Energy-Efficient Data Processing Using Accelerators

By

Amin Farmahini Farahani

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2015

Date of final oral examination: 12/22/2014

The dissertation is approved by the following members of the Final Oral Committee:
Nam Sung Kim, Associate Professor, Electrical and Computer Engineering
Katherine Morrow, Associate Professor, Electrical and Computer Engineering
Mark D. Hill, Gene M. Amdahl Professor and Department Chair, Computer Sciences
Mikko H. Lipasti, Philip Dunham Reed Professor, Electrical and Computer Engineering
Xinyu Zhang, Assistant Professor, Electrical and Computer Engineering

Dedicated to Zari, Mori, and Moien

Acknowledgments

This dissertation would not have been possible without the help of my family, professors, and friends. I would like to thank those who helped me in the past few years.

I owe my gratitude to a long list of people. I would like to thank Dr. Michael Schulte, my former adviser, for his guidance throughout my graduate studies even to this date. I would also like to thank my current advisers, Dr. Katherine Morrow and Dr. Nam Sung Kim, for their instruction and guidance throughout my graduate school career. I cannot express how grateful I am. I would like to thank Dr. David Wood for many things I learned from him. I would also like to thank Dr. Mark Hill, Dr. Mikko Lipasti, and Dr. Xinyu Zhang for being a member of my dissertation committee. Dr. Hill and Dr. Lipasti made time to meet with me and provided valuable feedback on my research and career opportunities.

I would like to express my gratitude to Dr. Wesley Smith, Tom Gorski, and Robert Fobes at the Physics Department for their support and cooperation in the early years of my graduate career.

During my internship at Nvidia and Qualcomm, I was fortunate to work under direction of Greg Palmer, Lacky Shah, Jeff Shabel, and Simon Booth. My experience of working with them greatly helped me develop my research aptitude. I am grateful for their support during and after the internship.

My fellow graduate students have played an important role in assisting me in these years. I would like to express my thanks to Atif Hashmi, Andrew Nere, Syed Zohaib Gilani, Arslan Zulfiqar, David Palframan, Vignyan Reddy, Dibakar Gope, Tony Gregerson, Jie Liu, Vinod Reddy, Jacob Adriaens, Chris Jenkins, Kyle Rupnow, Daniel Chang, Daniel Seemuth, Hadi Asgharimoghaddam, Mohammad Alian, Somayeh Sardashti, Hamid Reza Ghasemi, Aishwarya Nagarajan, Mitch Hayenga, Sean Franey, Paula Aguilera, Hao Wang, Zhenhong Liu, Shreesha

Srinath, Tony Nowatzki, Yuefi Zhao, Srinivasan Narayanamoorthy, Philip Garcia, Felix Loh, Henry Duwe, Charles Tsen, and Jungseob Lee for their support and company.

If I want to thank only one of my friends, it would definitely be Shirzad Malekpour. He has always been here for me in good times and bad. I am also very grateful to Anaram Shahravan for her support. I express my gratitude to my friends Diba Izadyar, Sheida Malekpour, Arsham Shahlari, Arash Sangari, Hasti Mirkia, Omid Forouzan, Elham Ahmadi, Arman Pazouki, Mozhdeh Rajaei, Hasan Tabatabaie, Chiya Saeidi, and Solmaz Forutan. I also owe a debt of gratitude to Shirin Malekpour and Peiman Hematti for helping me when I needed the most.

Finally, I would like to express my deep gratitude to my mother, father, and brother for supporting me. Their encouragement and love were the great power for me to overcome all troubles I faced.

I would like to take this opportunity to raise concern about the place that I worked in for a few years. I believe that the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison is going downhill in several aspects. I have been really proud of working in this department and I really hope that things change for the better. But I unfortunately do not see any good signs as of now. Returning back to the department in a couple of years, I long for a thriving department as it once used to be.

Abstract

Energy efficiency of computing systems has become crucial with the end of Dennardian scaling in which voltage scaling has stalled, thereby increasing power density with decreasing transistor size. One method to improve energy efficiency is to use accelerators specialized for a certain set of computing problems. Unlike traditional general-purpose processors, such an accelerator avoids the overhead of fetching and scheduling instructions. Supporting reconfigurability for accelerators can provide acceleration for a broader range of applications. This dissertation document investigates architectural techniques to enable energy-efficient data processing using reconfigurable accelerators and explores two aspects of this area: customizing L1 data caches for computing systems integrated with reconfigurable accelerators, and proposing a near-memory data processing architecture to decrease the energy consumption of data transfers between compute-engines and memory.

Data transfers between accelerators and memory are often a bottleneck for both performance and energy efficiency. This dissertation document demonstrates the potential of a configurable L1 data cache to exploit diversity in cache requirements across hybrid applications that use hardware accelerators. Configurable caches improve energy efficiency while maintaining high performance. One configurable feature is the cache topology; it can be reconfigured as a set of private L1 caches, or a single L1 cache shared by a processor and an accelerator. This dissertation also proposes a technique for L1 caches to provide a configurable tradeoff between the number of cache ports and its capacity.

To further reduce the overhead of transferring data between compute-engines and memory, this dissertation proposes NDA (Near-DRAM Acceleration), an architecture that stacks low-power, flexible reconfigurable accelerators atop off-chip commodity DRAM devices. The accelerators can thus access data directly near DRAM, without the need to bring data from off-

chip DRAM devices through the memory hierarchy of the host processor. To make this architecture attractive to industry and practical in the short run, we take commodity 2D DRAM devices and make minimal changes to DRAM devices to provide, in a practical way, high-bandwidth connections between accelerators and DRAM devices for the purpose of near-memory processing. The proposed NDA architecture requires no change to the host processor design and minimal changes to the commodity DRAM device while maintaining the compatibility with the standard DRAM interface and DIMM architecture. This dissertation explores three NDA microarchitectures to stack accelerators atop DRAM devices and analyzes the impact of supporting such NDA microarchitectures on DRAM area, timing, and energy in detail. The first NDA microarchitecture connects accelerators and DRAM through global I/O lines (inter-bank datalines) that are shared between all DRAM banks. In the second NDA microarchitecture, global I/O lines are doubled to increase the internal bandwidth between accelerators and DRAM. The third microarchitecture connects accelerators and DRAM through global datalines that are private to each DRAM bank, substantially increasing DRAM internal bandwidth. This dissertation also identifies various software and hardware challenges in implementing the proposed NDA architecture (e.g., processor-accelerator and DRAM-accelerator communications due to sharing the main memory system with the host processor) and provides cost-effective solutions.

Contents

List of Tables

List of Figures

1 Introduction

As transistors continue to scale with each new process technology, power density of a chip increases since operating voltage remains roughly constant. This results in higher energy consumption with the continued increase in number of chip transistors. As a result, energy consumption has become a major target for computer architecture researchers to tackle. Specialization of computing structures for required operations allows reduction in energy consumption and performance improvement by more efficiently using silicon area. This chapter motivates using specialized hardware accelerators to increase energy efficiency. It also introduces techniques to improve energy efficiency of data communication in accelerated systems and presents an architecture that uses accelerators for data processing near memory.

1.1 Motivation

Energy efficiency concerns have driven a shift from traditional computer architecture designs to heterogeneous design techniques, such as using specialized domain-specific or application-specific accelerators. Accelerators can reduce the energy penalty of instruction-based execution by implementing computations as circuits rather than programs stored in memory that must be retrieved instruction by instruction in order to be executed. The performance and energy efficiency benefits of specialized accelerators over general-purpose processors allow a designer to considerably improve system energy efficiency by trading the relatively cheap commodity of area for the expensive commodity of energy [1], [2].

Unlike ASIC cores that accelerate only a single specific computation (or a very small set of them), Reconfigurable accelerators are flexible structures can be specialized post-fabrication [3]

to accelerate a wide variety of computations by implementing different accelerator circuits at different times. Reconfigurable accelerators perform compute-intensive operations more quickly with less energy than traditional processors [4] by exploiting spatial parallelism in application *kernels* (compute-intensive portions of an application) to implement computations as parallel circuits.

Figure 1 shows the energy dissipation breakdown of an out-of-order processor and DRAM devices over 11 different scientific and multimedia applications (refer to Chapter 3 and Section 5.4.1 for evaluation methodology and benchmark descriptions). We categorize the energy dissipation into five components. The "instruction fetch and control" component comprises energy dissipation of fetching and scheduling instructions in the processor. The "execution" component represents the energy dissipated in processor execution units. The "on-chip transfer" component includes energy dissipation of the register file, cache hierarchy, LSQ, and buses. The "off-chip DRAM I/O" represents energy dissipation of the memory I/O interface. Finally, the "DRAM devices" component represents energy dissipation of memory subsystem. In our tested applications, 17% of the total system energy is spent in the overheads of fetching and scheduling

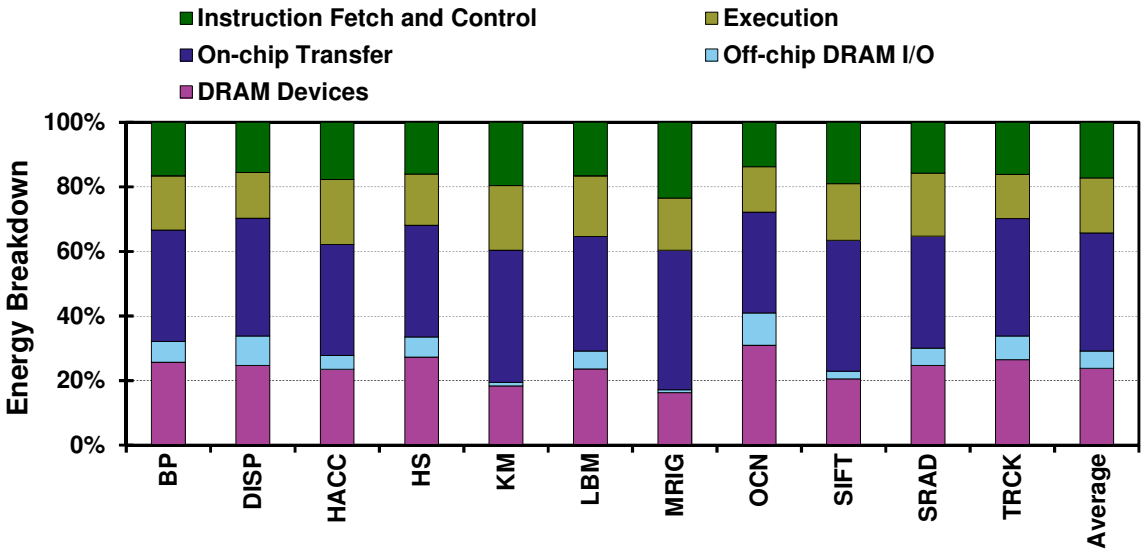


Figure 1. Combined processor and DRAM energy consumption.

instructions. This indicates significant potential for reconfigurable accelerators to reduce energy by flexibly, but efficiently, performing computation tasks without instruction-based execution. In addition, accelerators speed up application execution, reducing system's leakage energy.

However, the performance and energy efficiency benefits of systems with accelerators greatly depend upon how they are integrated into the memory hierarchy [5]. If they use a shared-memory communication paradigm, the data cache architecture has a strong effect on the system's performance and energy efficiency. However, conventional processor cache architectures do not necessarily provide an efficient solution for reconfigurable accelerators because of differences in memory access patterns and bandwidth requirements. Therefore, we propose techniques for improving energy-efficiency of the cache hierarchy in processor-accelerator systems (Section 1.2).

Tailoring the cache design to the needs of systems with reconfigurable accelerators improves energy efficiency considerably, but it still leaves room for improvement. An important source of energy inefficiency in current computing systems originates from moving data between where it is stored and where it is processed. The processor usually must transfer data from off-chip dynamic random access memory (DRAM) devices to its on-chip cache hierarchy and then into its register file before processing it. Energy consumed by this data movement is much higher than that consumed by actual data computation. For example, reading a data word from an L1 cache consumes about an order of magnitude more energy than performing an arithmetic operation on a data word [6]. Figure 1 shows that on average data movement (both on-chip and off-chip) constitutes 42% of total energy. As a result, data movement is the major component of the total system energy.

Furthermore, the contribution of data movement energy to the total system energy is projected to increase even further with technology scaling [7], [8]. Figure 2 compares the rate in

which data computation and data movement energy scales in future process technologies. Figure 2 shows that data computation energy (ALU operations) scales well in future processor technologies, but data movement energy on global wires does not decrease significantly because wire capacitance per distance unit (mm) does not scale as suitably as transistor switching capacitance [7]. Therefore, we expect that data movement energy dominates total system energy in future process technologies. To address challenges associated with ever-increasing data movement energy, we propose architectural innovations to enable local data processing using accelerators (Section 1.3).

This dissertation studies energy reduction and performance improvement opportunities through novel design techniques and heterogeneous architectures that exploit reconfigurable accelerators. To lay the foundation for this research, we begin by examining heterogeneous architectures that couple an embedded processor and a reconfigurable accelerator through a shared-memory cache hierarchy. We explore data cache requirements for these architectures and

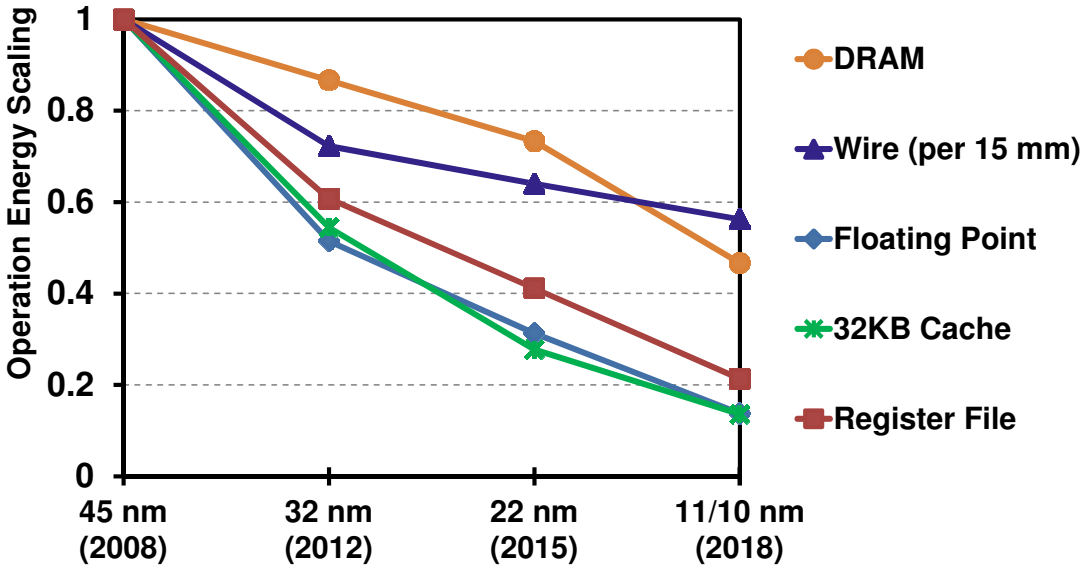


Figure 2. Energy scaling projections of DRAM access and communication, global wire transition, floating point arithmetic operation, L1 cache access, and register file access. Data is extracted from [7], [98], [161], [162]**.**

find cache design parameters that achieve high energy efficiency for different hybrid software/accelerated applications. Guided by this exploration, we investigate configurable cache design techniques that allow the designer to adapt the L1 data cache structure according to cache requirements across hybrid applications. To reduce data movement energy, We then propose a near-memory processing architecture that use reconfigurable accelerators stacked atop of commodity DRAM devices to process data close to where it is stored. We propose microarchitectures to increase the memory bandwidth between DRAM devices and reconfigurable accelerators in our near-memory processing architecture. Finally, we identify various software and hardware challenges in implementing our proposed architecture and provide novel cost-effective solutions.

1.2 Reconfigurable Cache Architectures for Processor-Accelerator Systems

Prior studies have primarily examined the performance of accelerator architectures or specific application implementations on reconfigurable computing platforms. Few efforts have examined the effects of cache design on energy efficiency of systems composed of embedded processors and reconfigurable accelerators implemented on a single chip using ASIC technology. In this dissertation, we revisit cache design decisions made for general-purpose multi-core processors based on the unique needs of a reconfigurable accelerator. Specifically, we study energy-efficient, high-bandwidth cache designs for embedded systems with reconfigurable accelerators.

We first investigate the impact of cache design and organization on energy efficiency of heterogeneous architectures that couple an embedded processor with a reconfigurable accelerator through a shared-memory cache hierarchy on a single chip. We call these architectures *processor-accelerator* architectures. In these architectures, software codes are run on the processor, while

application kernels are implemented in the accelerator. This results in faster computation and/or less energy consumption due to the specialized computational structures in the accelerator.

Accelerators exhibit different memory access pattern and memory access rate when compared to processors. Therefore, cache design requirements of these architectures are different from those of processor-only architectures. First, memory access patterns differ between a processor executing primarily or entirely sequential software code and a reconfigurable accelerator that implements a highly-parallel circuit. Second, the memory access rate of a reconfigurable accelerator is quite different from that of a processor running a functionally-equivalent software application. An accelerator performs potentially many parallel computations, and thus generally accesses memory at a higher rate than a processor. Unlike processors, accelerators also often issue separate bursts of read and write requests which favor high-bandwidth data communication methods to accommodate higher memory access rate [9], [10].

Therefore, reconfigurable accelerator's caches should be carefully designed based on the characteristics of the applications expected to use them. We explore the design space of L1 data caches to find a set of cache parameters that improve energy efficiency of processor-accelerator architectures. Guided by this exploration, we propose modifications to the L1 data cache level in processor-accelerator architectures to maximize energy efficiency of data caches in embedded systems in which processors and accelerators communicate through the cache hierarchy..

We propose a configurable data cache interface that allows the processor and accelerator to either share an L1 data cache or have their own private data caches. In fact, this interface allows having two different L1 data cache organizations in processor-accelerator architectures (acting as a single shared L1 data cache or two private L1 data caches). My goal is to provide an interface that enables a designer to select the organization with better energy efficiency based on the executing hybrid application.

Next, we propose a configurable cache port design that allows a designer to increase the number of cache ports by trading off cache capacity in processor-accelerator architectures. Some hybrid applications benefit from high-bandwidth caches due to reduction in execution time, while the performance of some other applications is not highly improved by multi-port caches. For these applications, using multi-port caches leads to unnecessary higher energy dissipation without noticeable performance improvement. Based on this observation, a configurable cache port design can improve the performance of hybrid applications that benefit from multi-port caches by configuring the cache as a multi-port cache, while it keeps the energy consumption low for other hybrid applications by configuring the cache as a single-port cache (Section 4.5). A configurable cache port design provides a tradeoff between L1 data port count and capacity to better support a variety of accelerated application kernels. To evaluate the configurable port cache design, we compare its performance and energy consumption with the various existing design techniques to implement multi-port caches.

1.3 NDA: Near-DRAM Acceleration

In approach described in the previous section, the accelerator is located near the processor, and acts as a co-processor. Data must be transferred from off-chip memory to on-chip caches before it can be processed by the accelerator. This data movement constitutes a large portion of the total system power consumption. To reduce data transfer (movement) energy, we can decrease data transfer distance by processing data *near* or *in* memory where it is stored. This motivates us to re-examine the previous processors-in-memory (PIM) architectures [11]–[23] that aimed at improving 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, high cost per DRAM bit [24], [25], and high design/verification cost

associated with custom DRAM architectures. Recently, 3D-stacking technology [26]–[30] 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* DRAM devices to reap the performance and energy-efficiency benefits of both accelerators and near-memory processing and demonstrated promising results [31]–[34]. 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 dissertation, we explore a near-memory processing architecture that we call NDA (Near-DRAM Acceleration). In NDA, contrary to prior work, accelerator logic is stacked atop *commodity* "2D" DRAM devices. The goal of NDA is to improve energy efficiency by exploiting local (near-memory) data processing. Commodity DRAM and logic are implemented on separate layers and connected using TSVs. In the NDA architecture, processing elements are physically close to memory devices, avoiding off-chip data communication. Thus, instead of moving data between compute-units and far memory in conventional systems, in this architecture data can be processed near DRAM devices, minimizing the distance of (and therefore energy consumed by) data transfers.

