Reconfigurable Dataflow Graphs For Processing-In-Memory

Charles F. Shelor
Computer Science and Engineering
University of North Texas
Denton, Texas, USA
Charles.shelor@gmail.com

Krishna M. Kavi
Computer Science and Engineering
University of North Texas
Denton, Texas, USA
Krishna.kavi@unt.edu

ABSTRACT

In order to meet the ever-increasing speed differences between processor clocks and memory access times, there has been an interest in moving computation closer to memory. The near data processing or processing-in-memory is particularly suited for very high bandwidth memories such as the 3D-DRAMs. There are different ideas proposed for PIMs, including simple in-order processors, GPUs, specialized ASICs and reconfigurable designs. In our case, we use Coarse-Grained Reconfigurable Logic to build dataflow graphs for computational kernels as the PIM. We show that our approach can achieve significant speedups and save energy consumed by computations. We evaluated our designs using several processing technologies for building the coarse-gained logic units. The DFPIM concept showed good performance improvement and excellent energy efficiency for the streaming benchmarks that were analyzed. The DFPIM in a 28 nm process with an implementation in each of 16 vaults of a 3D-DRAM logic layer showed an average speed-up of 7.2 over that using 32 cores of an Intel Xeon server system. The server processor required 368 times more energy to execute the benchmarks than the DFPIM implementation.

CCS CONCEPTS

• Computer systems organization ~ Data flow architectures

KEYWORDS

Dataflow Architectures, Coarse Grained Reconfigurable Logic, Processing in Memory, 3D-Stacked Memories

ICDCN '19, January 4–7, 2019, Navi mumbai, India

ACM ISBN 978-1-4503-6094-4/19/01...$15.00
https://doi.org/10.1145/3288599.3288605

1 Introduction

One of the major problems facing today's computer systems is the disparate speeds between processor instruction cycles and memory access times. This is not a new issue as this timing mismatch was recognized and known as the *memory wall* [42]. Advances in processor clock rates and architectures have outpaced improvements in memory bandwidth and access delay. Processors are running at 2 to 4 GHz range, while DRAM latency is 41.25 ns for the fastest DDR4 device from Micron [31]. The cache hierarchy in computer systems has been used to reduce the effects of the memory wall by providing fast access to data items on subsequent accesses. The use of multilevel memory caches, prefetching of data, multiple memory channels, and wide data paths mitigate the memory access delay, but there are still times when the processor must wait the 83 to 166 clocks for the requested data to arrive. Increasing computer system performance through multicore processors increases the pressure on the memory system as more memory requests are being generated. Conditions will occur where one or more cores must wait until an existing memory access completes before beginning its own memory access. With every fifth instruction [17] being a data request, the memory access delay and imperfect caching leads to high end servers being idle three out of four clocks [21]. Energy consumption of computer systems has been an increasing issue in recent years.

Advances in silicon technology have dramatically decreased the energy per computation for the processor core. However, the energy for memory accesses is increasing to achieve improved bandwidth and latency to attempt to match processor performance [34, 35]. The memory system is an increasingly significant fraction of the computing system energy use [26]. A 64-bit external memory access requires approximately 100 times the energy of a double precision computation [20, 9, 25].

Energy is particularly important to both high-performance applications and emerging Big Data and Deep Learning applications. For Exascale systems, the goals include a memory bandwidth of 4 TB/s at each node for 100,000 nodes with a maximum power budget of 20 MW [41]. Aggressive assumptions about memory technology improvements show that 70% of the power budget will be needed for memory accesses [43].

Demand for higher performance computer systems has pushed processor architectures to longer pipelines with multiple issue, out-of-order capabilities and larger memory caches to supply data.

These high-performance microarchitectural features require an energy overhead that reduces the energy efficiency of the processor. A 4-issue core has six integer ALU, two load-store, two floating point, and one integer multiply-divide. Only 26% of the energy is used by functional units that generate algorithmic results. The remaining energy is consumed by cache hierarchy, network on the chip, instruction scheduling, renaming registers and other logic needed for out of order execution.

Our architecture addresses both the execution and energy consumption. Execution performance is improved by moving computations closer to memory (that is, Processing in Memory) and eliminating traditional instruction pipelines with a reconfigurable graph describing a computation. Energy savings result from the elimination of instruction fetch/decode/issue cycles, cache memories and using lower clock frequencies.

The rest of the paper is organized as follows. Section 2 describes the technologies that enabled our work. Section 3 provides an overview of our dataflow processing-in-memory (DFPIM) architecture. Section 4 provides details of our experimental setup. Section 5 contains results of our evaluation and discussions. Section 6 includes research that is closely related to ours and Section 7 contains conclusions of our study.

2. Enabling Technologies

Our architecture is enabled by (hybrid) dataflow model of computation, coarse-grained reconfigurable logic and 3D-stacked memories with room for processing-in-memory logic.

2.1. Dataflow model represents a computation as a graph where the nodes represent operations on inputs received via the incoming edges and results are sent to other nodes via outgoing edges [3, 4, 5, 23, 24]. In our system, we deviate from the pure dataflow model. We use load units to bring input data from DRAM memory into local buffers. There are delay operations in the dataflow graphs to balance and synchronize all paths in the graph, eliminating the need for additional inputs to trigger when data is consumed. The dataflow graph is 'executed' only when all graph inputs for the next computation are available (not just inputs to nodes in the input layer). This pipelined execution also handles loop carried dependencies and simplifies memory ordering issues. Programmable state machines are used to implement looping structures within the dataflow graphs to increase graph execution independence from a host processor or controller. Figure 1 shows a dataflow graph representation of FFT. Detailed description of the operations is omitted due to space limitations.

