Blurring the Lines between Memory and Computation

Reetuparna Das
University of Michigan

Computer designers have traditionally separated the roles of storage and computation. Memories stored data. Processors computed them. Is this distinction necessary? A human brain does not separate the two so distinctly, so why should a computer? Before addressing this question, let us start with the well-known memory wall problem.[1]

What is the memory wall in today's context? The memory wall originally referred to the problem of growing disparity in speed between fast processors and slow memories. Since 2005 or so, as processor speed flatlined, memory latency has remained about the same. But as the number of processor cores per chip kept increasing, memory bandwidth and memory energy became more dominant issues. A significant fraction of energy is spent today in moving data back and forth between memory and computing units, a problem that is exacerbated in modern data-intensive systems.

How do we overcome the memory wall in today's computing world that is increasingly dominated by data-intensive applications? For well over two decades, architects have tried a variety of strategies to overcome the memory wall. Most of them have centered on exploiting locality. Here is an alternative: what if we could move computation closer to memory—so much that the line that divides computation and memory starts to blur?

The First Wave

Researchers discussed processing in memory (PIM) in the 1990s[2–6] (initial suggestions date back to as early as the 1970s[7]) as an alternative solution to scale the memory wall. The key idea was to physically bring the computation and memory units closer together by placing computation units inside the main memory (DRAM). But this idea did not quite take off back then, due to the high cost of integrating computational units within a DRAM die. Another factor may have been the fact that cheaper optimizations were still possible, thanks to Moore's law and Dennard scaling.

The advent of commercially feasible 3D chip stacking technology, such as Micron's Hybrid Memory Cube (HMC),[8] has renewed our interest in PIM. HMC stacks layers of DRAM memory on top of a logic layer. Computational units in the logic layer can communicate with memory through high-bandwidth through-silicon vias. Thanks to 3D integration technology, we can now take computational and DRAM dies implemented in different process technologies and stack them on top of each other.

The additional dimension in 3D PIM allows an order of magnitude more physical connections between the computational and memory units, and thereby provides massive memory bandwidth to the computational units.[9–15] The available memory bandwidth is so high in these systems that a general-purpose multicore processor with tens of cores is a poor candidate to take advantage of 3D PIM. The bandwidth of cheaper conventional DRAM is mostly adequate for these general-purpose processors. Better candidates are customized computational units that can truly take advantage of the abundant memory bandwidth in 3D PIM data-parallel accelerators, such as a GPU, or even better, customized accelerators such as Google's Tensor Processing Unit.[16]

Although 3D PIM is a clear winner in terms of memory bandwidth compared to conventional DRAM, its latency and energy advantages are perhaps exaggerated in literature. 3D PIM brings computation closer to DRAM memory. It has no effect on the energy spent accessing data within DRAM layers, DRAM refresh and leakage, and on-die interconnect in the logic layer, which together happen to be the dominant cost. To be clear, there is some memory latency and energy reduction as it eliminates communication over the off-chip memory channels by integrating computation in 3D PIM's logic

0272-1732/17/$33.00 © 2017 IEEE

layer. However, this benefit is not likely to be a big win and paves a smaller step toward reducing the steep data-movement overheads.[17,18]

The Second Wave

Although PIM brings computational and memory units closer together, the functionality and design of memory units remains unchanged. An even more exciting technology is one that dissolves the line that distinguishes memory from computational units. Nearly three-fourths of silicon in processor and main memory dies is simply to store and access data. What if we could take this memory silicon and repurpose it to do computation? Let us refer to the resulting unit as *Compute Memory*.

Compute Memory repurposes the memory structures, the ones that are traditionally used only to store data, into active computational units for near-zero area cost. Compute Memory's biggest advantage is that its memory arrays morph into massive vector computing units (potentially, one or two orders of magnitude larger than a GPU's vector units), as data stored across hundreds of memory arrays could be operated on concurrently. Because we do not have to move data in and out of memory, the architecture naturally saves the energy spent in those activities, and memory bandwidth becomes a meaningless metric.