Although, the NDA architecture is not dependent on any specific compute-engines, we use coarse-grain reconfigurable accelerators as compute-engines near memory since they have better performance and energy consumption than SIMD and GPU engines for most parallel workloads

[35]. Low energy is important since near-memory architectures have more stringent power/thermal constraints. In the NDA architecture, reconfigurable accelerators can take advantage of spatial parallelism in applications to speed up a wide range of applications. Furthermore, reconfigurable accelerators can process data much more energy-efficiently compared to processors since they are specialized units.

The NDA architecture does not require major changes to the underlying architecture of commodity DRAM devices apart from additional TSVs. Since even small design changes could affect DRAM's cost-per-bit, DRAM manufactures are not usually amenable to changing the internal architecture of DRAM devices. This is also particularly attractive to DRAM manufacturers looking to provide added value to their products without significant rework or requiring cooperation from large microprocessor companies.

In this dissertation, we discuss hardware challenges to interface reconfigurable accelerators with commodity DRAM devices using TSVs. We propose three microarchitectures to connect reconfigurable accelerators and DRAM devices in the NDA architecture. In the first microarchitecture, TSVs are connected to existing global I/O lines of DRAM. In this microarchitecture, the internal bandwidth between accelerators and DRAM devices is the same as DRAM external (off-chip) data bus. However, the internal DRAM bandwidth can be significantly higher than the off-chip DRAM bandwidth. This higher internal bandwidth can be exploited by a processor-near-memory architecture. Our next two proposed microarchitectures present the opportunity to exploit this higher bandwidth. Higher internal memory bandwidth between accelerators and DRAM devices can improve performance of data-intensive applications mapped to the NDA architecture. In the second microarchitecture, we double global I/O lines and connect TSVs to global I/O lines. In the third microarchitecture, we provide a separate data connection for each DRAM bank and connect TSVs to global datalines. Global datalines, contrary to global I/O

lines, are private to each DRAM bank. We then carefully analyze the impacts of our three microarchitectures on DRAM area, access energy, and timing parameters. For example, we analyze the impact of higher number of TSVs (required to enable higher internal bandwidth) on DRAM area.

We also present software changes required in the applications mapped to NDA to maintain compatibility with the existing DIMM architecture. We analyze performance and energy consumption of our proposed NDA architecture using simulation methods and compare it with conventional computing architectures. Our results show that the NDA architecture improves system performance significantly while also decreasing overall energy consumption.

1.4 Contributions

This dissertation aims to address the growing energy concerns of computing systems by investigating and proposing novel heterogeneous architectures that use reconfigurable accelerators for data processing. Although reconfigurable accelerators traditionally suffered from high area overhead and long reconfiguration time, transistor scaling and scheduling techniques have paved the way for their use in future computing systems. To enable energy-efficient data processing using accelerators, the research presented in this dissertation makes the following contributions:

Processor-accelerator architectures

- **Exploring the design space of L1 data caches for processor-accelerator architectures.** We evaluated the impact of cache design parameters such as capacity, number of ports, associativity, and private vs. shared organization on performance and energy consumption of different embedded applications. This investigation provides insights into the cache design requirements of the tested applications.

- **Proposing a configurable cache interface that allows the accelerator to either share the processor's L1 data cache or to use its own private L1 data cache.** The system can thus switch between two different organizations of the cache hierarchy based on the characteristics of the executing application to provide the best performance and/or energy efficiency.
- **Proposing an L1 data cache design with a configurable tradeoff between capacity and port count to tune cache bandwidth based on the currently-running application.** We examined the impact of this cache design on the performance and energy efficiency of embedded applications and compared it with existing design techniques such as true multi-port and multi-bank caches.

Processing-near-memory architectures

- **Proposing a near-memory processing architecture that we call NDA.** This architecture exploits 3D-stacking technology to enable energy-efficient local data processing by coupling reconfigurable accelerators on a logic layer and DRAM devices on a separate memory layer. The 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.
- **Proposing three NDA microarchitectures that can provide diverse DRAM-accelerator bandwidth.** We analyze the impact of supporting such microarchitectures on DRAM area, timing, and energy in detail.
- **Identifying software and hardware challenges and proposing solutions.** We study different 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.

1.5 Dissertation Organization

This dissertation is organized as follows: Chapter 2 provides a detailed overview of background and related work on configurable caches and processing-in-memory architectures. Chapter 3 discusses the experimental framework, including the simulation infrastructure and benchmarks development methodology. Chapter 4 introduces our configurable cache designs for the L1 cache in the processor-accelerator systems. Our contributions presented in this chapter have been published in the proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) in 2014. Chapter 5 introduces our new near-memory processing architecture and evaluates its performance and energy consumption. Our contributions presented in this chapter have been published in the IEEE Computer Architecture Letters (CAL) in 2014 and the proceedings of IEEE Symposium on High Performance Computer Architecture in 2015. Finally, Chapter 6 concludes the dissertation and summarizes key observations.

2 Background and Related Work

This chapter explores related work and background material that is central to this dissertation. Section 2.1 introduces reconfigurable accelerators and summarizes methods to integrate these accelerators into computing systems. Section 2.2 surveys previous work on high-bandwidth and configurable caches. Section 2.3 provides an overview of DRAM devices and covers previous architectures to enable in-memory computing.

2.1 Reconfigurable Accelerators

This section describes the architecture of coarse-grain reconfigurable accelerators and methods employed for transferring data between these accelerators and a system's memory hierarchy.

2.1.1 Coarse-grain Reconfigurable Accelerators (CGRAs)

A large body of work has investigated reconfigurable hardware for exploiting spatial parallelism in applications [36]. Some systems employ coarse-grained functional units designed to implement dataflow graphs [35], [37]–[41], while others use finer-grained units or FPGA-like hardware [42]–[45]. Coarse-grain reconfigurable accelerators (CGRAs) are composed of a grid of function units (FUs), which are also called processing elements (PEs). FUs operate on word-sized data, and can perform arithmetic and logic operations. A reconfigurable interconnect provides the capability for inter-FU communication as well as communication into and out of the accelerator. 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 [46].

CGRAs, such as those used in DySER [40] and SGMF [47], considerably improve performance and energy consumption compared to conventional processors by exploiting spatial data parallelism in application kernels and efficiently processing kernel's dataflow graphs [40], [43], [47]–[49]. In particular, CGRAs can practically eliminate the large energy overheads of fetching and scheduling instructions in conventional out-of-order processors. With recent advances in their compilers and architectures [50]–[54], CGRAs are gaining momentum in various applications and are integrated in several architectures [35], [37]–[41]. CGRAs improve the performance and energy-efficiency of many applications with high degrees of parallelism such as multimedia, signal processing, cryptography, scientific data analysis, face detection, voice recognition, and pattern matching [36], [55].

Compared to fine-grain reconfigurable accelerators (e.g., FPGAs), CGRAs are less flexible because of performing word-level operations (instead of working at a bit-level granularity). However, for the same reasons, CGRAs result in higher performance, lower energy consumption, and much smaller configuration data (which allows for a shorter configuration time) [49], [56]–[59].

Because this dissertation is not focused on the detailed low-level accelerator architecture, but rather its interaction with the rest of the system and the memory hierarchy in particular, we use an existing reconfigurable accelerator design rather than create a new one. We model a coarse-grained accelerator similar to the DySER architecture [40] with a heterogeneous grid of 32-bit functional units connected by a configurable routing fabric (Figure 3). Most functional units in this architecture can perform addition, subtraction, and a few logical operations, while a few functional units can perform complex operations such as multiplication. Like most CGRAs, the architecture also includes small local storage structures to keep intermediate data near the functional units that use them. This prevents CGRAs from polluting the cache (or memory) with data exhibiting low temporal locality that is only accessed once, and helps decrease energy consumption, particularly in streaming applications [4]. 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" [3]. 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

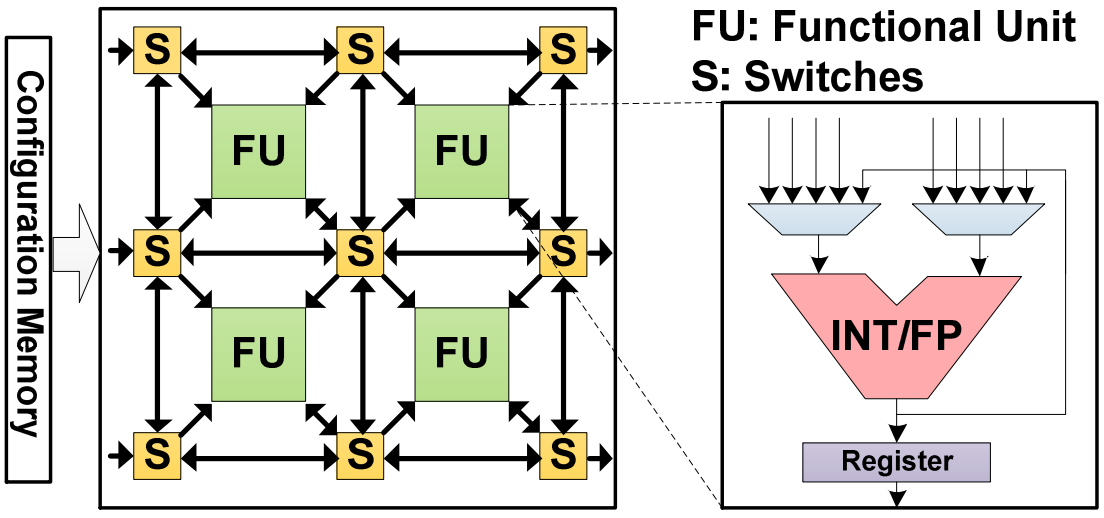


Figure 3. A CGRA with a grid of 2×2 functional units.

the CGRA. At configuration time, The CGRA reads the configuration data from the configuration memory (or main memory) to reprogram its functional units and switches. Although the experiments presented in this work assume the accelerators are CGRAs, the proposed techniques would also apply to other highly-parallel accelerator structures.

2.1.2 Interfaces between Reconfigurable Accelerators and Processors

This section briefly discusses different major architectural methods used to transfer data between a reconfigurable accelerator and the memory hierarchy in a heterogeneous system.

Local Buffer: In some cases, an accelerator is not integrated into the system's memory hierarchy; instead, a local buffer provides storage space for accelerator inputs and outputs, which are filled/fetched by the processor. The accelerator has no independent access to the processor data cache and main memory. The processor loads a batch of data into the accelerator's local buffer for the accelerator to consume. Once the accelerator finishes computation and writes its results back into the buffer, the processor copies those results to the memory hierarchy. The programmer must explicitly perform these data transfers. Many reconfigurable systems use this mechanism due to its architectural simplicity [60], [61].

Shared-Memory Cache Hierarchy: Instead of communicating through a separate local buffer, an accelerator may instead be integrated into a shared memory hierarchy [62], [63]. Direct access to the cache hierarchy helps accelerators load their required data on demand without any help from the processors.

There are several ways to organize the cache hierarchy for these architectures. The accelerator and processor may have private data caches, with shared input data loaded into each, increasing data duplication. The FARM prototype [64] coherently connects an FPGA board with private caches to two AMD boards through HyperTransport links. In the Many-cache memory

architecture [65], an FPGA-based accelerator uses multiple, multi-bank private caches; each cache targets a specific type of data or memory region.

Alternatively, one or more cache levels may be shared between the processor and accelerator. Sharing a data cache can greatly reduce communication traffic. For instance, a shared L2 cache was shown to provide higher performance and more energy-efficiency than a private L2 cache for applications with a great deal of interleaved software and hardware execution [5], [62]. Sharing the L1 could, however, improve performance of both accelerated and software-only execution if they share a working set that fits in the shared L1 cache.

In the Garp architecture [44], [45], an accelerator and a single processor share an L1 data cache. While the accelerator is active, it controls the memory hierarchy buses to load or store data from/to memory and has access to the same memory system as the processor. In the Tartan architecture [43], a special bus allows direct communication between a processor's register file and an accelerator, and they also share an L1 data cache. Choi *et al.* [66] investigates performance and area of multi-port caches in a system where a soft processor shares an L1 cache with several accelerators on an FPGA.

In this dissertation, we however investigate performance and energy efficiency of the cache hierarchy in a system where the whole system including the processor, caches, and coarse-grained accelerator is implemented using ASIC technology, thus having different requirements and characteristics.

2.2 Caches

Accelerators process data at a higher rate and therefore issue more requests to memory per time unit compared to processors. Therefore, cache bandwidth is critical for the performance of accelerators that communicate through a system's cache hierarchy. This section briefly presents

existing techniques to implement high-bandwidth caches and configurable caches that allow the cache design to be customized to application needs at runtime.

2.2.1 High-Bandwidth Caches

High cache bandwidth is necessary in systems where accelerators are connected to the cache hierarchy, in order to supply accelerators with data at the increased rate required by their increased computation capabilities. There are several types of multi-port cache designs that increase cache bandwidth [67]–[69]:

- *True (ideal) multi-porting*: all *N* cache ports operate independently, and *N* different addresses can be accessed simultaneously each cycle. Because true multi-porting incurs high area/power/delay costs, this approach is not feasible for large caches. Each extra port adds transistors to the SRAM cells and enlarges auxiliary circuits such as decoders and input/output circuits. Hence, it increases cache area, energy per access, leakage power, and hit time. Yet, if high bandwidth is needed, the increase in dynamic and leakage power may be offset by shorter execution time and reduced overall energy.
- *Time division multiplexing (virtual multi-porting or multi-pumping)*: the cache runs *N* times faster than the processor, providing the appearance of *N* ports. This technique does not scale to large port counts because of clock speed limits.
- *Cache replication*: a single-port cache is replicated *N* times, providing *N* read ports, but one write port is broadcast to the replicated caches to maintain coherence. Thus, cache area increases linearly with the number of read ports. Stores are costly in terms of energy consumption due to the broadcast.
- *Cache interleaving (multi-bank caches)*: the cache contents are split across *N* independently-addressed banks, allowing up to *N* simultaneous requests (at most one per bank). Requests to the same bank suffer from bank conflicts. Increasing the number of banks improves cache

access parallelism and cache access time (smaller banks), but increases area and wire delay of the arbitration and bank interconnection circuitry. This increases cache area and delay, limiting the feasible number of banks. In banked multi-ported caches, data can be split across the banks in multiple ways. Among them are *line interleaving* and *word interleaving*. Line-interleaving partitions the address space across multiple banks using a cache line granularity (Figure 4a). In word-interleaved multi-bank caches, the words within a cache line are distributed across multiple banks (Figure 4b). Requests to sequential words in word-interleave banked caches are served by different banks, reducing bank conflicts (and thus increasing performance) for sequential accesses as compared to line-interleave caches. However, word-interleave caches require multi-port tags or replicated tags to serve parallel requests to the same cache line [68]. The line tag must be replicated as many times as the number of banks, or the tag array must have as many ports as the number of banks. For example, in Figure 4b, tags for Banks 0 and 1 are duplicates. Note that there could be more than two words per cache line depending on the number of words per cache line and the number of banks.

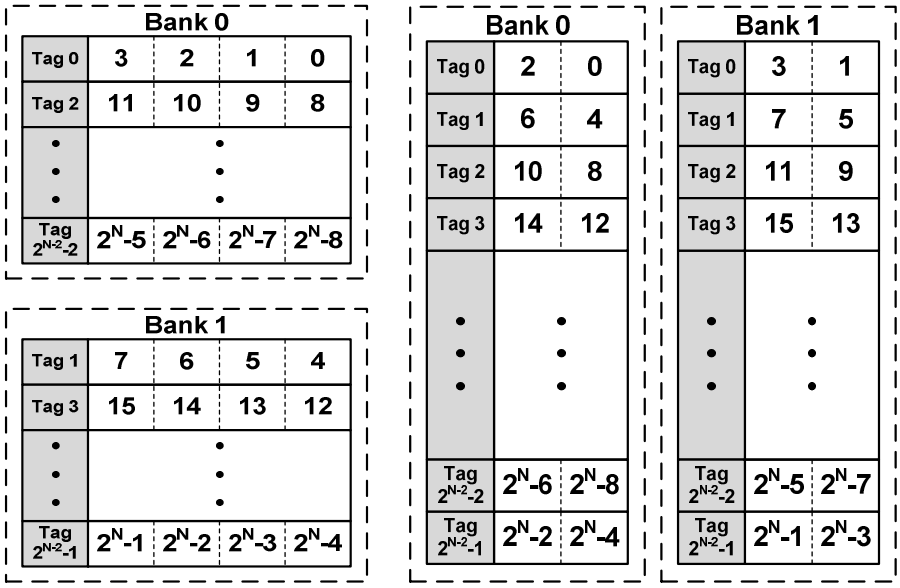


Figure 4. Interleaving schemes in dual-bank caches. (a) line interleaving (b) word interleaving.

2.2.2 Configurable Cache Architectures

Configurable caches provide the opportunity to tune cache parameters based on the requirements of the executing application. Much research work has studied configurable caches. This section mainly focuses on studies that are fundamental to this research.

Selective-ways [70] is a reconfigurable method that provides the ability to vary the number of cache ways in a set associative cache. The key goal is to reduce the dynamic energy of the data array in a set associative L1 cache by reducing the number of active ways. This method provides a flexible tradeoff between performance and energy. When performance degradation is small or tolerable, a subset of ways can be disabled. When high performance is required for cache-intensive applications, more cache ways can be enabled. Frequent change to the number active ways may incur high performance and energy overheads. Therefore, the number of active ways changes only prior to application's execution at context switch time (*i.e.*, process granularity). There are three approaches to handle data in a disabled way. That data can be (1) flushed, (2) only

accessible for coherence request, or (3) transferred to an enabled way. Frequent data flushing of disabled ways may impose substantial performance and energy overhead. The other approach is to make tag and data arrays of all ways accessible for coherence requests. Finally, data in a disabled way can be transferred to an enabled way when there is a hit to the disabled way by adding some logic to the cache datapath.

The *way-concatenation* method [71] provides the ability to vary the associativity of cache while still having the full cache capacity. By modifying the address decode logic, the method enables trading off fewer ways for more sets. The goal is to reduce the cache access energy by lowering associativity. The *V-way* method [72] provides the ability to vary the associativity of an L2 cache on a per-set basis. The method is based on the observation that applications do not have a uniform distribution of memory accesses across cache sets. This method reduces the miss rate by independently varying the associativity of sets based on the running application.

Selective-sets [73] provides the ability to vary the number of cache sets by enabling/disabling a subset of cache arrays. The goal is to reduce the energy consumption by resizing cache and power gating the disabled arrays [74]. Since resizing changes the cache look-up function (*i.e.*, set-mapping for cache lines), this method requires to store all the tags bits for the smallest possible cache size which impose area overheads. Moreover, dirty cache lines of disabled arrays should be written back to the lower level cache and all lines (clean or dirty) in cache arrays in which their set-mappings change after enabling the cache arrays should be flushed.

The hybrid *selective-sets-and-ways* method [75] combines both selective-ways and selective-sets to provide higher energy efficiency by allowing a wider range of possible cache sizes. Both selective-ways and selective-sets exploit the cache subarray technique in which a cache is divided into multiple subarrays to reduce both cache access latency and operation energy. This hybrid

method uses the subarray techniques to better match the required number of sets and ways to the application requirements for energy efficiency.

Ranganathan *et al.* [76] propose a reconfigurable cache technique to use unnecessary cache sub-arrays for different processor activities. The motivation is that some workloads do not use large caches and therefore parts of a cache (unnecessary cache sub-arrays) can be devoted to other activities such as lookup tables and prefetched data storage.

Some studies such as *MorphCache* [77] and *configurable cache hierarchy* [78] attempt to reconfigure a multi-level cache topology in a multiprocessor system. The motivation is that a fixed cache topology does not provide an optimal performance for all workload combinations running on multiple cores. Some other studies such as *Amoeba-Cache* [79] and c*onfigurable line size* [80] vary the cache line size to improve performance and energy efficiency based on the application requirements.

2.3 Memory

This section provides of an overview of commodity memory systems and discusses previous processor-in-memory proposals.

2.3.1 Dynamic Random-Access Memory (DRAM)

DRAM devices are generally used as main memory in many different systems from smartphones to servers. Each bit of data is stored in a memory cell that contains only a capacitor and a transistor, allowing DRAM devices to have much higher densities than SRAM, which usually use six transistors per bit. Due to the gradual electrical charge leakage, DRAM cells are periodically refreshed with a constant time interval (usually 64ms) to guarantee data integrity.

