NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules

Amin Farmahini-Farahani†, Jung Ho Ahn‡, Katherine Morrow†, and Nam Sung Kim†
†University of Wisconsin-Madison, ‡Seoul National University
farmahinifar@wisc.edu, gajh@snu.ac.kr, kati@engr.wisc.edu, nskim3@wisc.edu

Abstract

Energy consumed for transferring data across the processor memory hierarchy constitutes a large fraction of total system energy consumption, and this fraction has steadily increased with technology scaling. In this paper, we propose near-DRAM acceleration (NDA) architectures, which process data using accelerators 3D-stacked on DRAM devices comprising off-chip main memory modules. NDA transfers most data through high-bandwidth and low-energy 3D interconnects between accelerators and DRAM devices instead of low-bandwidth and high-energy off-chip interconnects between a processor and DRAM devices, substantially reducing energy consumption and improving performance. Unlike previous near-memory processing architectures, NDA is built upon commodity DRAM devices; apart from inserting through-silicon vias (TSVs) to 3D-interconnect DRAM devices and accelerators, NDA requires minimal changes to the commodity DRAM device and standard memory module architectures. This allows NDA to be more easily adopted in both existing and emerging systems. Our experiments demonstrate that, on average, our NDA-based system consumes 46% (68%) lower (data transfer) energy at 1.67× higher performance than a system that integrates the same accelerator logic within the processor itself.

1. Introduction

A major source of energy inefficiency in current systems originates from transferring data between where it is stored and where it is processed. The processor must transfer data from off-chip DRAM devices to its on-chip caches and then into its register file before processing it. Energy consumed for data transfers is much higher than actual computations. For example, transferring data from off-chip DRAM devices through the cache hierarchy to the register file consumes about two orders of magnitude more energy than performing a floating-point operation [1]. Furthermore, the fraction of data transfer energy in the total system energy is projected to increase with technology scaling [1, 2].

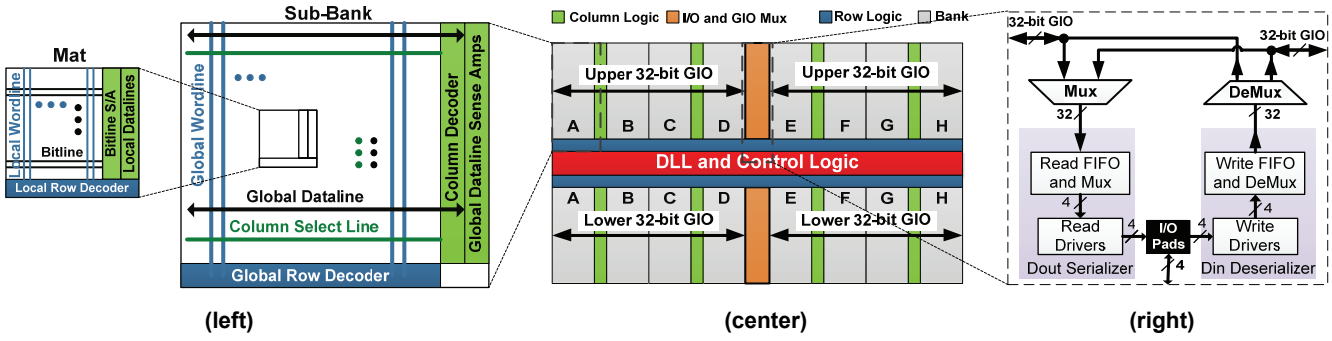
To reduce such data transfer energy, we can decrease data transfer distance by processing data near or in memory. This motivates us to re-examine the previous processors-in-memory (PIM) architectures [3–10] that aimed to improve performance by integrating processor logic and DRAM on the same die. Such PIM architectures, however, suffered from high manufacturing complexity (i.e., low yield), poor processor logic performance, large DRAM area per bit [11, 12], and design/verification challenges associated with custom DRAM architectures. Recently, 3D-stacking technology [13–16] has emerged as an alternative integration technology. It can solve some of the critical problems faced by the previous PIM architectures because it integrates logic and DRAM layers, each of which is manufactured with dedicated and separate process technology, with high-bandwidth and low-energy TSVs.

Leveraging 3D-stacking technology, researchers have proposed near-DRAM acceleration (NDA) architectures that integrate accelerator logic and *custom 3D* DRAM devices to reap the performance and energy-efficiency benefits of both accelerators and near-memory processing [17–20]. More specifically, they focus on either accelerator architecture where its memory system is separate from the host processor's main memory system (similar to a discrete GPU architecture) or the integration of and interaction between accelerators and DRAM using proprietary interfaces.

In this paper, we take *commodity 2D* DRAM devices and stack low-power and flexible accelerator logic atop DRAM. Our NDA architecture provides, in a practical way, high-bandwidth connections between logic and DRAM for near-memory processing. Our NDA architecture is seamlessly integrated with the host processor's main memory system based on standard off-chip memory modules. Specifically, we make the following contributions:

- We propose an NDA architecture that requires no change to the host processor design and minimal changes to the commodity DRAM device's I/O circuitry while maintaining the compatibility with the standard DRAM interface and DIMM architecture.
- We explore three NDA microarchitectures that can provide diverse DRAM-accelerator bandwidth using commodity DRAM. We analyze the impact of those microarchitectures on DRAM area, timing, and energy in detail.
- We identify various software and hardware challenges in implementing our NDA architecture (e.g., processor-accelerator and DRAM-accelerator communications due to sharing the main memory system with the host processor) and provide novel cost-effective solutions.

978-1-4799-8930-0/15/$31.00 ©2015 IEEE



(left) (center) (right)

Figure 1. Conventional DRAM organization. A DDR3 device with 8 banks of A to H and ×8 I/O interface (center), internal architecture of a DRAM bank and a mat (left), and DRAM I/O datapath (right).

Compared to a computing system integrating the same accelerator logic inside the processor, our NDA architecture can decrease data transfer energy by 64%-68% and increase performance by 19%-67%; NDA microarchitectures increase DRAM area by only 0.2%-3.3% with no negative impact on DRAM timing. Finally, our NDA architecture can also run legacy applications with practically no performance penalty while reaping its performance and energy efficiency benefits for NDA-mapped applications.

The remainder of this paper is organized as follows. Section 2 provides background information. Section 3 details our proposed NDA architecture. Section 4 discusses software considerations for NDA. Section 5 introduces our evaluation methodology. Section 6 demonstrates simulation results. Section 7 presents prior work. Finally, Section 8 concludes the paper.

2. Background

In this section, we provide an overview of DRAM and a description of a type of accelerator that was chosen for use in the tested NDA architecture.

2.1 DRAM Architecture

To better understand microarchitectures that enable low-cost, high-bandwidth connections between the accelerator logic and DRAM die, we first study the internal architecture of DRAM and its I/O circuitry. Figure 1(center) shows a DDR3 DRAM device architecture with an ×8 interface (8-bit data I/O per device) [21, 22]. This device consists of eight banks, each of which has two sub-banks located above and below the middle control logic. Each bank operates independently of other banks, but shares some resources such as DLLs and I/Os with other banks.

Each sub-bank is an array of memory cells that is divided into multiple mats. A mat is an array of 512×512 DRAM cells which has its own local row decoder, local wordlines, local datalines, bitlines, and bitline sense amplifiers. Each sub-bank includes a dedicated global row decoder and column decoder as illustrated in Figure 1(left). The global row decoder decodes the address and drives a particular global wordline. The local row decoders then use global wordlines to drive local wordlines. The bitline sense amplifiers latch data in each mat. Next, the global column decoder asserts the column select lines which drive data from the bitline sense amplifiers onto the global datalines through local datalines. The global datalines also have sense amplifiers to decrease data transfer latency.

The eight banks are divided into two groups of four left banks and four right banks. Each bank group shares a 64-bit inter-bank global I/O (GIO) lines (also known as inter-bank datalines). The GIO lines in this device are comprised of upper and lower 32-bit GIO lines. The GIO lines connect global datalines of each bank to data I/O logic pins shared by all banks. For each read operation, multiplexers depicted in Figure 1(right) select 64-bit data from the left or right 64-bit GIO lines; this data is then serialized before being sent through the 8-bit data I/O pins. Since the I/O data rate of DDR3 devices is 8× DRAM core clock frequency, the number of GIO lines is 8× the number of data I/O pins (i.e., a prefetch size of 8*n*, where *n* is the device I/O width).

2.2 Accelerator Architecture

Various accelerator architectures such as SIMT or SIMD processors would be compatible with this work. While each accelerator architecture provides a unique trade-off between performance/energy-efficiency improvement and programmability, we will separate such an impact by also evaluating a computing system that integrates the same accelerator architecture in a processor itself. This allows us to evaluate the net benefit of processing data near the DRAM regardless of a particular choice of accelerator architecture. In our experimentation, we assume that accelerators are a type of dataflow architecture—coarse-grain reconfigurable accelerators (CGRAs). CGRAs have been shown to provide significant performance and energy efficiency benefits [23].

A CGRA is typically comprised of a grid of word-sized functional units (FUs) that perform various arithmetic and

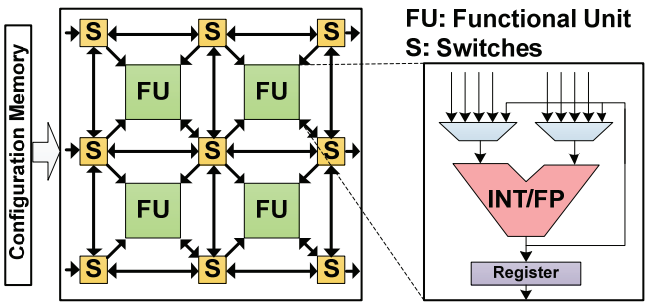


Figure 2. A CGRA with a grid of 2×2 functional units.