2.2. Coarse Grained Reconfigurable Logic (CGRL) is similar to FPGA, but the reconfigurability is at a functional block level and not at gate level. The CGRL fabric consists of functional units such as Integer ALUs, Floating Point Adders, Floating point multipliers, or other specialized functional units. The inputs of functional units can be connected to the outputs of other functional units, thus creating a dataflow graph representing a computation. Reconfiguring the input to output connections results in a new computational graph. We assume a partitionable crossbar interconnection network to communicate inputs and outputs. An example of a CGRL that is configured is shown in Figure 2.

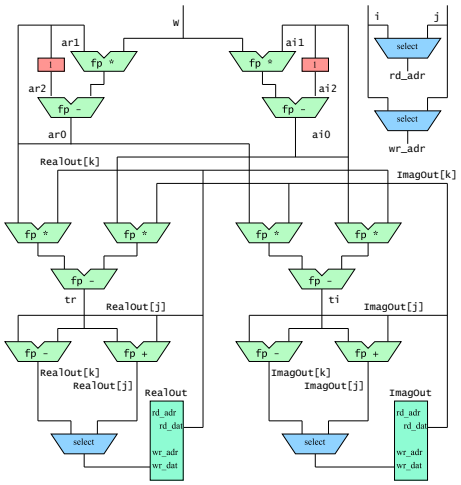

Figure 1: An Example Dataflow Graph for FFT

2.3. Processing in Memory (PIM) using 3D DRAM. One approach to mitigating the memory wall for applications that do not work well with caches is moving the processing of the data closer to the data itself [36, 43]. The advent of 3D-stacked DRAMS, which include a logic layer makes this Near Data Computing (NDC) or Processing-in-Memory (PIM).

The close, physical proximity of the stacked layers combined with the low capacitance of the TSV interconnect [30] provides a faster and lower power communication path than the standard memory controller to DRAM DIMM path through sockets and PCB traces. The multiple independent channels and high-speed serial links provides 256 GB/s for HBM [22, 31] and 160 GB/s for HMC [19, 32, 35].

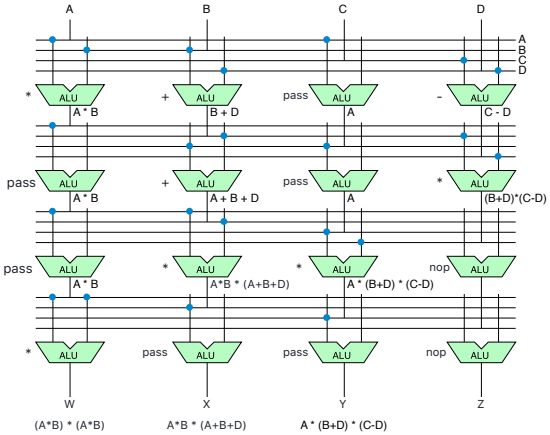

Figure 2. An Example of CGRL Configuration

3. Dataflow Processing In Memory (DFPIM)

DFPIM uses a hybrid dataflow technology to extract parallelism and pipelining for high performance computation in streaming data applications. The dataflow logic is configured into the application solution graph by using CGRL comprised of functional blocks and connectivity elements. The CGRL is implemented as PIM on the logic layer within a 3D stacked DRAM. Figure 3 shows a high-level architecture of the proposed dataflow PIM.

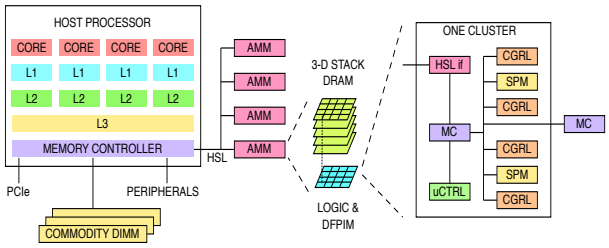

Figure 3: DFPIM Architecture

The left section represents the host computer for the DFPIM elements. This is a standard server or workstation computer system. The only feature that is not standard on current systems is a high-speed serial interface for connecting the accelerated memory modules. Processor manufacturers are incorporating these links in new products to take advantage of the higher bandwidth and lower energy of 3D stacked DRAM devices [7, 8].

There can be multiple accelerated memory modules (AMM) as shown in the figure. The center section shows an accelerated memory module expanded into the logic layer base and a stack of DRAMs, including representation of the sixteen independent vertical vaults. The logic layer base contains one memory controller and one DFPIM instance for each vault. A microcontroller is included on the logic layer to assist DFPIM configuration and minimize the communication between the host and DFPIM for optimum performance.

The right section shows an expanded view of the logic layer for one vault. The memory controller accesses the memory stack vault directly above the vault controller [2, 36, 43]. The memory controller communicates with the high-speed link for data transfers with the host. The DFPIM instance has load and store units within the CGRL that access the DRAM stack through the memory controller and buffer input data for the dataflow graphs. The DFPIM instance consists of the CGRL logic and the scratch pad memories that are local to the CGRL functional units. for implementing the dataflow graphs of the applications. There is also a link to the DFPIM microcontroller that is used to configure the CGRL and to initialize and store data from the scratch pad memories as needed.

3.1. DFPIM Operation

DFPIM operation can be divided into four phases. Initiation is performed by the host processor. Configuration is executed by the DFPIM micro-controller. Computation is executed by the DFPIM logic until the input data is exhausted. An update phase is conducted by the micro-controller for storing results.

The host computer initiates a DFPIM operation when a command that uses the DFPIM is executed. As an example, a user could enter a command via the keyboard. The operating system reads an executable file that contains the machine instructions that implement the given command. When the DFPIM is to be involved, there is a data segment within the file that must be transferred to the DFPIM. This is very similar to executing a command implemented in OpenCL or CUDA that involves a graphics processor. The code to be executed by the graphics processor is copied from a data segment of the host executable to the graphics processor to be used as the instructions to execute. The data segment directed to the DFPIM is copied through the high-speed link to an address dedicated to this.