The main memory system is comprised of one or multiple memory channels that work independently. Each memory channel has one or multiple Dual Inline Memory Modules

(DIMMs), which are circuit boards that hold a number of DRAM devices. DRAM devices in a DIMM can be grouped into ranks. All DRAM devices in a given rank work together in lockstep to provide data; thus all devices in a rank collectively service the same memory request. Each DRAM device is made up of multiple banks (each composed of arrays of memory cells, a row buffer, and row and column decoders) that can be accessed independently. DRAM devices allow concurrent bank operations to improve bandwidth. The most recently accessed row of each bank is stored in its associated row buffer. Only a subset of a row (called a column) is accessed in each memory transaction. Address bits determine the appropriate channel and rank in the main memory, the bank within the rank, and row and column in the bank. Command bits indicate the operation the device should perform. The *Activate* command brings an entire row of data from a particular bank in the DRAM arrays into the bank row buffer and activate (open) the row buffer. The *Read* command reads a column of data from the row buffer and places the data in the data I/O bus. Likewise, The *Write* command writes a column of data from the data I/O bus into the row buffer. The *Precharge* command restores data from the row buffer into DRAM cells and deactivates (close) the row buffer. The *Refresh* command activates and restores (precharges) a specific row in all banks [81].

2.3.2 Processor-in-Memory (PIM) Systems

Processor-in-memory (PIM) technology merges logic devices and memory cells onto the same silicon die to process data in memory. This improves performance by exploiting higher memory bandwidth and increases energy efficiency by exploiting local data processing. The PIM systems address the “memory wall” problem through faster processor-memory accesses and can benefit from lower memory access latency and significantly higher memory bandwidth compared to conventional systems, thereby enabling high-performance data processing. Moreover, in a conventional system, data is often moved from off-chip memory through on-chip devices (buses,

levels of cache, and register file) before it is processed. A PIM-based system eliminates much of this data movement by operating on data within the PIM chip, thereby localizing computations. This reduces the distance between where data is stored and where it is processed, leading to substantial energy saving.

Since conventional processors are not particularly designed to take advantage of high-memory bandwidth, processing units in PIM systems are usually specialized units such as vector processors or simple reconfigurable units. Also, generally several or many of these processing units operate in parallel to access memory concurrently (therefore increasing aggregate memory bandwidth). Integrating processing units with memory components can provide a way for high-performance and energy-efficient processing of data-intensive applications. This large and growing class of applications (such as biomedical applications) processes huge amounts of data and can often be implemented using a data-parallel approach well-suited to PIM systems.

There have been several efforts to unify processor logic and DRAM onto a single chip, including proposals for architectures, prototypes, and programming models as well as application case studies [11]–[22]. We briefly provide a summary of PIM architectures. The *Computational RAM* architecture [13], [14] unifies memory and logic by connecting SIMD processing elements to DRAM sense amplifiers through a wide bus. Similarly, the *Intelligent RAM* architecture [21] combines a vector processor and DRAM memory on a chip to enable fast data-parallel operations through wide memory interface [18]. The *FlexRAM* memory chip [16], which replaces a conventional memory chip, comprises a processor and a large number of simple compute engines that are interleaved with DRAM blocks. FlexRAM chip are connected for inter-chip communication. The processor runs the serial code and controls the execution flow, while simple compute engines process bulk of data in their local memory blocks [82]. The *Smart Memories* architecture [19], [83] is a tiled architecture composed of many processing units, caches, and

DRAM blocks in a same chip. The processing units, memory units, and interconnection units are reconfigurable to some degree. Thus, the programmer can program all these units to better support applications with different data structures and programming models. In *Data IntensiVe Architecture (DIVA)* [12], [15], processor-in-memory chips are composed of one or multiple tiles which each contains a processor and DRAM cells. The processor-in-memory chips are connected through a separate interconnect for inter-chip data communication. The *Active Pages* model [20] integrates many small pages of data and their associated functional units in a single chip. Data pages and functional units are implemented on DRAM and fine-grained reconfigurable fabric, respectively. Functional units operate only on data stored in their local data page. Inter-page communication is performed through the host processor. The recent *TCAM* [84] and *AC-DIMM* [85] 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, while memory (DRAM process) is optimized for capacity with fewer layers of metal to decrease manufacturing cost and improve yield. PIM systems overhaul the DRAM architecture to efficiently integrate processing units in memory devices, making their designs and verification challenging and time-consuming. Moreover, PIM systems suffer from high manufacturing costs and low yield because of integrating two different technologies (e.g., having additional metal layers compared to memory devices) [24], [25].

There are some proposals for processing data in the memory controller. These proposals reduce data movement through the cache hierarchy by adding specialized hardware to the memory controller. Yoo *et al.* [86] propose a near-memory processing architecture for a many-core system composed of 36 processor and memory tiles. To reduce the latency of memory operations, memory tiles are tightly coupled with special-purpose processors called *active*

memory processors (AMPs). This architecture enables offloading data-intensive operations to AMPs such that AMPs replace memory transactions over the on-chip network and computations on processor tiles. Some researchers have proposed adding specialized hardware to the memory controller for data processing. Ahn *et al.* [87] dedicate hardware to enable parallel execution of a specific class of atomic operations in the memory controller for data-parallel architectures. Wei *et al.* [88] propose augmenting the memory controller with a co-processor for vector, streaming, and bit manipulation operations. Fang *et al.* [89] propose an *active memory controller* (AMC) in which scalar and stream operations are performed in the memory controller. The goal is to reduce the latency of operation on data with low temporal locality and minimize memory traffic in a distributed shared-memory system. The AMC contains a hardware unit to perform pre-defined scalar and stream operations which are issued by the processor. In general, memory controllers augmented with processing hardware engines still suffer from high off-chip data energy and limited applicability due to supporting a limited class of applications.

Recently, there have been proposals to bond DRAM layers with application-specific accelerators [32] and GPGPU execution units [31]. The Hybrid Memory Cube (HMC) [90] and 3D-stacking memory and processors [91], [92] promise high memory bandwidth and new opportunities for local data processing. The HMC integrates 4 or 8 3D-stacked memory dies and a logic die on a package. The logic layer implements a memory controller and has a new interface to interface to the processor. The HMC packetizes memory accesses and exploits high-speed serial links. 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. Our approach is completely achievable using the present-day 3D-stacking technology and

does not require significant modifications to memory device design or their interface to the processor, making it compatible with commodity DRAM devices.

3 Experimental Framework

In this dissertation, we propose techniques to improve energy-efficiency of processor-accelerator architectures where the accelerators communicate through the cache hierarchy. We also propose a near-memory processing architecture that uses commodity DRAM devices and reconfigurable accelerators. To evaluate our proposed architectures, we use a combination of cycle-accurate simulation and power modeling over different benchmarks. In this chapter, we present the performance simulation infrastructure and power modeling tools that we have used in this research. We also describe our methodology to develop workloads for performance evaluation of our proposed architectures.

3.1 Performance Simulation Infrastructure

We use a simulation-based approach to evaluate performance and energy consumption of the proposed architectures and compare them with prior work. The studies in this dissertation entail the design of a simulation framework that integrates an out-of-order processor core, a coarse-grain reconfigurable accelerator (CGRA), cache hierarchy, and DRAM subsystem. For simulation and performance evaluation, We use the *gem5* simulator [93] which is extensively used for research on computer architecture in academia and research labs. The gem5 simulator is an event-driven simulator that models the processor core, cache hierarchy, and external memory. We have highly extended the gem5 simulator to support multi-port caches, multi-bank caches, non-temporal load/store memory accesses, and write combining buffers. We also carefully model a coarse-grain reconfigurable accelerator to account for its data computation and communication latency to the memory hierarchy. Therefore, our simulator models the functional and timing execution of the processor core, cache levels, CGRAs, DRAM devices, buses, and the interfaces

between these structures. For Chapter 5, we extended the memory controller component in gem5 [94] to support a 3D connection between CGRAs and DRAM devices and near-memory data processing capabilities. We have extended the ISA in gem5 to support accelerator invocation and control. All the results presented in this dissertation are based on the combined software and accelerated kernel execution, but do not include initialization and non-memory I/O operations.

3.2 Power Modeling

We use McPAT-based power models [95] to calculate power consumption of the processor core, on-chip cache hierarchy (using cacti) and other on-chip units such as buses. We have developed McPAT-based configuration models for a two-way out-of-order embedded processor similar to ARM Cortex-A9 [96], a four-way out-of-order high-performance processor similar to Intel Nehalem [97], and their cache hierarchy. We have also developed detailed scripts to automatically parse gem5 stats files (that include microarchitectural event counters) and generate McPAT activity factor input files. The generated activity factor input files and configuration models are then utilized by McPAT to output dynamic and leakage power consumption of different processor units.

We use cacti 6.5 [98] to estimate energy consumption of read/write operations from/to caches and their leakage energy consumption. When coherence is needed, we use a snooping MOESI cache coherence model and consider the overhead to maintain coherence. Similarly, we have developed script for parsing gem5 stats, parsing cacti outputs, and generating cache energy and power numbers.

We use DRAMPower [99] for evaluating power consumption of DRAM DDR devices and their I/O interface. We have modified DRAMPower to differentiate between communication through TSVs and communication through off-chip I/O. We have also integrated DRAMPower

with gem5. DRAMPower is a high-level power estimator tool that has been developed based on a *Power Calculator* technical note by Micron [100]. DRAMPower, similar to McPAT, has an XML interface to specify configuration model parameters which can be obtained from DRAM manufacture datasheets. DRAMPower output stats can be interpreted to extract different power components of a DRAM device. The DRAM *background* power includes power consumption of DRAM control logic and refresh operations. The *activation* power encompasses power consumption of activating row buffers and precharging bitlines. The *read/write* power refers to power consumed on reading/writing data from/to row buffers. Finally, the *interface* power represents power consumed to drive/terminate the output I/O bus [100], [101]. We denote I/O energy as a function of the number of I/O transactions and assume off-chip and TSV I/O consume 20 pJ/b and 4 pJ/b, respectively [102]. The common method to generate power estimates using DRAMPower is to feed DRAMPower with either low-level DRAM access traces (e.g., CAS, RAS commands with their addresses). DRAM traces could be large for memory intensive workloads. Therefore, we have modified DRAMPower to be fed with high-level stats (e.g., page hit rate, and number of activate commands, number of refreshes). This way the gem5 stats can be automatically fed to DRAMPower. This modification significantly improves speed and simplicity without sacrificing accuracy. The other method to evaluate DRAM power consumption is to use a power model developed by Rambus [103]. This power model, which is manufacture-independent, can be used to project the power consumption of both current and future DDR devices.

To characterize the energy efficiency of different caches in Chapter 0, we adopt the energy-delay product (EDP) [104] as the evaluation metric. In these metrics, delay is the execution time of the application, and energy is the consumed energy in the cache hierarchy (including L1 and L2 caches).

For evaluating area and power consumption of functional units in CGRAs, we developed Verilog models of different functional units using Synopsys DesignWare building block IPs [105] and synthesized the units using the Synopsys design compiler 2013 and TSMC 40nm low-power standard-cell library. We have developed an in-house power estimator to evaluate power consumption of CGRAs based on information extracted from synthesis results. The energy consumption of each kernel mapped to the CGRA functional units is based on the number and type of computations in the dataflow graph representing the kernel.

3.3 Benchmark Development

We have developed hybrid software/accelerated benchmarks that can be run on a heterogeneous architecture composed of a processor core and accelerator(s). For this research, we use the ARMv7 ISA and extend the ISA to support sending parameters from the processor to the accelerator and invoking application kernels on the accelerator.

To develop benchmarks, we profile the applications to find the kernels (compute-intensive and data-intensive operations) of each application. We then analyze each kernel to determine how much execution time is spent in it and whether or not the kernel can efficiently be mapped to reconfigurable hardware. We then partition the application into software (run by the processor) and hardware kernels (run by reconfigurable accelerators). To calculate the execution speed of kernels running on reconfigurable accelerators, we create dataflow graphs of the kernels in which nodes and edges of the graph represent operations and data movement, respectively. The chosen kernels are carefully analyzed and mapped to reconfigurable accelerators. Kernel latencies are based on the depth and type of computations in the dataflow graph after it is mapped to the CGRA functional units. The software code is then modified to replace kernel computation with appropriate instructions to transfer parameters and control between the processor and the

accelerator. We compile all benchmarks using GCC 4.6.3 with –O3 optimizations targeting the ARMv7-A architecture with the Thumb instruction set and VFPv3 (floating-point) extensions.

4 Reconfigurable Cache Architectures for Processor-Accelerator Systems

High-performance embedded systems often include one or more embedded processors coupled with more specialized accelerators. These accelerators improve both performance and energy efficiency because they are specialized for specific (or specific classes of) computations. Data communication between the accelerator and memory, however, is a potential bottleneck for both performance and energy-efficiency.

In this chapter, we first study two different processor-accelerator architectures for connecting a processor with a reconfigurable accelerator that are described in Section 4.1. My later investigations are based on these architectures. We then compare and evaluate the impact of L1 data cache design on performance and energy consumption of embedded processor-accelerator systems with shared memory. We study the potential of configurable caches to exploit diversity in cache requirements across hybrid software/accelerated applications to significantly improve energy-efficiency while maintaining high performance. Driven by these study insights, we propose two techniques for improving energy-efficiency of the cache hierarchy in processor-accelerator systems. The first technique adds configurability to the accelerator-cache interface to allow the accelerator to either share the processor's L1 data cache or use its own private L1 cache (Section 4.4). The second technique modifies the L1 cache structure to provide a configurable tradeoff between bandwidth (number of ports) and capacity (Section 4.5).

4.1 System Overview and Experimental Methodology

Previous studies have demonstrated that an accelerator performs better with a direct access to the memory hierarchy than an indirect access through a separate local buffer [5], [63], [106]. In

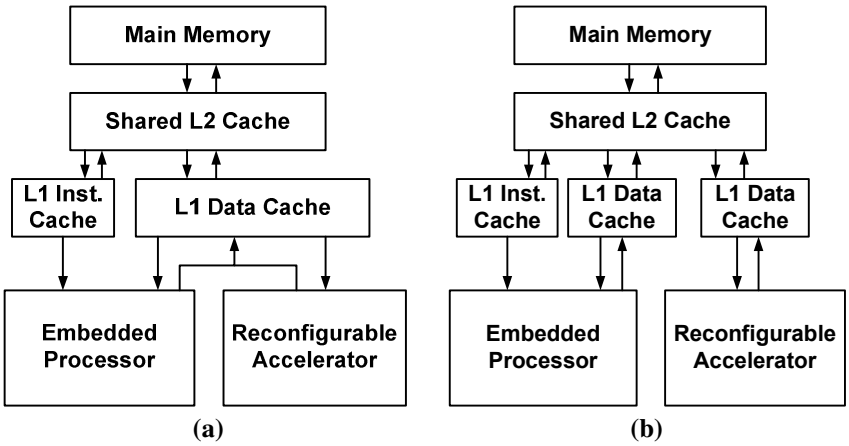


Figure 5. Processor-accelerator architectures: (a) Processor and accelerator with a shared L1 data cache, (b) Processor and accelerator with private L1 data caches.

this research, we thus focus on two variations of processor-accelerator architectures with a shared-memory organization in which the accelerator has a direct access to the cache hierarchy: one where the processor and accelerator share an L1 data cache (Figure 5a), and one where each has its own private L1 data cache (Figure 5b). Both architectures also contain a shared L2 cache. In this dissertation, we focus on a system with one processor core and one accelerator. However, our proposal could be extended to multi-core systems. In a multi-core system, processor cores could either share a single accelerator or each have a dedicated accelerator.

In this system, the accelerator directly issues its own memory requests to the shared memory hierarchy. Like the processor, it uses virtual addresses. For address translation, the TLB is shared between the processor and the accelerator [63], [106]. There is also a direct connection between the accelerator and processor that is used to communicate control information such as application kernel parameters through a fast, but low-bandwidth connection [63].

This work targets a high-end embedded system, but does not depend on a specific ISA or processor architecture. We chose to model a dual-issue out-of-order processor (OoO) similar to an ARM Cortex-A9 embedded processor with a 9-stage execution pipeline. Table 1 shows the key architecture parameters of the processor and caches. We assume the system is implemented

in 32 nm process technology with a supply voltage of 0.9V, and runs at 1GHz. For the accelerator, we use a CGRA with a heterogeneous grid of 8×8 32-bit functional units (See Section 2.1.1 for more details). The processor offloads application *kernels* onto the accelerator to improve performance and energy efficiency.

Table 1. Key architecture parameters for processor

Fetch/Decode/Dispatch/ Issue/Retire width	2/2/2/3/2	L1 Inst. Size	16KB
		L1 Inst. associativity	4-way
ROB entries	40	L1 access latency	3 cycles
IQ entries	12	L2 cache	512KB 8-way
LSQ entries	22	L2 access latency	12 cycles
Int ALUs	2	Main memory access latency	100 ns
FP ALUs	1		
Physical Registers	65	Cache line size	64B
Branch predictor	Tournament	MSHRs [107]	6 with 8 targets per register [108]
BTB entries	512		

4.2 Benchmarks

To evaluate performance and energy consumption of L1 data caches in processor-accelerator systems, we use a variety of applications from the Mediabench [109], Mediabench II [110], Parboil [111], and ERCBench [112]. We choose four multimedia applications (JPGD, MPG2D, ADPCM, SAD), one cryptographic application (AESE), and one scheduling application (PNS), as shown in Table 2, that are representative of typical embedded systems applications. In this chapter, we use these hybrid software/accelerated benchmarks to evaluate configurable cache architectures for embedded systems with reconfigurable accelerators. Section 3.3 discusses how we develop hybrid benchmarks.

Table 2. Benchmarks used in the evaluation of reconfigurable caches

Name	Description	# of Kernels	% Replaced Exec. Time	Domain	Characteristic
JPGD	JPEG decoder [110]	3	90.2%	Multimedia	Compute-intensive
MPG2D	MPEG2 decoder [110]	9	78.7%	Multimedia	Compute-intensive
ADPCM	Adaptive differential pulse-code modulation [109]	1	99.9%	Multimedia	Compute-intensive
PNS	Petri net simulation [111]	2	95.7%	Scheduling	Data-intensive
SAD	Sum of absolute differences [111]	1	94.0%	Multimedia	Data-intensive
AESE	128-bit AES encoder [112]	1	99.9%	Cryptographic	Compute-intensive

4.3 Motivation for Reconfigurable L1 Data Caches

The memory access patterns of processors differ from those of accelerators. Processors are designed for a broad range of applications, whereas accelerators, as specialized compute units, are designed for highly-parallel compute-intensive applications. These applications usually have simple control flow, and may represent a portion of a larger application. A circuit implemented in a reconfigurable accelerator requires few (or no) loop counters and array indices to be computed, stored, or fetched. Also, unlike processors, intermediate values are located internally to the computational structures; processors instead require memory load/store operations to access these values if they exceed the capacity of the register file. The memory behaviors thus differ, placing different demands on the data cache(s). We therefore revisit L1 data (L1D) cache design parameters in the context of processor-accelerator architectures. This investigation helps me find energy-efficient cache designs and guides me towards proposing new designs for L1 data caches to improve energy efficiency. My goal is to find energy-efficient cache designs with reasonable performance for such an architecture.

We explore the cache design space for different compute-intensive applications listed in Table 2. We examine three different architectures: (1) a processor-accelerator architecture with a *shared* L1D cache (Figure 5a), (2) a processor-accelerator architecture with *private* L1D caches (Figure 5b), and (3) a *processor-only* architecture. In all architectures, we explore a variety of cache design parameters to evaluate their effect on performance and energy consumption. These parameters include capacity (from 2KB to 64KB), number of read/write ports (from single- to triple-port), and set-associativity (from 1-way to 4-way). Since streaming applications rarely take advantage of the L2 cache, we do not vary the processor's L1 instruction cache and L2 cache; these structures retain the design parameters given in Table 1.

Table 3a and Table 3b summarize the above design exploration. For each application and architecture, Table 3a and Table 3b specify the cache design parameters that result in the best (lowest) EDP while maintaining performance (execution time) within some threshold (2%, 4%) of the baseline's performance. For example, the leftmost two result columns in Table 3a list design parameters that provide the greatest EDP savings with at most a 2% performance penalty. For comparison, we also list the parameters with the overall lowest EDP when this performance requirement is removed.

