Dataflow based Near Data Computing Achieves Excellent Energy Efficiency

Charles Shelor and Krishna Kavi
Department of Computer Science and Engineering
University of North Texas, USA

ABSTRACT

The emergence of 3D-DRAM has rekindled interest in near data computing (NDC) research. This article introduces dataflow processing in memory (DFPIM) which melds near data computing, dataflow architecture, coarse-grained reconfigurable logic (CGRL), and 3D-DRAM technologies to provide high performance and very high energy efficiency for stream oriented and big data application kernels. The application of dataflow architecture with a CGRL implementation provides a flexible, energy efficient computing platform. The initial evaluation presented in this paper shows an average speedup of 5.5 is achieved with an energy efficiency factor of 460.

Keywords

Dataflow, Computer Architecture, Near Data Computing, Processing-in-Memory, Coarse Grain Reconfigurable Logic, energy efficient computing, big-data, 3D stacked DRAM

1. INTRODUCTION

One of the major problems facing today's computer systems is the disparate timing between processor instruction cycles and memory access cycles, sometimes tagged the "memory wall" [17]. The cache hierarchy in processors has been used to mitigate the effects of the memory wall by providing fast access to data items on subsequent accesses. However, there are some classes of applications that do not exhibit repeated access to the same data items. These classes of applications achieve little to no benefit from the cache hierarchy. Examples of these classes include streaming and big data applications [6]. One approach to improving the performance of these applications is Near-Data-Computing (NDC) that includes Processing-in-Memory (PIM) as one form. This approach moves the processing closer to the memory to achieve faster access and higher bandwidth.

The recent commercialization of 3D, stacked DRAM [12] provides the opportunity to integrate processing on the logic layer of the stacked DRAM. While this logic layer provides high bandwidth access to the memory, there are limitations to the size and power of the processing elements. These limitations preclude the use of standard, high performance, out-of-order processors for PIM applications.

The use of dataflow techniques based on Coarse Grain Reconfigurable Logic (CGRL) offers processing capacity and power efficiency suitable for the PIM applications [16][7]. A dataflow processing-in-memory (DFPIM) structure using CGRL consists of a set of functional blocks with a reconfigurable interconnection. The interconnection of the blocks is configured to implement the dataflow graph of the PIM application. The dataflow paths are synchronized and pipelined such that a new element is typically computed on every clock cycle. The parallelism and pipelining provide high performance while requiring less energy than out-of-order processors. In this paper, we analyze the algorithm or source code to identify the kernels and generate dataflow graphs of the kernels. An XML representation of the dataflow graph is input to our DFPIM simulator which verifies the accuracy of the input by generating results for comparison to the benchmark reference data. The simulator also provides performance and energy estimates for the kernel.

Our initial findings show speedups averaging 23.3 with energy efficiency improvements averaging 596 times better than the host processor baseline for single instance analysis. Allowing the host processor to execute 32 instances of the kernel in parallel compared to a DFPIM configuration with multiple kernel instances still showed an average speedup of 5.5 with an energy efficiency improvement of a factor of 460 times.

Section 2 provides a detailed description of the DFPIM concept. Section 3 provides results and analysis of the DFPIM system. Section 4 discusses future work on DFPIM. Section 5 examines research that is closely related to DFPIM. Section 6 provides summary and conclusions for the DFPIM work presented in this paper.

2. DFPIM CONCEPT

Dataflow Processing-in-Memory (DFPIM) is based on melding three technologies. The dataflow paradigm extracts the available concurrency from the algorithms. Coarse Grain Reconfigurable Logic (CGRL) provides an efficient and flexible method to implement the dataflow processing. 3D-stacked DRAM includes a logic layer on which the CGRL can be implemented and provides low latency, high bandwidth access to memory.