The DFPIM logic accepts input data and generates results until it runs out of input data. If there is no input data ready for a particular clock cycle, the entire logic network waits for the data to become available. This is needed to ensure data stays synchronized through the computational sequence. If an exception condition is encountered it can be posted for detection after the computation has completed or it can be passed to the micro-controller which will terminate processing and notify the host processor that the operation has failed.

The update phase uses the micro-controller to download any results that are contained in scratchpad memories. The host processor is then notified that the requested operation has completed. In some cases, the results might be transferred to the host processor, in other cases it might just be an acknowledgement that the operation completed and the results are available at the requested location.

All DFPIM operations are based on physical address offsets since the DFPIM resides inside a physical memory. Any indirect or pointer accesses within the application must either be based on physical addresses or the application data must have been allocated as a large, continuous segment and all pointers are simply offsets within the segment. DFPIM applications are limited to the memory within the 3D-stacked component that contains the DFPIM logic. A communication network on the logic layer allows DFPIM elements to access data in other vaults, but this is likely to introduce delays.

3.2. DFPIM Layout

Figure 4 shows a possible floor plan for a DFPIM implementation. The DFPIM components use 50% of a 68 mm^2 stacked DRAM logic layer (the other 50% is set aside for memory controllers and TSVs). The illustration is drawn to scale using logic synthesis estimates for each block for a 28nm process technology. ARM core is used as the microcontroller for DFPIM.

Figure 4: An Example PIM Layout

In this layout the memory controllers are located in the lower left corner of each memory vault of an HMC like 3D stacked memory. The DFPIM logic is above and to the right of each memory controller. The logic units include integer units *I* (2 load, 1 store, 20 ALUs, four multiply units, some specialized units, two FIFOs), floating point units *F* (32 single precision adders and multipliers, ten double precision adders and multipliers) and a small local memory *M*. The interconnection bus is a 16 x 32 crossbar which can be segmented into smaller buses. This layout is only one example configuration.

4. Experimental Setup

In this paper we compare the execution times and energy consumed by DFPIM with a host system with two 14-core Intel Xeon E5-2683v3 processor running at 2GHz. Intel Performance Counter Monitor tools package [39] was used to monitor the power consumed by the CPUs during benchmark execution. We carefully isolated the execution time and energy consumed only for the benchmark kernels for a fair comparison with DFPIM. The execution and energy values for our DFPIM components (functional units and ARM core) are estimated using very detailed logic synthesis value using TSMC libraries for 7nm and 16nm FinFET and 28nm planar CMOS technologies and obtained through ARM Limited Artisan physical IP [27]. We evaluated eleven clock rates in six variants of 7nm libraries, eight clock rates in twelve variants of 16nm libraries and seven clock rates in four variants of 28nm libraries. From these 190 different synthesis runs for each DFPIM component, we selected the best configuration (clock rate and library) that results in optimal energy-delay values. Table 1 shows the selected libraries and clock frequencies for the three criteria of minimum energy, minimum energy-delay product, and the average of those two. We use the libraries and clock rates from the average column as a balance of energy and performance is desired.

Table 1: Optimal Silicon Libraries and Clock Rates

	Energy	Energy * Delay	E, E*D Ave
07nm	svt-c8, 1.0 GHz	ulvt-c8, 2.0 GHz	**lvt-c8, 1.8 GHz**
16nm	lvt-c16, 0.8 GHz	ilvt-c16, 1.5 GHz	**ilvt-c16, 1.0 GHz**
28nm	svt-c35, 0.6 GHz	svt-c30, 1.1 GHz	**svt-c30, 1.1 GHz**

4.1. Dataflow Graph Generation

We developed a backend to LLVM compiler [28, 29] to generate dataflow graphs. The portion of the C program that is targeted for execution on DFPIM is first identified. Using LLVM intermediate representation for the identified kernel code, a dataflow graph is generated. For our purpose the output is represented in XML representing the various functional units used by the graph and the connections between these units to form the graph. This XML code is used by our DFPIM simulator for producing execution results presented in Section 5.

4.2. DFPIM Simulator

Our simulator takes the input (in XML) generated by LLVM for each benchmark kernel, configures the functional units to represent the dataflow graph represented by the LLVM output, and executes the graph with inputs transmitted by the host processor. For our purpose we assume that both the host and DFPIM use the same address space and thus DFPIM will access the data from the shared 3D DRAM memory. Both host and DFPIM rely on physical addresses. The load units contained within DFPIM will buffer inputs for use by the computational functional units, and the store units copy results back to memory.

The execution delays and energy consumed by the various DFPIM logic components are based on the values obtained by our logic synthesis as described above in Section 4.

5. Results

In this section we describe the results of our experiments comparing the execution times and energy consumed by DFPIM with a host system as described in the previous section.

5.1. Benchmarks

We selected representative benchmarks from a wide-variety of application domains. The map-reduce benchmarks from HiBench [18], the map-reduce benchmarks from PUMA [1], the Rodinia benchmarks [6], SPEC benchmarks [38], and MiBench benchmarks [16] were reviewed. We selected benchmarks that had significant differences in their suitability for dataflow implementation. The seven benchmarks used in this paper are histogram, word occurrence count, fast Fourier transform, breadth first search, string match, linear regression, and SHA256. The SHA256 benchmark had three versions implemented in DFPIM for a total of nine analyses. We now describe these benchmarks.