Micron's Automata Processor (AP)[19,20] is an example for Compute Memory. It transforms DRAM structures to a Nondeterministic Finite Automata (NFA) computational unit. NFA processing occurs in two phases: state match and state transition. AP cleverly repurposes the DRAM array decode logic to enable state matches. Each of the several hundreds of memory arrays can now perform state matches in parallel. The state-match logic is coupled with a custom interconnect to enable state transition. We can process as many as 1,053 regular expressions in Snort (a classic network-intrusion detection system) in one go using little more than DRAM hardware. AP can be an order of magnitude more efficient than GPUs and nearly two orders of magnitude more efficient than general-purpose multicore CPUs! Imagine the possibilities if we can sequence a genome within minutes using cheap DRAM hardware.

AP repurposed just the decode logic in DRAMs. Could we do better? In our recent work on Compute Caches,[21,22] we showed that it is possible to repurpose SRAM array bit-lines and sense-amplifiers to perform in-place analog bit-line computation on the data stored in SRAM. A cache is typically organized as a set of sub-arrays; as many as thousands of sub-arrays, depending on the cache level.[23–25] These sub-arrays can all compute concurrently on several hundred thousands of data elements stored in them with little extensions to the existing cache structures, while incurring an overall area overhead of 4 percent. Thus, caches can effectively function as large vector computational units, whose operand sizes are orders of magnitude larger than conventional SIMD units. Of course, it also eliminates the energy spent in moving data in and out of caches. While our initial work supports few useful operations (logical, search, and copy), we believe that it is just a matter of time before we are able to support more complex operations (including comparisons, addition, multiplication, sorting).

Supporting Compute Caches' style-in-place, analog bit-line computing in DRAMs is more challenging. The problem is that DRAM reads are destructive—one reason why DRAMs need periodic refresh. Although in-place DRAM computing may not be possible, an interesting solution is to copy the data to a temporary row in the DRAM[8] and then do bit-line computing. This approach will incur extra copies, but retains the massive parallelism benefits.

Unlike DRAMs, bit-line computing may work well in a diverse set of nonvolatile memory technologies (RRAMs, STT-MRAMs, and Flash). Researchers have already found success in repurposing structures in emerging NVMs to build efficient ternary content-addressable memory (TCAM)[26] and neural networks.[27–29]

Computational memories can be massively data parallel—potentially, an order of magnitude more performance and energy efficient than modern data-parallel accelerators such as GPUs. Such dramatic improvements could have a transformative effect on applications ranging from genome sequencing to deep neural networks. However, capabilities of computational memories may not be as general purpose as GPUs are today, and may impose additional constraints in terms of where data is stored. Application developers may have to rework their algorithms to fully take advantage of Compute Memory. Modern data-parallel domain-specific language frameworks such as CUDA and Tensorflow can be adapted to help these developers. It may also require runtime and system software support to meet computational memory constraints such as data placement.

As the general-purpose core's efficiency flatlined over the past decade, both industry and academia have wholeheartedly embraced customization of computational units. It is high time for us to think about customizing memory units as well. While there are many ways that one could think of customizing memory, turning it into powerful accelerators is one of the more exciting avenues to pursue. Until recently, we have viewed computing and memory units as two separate entities. Even within a processor, caches and computational logic have operated as two separate entities that served different roles. The time has come to dissolve the line that separates them.