2.1 Dataflow

Dataflow is a style of computing where the data values "flow" from one operation to the next operation. Dataflow is highly concurrent and self-synchronizing [blocked]. Generating and analyzing dataflow graphs are integral components to optimizing compilers for high level programming

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

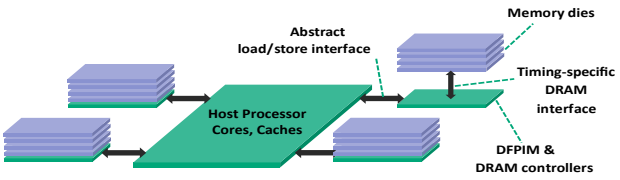

Figure 1: Example DFPIM system with Stacked DRAM.

languages. DFPIM utilizes the dataflow graphs of the PIM applications to extract parallelism, detect dependencies, and arrange pipelining of the application. Pure dataflow uses data availability at each operation to determine when to 'fire' the operation and generate a new result. This provides the self-synchronizing characteristic of dataflow. However, this also introduces substantial overhead in dataflow processing which has limited its commercial deployment.

Dataflow does not have a concept of memory as only values are utilized. DFPIM uses load units at the dataflow graph inputs to get the needed values from memory. The load units have 2 buffers per graph input that each hold a row of DRAM memory. The system will be accessing data from one row buffer while the other row buffer is being filled through a memory access. Similarly, there are store units to write rows back to memory as required. There are delay operations in the dataflow graphs to balance and synchronize the path lengths. The dataflow graph is 'executed' only when all graph inputs for the next computation are available. This single level of synchronization reduces the dataflow overhead by not requiring synchronization at each operation in the graph. The pipelined graphs also handle loop carried dependencies and simplifies memory ordering issues. Small configurable logic blocks based on look up tables are added to the dataflow and allow boolean decisions within the dataflow graphs. Programmable state machines are used to implement looping structures within the dataflow graphs to increase graph execution independence from a host processor or controller.

2.2 Coarse Grain Reconfigurable Logic

CGRL provides a set of functional blocks that are configured at run-time to implement an algorithm. Each functional block is implemented completely in silicon logic to minimize latency and power. This distinguishes CGRL from Fine Grain Reconfigurable Logic (FGRL) used by standard Field Programmable Gate Array (FPGA) devices where the functional block is implemented using programmable look-up tables interconnected with programmable switches. Dedicated functional units require less space, less power, and provide better timing. CGRL used as Coarse Grain reconfigurable accelerators (CGRA) provide significant energy efficiency and performance benefits [8]. The functional blocks interconnect with other functional blocks using programmable, bus-based routing. This arrangement allows one output to drive multiple inputs if a data value is used more than once.

2.3 3D-Stacked DRAM

3D-Stacked DRAM is a high density, high bandwidth memory subsystem that is created by stacking multiple DRAM semiconductor dies vertically and communicating through the stack using Through-Silicon-Vias (TSV) [11]. Chang [3]

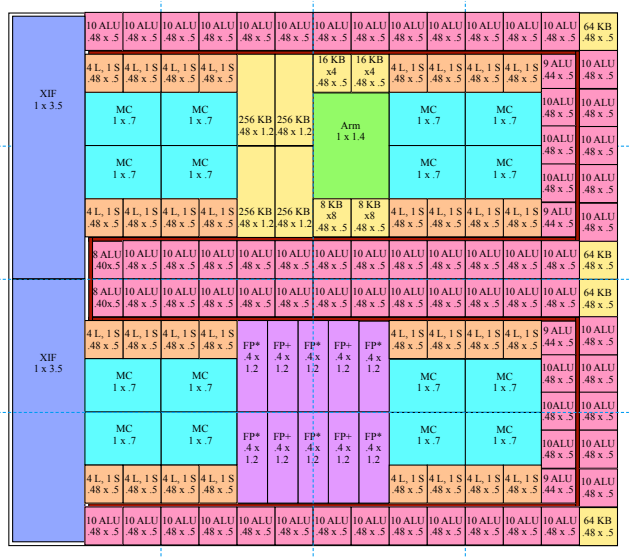

Figure 2: DFPIM layout Example.

has shown that the most significant performance benefit of stacked DRAM is increased bandwidth. The high bandwidth is required for high performance DFPIM operation as PIM applications have very high memory demands. The stacked DRAM configuration selected for DFPIM is illustrated in Figure 1. This is the same configuration selected by Zhang [19] and Scrbak [14] in giving the highest bandwidth between PIM and DRAM without thermal issues from the host processor.