Histogram. The histogram benchmark inputs an RGB image and generates a histogram of values for the red (R), green (G), and blue (B) components of the pixels. The benchmark code isolates the 8-bit color values from a 24-bit input with shift and mask operations. The pixel component values are used as addresses to three arrays (scratch pad memory in DFPIM) that returns the current count, increments it, and stores the new count. The input file contained 468,750,000 RGB pixels.

Word Count. The word occurrence count benchmark is based on tasks used in web indexing and searches. The first part of the benchmark inputs a character stream and isolates it into words. The second part of the benchmark creates a hash for the word and looks for the word in a hash table. If the word is found in the table its occurrence count is incremented, otherwise it is added to the table with a count of 1. The server implementation of the benchmark serially finds a word then processes the word, then looks for the next word. The DFPIM implementation has two sections. The first section processes the input looking for words. When a word is found it is put into a FIFO. The second section pulls a word from the FIFO, processes the word and then pulls the next word. The DFPIM uses word-wide comparison for word matching. The two sections work independently and are synchronized through the FIFO. The benchmark input was 94,858,002 bytes in length.

Fast Fourier Transform. The FFT benchmark processes a frame of time sampled data into a frame of frequency bins. The number of samples in the input frame is a power of 2, designated as N. The butterfly implementation of the algorithm is a triple nested

loop where the outer loop is repeated $\log_2(N)$ times. The middle loop iterates based on powers of 2 from $\log_2(N)$-1 to 1 while the inner loop iterates based on powers of 2 from 1 to $\log_2(N)$-1. The code within the inner loop is executed $\log_2(N) * N/2$. The FFT algorithm does benefit from caching as each data sample is accessed $\log_2(N)$ times during a frame processing. However, a program using an FFT is likely to process many frames in sequentially ordered streaming. The outer loop includes two *sine* and two *cosine* operations. As the DFPIM does not have sine and cosine blocks defined, the micro-controller must intervene and perform these operations. Alternately, they could be precomputed and stored in a scratch pad, eliminating the micro-controller involvement. This analysis was based on a frame size of 4096 samples and processing 500 frames of data. FFT is an example of a benchmark that is not very well suited for pure dataflow implementation.

Breadth First Search. The breadth first search benchmark is neither streaming nor cache friendly. It searches through a tree resulting in a random memory access pattern. The only advantage of the DFPIM is its faster access time to memory. The input file contains a tree with one million nodes.

String Match. The string match benchmark searches a text file for a list of keys. Whenever a match is found, it's location in the text file is saved. The algorithm first locates the end of the current word and then compares the word to each of the keys. The pointer to the word is stored in the results block when a match occurs. This analysis searched a 502 MiB file while searching for four keys.

Linear Regression. The linear regression benchmark takes a file of points and accumulates 5 information components: x-coordinate value, x-coordinate squared, y-coordinate value, y-coordinate squared, and x-coordinate times y-coordinate. The five accumulated values are returned when the end of the input data is reached. This benchmark was evaluated with a 670 MiB file.

SHA256. The SHA256 benchmark is a cryptographic application that creates a digest of a message that can later be used to guarantee the message has not been modified. A large sequence of rotation, logical and arithmetic operations are performed on the input data to generate the 32-byte message digest. Each round of the algorithm requires 6 rotate, 3 logical AND, 6 logical XOR, 2 logical OR, and 7 addition operations for a total of 24 operations per round. Sixty-four rounds are performed on each 64-byte input block for a total of 1536 operations per block Three DFPIM implementations of the SHA256 benchmark are used in this evaluation. The first is a straightforward implementation where an integer ALU is used for each of the operations. One input stream is processed at a time. The second implementation creates three new DFPIM components implementing macros of the processing round. This implementation is designated as *SHAmac*. The 24 integer ALUs per round are reduced to three special components and two integer ALUs. The reduced component count decreases power and energy while maintaining the same performance. The algorithm loops the result of each round to the start of the next round. Since the round takes three clocks to pass through the pipeline, each component is idle two-thirds of the time. The third implementation interleaves three different input streams to achieve three times the throughput of the first two implementations. This version is designated *SHAmac3*. The pipeline is fully utilized resulting in higher energy to obtain the better performance. A 50 MiB file is processed by the SHA256 benchmark in this evaluation. It should be noted that the two alternatives described here are not available with server implementation of SHA (since we cannot modify the functional units of the server).

5.2. Server Benchmark Results

The results of running the benchmarks on the server processor are shown in Table 2. The first eight rows of the table provide the measured data from the benchmark execution. The last five rows of the table are calculated values derived from the measured data. All three variants of the SHA256 benchmark are shown in this table even though the server data is the same for the three variants since there is only one implementation of SHA on the server. This keeps the four result tables consistent.

The *Base Clocks / Item* is the number of processor clocks needed to complete the benchmark while running on a single processor divided by the number of benchmark items processed. This baseline is compared to the clocks per item measured when the server is running 32 instances of the benchmark simultaneously to show how well the benchmark scales. The *Clocks (M)* row is the total number of processor clocks to complete the benchmark. The number is in millions of clock ticks. The *Freq (clk / usec)* entry is the actual operating frequency of the processor as reported by the hardware during the benchmark. This is measured to ensure the operating system did not change the frequency of the processors during operation.

The *Items / Proc (K)* measurement is the number of thousands of benchmark items processed on each of the 32 instances of the benchmark executed on the server. The *Item Size (bytes)* is the size of the benchmark item. The histogram benchmark processes pixels that are composed of a red, a green, and a blue component for a total of three bytes per pixel. The word count benchmark processes characters as input for a 1 byte size. The FFT benchmark operates on complex numbers with a floating-point real and imaginary values for a total of 8 bytes. Breadth first search operates on pointers with a size of 8 bytes. String match processes characters as input for an item size of 1 byte. The linear regression processes data points with an x component and y component that are both bytes. The SHA256 algorithm processes a 64-byte block of data as an item.