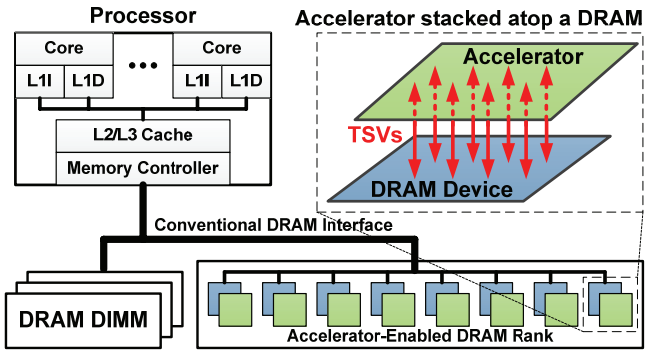


Figure 3. NDA organization.

logic operations, which are connected by a configurable interconnect fabric (Figure 2). CGRAs generally also include some amount of storage near each FU for intermediate results. A dataflow graph of a compute-intensive application kernel, containing nodes (operations) and edges (data communication), is mapped to the CGRA's FUs and interconnect either manually or using automated techniques [24]. The processor triggers CGRA reconfiguration at runtime to implement different kernels at different times, overwriting configuration memory with the new dataflow graph's configuration data. This allows the CGRA to act as "virtual hardware" [25]. Configuration data for the various dataflow graphs can be retained in main memory until it is needed, and is often cached in configuration memory local to the CGRA.

Exploiting spatial data parallelism in application kernels and efficiently processing kernel's dataflow graphs [26–30], CGRAs, such as those used in DySER [28] and SGMF [29], considerably improve performance and energy consumption compared to conventional processors; in particular, CGRAs can practically eliminate the large energy overheads of fetching and scheduling instructions in conventional out-of-order processors. Compared to fine-grain reconfigurable accelerators (e.g., FPGAs), CGRAs are less flexible, but this specialization results in higher performance, lower energy consumption, and much smaller configuration data (shorter configuration time) [26, 31, 32]. With recent advances in their compilers and architectures [30, 33–35], CGRAs are gaining momentum in various applications.

3. NDA Hardware Architecture

The NDA architecture is not dependent on a specific type of accelerator logic; the accelerators could be CGRAs as discussed in this paper, or could be SIMD/GPU/FPGA engines or even low-power cores. This paper focuses on CGRAs due to their improved performance and energy consumption versus SIMD and GPU engines for most parallel workloads [23]. Low energy is important since near-memory architectures have more stringent power/thermal constraints.

In order to facilitate energy-efficient near-memory processing, NDA stacks a CGRA on top of each DRAM device. The CGRA is connected to the internal DRAM I/O lines using TSVs. A conceptual view of the NDA architecture is illustrated in Figure 3. Each CGRA is connected only to its associated DRAM device and operates on data stored in that

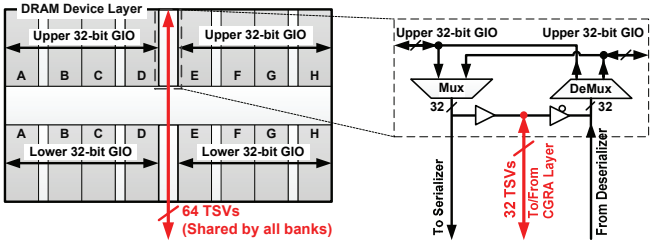


Figure 4. Accelerator-DRAM connection by connecting TSVs to existing GIO lines (Microarchitecture 1).

device independently of (and in parallel with) the CGRAs on the other DRAM devices on the DIMM(s).

NDA is architected such that it does not require changes to the processor or DIMM interface, and only minimal changes to the underlying DRAM design. Thus, NDA can be used with existing systems to accelerate NDA-enhanced applications. Furthermore, this architecture allows existing un-accelerated applications to run on NDA-equipped platforms without incurring any performance penalty [6].

Conventional processors can offload kernels to CGRAs. NDA transfers data through high-bandwidth and low-energy 3D interconnects between DRAM devices and their corresponding CGRAs without the processor's intervention and processes the data using the CGRAs. This minimizes data transfers through low-bandwidth and high-energy off-chip interconnects between the processor and DRAM devices. To maximize acceleration, the kernel data processed by CGRAs must be distributed evenly across DRAM devices to balance the computations across CGRAs.

3.1 Connection between Accelerator and DRAM

We propose three microarchitectures to connect a CGRA and a DRAM device using TSVs. In all the microarchitectures, we assume the TSV I/O energy is 4 pJ/b, which is 80% lower than the off-chip I/O energy of a DDR3 interface (Table 1).

Microarchitecture 1 (Connecting TSVs to existing GIO lines – NDA-1): A CGRA simply reads or writes data through the TSVs connected to the existing DRAM GIO lines without changing the underlying DRAM device design. TSVs are connected to the GIO lines between GIO multiplexers and data serializer/deserializer (cf. Figure 4). DRAM devices with ×8 and ×16 interfaces use 64 and 128 TSVs to transfer data between a DRAM device and its CGRA, respectively. This microarchitecture is similar to one developed for stacking a wide-I/O DRAM device atop a processor die [36]. As depicted in Figure 4, the same steering logic that directs data to/from the left or right set of banks also directs data through the TSVs to the CGRA. Hence, this microarchitecture has no area overhead apart from inserted TSVs (cf. Table 2). A CGRA transferring data through 3D interconnects is provided the same peak bandwidth per device as a processor transferring data through the off-chip interconnects because both the processor and the CGRA use the same mechanism to transfer data. However, the former consumes 51% lower energy per 8-byte read/write data transfer than the latter (cf. Table 2). The

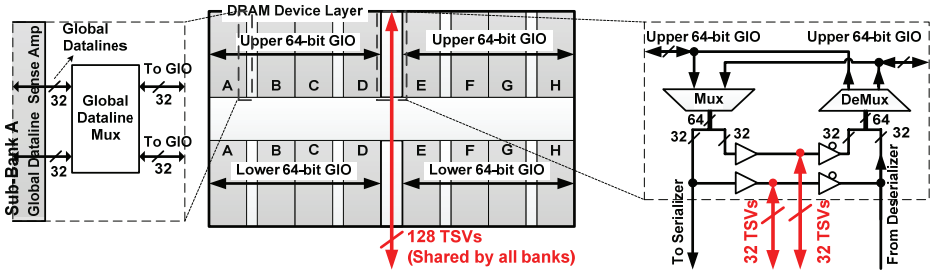


Figure 5. Accelerator-DRAM connection by connecting TSVs to doubled GIO lines (Microarchitecture 2).

CGRA-side memory controller (MC) manages data transfers between the two dies (cf. Section 3.2).

Microarchitecture 2 (Connecting TSVs to doubled GIO lines – NDA-2): Similar to Microarchitecture 1, we connect TSVs to GIO lines except that the number of GIO lines is doubled. By doubling the number of GIO lines and accordingly the number of TSVs, we double the CGRA-DRAM bandwidth (cf. Figure 5). This change requires doubling the number of global datalines, global dataline sense amplifiers, and local datalines, but halves the number of column select lines in each bank. The number of bitline sense amplifiers is not affected. Our SPICE simulation shows that this microarchitecture slightly increases the bank area and the GIO lines area compared to Microarchitecture 1 (cf. Table 2). However, DRAM timing parameters can remain unchanged by adjusting the repeater interval of GIO lines, as elaborated in [22]. Note that the read/write energy efficiency of this microarchitecture is better than that of Microarchitecture 1 since the overhead of sending address/command is amortized over a larger number of bits per data transfer transaction. To transfer data from/to a DRAM device, the CGRA uses all GIO lines while the processor uses only a half of GIO lines to reduce energy consumption of the long GIO lines. This can be supported by controlling data multiplexing at the boundary of global datalines and GIO lines (Figure 5).

Microarchitecture 3 (Connecting TSVs to banks' global datalines – NDA-3): In both Microarchitectures 1 and 2, the TSV connection is shared by all the banks in a DRAM device. However, each bank can be independently controlled, which can be exploited by a CGRA. In Microarchitecture 3, we instead provide each bank with a separate TSV connection to the CGRA. Thus, the CGRA can access data from each independently-addressed bank. This provides the opportunity to exploit the benefit of bank-level parallelism by accessing data from multiple banks concurrently, eliminating the limitation of sharing a TSV connection among all banks in a DRAM device. In this microarchitecture, we connect TSVs to global datalines of each bank, forming eight independently accessed TSV connections (Figure 6). This microarchitecture increases the CGRA-DRAM bandwidth substantially with only small overhead associated with TSVs (cf. Table 2). The CGRA access latency and energy decrease as well due to directly accessing data from bank's global datalines and bypassing GIO lines. The CGRA-side MC directs DRAM requests and responses between the CGRA and its corresponding bank.