The DFPIM instances are included in the base logic layer with the DRAM memory controllers. DFPIM instances are limited to 50% of an 80 mm^2 die with a total dissipated power (TDP) of 10 Watts consistent with Zhang [19]. The TDP for the DFPIM logic is therefore limited to 5 Watts.

2.4 DFPIM Layout

Figure 2 illustrates a possible floor plan for a DFPIM implementation. This illustration is drawn to scale using best available estimates for each of the block types for a 28 nm process technology. The Arm core, memory controller and SerDes block sizes were obtained from public sources. There are large blocks for the external SerDes interfaces and the memory controllers for the 16 vertical vaults of the stacked memory system.

The collection of DFPIM functional blocks form a serpentine path wrapping around the 16 memory controllers accessing the 16 vertical memory vaults in the stacked DRAM. The bold, red line represents the DFPIM connection bus channel. The DFPIM logic blocks are placed on both sides of the connection bus channel, which allows them to access the connection buses. The larger DFPIM blocks (floating point blocks, Arm processor, and 256 KB memories) utilize the horizontal space between the memory controller groups. There are 712 ALU equivalents, 32 load-store units, 10 floating point units, and 1.5 MB of memory in this configuration. Alternate configurations could delete floating point and increase memory, or increase floating point and decrease memory, or reduce ALUs to increase either floating point or memory. Smaller silicon geometries such as 22 nm or 14 nm

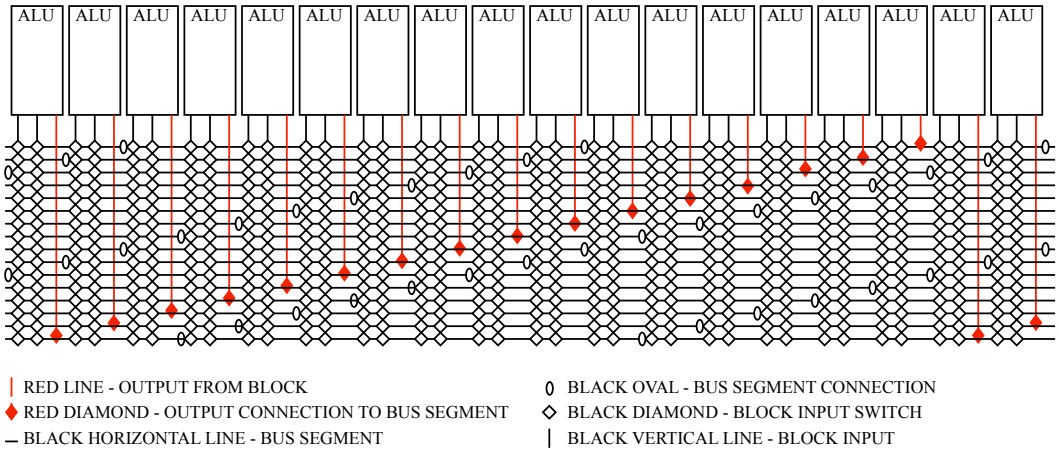

Figure 3: DFPIM Connection Bus.

would significantly increase the amount of DFPIM blocks implemented. A different layout of external interface and memory controllers would result in a different bus channel arrangement.

Figure 3 depicts the DFPIM connection bus. This depiction shows the top half of a connection bus channel. The lower half of the channel is a reflection of the top half. Each horizontal line represents a 32-bit data bus. Alternately, each bus can be two 16-bit buses or four 8-bit buses. There are a total of 32 of the buses. The rows of black, open diamonds represent a distributed AND-OR multiplexor structure that allows each input of the functional block to connect to any of the 32 horizontal bus segments. The red vertical lines with a closed diamond are the outputs of a DFPIM functional block that drives the horizontal segment where the diamond connects. Any of the ALUs connected to that segment can use that bus as an input. This forms a local 16-in by 32-out cross-point switch that includes the ability for multiple units to be driven by one output unit. The ovals on each bus represent bus switches that can extend the bus to the left or to the right. The bus segment switches are staggered to allow an overlap in horizontal bus segment coverage. The overlapping bus segments allows implementation of large dataflow graphs with a linear growth in switching area and power, rather than a quadratic growth for a full cross-point implementation. It also provides the flexibility to configure 32 pipelines of 20 units or 8 pipelines of 75 units.