Table 3a shows the results for processor-accelerator architectures with either a private or shared L1D cache, compared to a baseline processor-accelerator architecture with a shared 32KB, single-port 4-way set-associative L1D cache. Table 3b shows results for a similar exploration of a processor-only system, compared to a baseline processor-only system with a 32KB single-port 4-way set associative L1D cache. The baseline cache design for both tables is similar to L1D caches in several high-performance embedded processors such as those in the ARM-A9-based processors in the Tegra 3 SoC [113] and Freescale's e6500 processors [114].

Each table cell contains the cache design parameters that result in the best EDP that meets the required performance, relative to the baseline. These parameters are listed as a tuple (*x*:*y*:*z*), where *x* is the cache capacity, *y* is the number of read/write ports, and *z* is the degree of set associativity. For example, (16:3:2) stands for a 16KB, triple-port, and 2-way set associative data cache. In the case of shared or processor-only architectures, a single tuple describes the single L1D cache. For the private cache architecture we list two tuples; the processor's L1D parameters followed by the accelerator's L1D parameters. In each table cell beneath the tuple(s) that describe the best L1D cache design, we list the percent EDP savings that design provides over the baseline L1D cache design.

The table demonstrates that different applications demand different cache design parameters, and that these demands are also affected by whether or not an accelerator is present, and if so, how it interfaces with the cache hierarchy (i.e., private vs. shared L1D cache). As an example, the leftmost two result columns in Table 3a list design parameters that provide the greatest EDP savings with at most a 2% performance penalty. The results given in Table 3a and Table 3b lead to four key observations about L1D cache designs aimed to minimize EDP:

1. **L1D Organization (Shared vs. Private):** The energy and performance effects of using different L1D cache organizations are highly application-dependent. This is demonstrated by the fact that JPGD achieves its lowest overall EDP and execution time when the L1D cache is shared, whereas MPG2D and SAD achieve their lowest overall EDP and execution time when the L1D caches are private.

2. **L1D Size:** Hybrid applications (software+accelerated) require a wide range of L1D cache sizes from 2KB to 16KB to minimize EDP. Some applications (ADPCM and PNS) favor very

Table 3. L1D cache design parameters (*size*:*ports*:*associativity*) with the lowest EDP for each benchmark/architecture combination, subject to the listed maximum slowdown relative to the baseline L1D cache (a 32KB, single-port 4-way set associative L1D). In each table cell, the percentage is the EDP savings of a cache with the listed design parameters over that of the baseline cache.

(a) Processor-Accelerator Systems

Benchmark	Max 2% Slowdown		Max 4% Slowdown		No Slowdown Threshold	
	Private (Proc.), (Acc.)	Shared	Private (Proc.), (Acc.)	Shared	Private (Proc.), (Acc.)	Shared
AESE	(2:1:1), (8:1:1) 30%	(8:1:1) 31%	(2:1:1), (8:1:1) 30%	(8:1:1) 31%	(2:1:1), (8:1:1) 30%	(8:1:1) 31%
ADPCM	(2:1:1), (2:1:1) 44%	(2:1:1) 44%	(2:1:1), (2:1:1) 44%	(2:1:1) 44%	(2:1:1), (2:1:1) 44%	(2:1:1) 44%
PNS	(2:1:1), (2:3:1) 63%	(2:3:1) 63%	(2:1:1), (2:3:1) 63%	(2:3:1) 63%	(2:1:1), (2:3:1) 63%	(2:3:1) 63%
SAD	(4:1:1), (16:2:1) 50%	(16:2:1) 47%	(4:1:1), (16:2:1) 50%	(16:2:1) 47%	(4:1:1), (16:2:1) 50%	(16:2:1) 47%
JPGD	-	(16:2:2) 27%	-	(16:1:2) 30%	(2:1:1), (4:2:1) 18%	(8:2:1) 30%
MPG2D	(8:1:2), (4:2:1) 29%	(16:1:2) 24%	(8:1:1), (4:2:1) 30%	(8:2:1) 26%	(8:1:1), (4:1:1) 30%	(8:1:1) 29%

(b) Processor-only Systems

Benchmark	Max 2% Slowdown	Max 4% Slowdown	No Slowdown Threshold
ADPCM	(2:1:2) 17%	(2:1:2) 17%	(2:1:2) 17%
PNS	(4:1:1) 26%	(4:1:1) 26%	(4:1:1) 26%
SAD	(8:1:2) 21%	(8:1:2) 21%	(8:1:1) 32%
JPGD	(16:1:2) 28%	(8:1:2) 29%	(8:1:1) 31%
MPG2D	(16:1:2) 24%	(8:1:2) 25%	(8:1:1) 28%

small 2KB L1D caches, while others (SAD and MPGD) favor larger caches. This motivates a capacity-configurable cache design, where sections of the cache could be disabled to save energy when the full capacity is unnecessary.

3. **L1D SRAM Ports:** The energy-efficiency of some hybrid applications such as SAD, PNS, and JPGD, significantly increases by using multiple-port caches. However, the execution time of applications such as AESE and ADPCM is independent of port count. Port count configurability could provide a greater bandwidth by enabling more ports for applications such as SAD. Port count configurability could save energy by allowing ports to be disabled when they are not needed for applications such as AESE.
4. **L1D Associativity:** For most hybrid applications, direct-mapped caches result in a better EDP. However, a few applications favor 2-way set associative caches. For applications such as AESE, ADPCM, and PNS, set associative caches barely shorten the execution time, but significantly increase energy. The performance provided by 4-way set associative caches for applications such as MPG2D, JPGD, and SAD does not compensate for the increased energy consumption of the data cache, thereby degrading energy efficiency.

Based on observation 1, neither a shared nor a private L1D organization achieves the best energy efficiency across all applications. In Section 4.4, we therefore propose a simple architectural technique that provides a *configurable L1D cache organization*, where the accelerator can use a private L1D or one shared with the processor, based on the best choice for the executing application.

Based on the observations 2-4, no single combination of L1D design parameters provides the most energy-efficient L1D cache for all applications. The size, associativity, or number of ports of an L1D cache could be tuned based on the executing application to improve the system's energy efficiency. Prior studies have proposed adding reconfigurability to caches [76] to tune the

number of cache sets (such as *selective-sets* [73]), number of cache ways (such as *selective-ways* [70] and *way-concatenation* [71]), or both (such as *hybrid selective-sets-and-ways* [75]) to exploit cache requirement variability across applications to reduce cache energy dissipation with minimal impact on performance. However, no prior study has proposed a method to vary the number of cache ports. In this research, we propose a design technique that we call *configurable-port* to tune the number of cache ports across applications by trading cache capacity for port count. Section 4.5 describes the proposed technique in detail.

4.4 Configurable L1D Organization

We propose an L1D cache organization that can be configured to act as either a single shared L1D cache or two private L1D caches, allowing two different L1D cache topologies to exist in the same architecture. This low-overhead organization provides the opportunity to reconfigure the interface between the accelerator and the L1D caches based on the executing application.

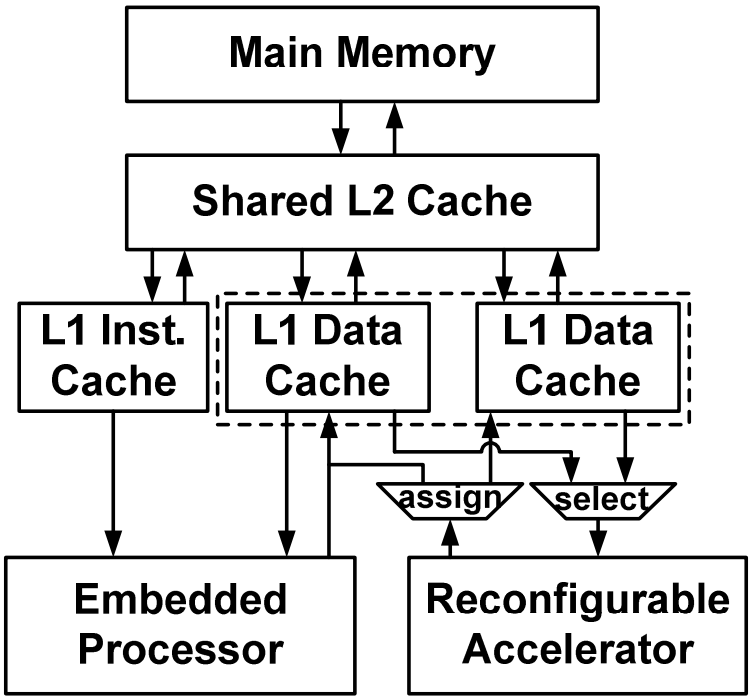


Figure 6. High-level structure of the proposed reconfigurable L1D cache organization.

My proposed L1D cache with configurable sharing is shown in Figure 6. The processor is always connected to its own L1D cache. The accelerator, however, can be connected either to the same L1D cache as the processor (one shared L1D cache), or to its own L1D cache (private L1D caches). In either cache organization (shared L1D or private L1D), the processor's memory requests are fulfilled by processor's data cache, but accelerator requests are directed either to the processor's L1D cache or the accelerator's L1D cache, depending on how the cache organization is currently configured. The configuration data sets memory bits that control the added routing logic labeled *select* and *assign* in Figure 6.

The cache organization exploits diversity in cache organization demands across applications; it can be reconfigured to provide applications with the organization that results in the best energy-efficiency (or performance, or whichever metric is desired). The reconfiguration can be performed prior to an application's execution or upon a context switch, based on profiling information for the application.

This configurable cache organization is also applicable to multi-core systems where each core has its own dedicated accelerator. In these systems, each processor-accelerator pair has a configurable L1D cache organization that is private with respect to the other processor-accelerator pairs. In this case, the cache organization of each processor-accelerator pair can be configured separately based on the application running on that pair. Note that in this design, we do not merge the two private caches to form a single, larger shared cache; rather, we disable the accelerator's private L1D cache when it is not in use.

4.4.1 Overhead

As shown in Figure 6, the added select and assign logic is very simple, and could be implemented using routing logic. The added logic needs to route data and address bits and some control bits using only one extra level of multiplexers. Therefore, this additional logic required to

support reconfigurability adds little area overhead to the L1D cache level design. The added delay is minor as well; the configuration bits are loaded prior to application execution and are retained until the application completes or a context switch. The path between the accelerator and its corresponding L1D is thus statically configured and not the result of a dynamic computation, limiting the latency and power impact. Therefore, we assume that the resulting structure can run at the speed of a conventional cache, yet provide the ability to selectively choose the desired organization when appropriate. In the case where the added logic increases the accelerator-cache access latency, the accelerator-cache communication could be increased by one cycle compared to the processor-cache communication to accommodate the extra logic delay. Note that extra logic in the accelerator-cache communication path does not affect the processor's access latency since the processor is directly connected to its dedicated cache.

When the cache organization is reconfigured from shared L1D to private L1D, the private L1D cache incurs the overhead of cold misses, which are included in the evaluation. When the cache organization is reconfigured from private L1D to shared L1D, the accelerator's L1D could be either power-gated or kept in retention mode to save energy [115]. In the former case where the L1D is power-gated, all dirty lines should be written back to L2 cache before the L1D is turned off. This may impose considerable performance and energy overheads if switching between shared and private L1D happens frequently. If switching occurs only on a context switch, then flushing dirty lines can be overlapped by context switch time, minimizing the performance overhead.

In the latter case where the L1D goes to retention mode, the L1D keeps its state, but only serves coherency requests from the L2 cache. In this case, the L2 cache behaves as if the L1D cache is active, initiating L1D coherence transactions when needed. However, the accelerator's memory requests are not issued to this L1D. The L1D cache in the retention mode consumes less

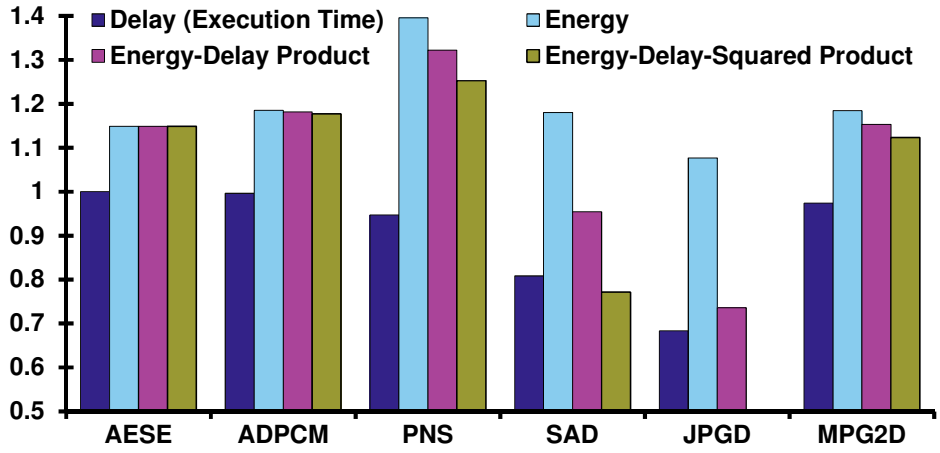


Figure 7. Performance and energy consumption of a shared L1D organization with (16:2:2) design parameters over a private L1D organization with (8:1:2), (8:2:1) design parameters.

energy in the active mode, even though energy consumption is not trivial. Since configuration bits are *only* loaded prior to application execution and cache organization reconfiguration occurs at the granularity of an individual process (i.e. on a context switch time), in the evaluation we assume the accelerator's L1D is power-gated and dirty lines are flushed prior to application execution.

4.4.2 Results

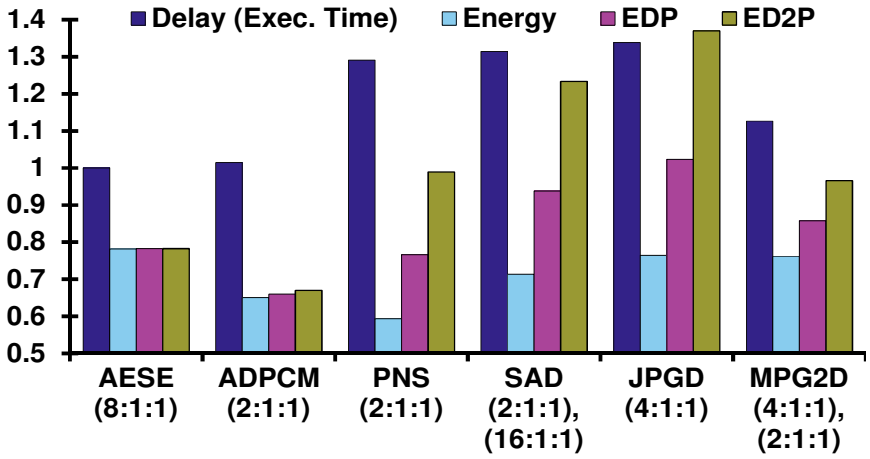
Our L1D cache design exploration in Section 4.3 indicates that when the L1D cache is shared, an L1D cache with (16:2:2) design parameters provides reasonable performance and energy efficiency across all tested applications. Also, when L1D caches are private, a processor's L1D cache with (8:1:2) design parameters and an accelerator's L1D cache with (8:2:1) design parameters provide reasonable performance while improving energy efficiency across all hybrid applications we tested. Therefore, if one has to choose *non-configurable* L1D caches, these cache design parameters could be the chosen parameters for shared and private cache organizations based on the applications we tested.

Figure 7 compares the execution time, energy, EDP, and ED^2P of a shared organization with (16:2:2) L1D cache design parameters over those of a private organization with (8:1:2), (8:2:1) design parameters. Note that the total L1D size in both organizations is the same. Figure 7 illustrates that the shared L1D organization with (16:2:2) design parameters has better or equal execution time for all benchmarks, while the private L1D organization with (8:1:2), (8:2:1) design parameters has better energy consumption for all benchmarks. With these cache design parameters, one would choose the private L1D organization when saving energy is critical (such as when the battery of the embedded device is low), while the shared L1D organization would be preferable when execution time and quality of service is more important (such as when the device is plugged in power source). In terms of EDP and ED^2P, some benchmarks have better energy efficiency on a shared L1D and some on a private L1D. For example, the shared L1D improves EDP of JPGD and SAD over the private L1D by 36% and 5%, respectively, while the private L1D improves EDP of PNS and MPGD over the shared L1D by 24% and 13%, respectively.

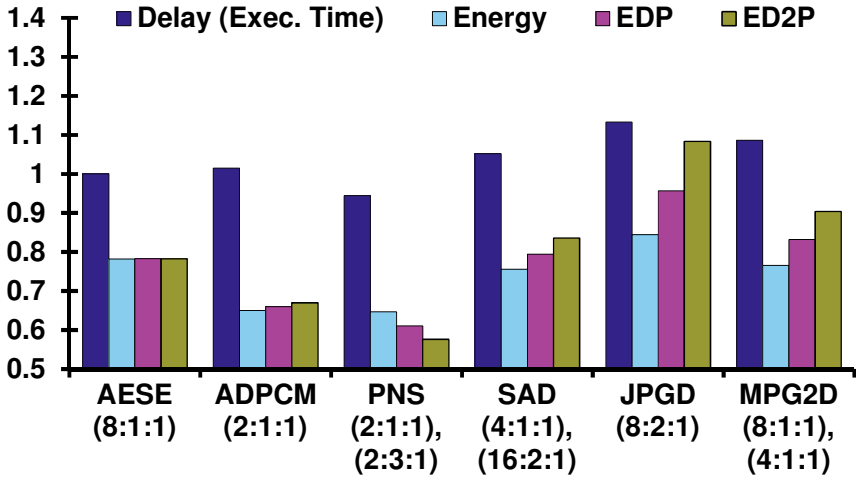
When *configurable* caches are used, one could choose from a wide range of possible cache configurations with different execution time and energy. We explore all potential combinations of cache design parameters and organization for all the benchmarks studied here. Note that best configuration is not determined and employed during runtime, but it is chosen based on application characteristics at compile time. Figure 8 shows the L1D cache configurations that result in the lowest energy, EDP, and ED^2P and compares their results with a shared L1D cache with (16:2:2) design parameters. Results show that the preferred L1D cache configuration and organization is highly dependent on the executing application and chosen optimization metric. A single cache configuration and organization provides the lowest energy, EDP, and ED^2P for AES and ADPCM, while other benchmarks require a different cache configuration to achieve their lowest energy, EDP, and ED^2P. In general, SAD, JPGD, and MPG2D benefit from larger caches

with more ports, while AESE, ADPCM, and PNS benefit from smaller single-port caches to achieve higher energy efficiency. In addition, Figure 8 indicates that AESE, ADPCM, and JPGD favor the shared cache organization, while SAD and MPG2D favor the private cache organization. All benchmarks see some reduction in energy, EDP, and ED^2P with this new configurable L1D organization, however the reduction is highly application dependent. The results show that the configurable L1D organization along with configurable caches can reduce energy, EDP, and ED^2P by up to 41%, 39%, and 44%, respectively (69%, 64%, and 78% improvement), over a fixed L1D organization.

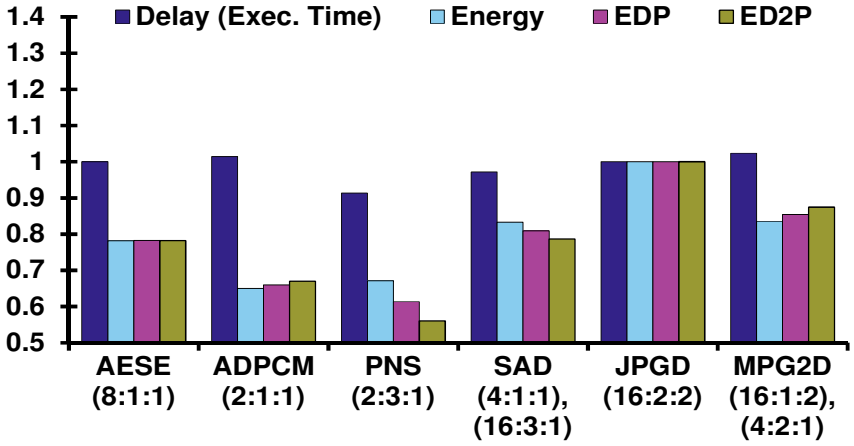
Overall, each combination of cache design parameters, organization, and application is placed in a different spot in the design space of energy and delay for processor-accelerator systems. Our configurable L1D organization provides the opportunity to choose the cache organization that best targets the desired optimization metric for the running application.



(a) Configurable L1D caches with lowest energy



(b) Configurable L1D caches with lowest energy-delay product (EDP)



(c) Configurable L1D caches with lowest energy-delay-squared product (ED^2P)

Figure 8. Result summary of configurable L1D caches normalized to an L1D cache with (16:2:2) design parameters (Lower is better).