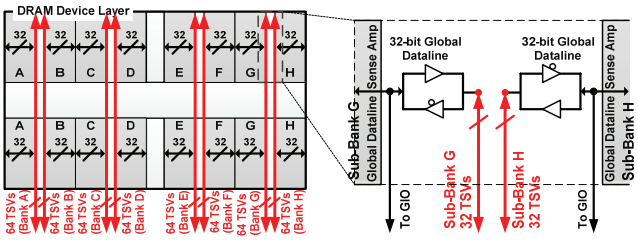


Figure 6. Accelerator-DRAM connection by connecting TSVs to global datalines of banks (Microarchitecture 3).

TSV overhead: Current 3D interconnect technology allows a TSV pitch size of 40-50µm for 3D DRAM stacking [36, 37], which provides enough room for PHY, test, and ESD protection [14]. To reduce TSV area overhead, finer grain TSVs with a pitch size of 5µm can be used for signals [15] with redundant TSVs to increase the yield [38]. ~280 TSVs are required for a 128-bit data connection between a CGRA and a DRAM device to account for the overheads of address, command, power, and ground [36]. This incurs an area overhead of only ~0.243 mm^2 by using coarse-grain TSVs for power/ground and fine-grain TSVs with 4:2 redundancy [38] for signals. Since the typical area of DDR3 and DDR4 devices ranges from 30 mm^2 to 100 mm^2 [39, 40], the area overhead of TSVs is negligible.

Summary: Table 2 lists the peak bandwidth of a DDR3-1600 device using off-chip I/O pads vs. using TSVs. The off-chip I/O bandwidth depends on the I/O clock frequency and I/O bit-width (with two transfers per I/O clock). The TSV bandwidth depends on the core clock frequency, the microarchitecture used to connect a CGRA with its DRAM device, and the number of TSVs. In Microarchitecture 1, a TSV-based connection provides the same peak bandwidth per DRAM device as the off-chip I/O bus connection. In Microarchitecture 2, the increase in the number of GIO lines doubles the TSV peak bandwidth, while off-chip I/O bandwidth remains unchanged. Microarchitecture 3 provides eight times the peak bandwidth of Microarchitecture 1 by accessing banks independently. Note that the processor and CGRA cannot access DRAM cells concurrently in any microarchitecture (cf. Section 3.2).

In all three microarchitectures, the underlying DRAM architecture remains intact, making the proposed microarchitectures low-cost and compatible with the industry standard DDR3 devices. To quantify the overheads and savings, we modeled the internal DRAM components using SPICE simulations in a 3-metal layer 28nm DRAM process and $6F^2$

Table 1. Area, timing, and energy parameters of an 8Gb DDR3-1600 ×8 DRAM device. ×8 = off-chip data I/O bit-width is 8.

Device	Row buffer size	Peak bandwidth	Core Freq.	I/O Freq.	Number of banks	Area	Access latency (*tCL*)	RD or WR energy without I/O	Off-chip I/O energy
DDR3-1600 ×8	1KB	12.8 Gbps	200 MHz	800 MHz	8	80 mm^2	13.75 ns	13 pJ/b [73]	20 pJ/b [22]

Table 2. Summary of bandwidth, area, timing, and energy of different microarchitectures to connect Accelerators and DRAM devices. Numbers are relative to DDR3-1600 ×8 parameters given in Table 1. S/A = Sense Amplifiers.

Connection between Accelerator and DRAM	Off-chip data I/O bit-width	Number of data TSVs	Peak bandwidth (Gbps)		Area change	Accelerator timing change	CPU timing change	Accelerator RD energy (without I/O)	CPU RD energy (without I/O)
			Through I/O pins	Through TSV pins					
NDA-1 (Connecting TSVs to existing GIO lines)	8	64	12.8	12.8	0.2% (TSV)	-	-	-51% (-7%)	-
	16	128	25.6	25.6	4.3% (S/A, pins, TSV)	-	-	-64% (-39%)	-12% (-31%)
NDA-2 (Connecting TSVs to doubled GIO lines)	8	128	12.8	25.6	3.3% (S/A, TSV)	-	-	-64% (-39%)	~0% (+0.3%)
	16	256	25.6	51.2	7.6% (S/A, pins, TSV)	-	-	-70% (-53%)	-12% (-30%)
NDA-3 (Connecting TSVs to global datalines)	8	512	12.8	102.4	1.2% (TSV)	Decrease tCL by 6ns	-	-59% (-28%)	~0% (+0.1%)
	16	1024	25.6	204.8	6.4% (S/A, pins, TSV)		-	-72% (-59%)	-12% (-30%)

DRAM cells. Table 2 indicates that our microarchitectures have a limited impact on the DRAM area. DRAM access latency (either from the processor or CGRA) does not increase either; the CGRA-DRAM access latency even decreases by 6ns in Microarchitecture 3. Table 2 also indicates that CGRAs consume less energy to transfer the same number of bits than processors. This is because the CGRAs obviate the need of serializing data and also the TSVs used for the CGRAs have better signal integrity and lower load capacitance than the pads, bumps, and PCBs used to connect a processor and DRAM packages. The wider datapath structures consume even less energy per bit because the overhead of sending and decoding a command/address is amortized over a larger number of bits per data transfer transaction. The impact of enabling connection to CGRA on normal access energy to the processor is minimal because GIO energy, which is affected by the growth in DRAM area, is much smaller than inter-package I/O energy. Table 2 summarizes this energy overhead of the three microarchitectures on normal DRAM accesses.

3.2 Memory Access by Accelerators

Memory controller: As shown in Figure 7, each CGRA includes an MC to issue memory requests to its DRAM device. Through TSVs, the CGRA-side MC sends command/address bits to the DRAM device to indicate the type of operation and the address involved (which in turn determines the bank, row, and column within the DRAM device). For all the CGRA-DRAM connections proposed in Section 3.1, the CGRA-side MC manages data transfers between the CGRA and DRAM device in compliance with DRAM timing constraints such as tFAW and tRRD [41].

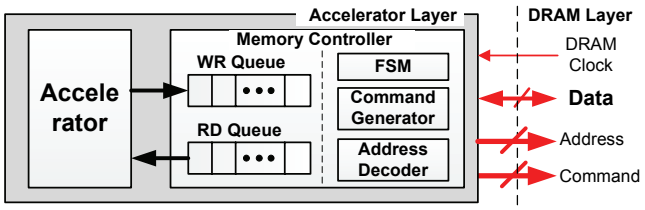


Figure 7. Accelerator stacked on top of a DRAM device.

The CGRA-side MC does not require complex request ordering and arbitration logic that exists in the processor-side MC. Moreover, many kernels include little or no data-dependent control flow and shun "pointer chasing." Thus, the memory request order can usually be optimized during kernel development. The processor-side MC is always responsible for providing DRAM clocks and refreshing the DRAM.

DRAM ownership transition between a processor and accelerators: When a kernel is launched, the processor-side MC hands over control of a DRAM rank to CGRA-side MCs. The processor-side MC stops sending commands (e.g., ACT and CAS) to the DRAM rank with active CGRAs. This allows avoiding frequent concurrent accesses by both the processor and the CGRAs (Section 3.3). When the ownership of a DRAM rank switches, its banks are in the precharged state. Thus, bank conflicts are avoided and the page policy (open or close) is managed by the owner. When processor cores (including ones that are not using CGRAs in a multi-programmed environment) need to access the DRAM rank with active CGRAs, CGRA operations are suspended (by writing to a DRAM mode register) and the CGRA-side MCs close (precharge) DRAM pages. At the same time, the processor-side MC assumes that all pages in that rank are currently closed when it takes back the control of the DRAM rank. Then, the processor-side MC can activate the required page and access data.

In a multi-programmed environment, processor cores might access the DRAM rank with active CGRAs. One way to alleviate the problem of such concurrent access is to partition main memory space into two groups: addresses mapped to conventional ranks, and addresses mapped to CGRA-enabled ranks. Memory allocation policies [42] can allocate application memory to a proper DRAM rank. Memory requests from applications that do not utilize CGRAs will not thus interfere with memory requests of active CGRAs. Similarly, the same ownership transition mechanism can be used when the processor-side MC must refresh the rank with active CGRAs. Although periodic refresh operations from the processor-side MC are unavoidable [43], refresh intervals (tREFI) are long enough (e.g., 7.8μs in DDR3) to allow

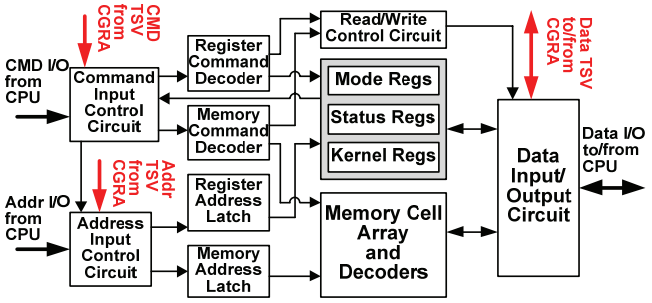


Figure 8. Access to mode, status, and kernel registers.

CGRAs to make considerable progress within them. CGRAs are aware of refresh intervals and can suspend their normal operations right before the refresh command from the processor-side MC. Recent techniques such as *adaptive refresh* can be used to further mitigate the refresh overhead [43]. Furthermore, on-die temperature sensors [21] can be provisioned to enable adjusting temperature-aware refresh intervals.