3. RESULTS AND ANALYSIS

3.1 Methodology

This paper focused on comparing the performance and energy between a standalone server processor and a DFPIM accelerated implementation. Future work will compare DFPIM to other NDC solutions such as multiple Arm cores and energy efficient GPGPUs. An example DFPIM implementation is illustrated in figure 4. The multicore host processor connects to the Accelerated Memory Modules (AMM) through a High Speed Link (HSL) such as that used in the Hybrid Memory Cube [12]. Each AMM contains a 3D stack of DRAM and a logic layer containing the DRAM memory controllers (MC), the HSL interface, the DFPIM controller (such as a small, in-order Arm processor), multiple CGRL segments, and scratch pad memories for the DFPIM dataflow graphs. A vertical 'vault' consists of a memory controller, access to the HSL interface, and some CGRL blocks and SPM.

Each benchmark was executed on a compute server system with two 14-core Intel Xeon E5-2683 processors running at 2.0 GHz. Runs were made for 1 to 32 instances of the benchmark. The useful capacity of the Xeon system was the number of benchmark instances that had incremental performance gain. Loss of cache effectiveness or saturation of the memory interface result in performance ceilings as more instances are executed simultaneously. The Intel Performance Counter Monitor tools package, version 2.11, was used to monitor the power consumed by the CPUs for each configuration. The time to complete the benchmark multiplied by the power consumed during that time yielded the energy to execute the benchmark. The memory bandwidth was computed by dividing the amount of data required by the benchmark by the time to execute the benchmark. The energy of the memory accesses was computed using the equations detailed in Pugsley [13]. These benchmarks read input data from a file or generate random input data. Only the portion of the benchmark that implemented the algorithm was timed for these tests which emulates using memory resident data.

Each benchmark was then simulated with the DFPIM simulator. This simulator is configured by reading an XML representation of the dataflow graph of the benchmark. The simulator then reads the input data file and provides that data to the dataflow graph in response to memory accesses. The interconnection, parallelism, and pipelining are modeled by the DFPIM simulator. At the end of the simulation run, the DFPIM generated results are compared to the host generated results to verify the accuracy of the dataflow graph implementation. The DFPIM simulator also indicates the number and types of CGRL elements used, the amount of simulated time required, the memory bandwidth used, and the estimated power used by the benchmark configuration. The useful capacity of the DFPIM system is computed based on the amount of CGRL resources used in the simulation with respect to the total amount of resources available.

The energy needed for each of the DFPIM dataflow graph component types used in the simulator was estimated by writing VHDL models of each component type and synthe-

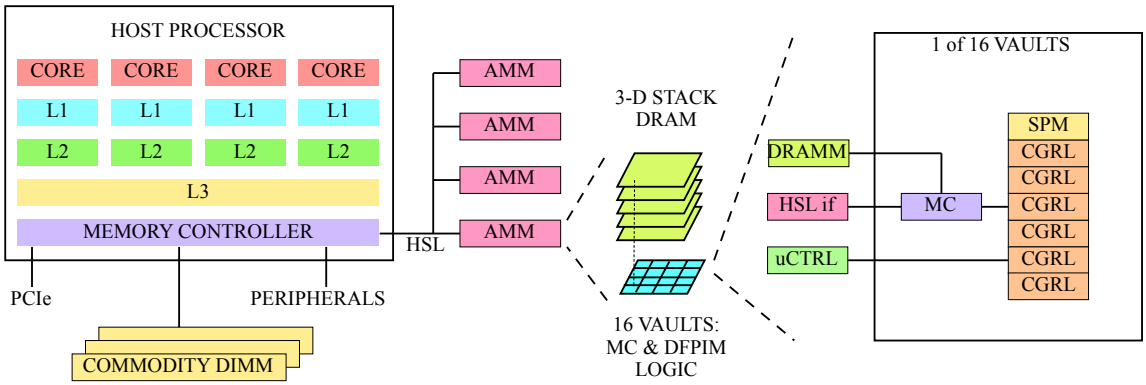

Figure 4: Example DFPIM System Implementation.

sizing them with a 28nm planar library and a 16nm FINFET library using Synopsys Design Compiler. The static and dynamic power values for each component were integrated into the DFPIM simulator to produce the final estimates for each benchmark based on number of components used and frequency of operation. The 16nm results were 25% faster, 1.6 times more energy efficient, and required 53% of the area than the 28 nm results.

