration for big data processing in memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 710–715.

[11] A. S. Cordeiro, S. R. dos Santos, F. B. Moreira, P. C. Santos, L. Carro, and M. A. Alves, "Machine learning migration for efficient near-data processing," in *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2021, pp. 212–219.

[12] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux, "Sinuca: A validated micro-architecture simulator," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015, pp. 605–610.