4.5 Configurable Ports

In Section 4.3, we demonstrated that multi-port caches improve execution time and even energy efficiency for some hybrid applications. However, multi-port caches barely improve the execution time of other hybrid applications, degrading their energy efficiency due to the higher energy dissipation of multi-port caches. To achieve better energy efficiency across a wide range of hybrid applications, we propose a technique that we call *configurable-port*. This technique enables the designer to increase the number of cache ports at the cost of smaller cache capacity when executing applications that require higher bandwidth. The configurable-port technique creates the illusion of multi-port caches using a set of simple single-port cache arrays. For applications that do not benefit from the additional port(s), the configurable-port technique enables the designer to reconfigure the cache as a conventional single-port cache to reduce energy dissipation. This single-port cache can be maintained as the same capacity as the multi-port configuration, or it can use the full capacity provided by the single-port cache arrays. Note that the configurable-port technique is orthogonal to the proposed configurable L1D organization. The two techniques can be used separately or together.

We exploit the idea of cache replication (discussed in Section 2.2) to implement the configurable-port technique. Starting from a single-port configuration, p identical cache arrays are first configured to the appropriate size and associativity, where p is the number of ports and cache having at least p arrays. For example, to make a dual-port 8KB cache, two cache arrays of 8KB each are required. Previous work on reconfigurable caches [75] discusses how to reconfigure cache arrays to the appropriate size and associativity. Each port is then connected to one of the cache arrays by configuring the relevant steering logic, as shown in Figure 9. Data is replicated across the cache arrays as many times as the number of ports. Hence, a read request is sent to a single cache array, but write requests are broadcast to all arrays for coherency.

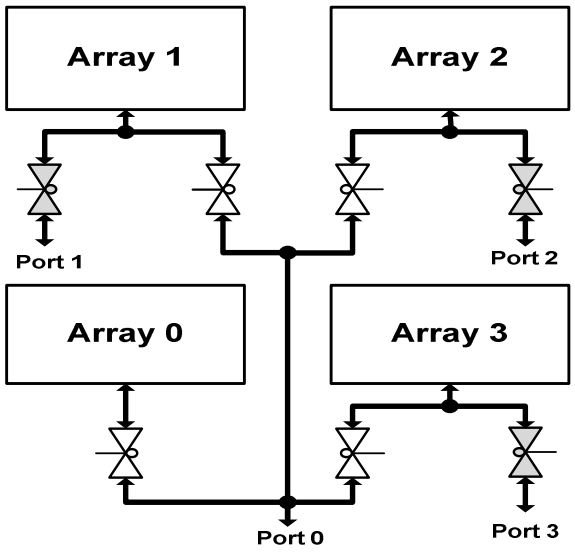


Figure 9. High-level structure of a configurable-port cache that can be configured as a single-, dual-, triple-, or quad-port cache. The added logics are indicated by shaded blocks.

Consequently, the configurable-port technique increases the number of cache ports (bandwidth) by reducing the cache capacity.

Figure 9 depicts the high-level structure of a configurable-port cache that can be configured as a single-, dual-, triple-, or quad-port cache. In this figure, write requests can only be issued by Port 0. The added steering logic required to support configurability are indicated by the shaded blocks. The steering logic (shown as transmission gates) are controlled by a simple function of address bits, configuration bits, and read/write bits. Setting the configuration bits therefore reconfigures the cache ports.

4.5.1 Overhead

To implement the proposed configurable-port technique, we can reuse the datapath logic in a multi-banked cache design to route data from/to one of arrays. Minor modifications to the control logic are also needed to generate control signals for the steering logic. Therefore, the implementation overhead to the L1D cache design is small. We also assume that the added logic also does not increase the critical path delay because the control signals for a configuration are set

Table 4. Energy consumption of different types of dual-port caches implemented in 32 nm technology.

Size	Cache type	Read access energy (pJ)	Write access energy (pJ)	Leakage power (mW)
8KB	True	19.5	29.8	0.5256
	Configurable	13.9	43.9	0.4932
	Line-interleave	15.6	19.6	0.5301
	Word-interleave	15.8	19.7	0.5350
16KB	True	22.7	31.4	0.6115
	Configurable	16.5	46.3	0.6746
	Line-interleave	17.0	25.1	0.6415
	Word-interleave	17.2	25.3	0.6450

before an application starts and do not change during application execution. In these results, we thus assume that a configurable-port cache has the same latency as the baseline cache.

Similar to the technique of configurable L1D organization, configurable-port cache designs enable the designer to configure the L1D when an application begins, at a context switch, and when an application completes. Any unused cache arrays can be power-gated to save energy using the existing mechanism typically available for SRAM [115]. Thus, dirty lines of power-gated cache arrays should be written back to the L2. In addition, all blocks (clean or dirty) in cache arrays in which their set-mappings change after enabling the cache arrays should be flushed [75]. To reduce the overhead of writing back dirty blocks, we only configure the cache when an application begins.

Multi-port caches created by the configurable-port technique have lower performance compared to true multi-port caches. To maintain coherence, configurable-port caches allow only a

single write at any given cycle, sending the write request to all cache arrays simultaneously. This degrades the performance of applications with high store-to-load ratio.

As shown in Table 4, energy dissipation of a configurable-port cache with two ports is different from that of a true dual-port cache. Since writes should be broadcast to multiple cache arrays, writes to configurable dual-port caches consume more energy than writes to true dual-port caches with the same size. However, reads to configurable dual-port caches consume less energy compared to corresponding true dual-port caches because of using simple single-port SRAM arrays rather than dual-port SRAM arrays. Thus, configurable-port caches reduce read energy (and potentially leakage), but increase write energy (included in our results). Multi-banked caches require crossbars to route requests and responses between the processor/accelerator and cache banks. In configurable-port caches, read requests can be sent to any cache arrays, obviating the need for crossbars. As a result, as shown in Table 4, reads to configurable dual-port caches consume slightly less energy than reads to dual-banked caches.

4.5.2 Results

In this section, we study the effect of high-bandwidth data caches and show that architectural techniques such as configurable-port improve the energy efficiency of judiciously-sized multi-port caches in accelerated systems with minimal impact on performance. We focus on performance and energy consumptions in our analysis since these are more of a concern than area in future embedded systems.

Execution setup. Based on the results in Section 4.3, we classify these benchmarks as either cache-insensitive or cache-sensitive. Cache-insensitive benchmarks, such as AESE and ADPCM, show little to no performance variation for the different cache parameters (port count, associativity, and size). AESE and ADPCM have few memory accesses per time unit and their rate of communication over computation is low. In fact, a small 4KB single-port L1D degrades

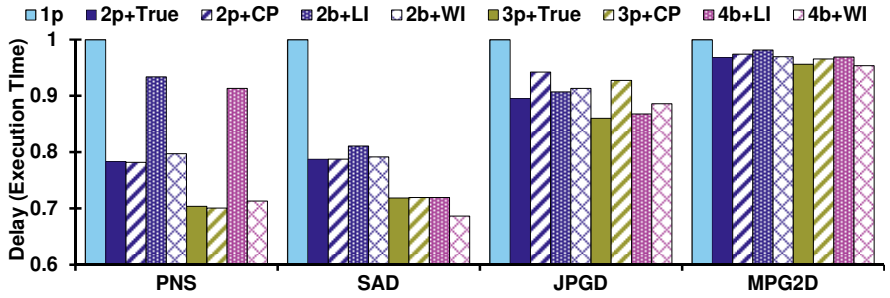
the performance of AESE and ADPCM by only 2% over a 64KB triple-port L1D, while it has clear energy advantages. In addition, AESE and ADPCM performance is approximately the same for both private and shared L1D topologies. Conversely, the performance of cache-sensitive benchmarks such as SAD and PNS is significantly affected by cache parameters. Therefore, we investigate the impact of the proposed configurable-port technique on execution time and energy dissipation of cache-sensitive benchmarks and compare it with true multi-port caches, line-interleave multi-bank caches, and word-interleave multi-bank caches.

To have a fair comparison, we use a 16KB two-way set associative cache in a shared L1D organization for all benchmarks, which provides reasonable performance/energy for all cache-sensitive benchmarks. The results are easily applicable, however, to other cache sizes and private L1D organization. In this evaluation, we assume the word size is four bytes for word-interleave banking.

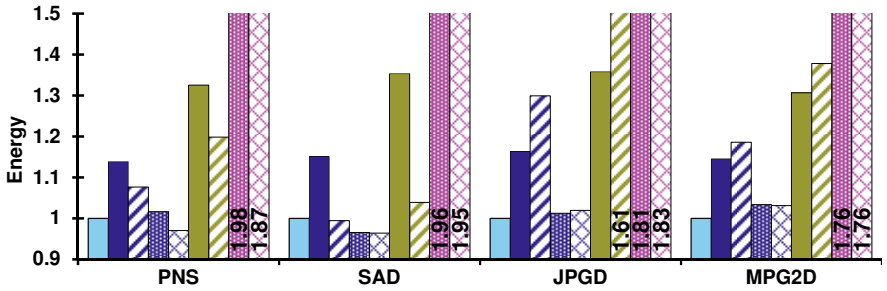
Comparison of multi-port cache techniques. Figure 10 compares different types of multi-port caches (true multi-port, configurable-port, line-interleave multi-bank, and word-interleave multibank caches) in terms of execution time, energy dissipation, ,EDP, and ED^2P. In general, this figure shows that for the cache-sensitive benchmarks, multi-bank and configurable multi-port caches improve execution time with a small increase in energy over single-port caches, but true multi-porting incurs high energy consumption.

Compared to a true dual-port L1D, a dual-bank L1D results in a slightly longer execution time, but improves EDP (the exception is the dual-bank line-interleave L1D for PNS). For PNS and SAD, which have low store-to-load ratio, execution time of a configurable multi-port L1D is comparable to that of a true multi-port L1D, and better than that of a multi-bank L1D. For these benchmarks, configurable multi-porting improves EDP and ED^2P considerably over true multi-porting. Furthermore, PNS and SAD have better execution time, EDP, and ED^2P when the

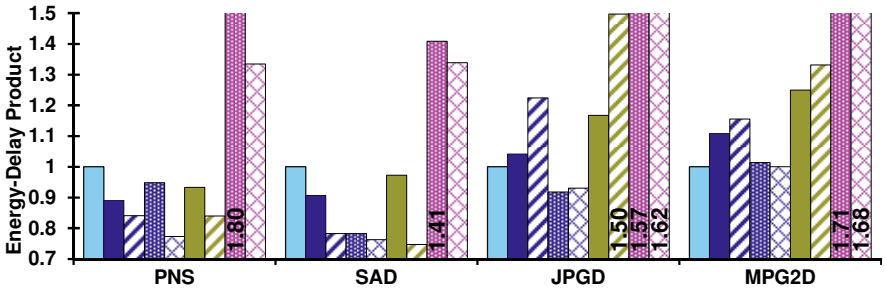
configurable-port L1D is configured with three instead of two ports. For JPGD and MPG2D, which have fair amount of stores, configurable multi-porting degrades execution time and EDP compared to true multi-porting. In general, configurable multi-porting and multi-banking achieve better EDP. A dual-bank L1D provides the most energy-efficient cache design for PNS, JPGD, and MPG2D, while configurable triple-port L1D provides the most energy-efficient cache design for SAD. For example, Figure 10 shows that a word-interleave dual-bank L1D reduces execution time and EDP of PNS by 20% and 23%, respectively, over a single-port cache. A line-interleave dual-bank L1D reduces execution time and EDP of JPGD by 9% and 8%, respectively, over a single-port cache. For SAD, configurable triple-port L1D reduces execution time and EDP by 28% and 25% over a single-port cache (improvement by 38% and 33%).



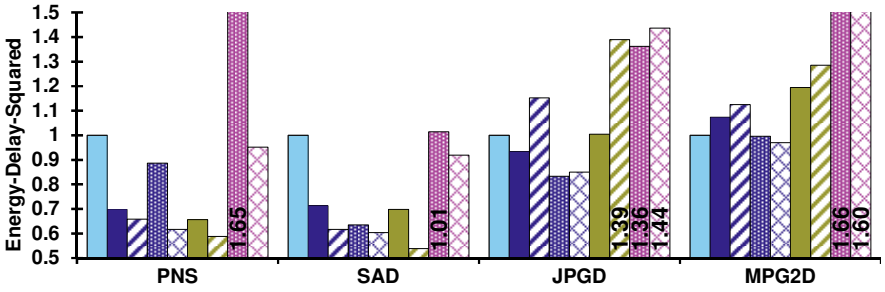
(a) Delay (execution time)



(b) Energy



(c) Energy-delay product (EDP)



(c) Energy-delay-squared product (ED^2P)

Figure 10. Results summary of configurable-port, true multi-port, and multi-bank caches normalized to a single-port cache. Caches are 16KB, 2-way set associative . The notations 'p', 'b', 'True', 'CP', 'LI', and 'WI' stand for ports, banks, true multi-porting, configurable multi-porting, line-interleave banking, and word-interleave banking, respectively.

Compared to an energy-hungry true triple-port L1D, a dual-bank L1D and a configurable dual-port L1D have longer execution time, but much better area cost and energy efficiency. Although a quad-bank L1D slightly improves performance over a dual-bank L1D, it is extensively energy inefficient due to extreme banking. On the other hand, while configurable triple-port improves EDP over configurable dual-port for PNS and SAD, it degrades EDP of JPGD and MPGD. The reason is that the extra port increases cache energy dissipation while it barely improves execution time of JPGD and MPGD, leading to higher EDP. Configurable-port can take advantage of variability of required cache ports across applications to resemble single-, dual-, or even triple-caches. True multi-port and multi-bank caches, on the other hand, have fixed structure and cannot be configured according to target applications.

Figure 10 also indicates that a word-interleave L1D performs equal or better than a line-interleave L1D for our cache-sensitive benchmarks. Note that even though word-interleave caches have marginally higher access energy and leakage power due to multi-port tags, their energy consumption is comparable with line-interleave caches because of the slight decrease in execution time.

Multi-port caches improve performance, but consume more energy. Multi-bank caches slightly degrade performance, but they have lower energy dissipation. Overall, we show that although high-bandwidth data caches consume higher energy compared to single-port caches, they can generally improve energy efficiency of some accelerated applications by reducing execution time and lowering leakage energy. For these applications, configurable-port caches can trade cache capacity for higher bandwidth. For other applications that do not benefit from higher bandwidth, the configurable-port caches can be configured as single-port caches to maintain lower access energy.

Effect of cache size on configurable-port. Figure 11 shows the performance of configurable dual-port caches with different sizes. Figure 11 confirms that, as expected, the miss rate and execution time decrease by increasing cache size. It also indicates that most benchmarks suffer from a high L1D miss rate. This is due to streaming nature of the benchmarks. This directs that an L1D prefetcher could improve performance (and potentially energy efficiency by lowering leakage energy). Figure 11d suggests that although EDP improves initially by increasing cache size, it degrades with further size increases. Thus, for each application, there is a specific break-even point for EDP when the energy cost of larger caches exceeds the energy benefit of shorter execution time. As shown in Figure 11d, For SAD and JPGD, a 16KB configurable dual-port L1D cache provides the most energy-efficient cache hierarchy. On the other hand, 2KB and 4KB configurable dual-port L1D caches deliver the lowest EDP for PNS and MPG2D, respectively. The best solution, therefore, is likely to be a 16KB configurable dual-port L1D (made of two single-port arrays of 16KB each) where some of the capacity can be disabled when not needed to reduce energy.

Overall, we show that although high-bandwidth data caches consume higher energy compared to single-port caches, they can generally improve energy efficiency of some accelerated applications by reducing execution time and lowering leakage energy. For these applications, configurable-port caches can trade cache capacity for higher bandwidth. For other applications that do not benefit from higher bandwidth, the configurable-port caches can be configured as single-port caches to maintain lower access energy.

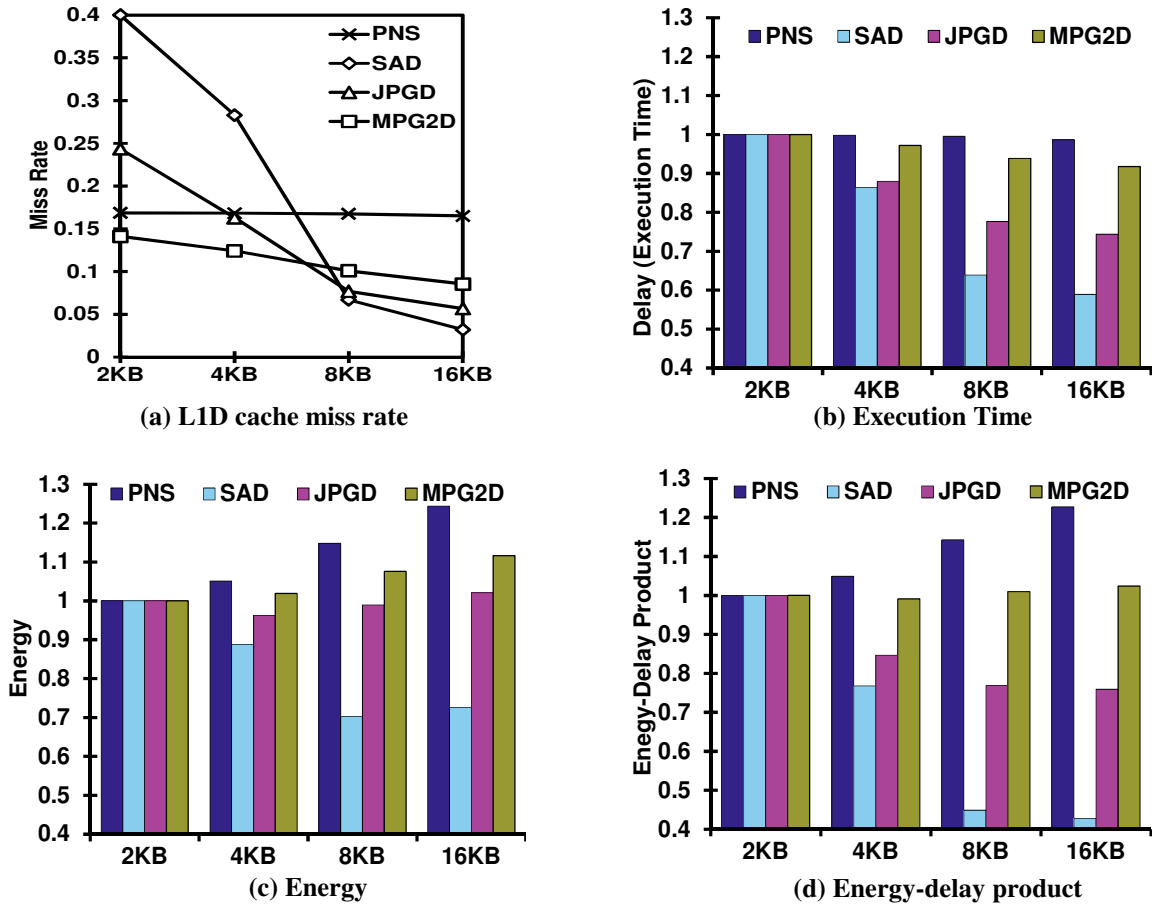


Figure 11. Results summary of configurable dual-port L1D caches with different sizes relative to a 2KB cache (Lower is better).

4.6 Summary

Accelerated embedded systems are designed to run a specific class of well-defined applications. We study a wide range of cache designs in these systems and find that configurable cache architectures can significantly improve energy-efficiency by varying cache requirements across applications. Therefore, we propose a configurable cache organization that allows shared and private L1 data caches to exist in the same architecture (Configurable L1D organization). Furthermore, we propose a novel cache design that provides a configurable tradeoff between cache capacity and cache bandwidth (Configurable-port L1D). Our simulation results show that

the Configurable L1D organization and Configurable-port L1D improve cache hierarchy energy-efficiency by up to 64% and 33%, respectively, over that of non-configurable caches. Our proposed approaches can be applied to multi-core systems where each core has a dedicated accelerator as well as systems where multiple cores share an accelerator.

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

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 chapter, we propose a near-DRAM acceleration (NDA) architecture, which process data using accelerators 3D-stacked on DRAM devices comprising off-chip main memory modules. The proposed NDA architecture aims at improving energy efficiency by exploiting local (near-memory) data processing. 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* 2D 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.