3.2 Benchmarks

There are a wide variety of benchmarks that could be used in an evaluation such as this. The purpose of DFPIM is to offload memory intensive kernels from the host processor to utilize the high bandwidth of 3D stacked DRAM and the parallelism of dataflow. We reviewed the map-reduce benchmarks from HiBench [10], the map-reduce benchmarks from PUMA [1], the Rodinia benchmarks [4], SPEC benchmarks [15], and MiBench benchmarks [9]. We picked benchmarks that had significant differences in the dataflow configuration of the kernels to evaluate DFPIM in a variety of applications. The six benchmarks used in this paper are histogram, word occurrence count, fast fourier transform, string match, linear regression, and breadth first search.

The benchmark kernels were extracted from the benchmarks by analyzing the source code and making dataflow graph representations. The output of the DFPIM simulator was verified against the output of the host processor execution to ensure the dataflow graphs were accurate representations of the benchmark functionality. An automatic extraction based on emitting dataflow graphs from the LLVM clang compiler is in process.

3.3 Results

Table 1 provides the results of our analysis. The six benchmarks are listed across the top of the table with the last column containing the average of the values across the benchmarks. The first row, Capacity, lists the number of DFPIM instances and Xeon instances of each benchmark that can run simultaneously. The second row, Speedup, is computed by the host processor execution time divided by the DFPIM execution time reported by the DFPIM simulator. The third row, Energy, is an energy efficiency ratio computed by the host processor energy divided by the DFPIM energy estimate from the simulation. All energy values used in this paper include both static and dynamic power components of the processor and DFPIM elements and the energy of the memory accesses and data transfers. The fourth row, Host BW, is the average memory bandwidth for the host execution in megabytes per second. The last row, PIM BW, is the average memory bandwidth for the DFPIM execution in megabytes per second.

The only benchmark that showed a significant degradation in Xeon performance due to memory access limitations was BFS. Performance stopped improving with 17 threads which set the useful capacity to 16 for BFS. The other five benchmarks scaled to 32 threads, although all of them had noticeable reductions in incremental improvement with more than 20 instances. The number of instances of DFPIM benchmarks can be constrained by the availability of CGRL element resources or by PIM power limitations. These values varied from 4 for the FFT benchmark which was limited by the number of floating point units that were available in the configuration being analyzed to 32 instances for the simple logic of the linear regression benchmark.

The DFPIM showed an average speedup of 5.5 for the six benchmarks. This is impressive since the Xeon host processor was being clocked 2.5 times faster than the DFPIM logic. The increased performance of the DFPIM resulted from the parallelism and pipelining achieved from the dataflow driven computations. This structure provided the ability to initiate and complete an iteration of the benchmark kernel on every clock cycle once the pipeline filled.

The average energy efficiency ratio of 460 is a combination of completing the computations in 18% of the time; the use of silicon technology optimized for low power rather than high speed; and the elimination of instruction fetch, instruction scheduling, instruction decode, reorder buffers, and high speed cache memory needed for high performance server processors.

The ability of DFPIM to better utilize the higher memory bandwidth provided by stacked DRAM is evident in the differences in memory bandwidth shown in the table. The ratio of memory bandwidth correlates to the speedup.

4. FUTURE WORK

The preliminary results shown in section 3 have demonstrated DFPIM is a viable approach to both performance improvements and energy efficiency for PIM applications. Continued development of DFPIM will include completely defining the set of functional blocks and the quantity of each type of functional block to include in a DFPIM cluster. The

	Hist	Word	FFT	BFS	Str-m	Lin-r	Ave
Capacity	16 | 32	8 | 32	4 | 32	16 | 16	20 | 32	32 | 32	16 | 30
Speedup	4.09	2.56	3.79	12.56	3.20	6.79	5.50
Energy	253	104	621	919	357	505	460
Host BW	9386	2471	1655	592	4004	7542	4275
PIM BW	34800	6336	25098	7430	12800	25600	19278

Table 1: Xeon Host Processor and DFPIM Comparison. These values are based on a 2.0 GHz host and a 0.8 GHz DFPIM.

definition will serve as a reference for generating dataflow graphs and development of the functional blocks. Each of the functional blocks will be modeled in VHDL. A synthesis tool will be used to characterize size, timing, and energy for each of the DFPIM blocks in 28nm planar and 16nm FINFET silicon technologies. The DFPIM simulator will be updated to include the area, timing and energy estimates from the logic synthesis results.