Memory addressing: CGRAs in NDA use physical memory addresses, and usually operate on big data structures (e.g., large images and matrices) that are brought into the memory by the processor at startup in large chunks. Big-data workloads rarely benefit from virtual memory features such as page swapping [44]. Therefore, to ensure the CGRAs read/write data from/into contiguous regions of physical memory, we adopt memory segmentations without paging for the region of memory address space accessed by CGRAs [44]. This segmentation approach maps contiguous regions of virtual memory to contiguous regions of physical memory which eliminates virtual memory overheads due to TLB misses for the large data structures accessed by CGRAs. The conventional page-based virtual memory is used for the rest of the address space. In NDA, the processor-side MC provides CGRAs with the physical starting address of data through system calls. Before invoking CGRAs, the processor must pin the memory regions (segments) accessed by CGRAs through the OS.

3.3 Processor-Accelerator Communication

The processor uses DRAM memory-mapped registers to communicate with CGRAs through the processor-DIMM interface. The processor-side MC transfers parameters such as data starting addresses, kernel configuration starting addresses, and kernel ID and triggers kernel execution/reconfiguration on CGRAs by writing to the *kernel registers* (Figure 8). This does not require changes to the processor-DIMM interface as existing MCs support accessing programmable *mode registers* in conventional DRAM. Once CGRAs start their execution, they can access the parameters from those register locations.

CGRAs can notify the processor of their execution status by writing to the *status registers* in their local DRAM devices. The processor periodically polls these memory-mapped status registers to check for kernel completion or exceptions (like *errno*). CGRA exceptions can be imprecise and terminate a kernel [45]. The status register read by the processor does not intervene in DRAM accesses by the CGRA since the status register which is supplied onto the off-chip bus uses a separate datapath in DRAM (Figure 8). Based on the kernel running on CGRAs and the data size of the kernel, the processor can accurately estimate the execution time of a kernel, thereby polling status when the kernel execution is about to finish or has just finished.

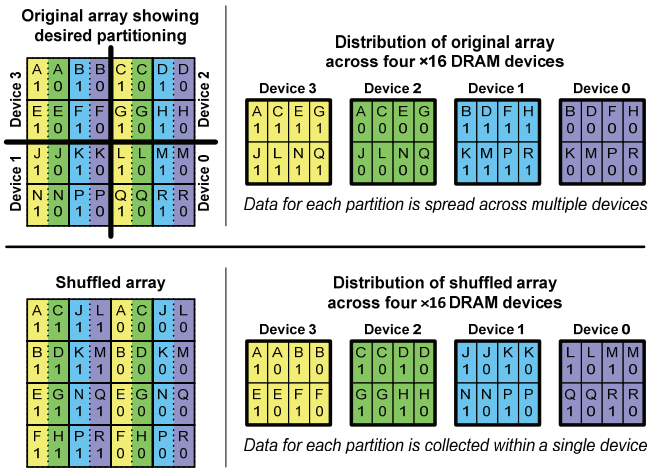


Figure 9. Data arrangement across devices for a 4×4 array of 32-bit words (A – R). Original data arrangement (top) and shuffled arrangement for NDA acceleration (bottom).

4. Software Considerations

4.1 Target Applications

NDA targets big-data applications with high parallelism and localized memory accesses which are traditionally deployed on Map-Reduce frameworks [46]. High parallelism enables employing CGRAs concurrently and localized memory accesses reduce inter-CGRA communication. Exploiting CGRAs for energy-efficient, accelerated data processing and high-bandwidth and low-energy 3D interconnects, NDA can process a large amount of data efficiently and quickly in the era of big data where huge data sets are being processed and analyzed constantly.

4.2 Data Arrangement for Accelerators

The conventional DIMM architecture interleaves each 64-bit data block amongst DRAM devices, each of which stores 8 and 16 bits for ×8 and ×16 DRAM devices, respectively. However, NDA requires that all data used by each CGRA be located within the DRAM device to which the CGRA is attached. Maintaining compatibility with the standard DIMM architecture and interface requires rearranging data both to (*i*) partition them across CGRAs for parallel processing and to (*ii*) ensure that all sub-words of a given data word are written to the same DRAM device.

Figure 9(top-left) shows an example of a 4×4 array of 32-bit words (A–R), where the four quadrants of the array should be distributed among the four ×16 DRAM devices in a DIMM as indicated. Here, each 32-bit data word consists of 16-bit upper and lower subwords, labeled "1" and "0". Figure 9(top-right) shows how this array is typically distributed across ×16 DRAM devices, separating the upper and

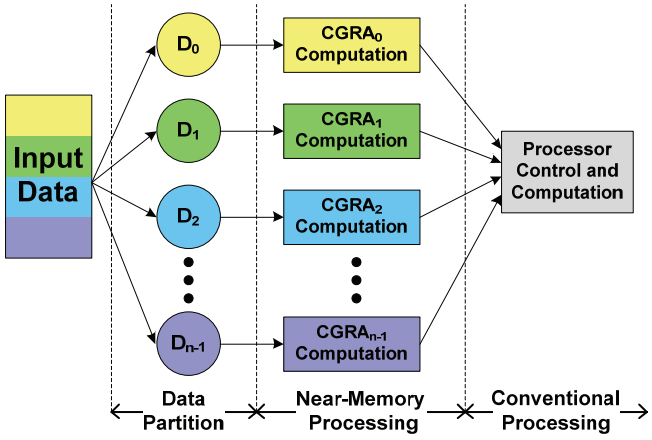


Figure 10. NDA programming model.

lower halves of each 32-bit word, and failing to group the data for each quadrant into a single device. Figure 9(bottom) shows a "shuffled" arrangement of the same data that results in the desired partitioning for NDA processing when it is written to the DRAM through the standard interface.

The processor shuffles CGRA input data during initialization. The once-per-execution overhead of input data shuffling is amortized over long-running applications that process a significant amount of data. Shuffling is not required for data that is both produced and consumed by CGRAs. For many applications, CGRA processing also reduces the local data within the DRAM device before it is further analyzed and reduced by the processor. This is similar to Map-Reduce programming model in which CGRAs perform map and local reduce operations and the processor performs global reduce and controls execution (Figure 10). If the CGRA result data is needed by the processor, the processor must "unshuffle" it before using it.

4.3 Programming for NDA

In NDA, Similar to SIMD/GPU engines, the processor is responsible for running the OS and irregular control-flow codes, and CGRAs execute application kernels.

Programming model: Figure 10 depicts the programming model for applications mapped to NDA. To enable concurrent CGRA computations with partitioned data, the processing is split into smaller computations. Data processing in many data-intensive applications in domains such as high-energy physics, biology, scientific computing, and geology can be split this way [47]. Some boundary data may need to be duplicated to reduce/eliminate inter-CGRA communication. The processor can orchestrate data sharing between CGRAs by reading/writing data from/to DRAM devices. After CGRA computations are completed, CGRAs output data is processed and merged by the processor.

Mapping programs to NDA: We analyze applications to find kernels with long execution time. We then divide the application into software (run by the processor) and hardware kernels (run by the CGRAs). The chosen kernels are converted to dataflow graphs, which are mapped to CGRAs. The software code is then modified to replace kernel computation with appropriate instructions to transfer parameters and control between the processor and CGRAs.

Program execution flow in NDA: After partitioning and shuffling data, the processor configures CGRAs to run the required kernels. The processor then sends kernel parameters to CGRAs and triggers execution. Once all processing using the current kernel is complete, the processor can read the CGRA output data and unshuffle it.

4.4 Data Coherence and Memory Consistency

There is no hardware mechanism to enforce data coherence between the processor and CGRAs. Since CGRAs have no access to the processor on-chip cache hierarchy, the data produced/modified by the processor and consumed by CGRAs should not be left in the processor's on-chip caches. To avoid inconsistencies between the processor's cache and main memory, the region of memory containing the data set consumed by CGRAs (i.e., CGRAs' shuffled input data) is defined as an un-cacheable region of memory. The processor does not usually consume data from this region of memory in the near future, if ever. Thus, it is not necessary to maintain cache coherence for this region of memory.

The processor accesses data in this region using *non-temporal instructions* (e.g., `MOVNTQ`, `MOVNTPS`, and `MASKMOVQ` in x86) that bypass the cache hierarchy. CGRAs usually operate on large data sets such as multi-dimensional matrices. The processor initializes/loads those data sets in memory using non-temporal instructions which are then consumed by CGRAs. Other data sets (including the data sets produced by CGRAs and consumed by the processor) can use hardware cache coherence as usual. Note that for the region of memory that is produced by CGRAs, the programmer should make sure no stale data is stored in the cache hierarchy before the processor accesses it. The processor and CGRAs do not operate on the same data set simultaneously. As a result, no changes to memory consistency are needed.

5. Evaluation Methodology

5.1 Benchmarks

Table 3 shows 11 different benchmarks from the San Diego Vision [48], Parboil [49], CORAL [50], SPLASH-2 [51], and Rodinia [52] benchmark suites that we use for evaluation. Table 3 also recaps the shuffling overhead of CGRAs' input data relative to the execution time of the benchmarks running on the baseline architecture in Section 5.3. The shuffling overhead numbers for iterative benchmarks are reported for 20 iterations. For the iterative benchmarks, as the number of iterations increases, the shuffling overhead of input data is amortized over more computation.