References

1. W.A. Wulf and S.A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *SIGARCH Computer Architecture News*, vol. 23, no. 1, 1995, pp. 20–24.
2. M. Gokhale, B. Holmes, and K. Iobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer*, 1995, vol. 28, no. 4, 1995, pp. 23–31.
3. Y. Kang et al., "FlexRAM: Toward an Advanced Intelligent Memory System," *Proc. Int'l Conf. Computer Design*, 1999, pp. 192–201.
4. P. Kogge, "Execube: A New Architecture for Scaleable MPPs," *Proc. Int'l Conf. Parallel Processing*, vol. 1, 1994, pp, 77–84.
5. M. Oskin, F. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," *Proc. 25th Ann. Int'l Symp. Computer Architecture*, 1998, pp. 192–203.
6. D. Patterson et al., "A Case for Intelligent RAM," *IEEE Micro*, vol. 17, no. 2, 1997, pp. 34–44.
7. H.S. Stone, "A Logic-in-Memory Computer," *IEEE Trans. Computers*, vol. C-19, no. 1, 1970, pp. 73–78.
8. *Hybrid Memory Cube Specification*, 2014; http://hybridmemorycube.org.
9. J. Ahn et al., "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," *Proc. 42nd Ann. Int'l Symp. Computer Architecture*, 2015, pp. 336–348.
10. A. Farmahini-Farahani et al., "NDA: Near-Dram Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," *Proc. IEEE 21st Int'l Symp. High Performance Computer Architecture*, 2015, pp. 283–295.
11. D. Kim et al., "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 380–392.
12. S. Pugsley et al., "NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software*, 2014, pp. 190–200.
13. V. Seshadri et al., "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," *Proc. 46th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2013, pp. 185–197.
14. D. Zhang et al., "Top-PIM: Throughput-Oriented Programmable Processing in Memory," *Proc. 23rd Int'l Symp. High-Performance Parallel and Distributed Computing*, 2014, pp. 85–98.
15. Q. Zhu et al., "A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing," *Proc. IEEE Int'l 3D Systems Integration Conf.*, 2013, doi:10.1109/3DIC.2013.6702348.
16. N.P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *Proc. 44th Ann. Int'l Symp. Computer Architecture*, 2017, pp. 1–12.
17. K. Bergman et al., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, DARPA, 2008.
18. B. Dally, "Power, Programmability, and Granularity: The Challenges of ExaScale Computing," *Proc. IEEE Int'l Parallel Distributed Processing Symp.*, 2011, p. 878.
19. Micron Automata Processing; www.micronautomata.com.
20. P. Dlugosch et al., "An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 12, 2014, pp. 3088–3098.
21. S. Aga et al., "Compute Caches," *Proc. 23rd Int'l Symp. High Performance Computer Architecture*, 2017, pp. 481–492.
22. S. Jeloka et al., "A Configurable TCAM/BCAM/SRAM using 28nm Push-Rule 6T Bit Cell," *Proc. IEEE Symp. VLSI Circuits*, 2015, pp. C272–C273.
23. W.J. Bowhill et al., "The Xeon R Processor E5-2600 v3: A 22 nm 18-Core Product Family," *J. Solid-State Circuits*, vol. 51, no. 1, 2016, pp. 92–104.
24. W. Chen et al., "A 22nm 2.5 MB Slice On-Die L3 Cache for the Next Generation Xeon R Processor," *Proc. IEEE Symp. VLSI Technology*, 2013, pp. C132–C133.
25. M. Huang et al., "An Energy Efficient 32-nm 20-MB Shared On-Die L3 Cache for Intel R Xeon R Processor E5 Family," *J. Solid-State Circuits*, vol. 48, no. 8, 2013, pp. 1954–1962.
26. Q. Guo et al., "Resistive Ternary Content Addressable Memory Systems for Data-Intensive Computing," *IEEE Micro*, vol. 35, no. 5, 2015, pp. 62–71.
27. M.N. Bojnordi and E. Ipek, "Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning," *Proc. IEEE Int'l Symp. High Performance Computer Architecture*, 2016, pp. 1–13.
28. P. Chi et al., "Prime: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 27–39.
29. A. Shafiee et al., "Isaac: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *Proc. 43rd Int'l Symp. Computer Architecture*, 2016, pp. 14–26.

Reetuparna Das is an assistant professor in the Electrical Engineering and Computer Science Department at the University of Michigan. Contact her at reetudas@umich.edu.