The dataflow graph conversion process will be improved to provide a more automated flow to reduce or eliminate manual efforts in the benchmarking process. This will allow a larger number of benchmarks to be analyzed. The benchmark analysis will be expanded to include comparisons of other PIM implementations in addition to comparing to the host processor.

5. RELATED WORK

TOP-PIM [18] is a very similar approach to DFPIM. The principle difference being the use of GPGPU devices as the processor component in the PIM. This study showed a mean decrease in performance of 25% for a 22 nm technology and a mean increase in performance of 8% for a 16 nm technology. The energy savings was shown to be 76% and 86% respectively when including the memory power. The parallelism of dataflow and flexibility of CGRL work to provide better performance at comparable energy savings. There were no benchmarks in common between the two studies so a direct comparison cannot be stated until DFPIM expands its benchmark coverage to include those used by Zhang.

Single Graph Multiple Flows (SGMF) [16] uses a dynamic dataflow paradigm and CGRL to compare to an Nvidia Fermi streaming multiprocessor. The application arena for SGMF is compute intensive applications so it is not suitable as a PIM. However, the advantages of using dataflow with CGRL is shown in this paper with an average speedup of 2.2 and energy efficiency of 2.6 for the 64 token case.

The use of a low power embedded processor as a PIM is addressed in Scrbak [14]. The embedded processor is limited to memory accesses at a cache line resolution. It also requires energy for instruction fetch and decode and a cache subsystem that is not needed by DFPIM. The parallelism of dataflow CGRL provides higher performance than a single instruction stream processor running at the same clock rate.

The Tesseract PIM in [2] uses multiple in-order processors in a Hybrid Memory Cube [12]. The memory bandwidth restrictions of the in-order cores are mitigated by prefetching mechanisms. The internal crossbar network allows the Tesseract processors to communicate without host processor intervention. This allows them to be used for the reduce task workload as well as the map task workload. The Tesseract PIM performance is significant for multi-threaded message passing applications. DFPIM has not been evaluated in these types of applications.

The Near DRAM Accelerator (NDA) [5] utilizes a dataflow network of functional devices to reduce energy by 46% and increase performance by 1.67 speedup. The NDA does not include sequencing functional units nor scratch pad memories which we have shown to be necessary for best performance in some benchmarks. The NDA connects each accelerator to a single DRAM die rather than a 3D-DRAM stack used by DFPIM. This results in a higher accelerator-to-memory cost ratio as a single DFPIM can support 4 or 8 DRAM dies.

The Heterogenous Reconfigurable Logic (HRL) near data processing [7] uses CGRL functional units and bus based routing as well as dedicated memory load and store units. This paper illustrates the area, performance, and energy advantages of mixed granularity systems such as HRL and DFPIM. The HRL system requires 8 memory stacks to achieve an average 2.5 speedup, while DFPIM gets a 3.6 speedup with a single memory stack. Part of this is attributable to the difference between the 45 nm process of HRL and the 28 nm process of DFPIM. DFPIM uses a flexible, partitioned bus rather than the mesh network of the HRL which may allow more efficient implementation of some dataflow graphs.

6. CONCLUSION

In this paper we have proposed and evaluated a hybrid dataflow technology using coarse grain reconfigurable logic as a highly energy efficient solution for near data computing within a stacked DRAM module. The parallelism of dataflow implemented in a low power semiconductor process on the logic layer of the module provides both a significant speedup of 5.5 and a very high energy efficiency of 460 for the presented benchmarks. Our work shows that the hybrid dataflow approach with sequencers, scratch-pad memories, and FIFOs implement multilevel looping and asynchronous interactions within the application kernels without the host intervention that is needed if a pure dataflow approach is used.

Acknowledgment

This work is conducted in part with support from the NSF Net-centric IUCRC and its members. Many friends and reviewers offered suggestions that improved the research and this paper.