5.2 Architectural Components

CGRAs: FUs in our CGRAs are unified dual-mode integer/floating point units, meaning that they can operate in either integer mode or floating point mode. Most FUs can perform addition, subtraction, and a few logical operations, while a few FUs can perform complex operations such as

Table 3. Benchmarks used in our evaluation

Name	Description	# of Kernels	Iterative?	Replaced Exec. Time	Shuffling Overhead
BP	Back propagation [52]	4	Yes	99.9%	1.6%
DISP	Disparity map [48]	10	No	99.9%	0.1%
HACC	Hardware accelerated cosmology code [50]	2	Yes	99.9%	1.5%
HS	Hotspot [52]	2	Yes	99.9%	1.1%
KM	K-means clustering [52]	1	Yes	99.9%	0.7%
LBM	Lattice-Boltzmann method fluid dynamics [49]	1	Yes	99.9%	1.3%
MRIG	Magnetic resonance imaging gridding [49]	1	No	98.5%	0.14%
OCN	Ocean movements [51]	16	Yes	81.7%	1.6%
SIFT	Scale-invariant feature transform [48]	4	No	92.7%	0.1%
SRAD	Speckle reducing anisotropic diffusion [52]	3	Yes	99.9%	0.2%
TRCK	Feature tracking [48]	8	No	79.3%	2.0%

Table 4. Design parameters of a CGRA with 64 32-bit FUs

Unit	Latency (Cycles)	Count	Area (μm^2)	Energy per Op.
INT ALU	1	40	3589	2.2 pJ
FP ALU	5		4762	7.1 pJ
INT MUL	2	20	6507	13.1 pJ
FP MUL	4		6565	11.3 pJ
INT DIV	8	4	10596	30.1 pJ
FP DIV	9		15484	27.7 pJ
Switch	-	81	4236	1.11 pJ

multiplication and division. Table 4 summarizes the details of a CGRA with 64 FUs used in this paper. The types of FUs and the number of each type present the required operations of our benchmarks. Numerical methods approximate rarely used operations, such as square root.

For evaluating area and power consumption of FUs and switches in our CGRAs, we developed their Verilog models using Synopsys DesignWare building block IPs and synthesized them using the Synopsys design compiler 2013 and TSMC 40 nm low-power standard-cell library. We optimized them for a clock frequency of 800 MHz. Table 4 indicates that the area of a 64-FU CGRA is ~0.832 mm^2. CGRAs consume very little dynamic power because of their simple datapath and low frequency. In our benchmarks, the dynamic power of switches and FUs in a CGRA is only 3-17 mW, translating to a power density of 4-21 mW/mm^2. Also, the power gird of existing modern DRAM dies would further reduce CGRA temperature variations by acting as a heat spreader. Thus, CGRAs have minimal thermal impact on a DRAM device. Based on our estimates using McPAT [53], the area of a CGRA-side MC is ~0.21 mm^2.

Processor: We use a four-way OoO processor with a 16-stage execution pipeline. Table 5 shows the key architecture parameters of the processor and its caches.

Memory: We use DDR3-1600 DRAM devices with data I/O sizes of either 8 bits (×8) or 16 bits (×16), which are common for DDR3. In ×8 and ×16 interfaces, there are eight and four memory devices in a given rank, respectively, to form a 64-bit bus between the memory and MC. In general, DDR3 devices have a burst-length of 8, transferring 64 bytes of data in each memory transfer. Table 6 contains detailed timing parameters of DRAM subsystem used in our evaluation.

I/O: We denote I/O energy as a function of the number of I/O transactions and assume off-chip and TSV I/O energy is 20 pJ/b and 4 pJ/b, respectively.

Table 5. Key processor architecture parameters

Pipeline width	4	I/D L1 caches	32KB 4-way
ROB/IQ/LQ/SQ entries	128/36/48/32	I/D L1 ports	1/2
Integer and FP ALUs	Nehalem-like	L2 cache	512KB 8-way
Int & FP physical Regs	128 and 128	Cache line size	64 bytes
Branch predictor	Tournament	L1/L2 latency	3/16 cycles
BTB entries	2048	L1/L2 MSHRs	10/16

Table 6. DRAM subsystem parameters

DDR3-1600-11-11-11-28	
tRCD=13.75ns, *tCL*=13.75ns, *tRP*=13.75ns, *tRAS*=35.0ns, *tCCD*=5.0ns, *tWTR*=7.5ns, *tWR*=15.0ns, *tRTP*=7.5ns, *tRTW*=2.5ns, *tRRD*=6.25ns, *tFAW*=40.0ns, *tBURST*=5.0ns, *tRFC*=300.0ns, *tREFI*=7.8μs	
Device row buffer size	1KB/2KB for ×8/×16
Memory Management	
Page interleaving, Open page policy, 40/40-entry read/write request queues per MC, FR-FCFS scheduler	

5.3 Evaluated Architectures

We compare the performance of NDA against a diverse set of architectures, summarized in Table 7. In the "Baseline" architecture, we use a conventional four-way processor (Table 5) running at 2GHz and DDR3-1600 memory subsystem (Table 6) without any CGRAs. In "Base 4×Para," we assume that the performance of the evaluated applications perfectly scales with the number of cores whereas in reality, parallel application performance is strongly dependent on how well the applications are parallelized. In "Base+CGRAs," CGRAs are integrated with the processor [30, 54–56]; CGRAs transfer data through the off-chip interconnects between the processor and DRAM devices. In NDA, four CGRAs are stacked atop each ×8 DRAM device to provide the same total number of CGRAs as "Base+CGRAs." In NDA, we conservatively assume that it takes 1.25ns to latch (to minimize skew) and transfer data between a DRAM device and a CGRA over TSVs.

5.4 Simulation Platform

We extended the gem5 simulator [57] with the ARM ISA to support CGRA execution, 3D-stacked memory, and non-temporal memory accesses. We model the functional and timing execution of the processor, cache levels, CGRAs, DRAM devices, and interfaces among them. We extended the gem5 MC to support CGRA-DRAM 3D connections.

We use McPAT-based power models [53] to calculate power consumption of processor core, caches, and other on-chip units. We have developed an in-house power estimator to evaluate power consumption of CGRAs based on information given in Table 4. We assume both the processor and CGRAs are implemented in 40 nm ASIC technology. We use DRAMPower [58] for evaluating power consumption of DRAM devices and their I/O interface. We have modified

Table 7. Summary of different architectures evaluated in this paper

Abbreviation	# CGRAs	Description
Base	-	4-way OoO processor at 2GHz (Table 5) and DDR3-1600 ×8 memory (Table 6)
Base 4×Para	-	Same as the baseline with four threads running concurrently with hypothetical ideal thread parallelism
Base+CGRAs	32	Same as the baseline with 32 on-chip CGRAs co-located with the processor
NDA-1	32	Same as the baseline with 4 CGRAs stacked atop each ×8 DRAM device using Microarchitecture 1
NDA-2	32	Same as the baseline with 4 CGRAs stacked atop each ×8 DRAM device using Microarchitecture 2
NDA-3	32	Same as the baseline with 4 CGRAs stacked atop each ×8 DRAM device using Microarchitecture 3

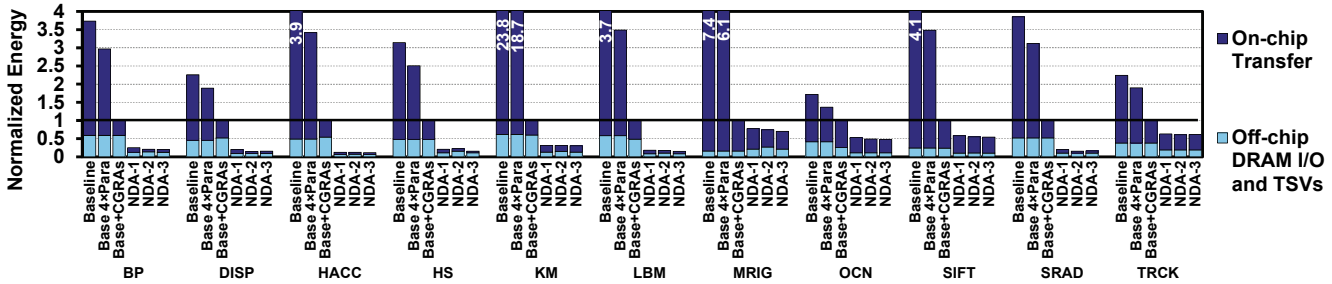


Figure 11. On- and off-chip data movement energy normalized to that of "Base+CGRAs."

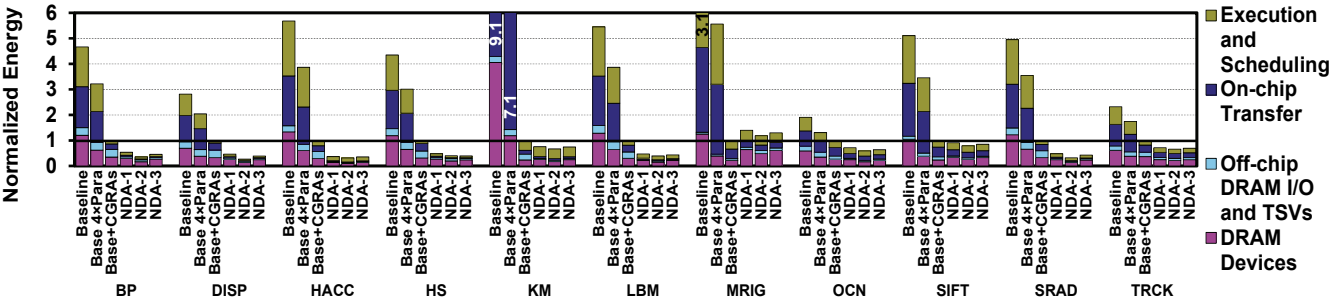


Figure 12. Combined processor, DRAM, and CGRA energy consumption normalized to that of "Base+CGRAs."