5.1 Background

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

5.1.1 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 I/O circuitry. Figure 12(center) shows a DDR3 DRAM device architecture with an ×8 interface (8-bit data I/O per DRAM device) [102], [103], [116]. 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.

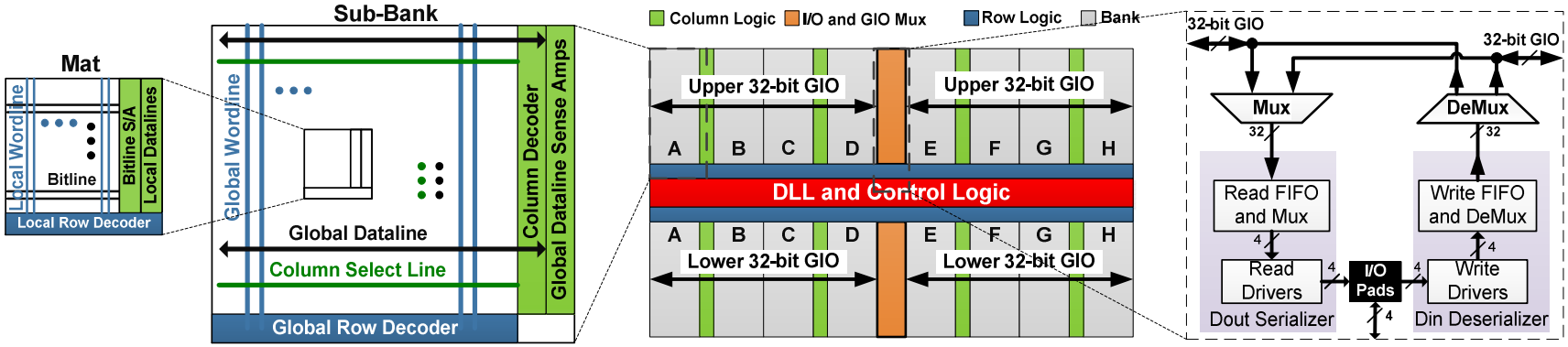


Figure 12. Conventional DRAM organization. A DDR3 device with 8 banks of A to H and ×8 I/O interface (center), internal architecture of a DRAM bank and a mat (left), and DRAM I/O datapath (right).

Each sub-bank is an array of memory cells that is divided into multiple mats. A mat typically is a sub-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 12(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. In ×8 DRAM devices, each bank group shares a 64-bit inter-bank global I/O (GIO) lines (also known as inter-bank datalines). The GIO lines in an ×8 device are comprised of upper and lower 32-bit GIO lines. In ×16 DRAM devices, each bank group shares a 128-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 12(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 times the 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).

5.1.2 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 integrating 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 data-flow architecture—coarse-grain reconfigurable accelerators (CGRAs). CGRAs have been shown to provide significant performance and energy efficiency benefits [35]. See Section 2.1.1.

5.2 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 dissertation, or could be SIMD/GPU/FPGA engines or even low-power cores. This dissertation focuses on CGRAs for near-memory processing due to their improved performance and energy consumption versus SIMD and GPU engines for most parallel workloads [35]. 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 13. Each CGRA is connected only to its associated DRAM device and operates on data stored in that 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.

Conventional processors can offload kernels to CGRAs. The processor communicates with the CGRA through a memory-mapped I/O interface that operates similarly to mode registers in

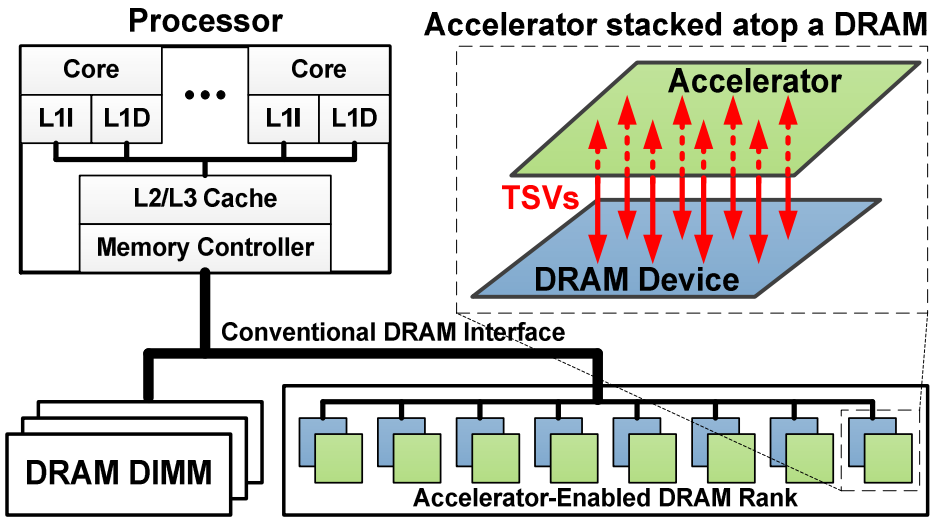


Figure 13. NDA organization.

conventional DRAM systems, and that does not require changes to the processor-DIMM interface. The processor sends kernel configuration parameters and address generation parameters for the data to be processed before triggering kernel execution using this interface. The processor periodically polls a memory-mapped status register to check for kernel completion.

The NDA architecture 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.

5.2.1 Connection between Accelerator and DRAM

We propose three microarchitectures to connect a CGRA and a DRAM device using TSVs. In all the approaches, we assume the TSV I/O energy for transferring data and control data is 4 pJ/b which is 80% lower than the off-chip I/O energy of a DDR3 interface (Table 5). Furthermore, in

all the microarchitectures, the CGRA-side memory controller (MC) manages data transfers between the CGRA and DRAM dies (Section 5.2.2).

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 architecture of the DRAM device. TSVs are connected to the GIO lines between GIO multiplexers and data serializer/deserializer (Figure 14). DRAM devices with ×8 and ×16 interfaces use 64 and 128 TSVs to transfer data between a DRAM device and its CGRA, respectively. Therefore, ×8 and ×16 DDR3 DRAM devices transfer 8 and 16 bytes of data between a DRAM device and CGRA in each memory transaction. This microarchitecture is similar to one developed for stacking a wide-I/O DRAM device atop a processor die [117]. As depicted in Figure 14, 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. Table 6 presents detailed information of area, energy, and timing specifications of this microarchitecture. A CGRA transferring data through 3D interconnects is provided the same peak bandwidth per device as a processor transferring data

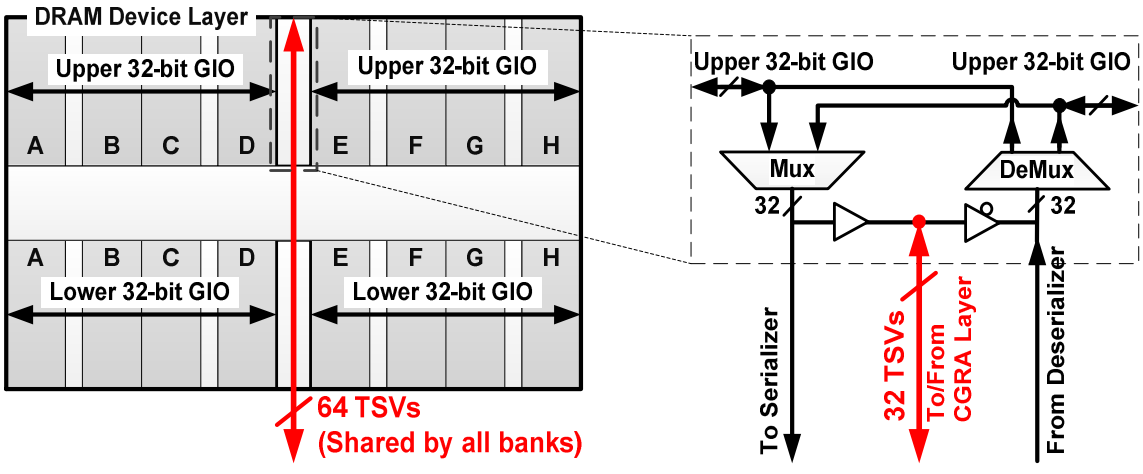


Figure 14. Accelerator-DRAM connection by connecting TSVs to existing GIO lines.

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 (Table 6). There are two main reasons for lower data transfer energy on the CGRA-DRAM connection over that on the processor-DRAM off-chip connections. The first reason is that data transferred to CGRAs does not move through the serializer/deserializer of commodity DRAM (circuit shown in Figure 12(right)). The second reason is that TSVs are 80% more energy-efficient than off-chip interconnect.

Microarchitecture 2 (Connecting TSVs to doubled GIO lines – NDA-2): Similar to Microarchitectures 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 as shown in Figure 15. 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. However, DRAM timing parameters can remain unchanged by adjusting the repeater interval of GIO lines, as elaborated in [102]. Table 6 presents detailed information of area, energy, and timing specifications of this microarchitecture. Note that

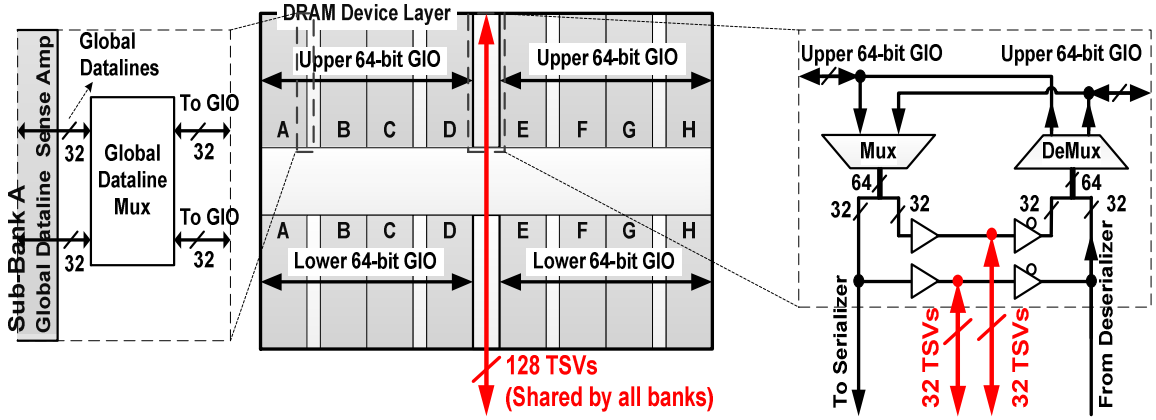


Figure 15. Accelerator-DRAM connection by connecting TSVs to doubled GIO lines.

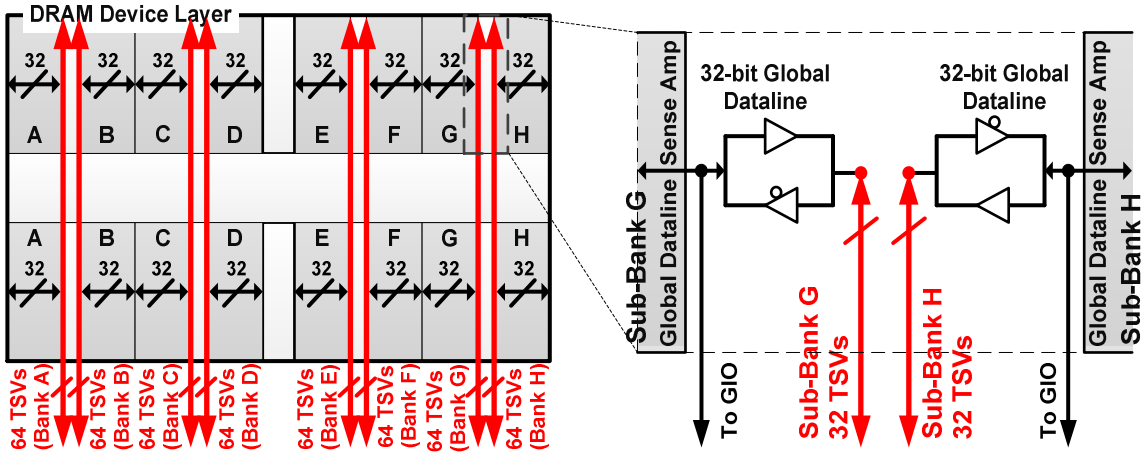


Figure 16. Accelerator-DRAM connection by connecting TSVs to global datalines of banks.

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 using "Global Dataline Multiplexers" as shown in Figure 15.

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. A 3D-stacked CGRA can exploit this to separately transfer data/address/command from/to each DRAM bank. 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 16). This microarchitecture increases the

CGRA-DRAM bandwidth substantially with only small overhead associated with TSVs. Table 6 presents the area overhead of this microarchitecture. 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.

TSV overhead: Current 3D interconnect technology allows a TSV pitch size of 40-50µm for 3D DRAM stacking [117], [118], which provide enough room for PHY, test, and ESD protection [28]. To reduce TSV area overhead, finer grain TSVs with a pitch size of 5µm can be used for signals [29] with redundant TSVs to increase the yield [119]. For data connections between a CGRA and a DRAM device, the overheads of address, command, power, and ground [117] should also be accounted for. We use fine-grain TSVs with a pitch size of 5µm and 100% redundancy for signals and coarse-grain TSVs with a pitch size of 45µm for power/ground. For instance, for a 128-bit data connection we use 256 TSVs for data, 64 TSVs for address and command, and 116 TSVs for power/ground. This incurs an area overhead of only ~0.243 mm^2. Since the typical area of DDR3 and DDR4 devices ranges from 30 mm^2 to 100 mm^2 [120]–[122], the area overhead of TSVs is negligible.

In Microarchitecture 3, there are eight 64-bit connections between a CGRA and a DRAM device. We conservatively bundle those eight connections into four 128-bit connection groups to share power/ground TSVs between data signals in a given connection group. In the worst-case scenario, TSVs in a given connection group can be arranged in a square of 495µm × 495µm. Note that arranging TSVs in rectangle results in a smaller area overhead. As shown in Table 5, we use an 8Gb DRAM chip with a 8mm × 10mm die area. Therefore, the TSV area overhead of Microarchitecture 3 would be (0.495×0.495×4)/(8×10) = 1.2%. This TSV area overhead does not include any added space area in the DRAM layout which is imposed by placing TSV connection

groups. In the worst-case scenario where the added space cannot be filled up with any other logic, the four TSV connection groups in Microarchitecture 3 increase DRAM die area by 26.9% to 8.49mm×11.96mm.

Summary: Table 6 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 (Section 5.2.2).

Table 5. 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/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 [123]	20 pJ/b [102], [124]

Table 6. 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 5. S/A = Sense Amplifiers. Some data in this table has been obtained by Prof. Ahn at Seoul National University.

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					
Microarchitecture 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%)
Microarchitecture 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%)
Microarchitecture 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%)

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 $6F^2$ DRAM cells. Table 6 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 6 also indicates that CGRAs consume less energy to transfer the same number of bits than processors. This is because 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 6 summarizes this energy overhead of the three microarchitectures on normal DRAM accesses.

5.2.2 Memory Access by Accelerators

Memory controller: As shown in Figure 17, 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 5.2.1, the CGRA-side MC manages data transfers between the CGRA and DRAM device in compliance with DRAM timing constraints such as tFAW and tRRD [125]. The CGRA-side MC does not require complex request ordering and arbitration logic

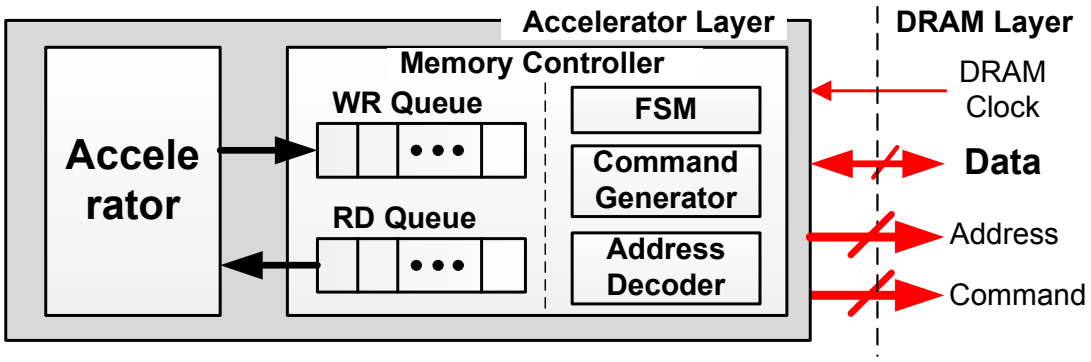


Figure 17. Accelerator stacked on top of a DRAM device.

that exists in the processor-side MC. Generally, many accelerated kernels include little or no data-dependent control flow and avoid "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 the 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. 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. After the processor offloads a kernel to CGRAs, it busy-waits until the CGRA operations complete to avoid frequent concurrent accesses by both the processor and the CGRAs (See Section 5.2.3). 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 method 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 [126] 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. An alternate form of this method is to limit the number of banks accessible by CGRAs to a specific set of banks. Therefore, the processor and CGRAs each have access only to their own dedicated set of banks. The drawback of this approach is to impose capacity constraints on allowable CGRA working set size, making CGRA programming more difficult.

The other way is to lock up the DRAM rank with active CGRAs and delay interfering memory accesses to the same rank until CGRA computations are completed. This, however, might significantly increase the response latency of the interfering task. The other way could be to use an MC that is shared between the processor and CGRAs. The shared MC can manage

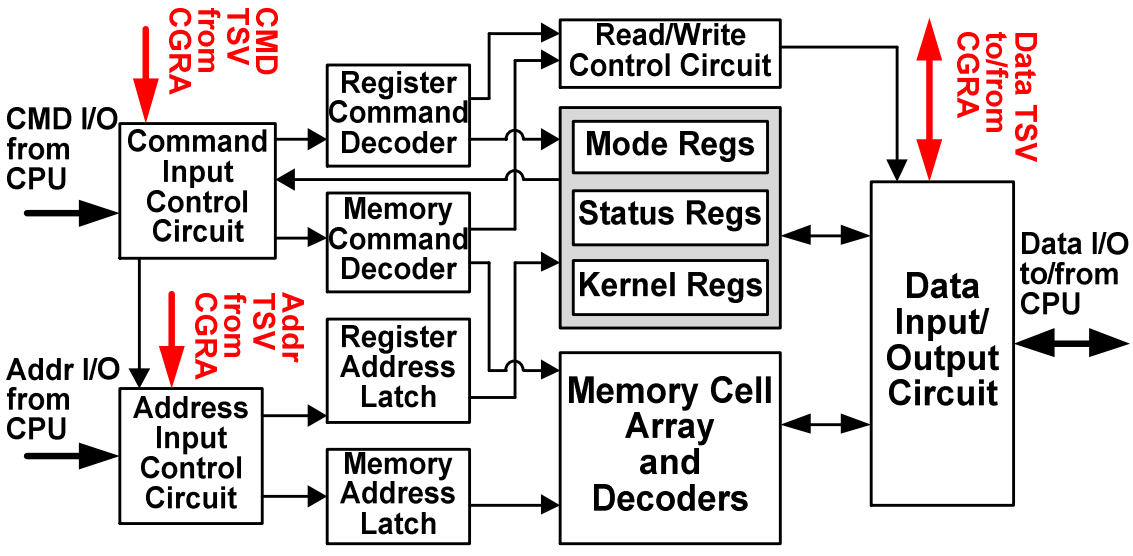


Figure 18. Access to mode, status, and kernel registers.

concurrent accesses based on latency and performance constraints. Although this approach is not feasible in the NDA architecture due to physical placement of CGRA-side MC and processor-side MC in different chips, this could be mapped to DRAM stacks that use a central MC in their logic layer (e.g. HMC).

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 [127], refresh intervals (tREFI) are long enough (e.g., 7.8µs in DDR3) to allow CGRAs to make considerable progress within them. CGRAs can be aware of refresh intervals and can suspend their normal operations right before the refresh command from the processor-side MC. One method to make the CGRA-side MC aware of refresh intervals is to have the processor-side MC send a refresh command right before it hands over its control to CGRA-side MC. Then, the CGRA-side MC can simply predict when the next refresh command will arrive from the processor-side MC. Recent techniques such as *adaptive refresh* [127] and *retention-aware refresh* [128] can be used to further mitigate the refresh overhead. Furthermore, on-die temperature sensors [116] 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 [129]. Therefore, to ensure 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 [129]. 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.