7. REFERENCES

[1] Ahmad, F., Lee, S., Thottethodi, M., Vijaykumar, T.N.: Puma: Purdue mapreduce benchmarks suite. Tech. rep., Purdue University (2012), http://docs.lib.purdue.edu/ecetr/437

[2] Ahn, J., Hong, S., Yoo, S., Mutlu, O., Choi, K.: A scalable processing-in-memory accelerator for parallel

graph processing. In: Proceedings of the 42nd Annual International Symposium on Computer Architecture. pp. 105–117. ISCA '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2749469.2750386

[3] Chang, D., Byun, G., Kim, H., Ahn, M., Ryu, S., Kim, N., Schulte, M.: Reevaluating the latency claims of 3d stacked memories. In: Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific. pp. 657–662 (Jan 2013)

[4] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. pp. 44–54 (Oct 2009)

[5] Farmahini-Farahani, A., Ahn, J.H., Morrow, K., Kim, N.S.: Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In: 2015 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 283–295. IEEE Conference papers (March 2015), http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7056040

[6] Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A., Ailamaki, A., Falsafi, B.: A case for specialized processors for scale-out workloads. Micro, IEEE 34(3), 31–42 (May 2014)

[7] Gao, M., Kozyrakis, C.: Hrl: Efficient and flexible reconfigurable logic for near-data processing. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). pp. 126–137 (March 2016)

[8] Govindaraju, V., Ho, C.H., Nowatzki, T., Chhugani, J., Satish, N., Sankaralingam, K., Kim, C.: Dyser: Unifying functionality and parallelism specialization for energy-efficient computing. Micro, IEEE 32(5), 38–51 (Sept 2012)

[9] Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R.: Mibench: A free, commercially representative embedded benchmark suite. In: Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on. pp. 3–14 (Dec 2001)

[10] Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on. pp. 41–51 (March 2010)

[11] Loh, G.: 3d-stacked memory architectures for multi-core processors. In: Computer Architecture, 2008. 35th International Symposium on. pp. 453–464 (June 2008)

[12] Micron Technology: Hmc high-performance memory brochure (nov 2014), http://www.micron.com/~/media/documents/products/product-flyer/brochure_hmc.pdf

[13] Pugsley, S.H., Jestes, J., Zhang, H., Balasubramonian, R., Srinivasan, V., Buyuktosunoglu, A., Davis, A., Li, F.: Ndc: Analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads. In: Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. pp. 190–200. IEEE (March 2014)

[14] Scrbak, M., Islam, M., Kavi, K., Ignatowski, M., Jayasena, N.: Processing-in-memory: Exploring the design space. In: Pinho, L.M.P., Karl, W., Cohen, A., Brinkschulte, U. (eds.) Architecture of Computing Systems Đ ARCS 2015, Lecture Notes in Computer Science, vol. 9017, pp. 43–54. Springer International Publishing (2015), http://dx.doi.org/10.1007/978-3-319-16086-3_4

[15] Spec benchmarks, https://www.spec.org/benchmarks.html

[16] Voitsechov, D., Etsion, Y.: Single-graph multiple flows: Energy efficient design alternative for gpgpus. In: Proceeding of the 41st Annual International Symposium on Computer Architecuture. pp. 205–216. ISCA '14, IEEE Press, Piscataway, NJ, USA (2014), http://dl.acm.org/citation.cfm?id=2665671.2665703

[17] Wulf, W.A., McKee, S.A.: Hitting the memory wall: Implications of the obvious. SIGARCH Comput. Archit. News 23(1), 20–24 (Mar 1995), http://doi.acm.org/10.1145/216585.216588

[18] Zhang, D.P., Jayasena, N., Lyashevsky, A., Greathouse, J., Meswani, M., Nutter, M., Ignatowski, M.: A new perspective on processing-in-memory architecture design. In: Proceedings of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness. pp. 7:1–7:3. MSPC '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2492408.2492418

[19] Zhang, D., Jayasena, N., Lyashevsky, A., Greathouse, J.L., Xu, L., Ignatowski, M.: Top-pim: Throughput-oriented programmable processing in memory. In: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing. pp. 85–98. HPDC '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2600212.2600213