DRAMPower to support TSV-based connection and have integrated it with gem5.

6. Results

In this section, we evaluate the performance and energy-efficiency of NDA. Unless otherwise indicated, we use ×8 DRAM devices and CGRAs with 64 FUs at 800 MHz. We consider dynamic and leakage energy of the processor, DRAM devices, and CGRAs. While CGRAs are operating, the processor runs at 1/16th of its operating frequency (reflected in our results) and can periodically check the status of CGRAs by reading CGRA status registers through the processor-memory interface after a certain delay (e.g., 10000 idle loop iterations).

Data transfer energy: NDA improves data transfer energy by performing computation near off-chip DRAM devices. Figure 11 plots the energy consumed to transfer data through the memory I/O interface (off-chip and/or TSV interconnect) and through the register file, cache hierarchy, LSQ, and buses (on-chip transfer) for the baseline and NDA architectures. NDA can greatly reduce data transfer energy for several reasons. First, most of these benchmarks have large working sets with low temporal locality that do not fit within processor caches. Second, for applications that have temporal locality, CGRA's internal registers allow local storage of data. Finally, NDA uses low-energy TSVs consuming 5× less energy than off-chip interconnects. Overall, NDA-1, -2, and -3 reduce average data transfer energy by 64%, 66%, and 68% over "Base+CGRAs," respectively. NDA also reduces average data transfer energy by 89% over "Base."

Note that "Base 4×Para" decreases on-chip cache (leakage) energy by 75% compared to "Base" due to reduced application execution time. "Base+CGRAs" also decreases average on-chip data transfer energy by 79% over "Base," because CGRA's internal registers can reduce accesses to the processor cache hierarchy [59].

Total energy: NDA improves total energy consumption by exploiting energy-efficient CGRAs near DRAM devices. Figure 12 shows the total energy dissipation breakdown of baseline and NDA. We categorize the energy dissipation into four components. "Execution and Scheduling" comprises energy dissipation of (*i*) fetching and scheduling instructions in the processor and (*ii*) executing instructions in both the processor and CGRAs. "On-chip Transfer" represents energy dissipation of the register file, cache hierarchy, LSQ, and buses. "Off-chip DRAM I/O and TSVs" represents energy dissipation of the memory I/O interface. Finally, "DRAM Devices" represents energy dissipations in DRAM devices.

On average, NDA-1, -2, and -3 consume 34%, 46%, and 39% lower total energy than "Base+CGRAs." NDA also consumes 84% and 78% lower energy than "Base" and "Base 4×Para," respectively. Due to the absence of the overhead of fetching and scheduling instructions in the processor for kernel executions, NDA using CGRAs processes data

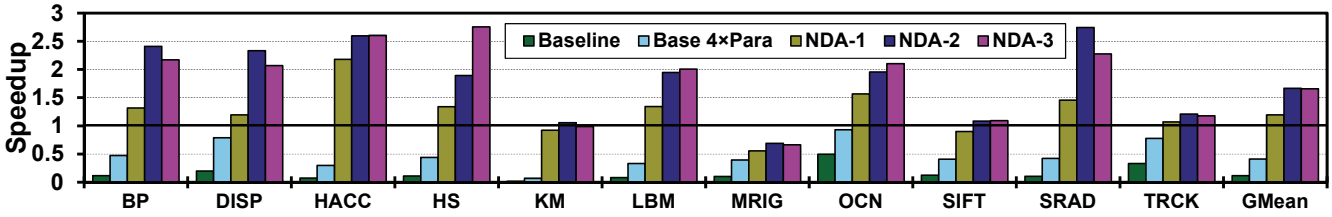


Figure 13. Performance comparison of different architectures normalized to "Base+CGRAs."

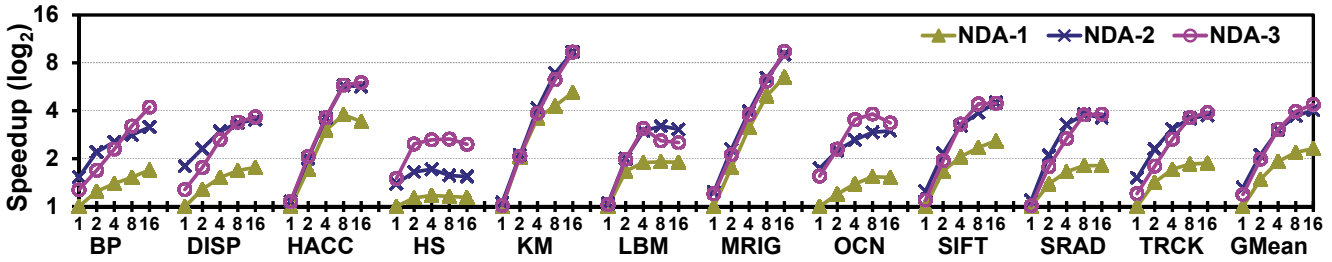


Figure 14. Performance of different NDA microarchitectures to connect CGRAs and DRAMs with varying the number of stacked CGRAs (*x* axis) normalized to the performance of NDA-1 with one CGRA stacked atop a DRAM.

much more energy efficiently than processors. Moreover, NDA substantially reduces data transfers through cache hierarchy, register file, and bus. NDA also reduces leakage and background energy of the processor and DRAM devices, respectively, because CGRAs speed up application execution. Finally, NDA considerably decreases DRAM device energy because CGRAs generally have more well-defined and regular memory access patterns compared to processors [60, 61]. This leads to higher DRAM page (row buffer) hit rate and thus lower DRAM activate energy. Only for MRIG and SIFT, NDA incurrs more DRAM accesses (i.e., more DRAM access and activation energy) than "Base+CGRAs." Higher DRAM energy in MRIG is due to its non-uniform data distribution and gather memory operations and higher DRAM energy in SIFT is due to its temporal data accesses that cannot be fully filtered out by CGRA's internal registers. For such applications, the processor's on-chip cache hierarchy can reduce DRAM accesses.

Speedup Comparison: NDA also improves the performance of target applications by using parallel and accelerated processing. Figure 13 compares the performance of NDA with baseline architectures. The speedup numbers include execution time of both software and accelerated kernel(s). On average, NDA-2 and NDA-3 provide 14.1× and 4.0× higher performance than "Base" and "Base 4×Para." High speedups originate from (*i*) concurrent execution of CGRAs, (*ii*) data-level parallelism exploited by CGRAs, and (*iii*) slightly lower memory access latency, and 2× and 8× higher memory bandwidth for NDA-2 and -3, respectively.

NDA-1 provides the same aggregated memory bandwidth as "Base" and "Base+CGRAs" for CGRAs. However, NDA-1 provides 10.1× higher average speedups than "Base" because its CGRAs exploit data parallelism. NDA-1 provides 19% higher average speedup than "Base+CGRAs" although "Base+CGRAs" also uses CGRAs to exploit the same data parallelism as NDA. The reason is that parallel execution of CGRAs integrated with the processor causes more memory contentions, leading to higher DRAM bank conflicts and lower page hit rate. In NDA, on the other hand, CGRAs' DRAM accesses are local to their DRAM devices. Finally, NDA also benefits from transferring data through 3D-interconnects with 2ns lower memory access latency (and 2× and 8× higher bandwidth for NDA-2 and -3) and bypassing data transfers across the processor cache hierarchy. NDA-2 and NDA-3 increase average performance by 67% and 66% over "Base+CGRAs," respectively, due to higher DRAM bandwidth. Among the tested benchmarks, MRIG is the only one that does not take advantage of NDA, where NDA-2 performs 31% worse than "Base+CGRA" because MRIG exhibits notable temporal data locality and benefits from the processor on-chip cache hierarchy.

Figure 13 indicates that NDA-3 performs almost the same as NDA-2 although its CGRA-DRAM peak bandwidth is 4× higher. To achieve peak bandwidth in NDA-3, all 8 DRAM banks should be utilized concurrently, while NDA-2 achieves its peak bandwidth with streaming accesses. Moreover, memory transactions in NDA-3 are half of the width of those in NDA-2, resulting in more transactions in NDA-3 for wide memory accesses. We expect that optimizing memory access patterns of the benchmarks to exploit bank-level parallelism could improve NDA-3 performance.

Sensitivity to CGRA-DRAM connection type: Having different memory bandwidth characteristics, our three NDA microarchitectures lead to different performance results. Figure 14 compares the performance improvement of NDA-2 and -3 over NDA-1 with one CGRA stacked atop a DRAM device. NDA-1 provides the lowest bandwidth among our three microarchitectures. To evaluate the benefit of higher CGRA-DRAM bandwidth, we vary the number of CGRAs stacked atop each DRAM device (*x* axis). Figure 14 includes only accelerated kernel execution since CGRA-DRAM connection type has no impact on the processor performance. Figure 14 indicates that the performance of some benchmarks such as KM scales well with the number of CGRAs while that of other benchmarks such as HS caps out with more CGRAs due to memory bandwidth limitations.