Error Correcting Code (ECC): For the memory address space used by the processor we can still use conventional ECC mechanisms. For the memory space used by CGRAs, we can embed the complete ECC code word for each data word packed/shuffled in each DRAM device which is called embedded ECC [130]–[132].

5.2.3 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 18). 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 [133]. 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 18). 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.

5.3 Software Considerations

5.3.1 Target Applications

NDA targets big-data applications with high parallelism and localized memory accesses which are traditionally deployed on Map-Reduce frameworks [92]. 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 [134].

5.3.2 Data Arrangement for Accelerators

The conventional DIMM architecture interleaves each 64-bit data block amongst DRAM devices in a given rank, 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 19(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 19(top-right) shows how this array is typically distributed across ×16 DRAM devices, separating the upper and lower halves of each 32-bit word, and failing to group the data for each quadrant into a single device. Figure 19(bottom) shows a “shuffled” arrangement of the

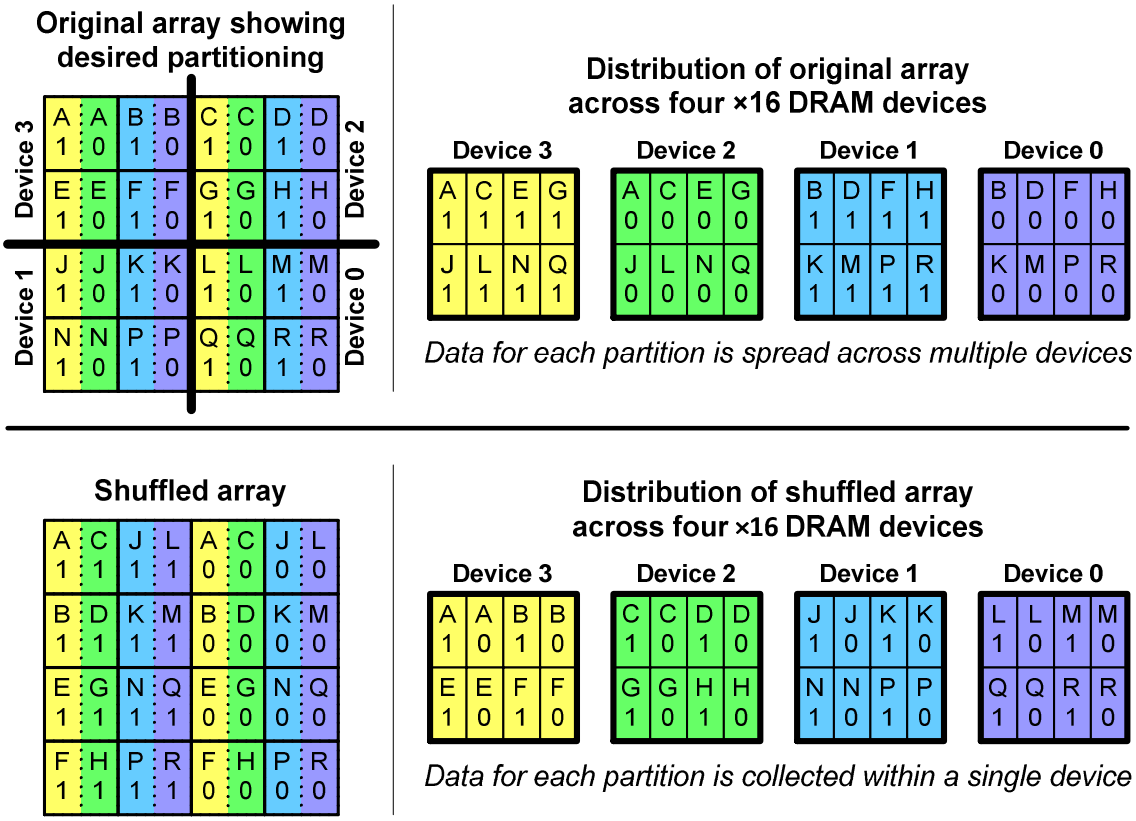


Figure 19. Data arrangement across devices for a 4×4 array of 32-bit words (A – R). Original data arrangement (top) and shuffled arrangement for NDA acceleration (bottom).

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 20). If the CGRA result data is needed by the processor, the processor must “unshuffle” it before using it.

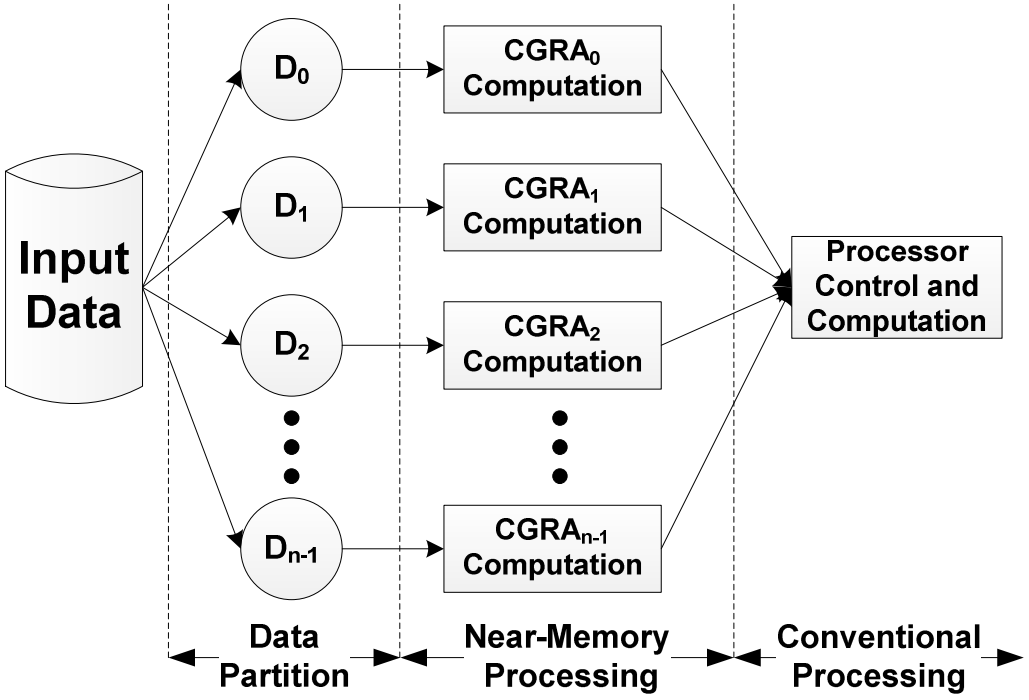


Figure 20. NDA programming model.

5.3.3 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 20 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 [135].

In case some boundary data is needed by more than one CGRA, this data need to be replicated to reduce/eliminate inter-CGRA communication. This replication can be performed when the CGRA input data is shuffled and the data structures initialized. If direct data communication between CGRAs is required (inter-CGRA data sharing), the processor can orchestrate data sharing between CGRAs by reading data from one DRAM device and write it to

another DRAM device. This is a time-taking process because the processor should first perform a DRAM read and then a DRAM write operation for each piece of shared data and only a portion of DRAM bandwidth is used in each DRAM operation. As a result, data sharing between CGRAs should be infrequent to keep the communication overhead low; meaning our NDA architecture is not a good fit for applications with high amount of data sharing between threads. 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 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: The pseudo code in Figure 21 shows the program execution flow of applications mapped to 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. The processor busy-waits until CGRA computations are finished. Once all processing using the current kernel is complete, the processor can read the CGRA output data and unshuffle it.

```
1: Partition data structures and allocate memory
2: Shuffle (arrange) input data
3: Configure CGRAs
4: While (data processing not complete) {
    5: Send kernel parameters and trigger kernel executions on CGRAs (spawn computation)
    6: Check CGRA computation status
    7: If CGRA computation not finished, go to 6
}
8: Unshuffle CGRA output data, if needed
```

Figure 21. Execution flow of programs mapped to NDA.

5.3.4 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[1]) 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

[1] ARMv8 has some non-temporal *hint* instructions such as `LDNP` and `STNP`.

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. When CGRAs are working, the processor periodically checks the status of CGRA computations until they are completed. As a result, no changes to memory consistency are needed.

5.4 Evaluation Methodology

In this section, we present our methodology to evaluate and quantitatively compare our proposed NDA architecture with prior work.

5.4.1 Benchmarks

Table 7 shows 11 different benchmarks from the San Diego Vision [136], Parboil [111], CORAL [137], SPLASH-2 [138], and Rodinia [139] benchmark suites that we use for evaluation. These benchmarks are scientific (e.g., LBM, KM, and MRIG) and multimedia (e.g., SIFT, TRCK, and DISP) workloads. Table 7 also recaps the shuffling overhead of CGRAs' input data relative to the execution time of the benchmarks running on the baseline architecture in Section 5.4.3. 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.

Table 7. Benchmarks used in our evaluation

Name	Description	# of Kernels	Iterative?	Replaced Exec. Time	Shuffling Overhead
BP	Back propagation [139]	4	Yes	99.9%	1.6%
DISP	Disparity map [136]	10	No	99.9%	0.1%
HACC	Hardware accelerated cosmology code (N-body) [137]	2	Yes	99.9%	1.5%
HS	Hotspot [139]	2	Yes	99.9%	1.1%
KM	K-means clustering [139]	1	Yes	99.9%	0.7%
LBM	Lattice-Boltzmann method fluid dynamics [111]	1	Yes	99.9%	1.3%
MRIG	Magnetic resonance imaging cartesian gridding [111]	1	No	98.5%	0.14%
OCN	Ocean movements [138]	16	Yes	81.7%	1.6%
SIFT	Scale-invariant feature transform [136]	4	No	92.7%	0.1%
SRAD	Speckle reducing anisotropic diffusion [139]	3	Yes	99.9%	0.2%
TRCK	Feature tracking [136]	8	No	79.3%	2.0%

5.4.2 Architectural Components

CGRAs: Functional units (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 multiplication and division. Table 8 summarizes the details of a CGRAs with 32 FUs and 64 FUs used in this dissertation. 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 [105] and

synthesized them using the Synopsys design compiler 2013 and TSMC 40 nm low-power standard-cell library. We optimized them for two clock frequencies of 400 MHz and 800 MHz. Based on Table 8, we estimate that the area of a 64-FU CGRA is ~0.832 mm^2 in 40 nm technology targeted at 800 MHz. Moreover, based on our estimates using McPAT [95], the area of a CGRA-side MC is ~0.21 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. Moreover, CGRAs are stacked on top of DRAM devices (not bottom). Thus, they can dissipate heat without passing the DRAM die. Thus, we believe that CGRAs have minimal thermal impact on a DRAM device. Note that the power gird of existing modern DRAM dies would further reduce CGRA temperature variations by acting as a heat spreader.

Table 8. Design parameters of 32-bit CGRA with 64 and 32 FUs

Unit	Latency (Cycles)	Count		Area (μm^2)		Energy per Operation (pJ)		Functionality
		32-FU CGRA	64-FU CGRA	Optimized for 400 MHz	Optimized for 800 MHz	Optimized for 400 MHz	Optimized for 800 MHz	
INT ALU	1	20	40	2997	3589	2.1	2.2	Int. Add/Sub/Comp/ Logic/Shift
FP ALU	5			3830	4762	4.4	7.1	FP Add/Sub/Fp2Int /Int2Fp/Comp
INT MUL	2	10	20	6270	6507	12.6	13.1	Integer multiplicator
FP MUL	4			5630	6565	10.0	11.3	Floating-point multiplicator
INT DIV	8	2	4	10072	10596	27.6	30.1	Integer divider
FP DIV	9			13404	15484	24.4	27.7	Floating-point divider
Switch	-	45	81	4236 [40]		1.11		-

Processor: This proposal does not depend on a specific ISA or processor architecture. We use a four-way OoO processor with a 16-stage execution pipeline. Table 9 shows the key architecture parameters of the processor and its caches.

Table 9. Key processor architecture parameters

Fetch/Decode/Dispatch/ Issue/Retire width	4/4/4/5/4	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

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 between the MC and DRAM devices. Both interfaces have 8 banks per rank. However, ×8 and ×16 DRAM devices have a different row buffer size of 1KB and 2KB per device, respectively [140]. Table 10 contains detailed timing parameters of DRAM subsystem and the MC used in our evaluation.

Table 10. DRAM subsystem parameters

DDR3-1600-11-11-11-28 [140]	**Symbol**	**Latency (ns)**
Activate to read or write delay	tRCD	13.75
Column access latency	tCL	13.75
Row precharge command period	tRP	13.75
Activate to precharge command period	tRAS	35.0
Read to read command delay	tCCD	5.0
Write to read delay	tWTR	7.5
Write Recovery	tWR	15.0
Minimum time between a read command and a precharge command	tRTP	7.5
Read to write delay	tRTW	2.5
Activate to activate delay	tRRD	6.25
Four activate windows	tFAW	40.0
Burst transfer delay	tBURST	5.0
Refresh cycle time	tRFC	300.0
Refresh command interval	tREFI	7800.0
Device row buffer size	1KB/2KB for ×8/×16	

Memory Management	**Description**
Address mapping	Page interleaving
Row policy	Open page
Processor-side memory controller	40/40-entry read/write request queues, FR-FCFS scheduler

5.4.3 Evaluated Architectures

We compare the performance of the NDA architectures against a diverse set of architectures. Table 11 summarizes different architectures that we use for evaluation in this proposal. In the "Baseline" architecture, we use a conventional four-way processor (Table 9) running at 2GHz and DDR3-1600 memory subsystem (Table 10) without any CGRAs. In the "Base 4×Para" architecture, we assume that the performance of 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 the "Base+CGRAs" architecture, CGRAs are integrated with the processor [37]–[39], [41], [54], [141]; CGRAs transfer data through the off-chip interconnects between the processor and DRAM devices. In the NDA architectures, DRAM devices are coupled with CGRAs to enable near-memory processing. In the NDA architectures, four CGRAs are stacked atop each ×8 DRAM device to provide the same total number of CGRAs as the "Base+CGRAs" architecture. In all the NDA architectures (NDA-1, NDA-2, and NDA-3), 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.

5.5 Results

In this section, we evaluate performance and energy-efficiency of the NDA architectures. Unless otherwise indicated, we use ×8 DRAM devices and CGRAs with 64 FUs at 800 MHz. We consider both dynamic and leakage energy of the processor, DRAM devices, and CGRAs. While CGRAs are operating, the processor runs at $1/16^{th}$ of its operating frequency 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). Future work can study using idle C-states in the processor core when CGRAs are operating.

5.5.1 Data Transfer (Movement) Energy

NDA improves data transfer energy by performing computation near off-chip DRAM devices. Figure 23 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 the "Base+CGRAs" architecture, 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 [4].

Table 11. Summary of different architectures evaluated in this chapter

Abbreviation	# CGRAs	Description
Base	-	4-way OoO processor at 2GHz (Table 9) and DDR3-1600 ×8 memory (Table 10)
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

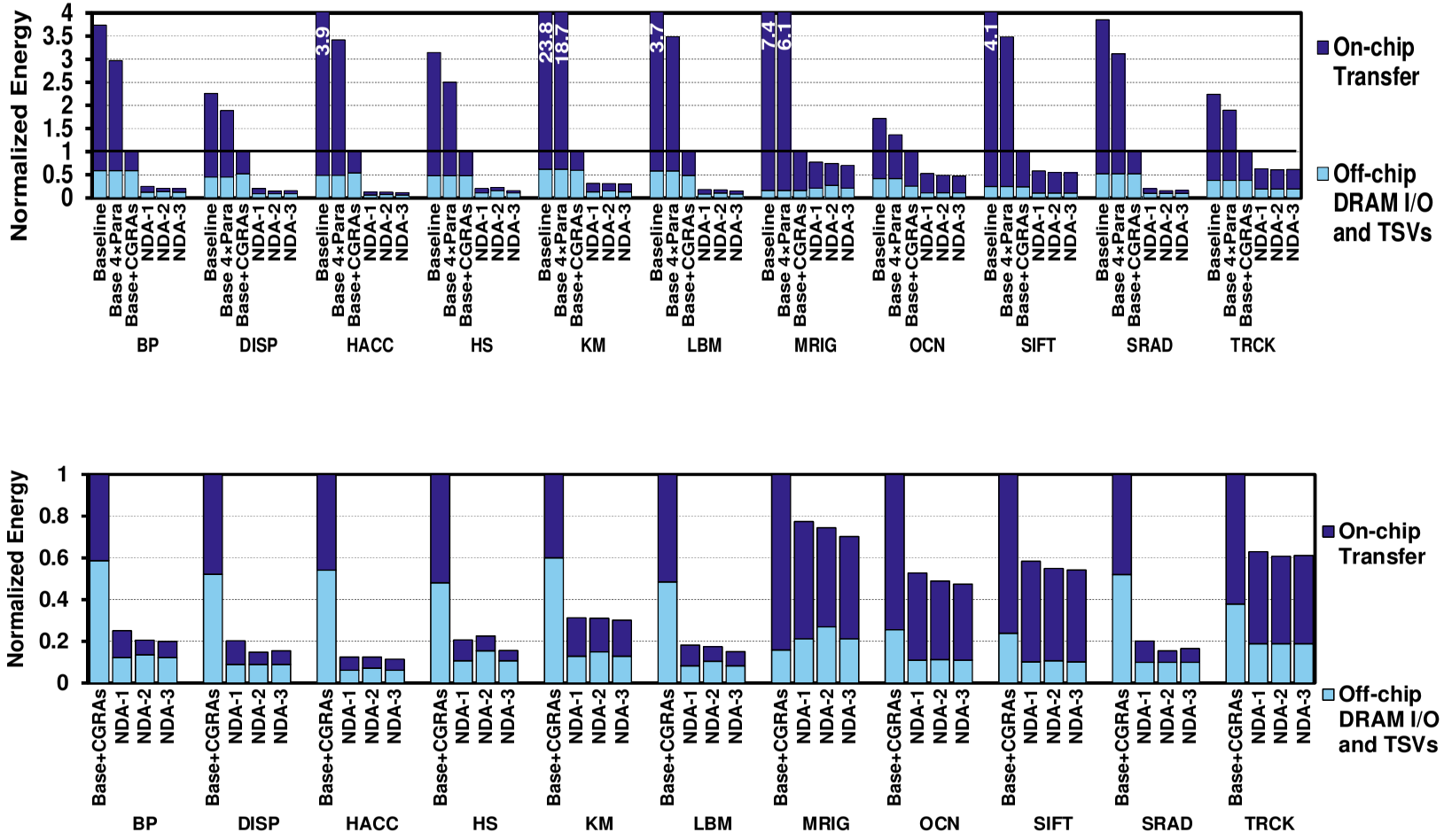


Figure 21. On- and off-chip data movement energy for different architectures normalized to that of the "Base+CGRAs" architecture.