The *CPU Power (W)* row contains the measured CPU power in Watts for the benchmark. The *Mem Power (W)* row provides the measured memory power in Watts. There is very little variation in power for the different benchmarks. The power measurements showed more correlation to the number of cores that were active than the type of activity being performed.

Table 2: Server Benchmark Results

	Hist	Word	FFT	BFS	Str M	Lin R	SHA256	SHAmac	SHAmac3
Base Clocks / Item	45.08	18.57	348.85	82.08	12.28	25.08	2160.19	2160.19	2160.19
Clocks (M)	299.30	76.70	776.64	164.64	262.74	372.49	131.07	131.07	131.07
Freq (clk/usec)	1997	1997	1997	1997	1997	1997	1997	1997	1997
Items / Proc (K)	4882	2964	2048	1000	16454	10986	51	51	51
Item Size (bytes)	3	1	8	8	1	2	64	64	64
CPU Power (W)	126.17	125.86	125.62	125.35	125.73	124.81	126.34	126.34	126.34
Mem Power (W)	2.27	2.07	1.96	2.09	2.02	2.02	2.02	2.02	2.02
Kernel percent	99.99	99.99	98.88	99.25	99.99	99.99	95.81	95.81	95.81
Execution Time (S)	0.1499	0.0384	0.3889	0.0824	0.1316	0.1865	0.0656	0.0656	0.0656
Server Energy (J)	19.250	4.913	49.617	10.507	16.808	23.657	8.425	8.425	8.425
Bandwidth (MB/S)	3127.6	2469.8	1348.1	3105.1	4002.1	3769.6	1604.5	1604.5	1604.5
Clocks per item	61.30	25.87	379.22	164.64	15.97	33.91	2549.05	2549.05	2549.05
Congestion Factor	0.36	0.39	0.09	1.01	0.30	0.35	0.18	0.18	0.18

The *Kernel percent* measurement indicates the percentage of benchmark clocks that were used for execution of the benchmark kernel section.

The *Execution Time (S)* is derived from the total clocks executed divided by the clock frequency and expressed in seconds.

The *Server Energy (J)* is computed by adding the CPU power and memory power and multiplying the sum by the execution time. This expresses the energy in Joules.

The *Bandwidth (MB/S)* is the number of benchmark items times the size of each item divided by the execution time. It does not include any instruction accesses, incidental cache hits, or memory accesses from algorithmic overhead such as loop indexing or address calculations.

The *Clocks per item* metric is the number of processor clock cycles divided by the number of benchmark items of a benchmark instance running on each of 32 processors. As the actual number of clock cycles required per item is given in the base clocks per item, any additional cycles must be attributed to congestion in accessing memory or other resources. The *Congestion Factor* row expresses this congestion as a percentage of the base clocks per item. There is a noticeable trend for higher memory bandwidths to have higher congestion factors.

5.3. DFPIM Benchmark Results

Table 3 displays the measurements for the DFPIM benchmarks. We analyzed three different silicon technologies as described in Section 4. This resulted in a separate measurement for each technology. The units used in this table are consistent with the units in Table 2 allowing the numbers to be directly compared. The data for the server processor was collected with 32 active cores. The DFPIM has 16 vaults and each DFPIM executes an instance of the benchmark (server system represents twice as many "cores" that are executing twice as many copies of the benchmark kernel when compared DFPIM implementation). Therefore, the *Items / Vault (K)* will be double the number of items per process used for the server processor data.

The *DF Power.. (W)* is the power of the DFPIM components which is equivalent to the CPU power of the server data (and we show the values for 28, 16 and 7nm versions. The *Mem Power.. (W)* is the power of the memory accesses within the stacked DRAM. The *Clocks per item* measurement does not have three components as the underlying silicon technology does not impact the dataflow pipeline organization resulting in the same number of clocks for each technology. The lower clocks per item in the DFPIM results compared to the server processor results show the benefits of the dataflow parallelism and pipelining. The power values show the benefits of low power silicon libraries and not pushing the technology to its performance limits (running at lower frequency than maximum possible).

The advantages of using specialized functional units for the SHAmac version of the SHA256 benchmark can be seen in the DF Power values. There is no difference in timing as the benchmark round still requires three clocks. Interleaving three benchmark instances in the SHAmac3 version does show a factor of three timing improvement. This version shows an increase in power as each component is active every cycle while components are only active for 1 in 3 clocks in the other two SHA256 versions.