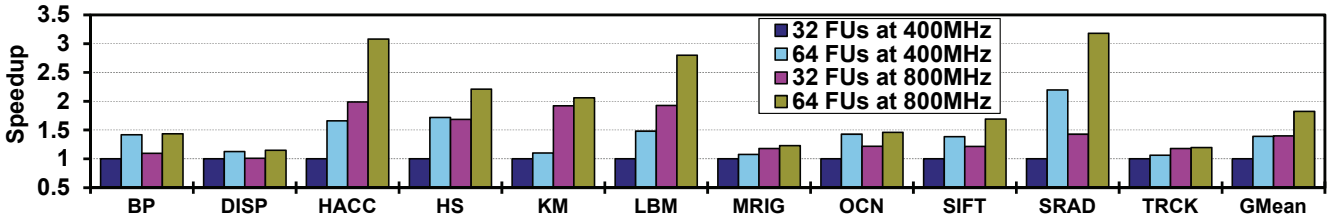


Figure 15. Performance of the NDA-1 architecture by varying the number of functional units and frequency of CGRAs (normalized to the performance of NDA-1 using 32-FU CGRAs at 400MHz). One CGRA is stacked atop each ×8 DRAM.

NDA-2 and NDA-3 achieve 74% and 89% higher average performance than NDA-1, respectively (with 16 CGRAs per DRAM device); the evaluated benchmarks can exploit the higher bandwidth of NDA-2 because they mostly tend to access data in streaming fashion. NDA-3 provides the highest bandwidth and thus most benchmarks can achieve their highest performance when the number of CGRAs is 16. Note that increasing the number of CGRAs to 16 slightly decreases the performance of HS, LBM, and OCN. This is because more CGRAs for these benchmarks increase memory contentions (i.e., DRAM bank conflicts). We expect that optimizing the benchmarks to exploit bank-level parallelism can lead to significant performance improvements of NDA-3.

Sensitivity to CGRA computing capability: We evaluate the impact of the number of FUs and frequency of CGRAs on the accelerated kernel execution time. Figure 15 compares the performance of NDA-1 using CGRAs with various computing capability. For this evaluation, note that one CGRA is stacked atop each DRAM device. For data-intensive benchmarks such as DISP and TRCK, the computing capability of CGRAs has small impact on performance, while for compute-intensive benchmarks such as HACC, LBM, and SRAD, the more CGRA's computing capability can significantly improve performance. Some benchmarks such as KM benefit more from higher frequency CGRAs, while benchmarks such as SRAD and SIFT benefit more from more FUs. On average, a 64-FU CGRA at 800MHz achieves 82% higher performance than a 32-FU CGRA at 400MHz on our benchmarks.

NDA×8 versus ×16: As explained in Section 3.1, our proposed NDA architecture is also applicable to DRAM ×16 devices. There are eight and four DRAM ×8 and DRAM ×16 devices in a given rank. Thus, to have the same computing capability, NDA×16 requires twice the number of CGRAs per DRAM device compared to NDA×8. Our experiments demonstrate that NDA×16 performance is on average 1.7% higher than NDA×8 for our evaluated benchmarks. The row-buffer size of each DRAM×16 device is twice that in DRAM×8 devices. As a result, NDA×16 has higher performance for streaming benchmarks (e.g., 23% and 9% better performance for DISP and TRCK on NDA×16 versus NDA×8). Conversely, concurrent execution of more CGRAs per DRAM device increases bank conflict rate. As a result, HS and SRAD perform 44% and 22% better on NDA×8 versus NDA×16.

7. Related Work

In this section, we briefly discuss related work on reconfigurable accelerated processing and processing in memory.

Spatial reconfigurable accelerators. A large body of work has investigated reconfigurable hardware for exploiting spatial parallelism [62]. Spatial accelerators improve the performance and energy efficiency of kernels in domains such as multimedia, signal processing, cryptography, big data analysis, and pattern matching. The reconfigurable hardware can be used as either tightly-integrated units in processor's datapath, coprocessors, or off-chip accelerators. Some proposed architectures employ a grid of coarse-grain functional units [30, 54–56], while others use finer grain units or FPGA-like hardware [27, 63–65]. Although our work is not limited to specific reconfigurable hardware, we have used a tiled array of coarse-grain reconfigurable functional units because they tend to be more efficient in energy and faster compared to FPGAs [26, 31, 32].

Processors in memory (PIM). There have been numerous efforts to integrate processor logic and DRAM onto a single chip to enable processing in memory [3–9]. PIM systems exploit lower access latency and higher aggregate bandwidth to enable high-performance local data processing. The *Computational RAM* architecture [5] unifies memory and logic by connecting SIMD processing elements to DRAM sense amplifiers through a wide bus. Similarly, *Intelligent RAM* [9] combines a vector processor and DRAM memory on a chip to enable fast data-parallel operations through a wide memory interface. The *FlexRAM* chip [6], which replaces a conventional memory chip, comprises a processor and a large number of simple compute engines that are interleaved with DRAM blocks. *Smart Memories* [7] is a tiled architecture composed of many processing units, caches, and DRAM blocks in a same chip with reconfigurable functionality. In *DIVA* [4], PIM chips are composed of one or more tiles where each contains a processor and DRAM cells. The *Active Pages* model [8] integrates many small pages of data implemented on DRAM and their associated functional units implemented on reconfigurable fabric. The recent *TCAM* [66] and *AC-DIMM* [67] proposals enable in-memory associative computing using PCM and STT-MRAM technologies.

Memory and logic technologies in PIM systems have different characteristics. Logic is optimized for performance, whereas DRAM is optimized for capacity with fewer metal layers to reduce manufacturing cost and improve yield. PIM

systems suffer from high manufacturing costs and low yield because of integrating two different technologies [11, 12]. Moreover, they overhaul the DRAM architecture to efficiently integrate processing units in the memory devices, making their design and verification challenging and time-consuming. In contrast, our proposal is completely achievable using the present-day 3D-stacking technology and does not require significant modifications to DRAMs or their interface to the processor.

Processing in memory controller (MC). Processing in MC has been proposed to reduce data movement through the cache hierarchy by adding specialized hardware to the MC. Ahn *et al.* [68] dedicate hardware to enable parallel execution of a specific class of atomic operations in the MC for data-parallel architectures. Wei *et al.* [69] augment an MC with a co-processor for vector, streaming, and bit manipulation operations. Fang *et al.* [70] propose an *active memory controller* (AMC) that performs scalar and stream operations. In general, MCs augmented with hardware engines still suffer from high off-chip data energy and limited applicability due to supporting a limited class of applications.

The Hybrid Memory Cube (HMC) [71] and 3D-stacking memory and processors [46, 72] promise high memory bandwidth and new opportunities for local data processing. Recently, there have been proposals to bond DRAM layers with application-specific accelerators [18] and GPGPU execution units [17]. Even if our proposal can be exercised in an HMC-like structure or 3D-stacked memory by embodying the processing units in the logic layer, we take a different approach by processing data *near conventional and commodity DRAM devices* to make impact in the near future.

8. Conclusions

We have proposed a new architecture, NDA, to enable accelerated data processing near main memory. NDA concurrently reduces energy and enhances throughput in data movement by stacking coarse-grain reconfigurable accelerators on top of conventional DRAM devices and by off-loading data-intensive operations to the accelerators. Apart from inserting through-silicon vias to 3D-interconnect DRAM devices and CGRAs, NDA requires minimal changes to the underlying DRAM design and no change to processor design, enabling NDA to be more easily adopted in existing and emerging systems for energy reduction and performance improvement. The simulation results show that NDA significantly improves the performance of a wide range of evaluated applications (3.2-60.6×) and reduces their total energy consumption (63-96%).

Acknowledgments

We thank our anonymous reviewers and our shepherd, Babak Falsafi, for their insightful comments. We also thank Young Hoon Son for his help on the DRAM modeling and SPICE simulations. This work was supported in part by the NSF (CCF-0953603, CNS-1217102, and CNS-0952425), and DARPA (HR0011-12-2-0019). Jung Ho Ahn was supported by the Basic Science Research Program through the NRF funded by the MSIP (2012R1A1B4003447). Nam Sung Kim has a financial interest in AMD and Samsung Electronics.