Table 3: DFPIM Benchmark Results

	Hist	Word	FFT	BFS	Str M	Lin R	SHA256	SHAmac	SHAmac3
Clocks (M)	9.77	6.02	49.27	57.56	33.57	21.97	19.75	19.75	6.58
Freq28 (clk/usec)	1100	1100	1100	1100	1100	1100	1100	1100	1100
Freq16 (clk/usec)	1300	1300	1300	1300	1300	1300	1300	1300	1300
Freq07 (clk/usec)	1800	1800	1800	1800	1800	1800	1800	1800	1800
Items / Vault (K)	9765	5928	4096	2000	32909	21972	102	102	102
Item Size (bytes)	3	1	8	8	1	2	64	64	64
DF Power28 (W)	0.2000	4.6165	1.7990	0.2885	0.5598	0.6358	0.2899	0.0972	0.3803
DF Power16 (W)	0.1313	2.8157	1.1809	0.1845	0.3587	0.4120	0.1840	0.0634	0.2442
DF Power07 (W)	0.0779	1.8954	0.7514	0.1132	0.2198	0.2690	0.1063	0.0376	0.1556
Mem Power28 (W)	2.0329	0.9828	0.8165	0.6148	0.9807	1.5119	0.6437	0.6437	0.9909
Mem Power16 (W)	2.3170	1.0760	0.8795	0.6411	1.0736	1.7014	0.6752	0.6752	1.0857
Mem Power07 (W)	3.0274	1.3091	1.0369	0.7070	1.3058	2.1750	0.7542	0.7542	1.3224
Exec Time28 (S)	0.0089	0.0055	0.0448	0.0523	0.0305	0.0200	0.0180	0.0180	0.0060
Exec Time16 (S)	0.0075	0.0046	0.0379	0.0443	0.0258	0.0169	0.0152	0.0152	0.0051
Exec Time07 (S)	0.0054	0.0033	0.0274	0.0320	0.0186	0.0122	0.0110	0.0110	0.0037
DF Energy28 (J)	0.0261	0.0336	0.1484	0.0815	0.0631	0.0537	0.0160	0.0142	0.0123
DF Energy16 (J)	0.0221	0.0198	0.0960	0.0569	0.0466	0.0423	0.0109	0.0101	0.0091
DF Energy07 (J)	0.0185	0.0115	0.0571	0.0352	0.0327	0.0327	0.0067	0.0063	0.0065
BW28 (MB/S)	52800.0	17322.8	11704.8	4892.5	17254.9	35200.0	5866.6	5866.6	17599.4
BW17 (MB/S)	62400.0	20472.4	13832.9	5782.0	20392.2	41600.0	6933.2	6933.2	20799.2
BW07 (MB/S)	86400.0	28346.4	19153.2	8005.8	28235.3	57600.0	9599.9	9599.9	28798.9
Clocks per item	1.00	1.02	12.03	28.78	1.02	1.00	192.00	192.00	64.00

5.4. Server to DFPIM 28 nm Comparison

The Intel Xeon E5-2683v3 processor used in this evaluation is implemented in a 22 nm FinFET silicon process. It is being compared to the DFPIM in a 28 nm planar process. This gives the server processor a moderate technology advantage. The lower production volume of a PIM logic layer compared to a server processor would favor the lower development and production cost of the 28 nm planar technology making this a reasonable comparison. The benefits of the two smaller FinFET technologies are shown in Table 4 for applications needing the additional performance while maintaining low energy.

The speedup 28 shows the execution time on server compared to that on DFPIM using 28nm technology.

The large histogram speedup is a result of computing the red, green, and blue pixel components in parallel. The DFPIM ability for single clock-cycle read-modify-write of the scratch pad memories is another factor contributing to the large histogram speedup. The word occurrence count benchmark speedup results from separate, independent character and word processing sections and the DFPIM capability to perform a full word comparison in a single clock. The FFT speedup is achieved by parallelism and pipelining to achieve 14 algorithm operations per clock cycle. The breadth first search benchmark has only a marginal speedup due to its unstructured and limited parallelism. The string match benchmark speedup results primarily from processing the four keys in parallel and independent character and word processing sections. The linear regression benchmark performs all five updates in parallel with the three multiplications pipelined for eight operations per clock. The standard SHA256 algorithm provides an average of eight operations per clock, but is limited to a threeclock pipeline latency due to data dependencies. Interleaving three benchmark instances increases the speedup for SHAmac3.

Likewise, E-ratio 28 shows the energy consumed on server compared to the energy consumed on DFPIM using 28nm technology. \ The energy ratio of 334.4 for the FFT indicates performing an FFT on a server processor requires 334 times that of DFPIM implementation. The speedup is then multiplied with the energy ratio to get a ratio of the energy-delay products of the benchmark implementations. The Table also includes the memory bandwidths taken directly from Table 3.

Table 4: Server vs 28nm DFPIM comparisons

	Hist	Word	FFT	BFS	Str M	Lin R	SHA256	SHAmac	SHAmac3
Speedup 28	16.9	7.0	7.9	1.6	4.3	9.3	3.2	3.2	7.5
Server energy (J)	19.250	4.913	49.617	10.507	16.808	23.657	8.425	8.425	8.425
E-ratio 28	737.3	146.3	334.4	128.9	266.5	440.5	525.6	592.3	686.3
S * E-ratio 28	12425	1025	2646	201	1149	4109	1667	1878	5158
Server BW	3127.6	2469.8	1348.1	3105.1	4002.1	3769.6	1604.5	1604.5	1604.5
DFPIM 28 BW	52800.0	17322.8	11704.8	4892.5	17254.9	35200.0	5866.6	5866.6	17599.4

5.5. DFPIM 28 to DFPIM 16 nm and 7 nm Comparison

The 28 nm planar technology has been in production since 2009. The 16 nm technology began production in 2013 and the 7 nm technology began production in 2017. The newer technologies offer both performance and energy efficiency benefits. These benefits are quantified in Table 5. The first section of the table uses all three DFPIM time values from Table 3. The *speedup 16 (speedup 7)* row is the DFPIM 28 execution time divided by the DFPIM 16 (DFPIM 7)execution time. Likewise, E-ratio 16 and E-ratio 7 show the energy comparisons of 28, 16 and 7nm technologies

Table 5. Comparing 28nm, 16nm and 7nm DFPIM results

	Hist	Word	FFT	BFS	Str M	Lin R	SHA256	SHAmac	SHAmac3
Speedup 16	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
Speedup 7	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6
E-ratio 16	1.18	1.69	1.55	1.43	1.35	1.27	1.46	1.41	1.34
E-ratio 7	1.41	2.92	2.60	2.31	1.93	1.64	2.39	2.25	1.90
S * E-ratio 16	1.40	2.00	1.83	1.69	1.60	1.50	1.73	1.67	1.59
S * E-ratio 7	2.31	4.78	4.25	3.79	3.15	2.69	3.91	3.68	3.10

The speedup, energy ratio, and speedup * energy product ratio factors are multiplicative with the values for the server to DFPIM 28 nm comparison shown in Table 4. Thus, the average speedup of 7.2 for server to DFPIM 28 nm becomes 11.5 (7.2 * 1.6) for the server to the DFPIM 7 nm speedup. The energy efficiency of 368 for server to DFPIM 28 nm becomes 810 for server to DFPIM 7nm.

6. Related Works

We only include works that rely on dataflow like processing. There are too many proposals for PIM or Near Data Processing that use conventional processing architectures or GPUs.

The Near DRAM Accelerator (NDA) [10] utilizes a dataflow network of functional devices to reduce energy by 46% and increase performance by an average 1.67 times. The NDA does not include sequencing functional units or scratch pad memories which DFPIM has shown to be necessary for improved 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.

Gan uses a reconfigurable dataflow architecture [11] to implement stencil operations for atmospheric modeling. The FPGA implemented system achieved a speedup of 18 compared to a server processor. The server processor used 427 Watts, while the FPGA hardware added 523 Watts to achieve the speedup. The overall power efficiency was 8.3. The power required is not suitable for a PIM application, but the performance gain showed the effectiveness of a dataflow implementation.

The Heterogenous Reconfigurable Logic (HRL) near data processing [12] 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 7.2 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.

HRL does not have the programmable state machine for sequencing and depends on the host for looping.

The DySER system integrates dataflow graph processing into the pipeline of a processor essentially transforming the dataflow graph into a processor instruction [13, 14]. The CPU instruction fetch and single memory access per instruction greatly limits the performance of DySER. Harmonic mean speedup ranged from 1.3 on SPECint benchmarks to 3.8 on GPU benchmarks. Being integrated into the processor pipeline restricts the parallelism and pipelining that a full dataflow construct can provide.

The bundled execution of recurring traces (BERET) research implements basic blocks as a dataflow subgraph in a coprocessor [15]. Each subgraph is executed through the CPU coprocessor interface. A set of eight subgraphs were selected through trace analysis to be implemented. The system resulted in a 19% performance improvement and a 35% energy savings. A coprocessor implementation of a standard eight subgraphs limits the capability of a full dataflow approach.

Single Graph Multiple Flows (SGMF) [40] uses a dynamic dataflow paradigm and CGRL to compare with an Nvidia Fermi streaming multiprocessor. The applications for SGMF are 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.

The Wave Computing dataflow based neural net accelerator [33] uses an array of small processors to execute basic block instructions. Each processor is assigned a basic block and accepts data from its predecessors and provides data to its successors. The processor contains a 256-entry instruction RAM and a 1KB data RAM. The network routing forms the dataflow graph. Current implementation of a Wave compute appliance consists of four data processing units per board, multiple boards per chassis and multiple chassis. This is not suitable for a low power PIM implementation

7. Conclusions

In this paper we described a processing-in-memory accelerator based on dataflow computing model and we show that our system can be used for distributed applications such as Big Data analytics.

We used careful and extensive logic syntheses to obtain execution and energy values for our DFPIMs components using 28nm, 16nm and 7nm technologies. We developed a backend to LLVM to generate dataflow graphs from C code kernels identified for PIM processing. These graphs are then used by our simulator, which executes the graph with inputs and generates results. We have verified the correctness of execution by our simulator by comparing the results generated from an execution on a host processor that uses Intel Xeon cores.

We compared the performance and energy values of DFPIM implementations with those obtained from our baseline host consisting of 2 14-core Intel Xeon processors for a variety of benchmarks. We evaluated three different versions of DFPIM using 28nm, 16nm and 7nm technologies. We compared 28nm planar version of DFPIM with the host (which uses 22 nm FinFET technology).

The DFPIM concept showed good performance improvement and excellent energy efficiency for the streaming benchmarks that were analyzed. The DFPIM in a 28 nm process with an a DFPIM core in each of 16 vaults showed an average speedup of 7.2 over 32 cores in the server system. The server processor required 368 times more energy to execute the benchmarks than the DFPIM implementation. These values result from the parallelism and pipelining available in the DFPIM architecture and the use of low power libraries in the silicon process.

Better performance and higher energy efficiency are possible by using the more recently available 16 nm and 7 nm silicon technologies. The 16 nm technology provides a modest speedup of 1.2 with an energy efficiency improvement of 1.4 compared to the 28 nm. The 7 nm technology provides a speedup of 1.6 with an energy efficiency of 2.2 compared to the 28 nm.

The 7 nm DFPIM implementation has an average speed-up of 11.5 with an energy efficiency ratio of 810 when compared to the server processor system.

Acknowledgements. This research is supported in part by NSF Net-centric Industry/University Cooperative Research Center and its industrial memberships.

8. References

[1] F. Ahmad, S. Lee, M. Thottethodi, and T. N. Vijaykumar, *Puma: Purdue mapreduce benchmarks suite*, Tech. report, Purdue University, 2012.

[2] B. Akin, F. Franchetti, and J. C. Hoe, *Data reorganization in memory using 3d-stacked dram*, Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on, June 2015, pp. 131–143.

[3] Arvind, *Data flow languages and architectures*, Proceedings of the 8th Annual Symposium on Computer Architecture (Los Alamitos, CA, USA), ISCA '81, IEEE Computer Society Press, 1981, pp. 1–.

[4] Arvind and D. E. Culler, *Dataflow architectures*, Annual Review of Computer Science Vol. 1, 1986 (Joseph F. Traub, Barbara J. Grosz, Butler W. Lampson, and Nils J. Nilsson, eds.), Annual Reviews Inc., Palo Alto, CA, USA, 1986, pp. 225–253.