References

[1] S. W. Keckler et al., "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep. 2011.
[2] S. Borkar, "Role of Interconnects in the Future of Computing," *J. Light. Technol.*, vol. 31, no. 24, pp. 3927–3933, Dec. 2013.
[3] M. F. Deering et al., "FBRAM: a new form of memory optimized for 3D graphics," in *Computer Graphics and Interactive Techniques*, 1994, pp. 167–174.
[4] J. Draper et al., "The architecture of the DIVA processing-in-memory chip," in *Intl. Conf. on Supercomputing*, 2002, pp. 14–25.
[5] D. G. Elliott et al., "Computational Ram: A memory-SIMD hybrid and its application To DSP," in *IEEE Custom Integrated Circuits Conference*, 1992, pp. 30.6.1–30.6.4.
[6] D. Keen et al., "FlexRAM: Toward an advanced intelligent memory system," in *Intl. Conf. on Computer Design: VLSI in Computers and Processors*, 1999, pp. 192–201.
[7] K. Mai et al., "Smart Memories: a modular reconfigurable architecture," in *Intl. Symp. on Computer Architecture*, 2000, pp. 161–171.
[8] M. Oskin et al., "Active Pages: A computation model for intelligent memory," in *Intl. Symp. on Computer Architecture*, 1998, pp. 192–203.
[9] D. Patterson et al., "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.
[10] G. H. Loh et al., "A processing in memory taxonomy and a case for studying fixed-function PIM," in *Workshop on Near-Data Processing*, 2013.
[11] D. Patterson et al., "Intelligent RAM (IRAM): the industrial setting, applications, and architectures," in *Intl. Conf. on Computer Design*, 1997, pp. 2–7.
[12] M. Chu et al., "High-level programming model abstractions for processing in memory," in *Workshop on Near-Data Processing*, 2013.
[13] R. G. Dreslinski et al., "Centip3De: A 64-Core, 3D Stacked Near-Threshold System," *IEEE Micro*, vol. 33, no. 2, pp. 8–16, Mar. 2013.
[14] B. Black, "Die stacking is happening (keynote)," in *Intl. Symp. on Microarchitecture*, 2013.
[15] D. H. Kim et al., "3D-MAPS: 3D massively parallel processor with stacked memory," in *IEEE Intl. Solid-State Circuits Conference*, 2012, pp. 188–190.
[16] D. H. Woo et al., "POD: A 3D-Integrated Broad-Purpose Acceleration Layer," *IEEE Micro*, vol. 28, no. 4, pp. 28–40, Jul. 2008.
[17] D. P. Zhang et al., "TOP-PIM: throughput-oriented programmable processing in memory," in *Intl. Symp. on High Performance Parallel and Distributed Computing*, 2014.
[18] Q. Zhu et al., "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *IEEE Intl. 3D Systems Integration Conf.*, 2013, pp. 1–7.
[19] R. Sampson et al., "Sonic Millip3De: A massively parallel 3D-stacked accelerator for 3D ultrasound," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 318–329.
[20] A. Farmahini-Farahani et al., "DRAMA: An Architecture for Accelerated Processing near Memory," *IEEE Comput. Archit. Lett.*, 2014.
[21] C. Park et al., "A 512-Mb DDR3 SDRAM Prototype With CIO Minimization and Self-Calibration Techniques," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 831–838, Apr. 2006.
[22] Y. H. Son et al., "Reducing memory access latency with asymmetric DRAM bank organizations," in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 380–391.
[23] V. Govindaraju et al., "DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep. 2012.

[24] T. Nowatzki et al., "A general constraint-centric scheduling framework for spatial architectures," in *Conf. on Programming Language Design and Implementation*, 2013, pp. 495–506.

[25] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[26] Y. Huang et al., "Elastic CGRAs," in *Intl. Symp. on Field Programmable Gate Arrays*, 2013, pp. 171–180.

[27] M. Mishra et al., "Tartan: evaluating spatial computation for whole program execution," in *Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 163–174.

[28] V. Govindaraju et al., "Dynamically specialized datapaths for energy efficient computing," in *High Performance Computer Architecture*, 2011, pp. 503–514.

[29] D. Voitsechov and Y. Etsion, "Single-graph multiple flows: Energy efficient design alternative for GPGPUs," in *Intl. Symp. on Computer Architecture*, 2014, pp. 205–216.

[30] A. Parashar et al., "Triggered instructions: a control paradigm for spatially-programmed architectures," in *Intl. Symp. on Computer Architecture*, 2013, pp. 142–153.

[31] J. L. Tripp et al., "A Survey of Multi-Core Coarse-Grained Reconfigurable Arrays for Embedded Applications," 2008.

[32] R. Hartenstein, "Coarse grain reconfigurable architecture," in *Asia South Pacific Design Automation Conference*, 2001, pp. 564–570.

[33] C. Kim et al., "ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications," in *Intl. Conf. on Field-Programmable Technology*, 2012, pp. 329–334.

[34] W.-J. Lee et al., "A scalable GPU architecture based on dynamically reconfigurable embedded processor," in *High Performance Graphics, Posters*, 2011.

[35] N. R. Miniskar et al., "Function inlining and loop unrolling for loop acceleration in reconfigurable processors," in *Intl. Conf. Compilers, architectures and synthesis for embedded systems*, 2012, pp. 101–110.

[36] J.-S. Kim et al., "A 1.2 V 12.8 GB/s 2 Gb Mobile Wide-I/O DRAM With 4x128 I/Os Using TSV Based Stacking," *IEEE J. Solid-State Circuits*, vol. 47, no. 1, pp. 107–116, Jan. 2012.

[37] T. Sekiguchi et al., "1-Tbyte/s 1-Gbit DRAM Architecture Using 3-D Interconnect for High-Throughput Computing," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 828–837, Apr. 2011.

[38] Uksong Kang et al., "8Gb 3D DDR3 DRAM using through-silicon-via technology," in *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, 2009, pp. 130–131,131a.

[39] K.-N. Lim et al., "A 1.2V 23nm 6F2 4Gb DDR3 SDRAM with local-bitline sense amplifier, hybrid LIO sense amplifier and dummy-less array architecture," in *IEEE Intl. Solid-State Circuits Conference*, 2012, pp. 42–44.

[40] K. Sohn et al., "A 1.2V 30nm 3.2Gb/s/pin 4Gb DDR4 SDRAM with dual-error detection and PVT-tolerant data-fetch scheme," in *IEEE Intl. Solid-State Circuits Conference*, 2012, pp. 38–40.

[41] M. Shevgoor et al., "Quantifying the relationship between the power delivery network and architectural policies in a 3D-stacked memory device," in *Intl. Symp. on Microarchitecture*, 2013, pp. 198–209.

[42] A. R. Lebeck et al., "Power aware page allocation," in *Intl. Conf. on Architectural Support for programming Languages and Operating Systems*, 2000, vol. 28, no. 5, pp. 105–116.

[43] J. Mukundan et al., "Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems," in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 48–59.

[44] A. Basu et al., "Efficient virtual memory for big memory servers," in *Intl. Symp. on Computer Architecture*, 2013, pp. 237–248.

[45] D. Y. Deng et al., "Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric," in *Intl. Symp. on Microarchitecture*, 2010, pp. 137–148.

[46] S. Pugsley et al., "Comparing Different Implementations of Near Data Computing with In-Memory MapReduce Workloads," *IEEE Micro*, vol. 34, no. 4, pp. 44–52, 2014.

[47] J. Ekanayake et al., "MapReduce for Data Intensive Scientific Analyses," in *IEEE Intl. Conf. on eScience*, 2008, pp. 277–284.

[48] S. K. Venkata et al., "SD-VBS: The San Diego Vision Benchmark Suite," in *International Symposium on Workload Characterization*, 2009, pp. 55–64.

[49] J. Stratton et al., "Parboil: A revised benchmark suite for scientific and commercial throughput computing," 2012.

[50] "CORAL benchmark codes," 2014. [Online]. Available: https://asc.llnl.gov/CORAL-benchmarks/.

[51] S. C. Woo et al., "The SPLASH-2 programs: characterization and methodological considerations," in *Intl. Symp. on Computer Architecture*, 1995, pp. 24–36.

[52] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Intl. Symp. on Workload Characterization*, 2009, pp. 44–54.

[53] S. Li et al., "The McPAT Framework for Multicore and Manycore Architectures," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 1, pp. 1–29, Apr. 2013.

[54] B. Mei et al., "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *Field Programmable Logic and Application*, 2003, pp. 61–70.

[55] H. Schmit et al., "PipeRench: A virtualized programmable datapath in 0.18 micron technology," in *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference (Cat. No.02CH37285)*, 2002, pp. 63–66.

[56] M. A. Watkins and D. H. Albonesi, "ReMAP: A Reconfigurable Heterogeneous Multicore Architecture," in *Intl. Symp. on Microarchitecture*, 2010, pp. 497–508.

[57] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Arch. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[58] K. Chandrasekar et al., "DRAMPower: Open-source DRAM power & energy estimation tool." [Online]. Available: http://www.drampower.info.

[59] R. Hameed et al., "Understanding sources of inefficiency in general-purpose chips," in *Intl. Symp. on Computer Architecture*, 2010, pp. 37–47.

[60] R. Hou et al., "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *High Performance Computer Architecture*, 2011, pp. 312–320.

[61] A. Farmahini-Farahani et al., "Energy-Efficient Reconfigurable Cache Architectures for Accelerator-Enabled Embedded Systems," in *Intl. Symp. on Performance Analysis of Systems and Software*, 2014, pp. 211–220.

[62] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Design, Automation and Test in Europe*, 2001, pp. 642–649.

[63] Z. A. Ye et al., "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *Intl. Symp. on Computer Architecture*, 2000, pp. 225–235.

[64] J. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Field-Programmable Custom Computing Machines*, 1997, pp. 12–21.

[65] T. Callahan et al., "The Garp architecture and C compiler," *Computer (Long. Beach. Calif).*, vol. 33, no. 4, pp. 62–69, 2000.

[66] Q. Guo et al., "A resistive TCAM accelerator for data-intensive computing," in *Intl. Symp. on Microarchitecture*, 2011, pp. 339–350.

[67] Q. Guo et al., "AC-DIMM: associative computing with STT-MRAM," in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 189–200.

[68] J. Ahn et al., "Scatter-Add in Data Parallel Architectures," in *Intl. Symp. on High-Performance Computer Architecture*, 2005, pp. 132–142.

[69] R. B. T. Mingliang Wei, Marc Snir, Josep Torrellas, "A near-memory processor for vector, streaming and bit manipulation workloads," in *UIUC Tech. Report*, 2005.

[70] Z. Fang et al., "Active memory controller," *J. Supercomput.*, vol. 62, no. 1, pp. 510–549, Jan. 2012.

[71] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips 23*, 2011.

[72] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *Intl. Symp. on Computer Architecture*, 2008, pp. 453–464.

[73] S. O et al., "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *Intl. Symp. on Computer Architecture*, 2014.