[5] Arvind and R. S. Nikhil, *Executing a program on the mit tagged-token dataflow architecture*, IEEE Transactions on Computers 39 (1990), no. 3, 300–318.

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

[7] Xilinx Corp, *UltraScale Architecture and Product Data Sheet: Overview*, Jan 2018.

[8] Nvidia17 Corporation, *Nvidia TESLA V100 GPU Architecture*, 2017.

[9] Elpida Corp, *Elpida begins sample shipments of ddr3 sdram (x32) based on tsv stacking technology*, 2011.

[10] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N.S. Kim, *NDA: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules*, 2015 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE Conference papers, March 2015, pp. 283–295.

[11] L. Gan, H. Fu, W. Luk, C. Yang, W. Xue, and G. Yang, *Solving mesoscale atmospheric dynamics using a reconfigurable dataflow architecture*, IEEE Micro 37 (2017), no. 4, 40–50.

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

[13] V. Govindaraju, C. H. Ho, and K. Sankaralingam, *Dynamically specialized datapaths for energy efficient computing*, 2011 IEEE 17th International Symposium on High Performance Computer Architecture, Feb 2011, pp. 503–514.

[14] V. Govindaraju, Chen-Han Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and Changkyu Kim, *Dyser: Unifying functionality and parallelism specialization for energy- efficient computing*, Micro, IEEE 32 (2012), no. 5, 38–51.

[15] S. Gupta, S. Feng, A. Ansari, S. Mahlke, and David August, *Bundled execution of recurring traces for energy-efficient general purpose processing*, Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (New York, NY, USA), MICRO-44, ACM, 2011, pp. 12–23.

[16] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, *Mibench: A free, commercially representative embedded benchmark suite*, Workload Char- acterization, 2001. WWC-4. 2001 IEEE International Workshop on, Dec 2001, pp. 3–14.

[17] J. L. Hennessy and D. A. Patterson, *Computer architecture, fifth edition: A quan- titative approach*, 5th ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

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

[19] Hybrid Memory Cube Consortium, *Hybrid memory cube specification 2.1*, 2014.

[20] International Technology Roadmap for Semiconductors, *Itrs interconnect working group, 2012 update*, 2012.

[21] J. Jeddeloh and B. Keeth, *Hybrid memory cube new dram architecture increases density and performance*, 2012 Symposium on VLSI Technology (VLSIT), June 2012, pp. 87–88.

[22] JEDEC Solid State Technology Association, *Jesd235a high bandwidth memory (HBM) dram*, 2015.

[23] K. M. Kavi, R. Giorgi, and J. Arul, *Scheduled dataflow: execution paradigm, architecture, and performance evaluation*, IEEE Transactions on Computers 50 (2001), no. 8, 834–846.

[24] K. M. Kavi, C. Shelor, and D. Pace, *Concurrency, synchronization, and speculation - the dataflow way*, Advances in Computers 96 (2015), 47–104.

[25] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, *Gpus and the future of parallel computing*, IEEE Micro 31 (2011), no. 5, 7–17.

[26] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, *Energy management for commercial servers*, Computer 36 (2003), no. 12, 39–48.

[27] Arm Limited, *Arm Physical IP*, 2017.

[28] LLVM Project, *The llvm compiler infrastructure*, 2018.

[29] ___, *Writing an LLVM Pass*, 2018.

[30] G.H. Loh, *3d-stacked memory architectures for multi-core processors*, Computer Architec- ture, 2008. 35th International Symposium on, June 2008, pp. 453–464.

[31] Micron Technology, *16gb: x16 twindie single rank ddr4 sdram datasheet*, nov 2014.

[32] ___, *Hmc high-performance memory brochure*, jun 2016.

[33] C. Nicol, *A Dataflow Processing Chip for Training Deep Neural Networks*, Hot Chips: A Symposium on High Performance Chips, August 2017.

[34] D. A. Patterson, *Latency lags bandwith*, Commun. ACM 47 (2004), no. 10, 71–75. [90] J. Thomas Pawlowski, *Hybrid memory cube (HMC)*, 2011.

[35] J. Thomas Pawlowski, *Hybrid memory cube (HMC)*, 2011.

[36] M. Scrbak, M. Islam, K. M. Kavi, M. Ignatowski, and N. Jayasena, *Processing-in-Memory: Exploring the Design Space*, Architecture of Computing Systems ARCS 2015 (Lus Miguel Pinho Pinho, Wolfgang Karl, Albert Cohen, and Uwe Brinkschulte, eds.), Lecture Notes in Computer Science, vol. 9017, Springer International Publishing, 2015, pp. 43–54.

[37] SK hynix, *DRAM HBM products*, 2018.

[38] *SPEC Benchmarks*, https://www.spec.org/benchmarks.html.

[39] P. Fay, T. Willhalm, R. Dementiev, *Intel Performance Counter Monitor - A Better Way to Measure CPU Utilization*, January 2017.

[40] D. Voitsechov and Y. Etsion, *Single-Graph Multiple Flows: Energy Efficient Design Alternative for GPGPUs*, Proceeding of the 41st Annual International Symposium on Com- puter Architecture (Piscataway, NJ, USA), ISCA '14, IEEE Press, 2014, pp. 205–216.

[41] A. White, *Exascale Challenges: Applications, Technologies, and Co-design*, From Petascale to Exascale: R&D Challenges for HPC Simulation Environments, 03 2011.

[42] W. A. Wulf and S. A. McKee, *Hitting the Memory Wall: Implications of the Obvious*, SIGARCH Comput. Archit. News 23 (1995), no. 1, 20–24.

[43] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, *TOP-PIM: Throughput-oriented Programmable Processing in Memory*, Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing (New York, NY, USA), HPDC '14, ACM, 2014, pp. 85–98