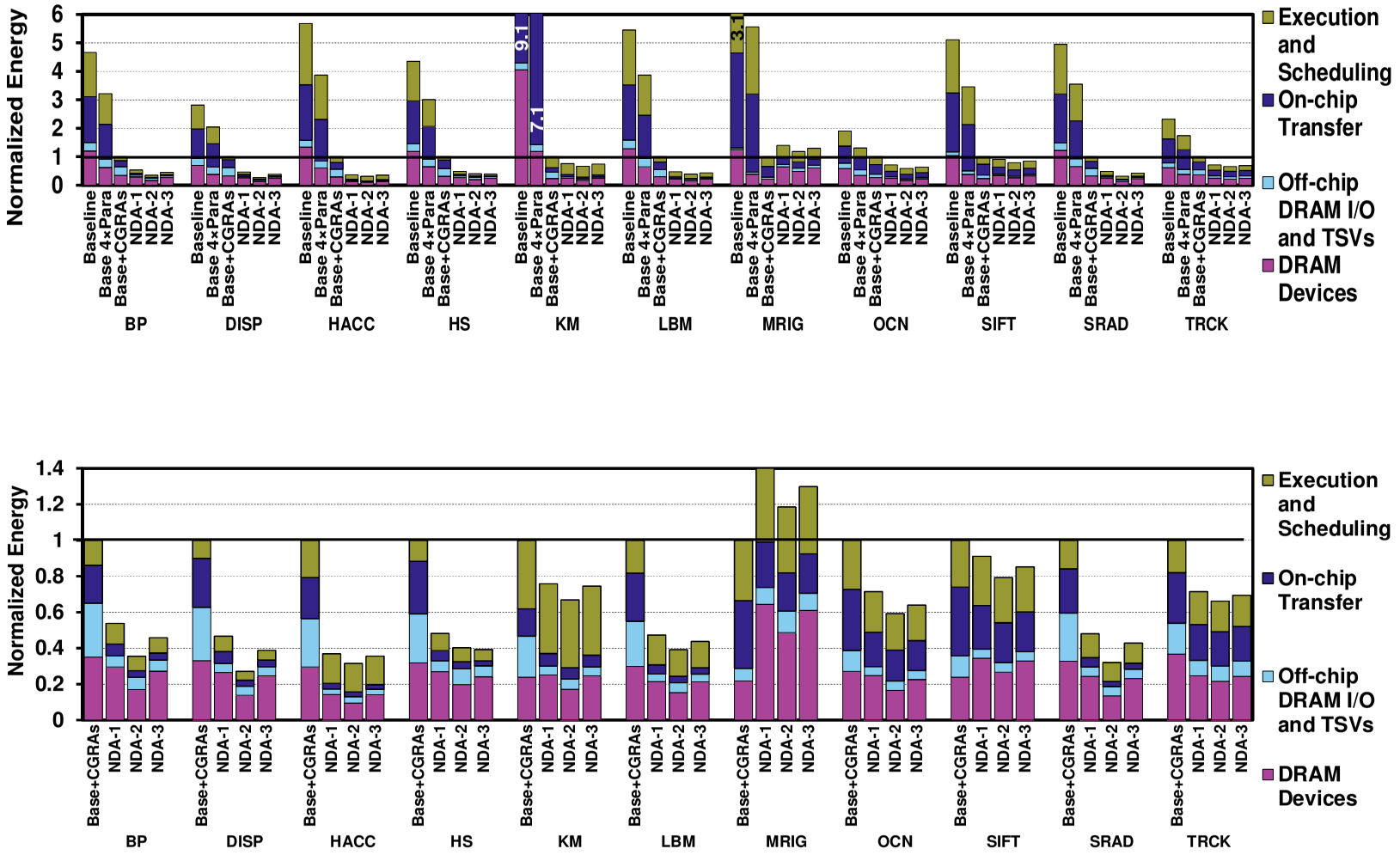


Figure 22. Combined processor, DRAM, and CGRA energy consumption for different architectures normalized to that of the "Base+CGRAs" architecure.Figure 23. On- and off-chip data movement energy for different architectures normalized to that of

5.5.2 Total Energy

NDA improves total energy consumption by exploiting energy-efficient CGRAs near DRAM devices. Figure 22 shows the total energy dissipation breakdown of baseline and NDA for all the evaluated benchmarks. We categorize the energy dissipation into four components. "scheduling and execution" 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.

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 absence of the overhead of fetching and scheduling instructions in the processor for kernel executions, NDA using CGRAs processes data 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 accelerate application execution (See Section 2.1.1 for more details on CGRAs). Reduced execution time reduces leakage energy consumption. Finally, NDA considerably decreases DRAM device energy because CGRAs generally have more well-defined and regular memory access patterns compared to processors [10], [142]. This leads to higher DRAM page (row buffer) hit rate and thus lower DRAM activate energy. Only for MRIG and SIFT, NDA incurs 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 fully be filtered out by CGRA's internal registers. For such applications, the processor's on-chip cache hierarchy can reduce DRAM accesses.

5.5.3 Speedup Comparison

NDA also improves performance of target applications by using parallel and accelerated processing. Figure 25 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 operate independently from other CGRAs, Thus, each CGRAs' DRAM accesses do not interfere with other CGRAs' accesses as each CGRA has access only to its local DRAM device. 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 25 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.

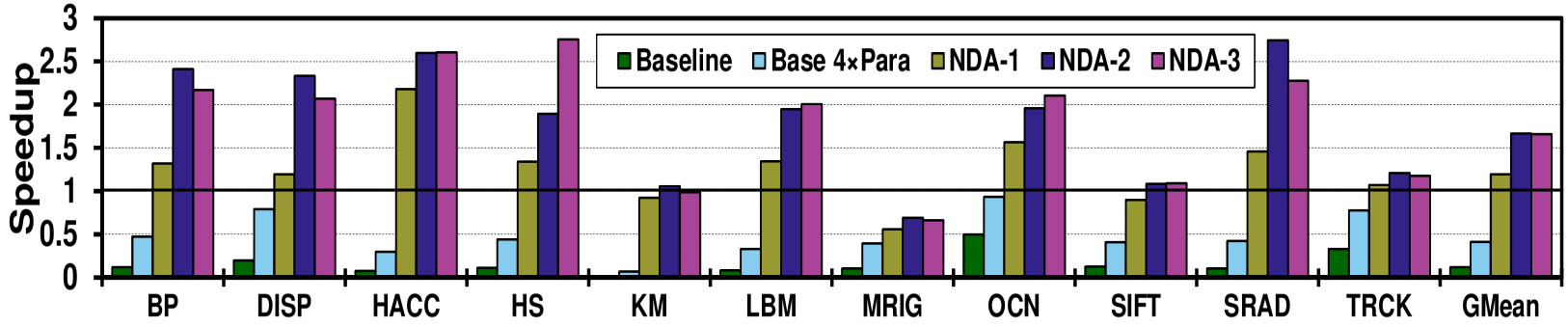


Figure 25. Performance comparison of different architectures normalized to the "Base+CGRAs" architecture.

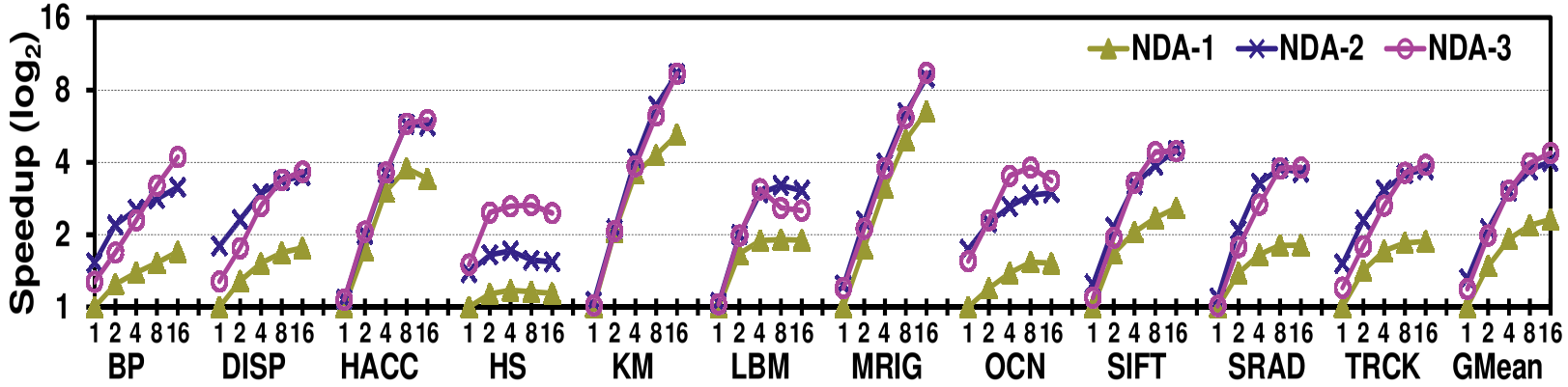


Figure 26. Performance of different NDA approaches to connect CGRAs and DRAMs with varying the number of stacked CGRAs (*x* axis) normalized to performance of NDA-1 with one CGRA stacked atop a DRAM. CGRAs have 64 FUs running at 800 MHz and are stacked atop ×8 DRAM devices.

5.5.4 Sensitivity to CGRA-DRAM Connection Type

Having different memory bandwidth characteristics, our three NDA microarchitectures lead to different performance results. Figure 26 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 26 includes only accelerated kernel execution since CGRA-DRAM connection type has no impact on the processor performance. Figure 26 indicates that performance of some benchmarks such as KM scales well with the number of CGRAs while performance of other benchmarks such as HS caps out with more CGRAs due to memory bandwidth limitations.

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 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 improvement of NDA-3.

5.5.5 Sensitivity to CGRA Computing Capability

We evaluate the impact of the number of FUs and frequency of CGRAs on the accelerated kernel execution time. For this evaluation, we use CGRAs containing 32 and 64 FUs running at 400MHz and 800MHz. Figure 27 compares the performance of NDA-1 using CGRAs with various computing capability. For this evaluation, only one CGRA is stacked atop each DRAM

device. Figure 27 includes only accelerated kernel execution since CGRA's computing capability has no impact on the processor performance. 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.

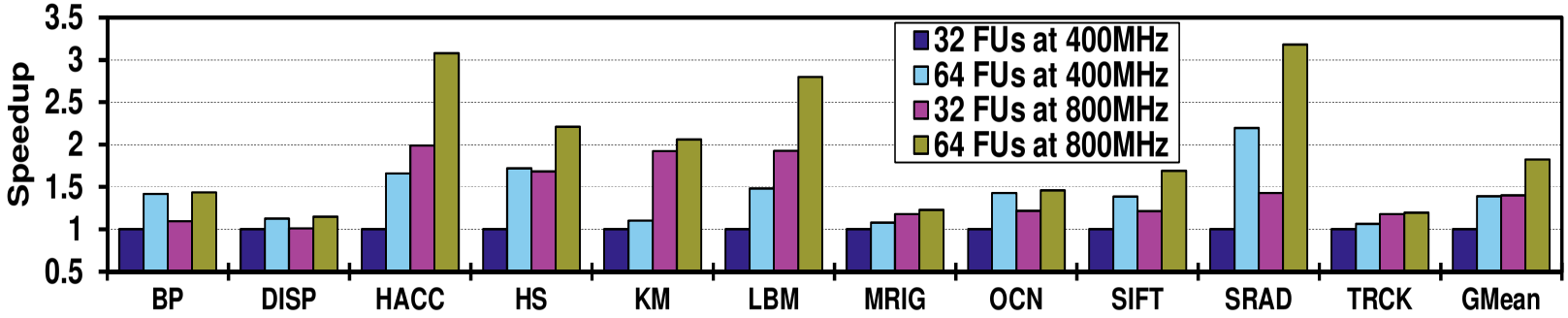


Figure 27. Performance of the NDA-1 ×8 architecture by varying the frequency of CGRAs and number of functional units in CGRAs (normalized to performance of NDA-1 ×8 using 32-FU CGRAs at 400MHz). One CGRA is stacked atop each ×8 DRAM.

5.5.6 NDA×8 versus ×16

As explained in Section 5.2.1, 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, some benchmarks such HS and SRAD perform 44% and 22% better on NDA×8 versus NDA×16.

5.6 Summary

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 our NDA architectures significantly improve performance of a wide range of evaluated applications by 3.2-60.6× and reduce their total energy consumption by 63-96% over an aggressive out of order processor. Compared to a computing system integrating the same accelerator logic inside the processor, our NDA architectures decrease data transfer

energy by 64%-68% and increase performance by 19%-67%, respectively. We show that NDA architectures increase DRAM area by only 0.2%-3.3% with no negative impact on DRAM timing.

6 Conclusion

Reconfigurable architectures have been studied in the past, but energy concerns make researchers revisit architectures based on coarse-grain reconfigurable accelerators with fresh eyes. Heterogeneous architectures that take advantage of dark silicon to realize accelerators are highly expected in the near future. Reconfigurable accelerators use currently cheap silicon area to energy-efficiently speed up a variety of application kernels. The dissertation has investigated opportunities to provide energy-efficient computing using heterogeneous reconfigurable architectures. Our investigations have led to two primary proposals for heterogeneous reconfigurable architectures that use accelerators for data processing. First, we have proposed new data cache designs for systems that interface reconfigurable accelerators with the cache hierarchy. Second, we have also proposed a novel near-memory memory architecture to enable energy-efficient computing by stacking reconfigurable accelerators on top of commodity DRAM devices.

In Chapter 4, we have evaluated a wide range of cache designs in embedded systems that couple a processor and a coarse-grain reconfigurable accelerator on a single chip through a shared-memory cache hierarchy and have found that configurable cache architectures can significantly improve energy-efficiency by varying cache requirements across target applications. Therefore, we have investigated a configurable cache organization that allows shared and private L1 data caches to exist in the same architecture. Furthermore, we have investigated a novel cache design that provides a configurable tradeoff between cache capacity and cache bandwidth.

In Chapter 5, we have proposed a near-memory processing architecture that we call NDA (Near-DRAM Acceleration). The NDA architecture takes commodity “2D” DRAM devices (not existing 3D DRAM) and stacks low-power, low-area, and flexible coarse-grain reconfigurable

accelerators atop DRAM. Therefore, the NDA architecture extends a conventional system by accelerator-enabled DRAM devices to process data near where it is stored. In the NDA architecture, the accelerators can directly access data from their corresponding DRAM device and provides, in a practical way, high-bandwidth connections between the accelerators and DRAM devices only for the purpose of near-memory processing. The NDA architecture promises a new local data processing approach that significantly reduces data movement between storage units and processing units (i.e., data movements from the off-chip DRAM through on-chip caches, buses, and register file). We believe that using commodity DRAM devices provides a cost-effective approach for reducing data movement energy in the short run.

In this dissertation, we have shown that the proposed NDA architecture can significantly reduce energy, while achieving high speedups compared to conventional systems. We have also demonstrated that the NDA architecture does not require design changes to the processor(s), the processor-memory interface, or the underlying architecture of commodity DRAM devices, apart from the additional TSVs. This is particularly attractive to DRAM manufacturers looking to provide added value to their products without significant rework or requiring cooperation from large microprocessor companies.

We have proposed three microarchitectures in the NDA architecture to provide high-bandwidth connection between reconfigurable accelerators and DRAM devices by taking advantage of high internal bandwidth of DRAM devices. Higher memory bandwidth could potentially improve the performance of data-intensive kernels running on the accelerators. The off-chip bandwidth of DRAM devices are constrained by their off-chip I/O bus width and frequency. However, DRAM devices are essentially multi-banked devices with wide row buffers that store the most recently accessed data. This means that DRAM devices could provide much higher bandwidth if not constrained by requirements of their off-chip buses. We break this

constraint by proposing microarchitectures that intercept data from DRAM's internal buses and transfer data between DRAM and accelerators over TSVs. Our proposed microarchitectures increase the DRAM-accelerator bandwidth with little area overhead on DRAM devices.

6.1 Future Directions

During the course of this desertion research, we have discovered future research directions that have not been studied deep enough by the academia or industry. Currently, near-memory processing is a new avenue of research that certainly requires more studies.

Virtual memory, coherence, consistency issues in near-memory processing architectures: Although we have considered and discussed these issues in the NDA architecture, more thorough investigation could be done as future work. The accelerators in the NDA architecture use physical addresses to access data from memory. Using physical addresses raises permission (read or write) and security challenges. Input-output memory management units (IOMMUs) [143] could potentially address permission and security challenges in near-memory processing architectures. Compute engines near-memory could employ IOMMUs to enable virtual memory addressing and access permissions for compute engines. Efficient integration of IOMMUs with near-memory compute-engines, their impacts on the design cost and performance, and their benefits and drawbacks in near-memory processing architectures could be studied thoroughly.

In the NDA architecture, the cache coherence and consistency model of the host processor are left unaltered. We use un-cacheable regions of memory for CGRA address space that require no changes to the host processor. The programmer specifies un-cacheable regions and is responsible to maintain coherence. A study investigating hardware coherence techniques could be performed.

Moreover, memory consistency could also be revisited to enable concurrent accesses of processor and CGRAs to memory.

Near-memory processing using non-commodity DRAM devices: Our NDA architecture extends the traditional systems to enable near-memory processing using commodity DRAM devices. However, we believe other forms of near-memory architectures could be studied in the future. Non-commodity DRAM stacks such as HMC [90] and Wide I/O 2 [144] are inherently aimed at 3D architectures. These DRAM stacks target higher bandwidth and lower power consumption which are the fundamental bottlenecks of commodity DDR devices. This makes HMC and Wide I/O 2 a good fit for near-memory processing provided their manufacturing cost plummets and they enjoy wide adoption by consumers.

Near-memory processing on clusters: The NDA architecture uses a single host processor to control the flow of execution on accelerator-enabled DRAM ranks. A researcher could investigate architectures for a cluster of nodes in which each node contains a host processor and compute-engines near memory. In this cluster, the host processors not only control the execution on their near-memory compute engines, they also manage data transfers between nodes. This cluster could be used for processing large distributed datasets and in-memory databases. One step further, the host processors can be removed to reduce the cluster cost. In this case, the near-memory compute engines should be enabled (e.g. by using low-power embedded cores) to perform inter-node communication as well as execution control.

Compute-engines for near-memory processing: In the NDA architecture, we used CGRAs for near-memory processing. One could investigate using other types of compute-engines such as GPUs [31], [145], SIMD units [30], low-power cores [92], [146], [147], FPGAs, ASICs [34], and even heterogeneous cores [148] for target applications. Besides performance and energy-efficiency, their thermal impacts and ease of programmability could also be studied.

Programming model for near-memory processing: A programming model helps programmers exploit the hardware platform and energy efficiency benefits of new architectures and makes programming easier by abstracting architectural details. A good programming model could significantly improve the adoption of near-memory processing architectures. Low-level APIs to access CGRAs (loading a kernel, sending parameters, checking status, etc.) could be studied as they pave the way for high-level programming models. Next, data structures for near-memory processing, data partitioning among CGRAs, labor division between processor and CGRAs, data communication between CGRAs and the processor, etc. could be investigated [25], [82]. High-level programming models should make programming of the heterogeneous and distributed near-memory processing architectures easier for general programmers.

Interconnects for near-memory processing: To enable data processing in compute engines of the logic layer, data should first be transferred from memory dies to the logic layer and then from the logic layer to target compute engines. In this dissertation, we have focused on microarchitectures to transfer data between the DRAM die and logic layer. However, on-die interconnect networks to route data within a logic layer are left as future work. An interconnect network in a logic layer manages data transfers among on-die compute engines as well as data transfers between the memory controller(s) in the logic layer and compute engines [149], [150]. The other scope of research could be design exploration for off-chip interconnects that manage data transfers within a cluster of NDA-enabled nodes. An example of this type of interconnect is inter-HMC networks [151].

In-SSD Processing: Near-memory processing architectures promise a new approach for improving energy efficiency of a wide range of big-data workloads. However, there exists an important list of applications that cannot be efficiently mapped to near-memory architectures, (or NDA in particular). An example of these application kernels are scan (filter) workloads which are

used in database [152], search [153], and biomedical [154] application domains. These applications usually have regular data flow, low control complexity, and large working sets that cannot be stored in physical memory. In these kernels, the streaming data is filtered or preprocessed to generate a reduced set of data that is processed more thoroughly. Due to the data filtering nature of these kernels, the high shuffling overhead of these kernels prevents their efficient implementations on the NDA architecture (unless the input set is stored in a shuffled fashion). However, these kernels can benefit from processing in Solid State Drives (SSDs). Recently, there have been some preliminary proposals to enable data processing in SSDs [155]–[160], but more investigations could lay a foundation for practical in-SSD processing.

7 Publications

1. **A. Farmahini-Farahani**, J. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM Acceleration Architecture leveraging Commodity DRAM Devices and Standard Memory Modules," accepted for publication in *High-Performance Computer Architecture (HPCA)*, 2015.
2. **A. Farmahini-Farahani**, J. Ahn, K. Morrow, and N. S. Kim, "DRAMA: An Architecture for Accelerated Processing near Memory," *Computer Architecture Letters (CAL)*, 2014.
3. **A. Farmahini-Farahani**, N. S. Kim, K. Morrow, "Energy-Efficient Reconfigurable Cache Architectures for Accelerator-Enabled Embedded Systems," in *IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2014, pp. 211-220.
4. P. Aguilera, J. S. Lee, **A. Farmahini-Farahani**, N. S. Kim, K. Compton, "Process Variation-aware Workload Partitioning Algorithms for GPUs Supporting Spatial Multitasking," accepted for publication in *IEEE/ACM Design Automation and Test in European (DATE)*, Mar. 2014, pp. 1-6.
5. **A. Farmahini-Farahani**, H. Duwe, M. Schulte, and K. Compton, "Modular Design of High-throughput, Low-latency Sorting Units," *IEEE Trans. on Computers*, vol. 62, no. 7, pp. 1389-1402, July 2013.
6. P, Klabbers, M. Bachtis, J. Brooke, M. Cepeda Hermida, K. Compton, S. Dasu, **A. Farmahni-Farahani**, S. Fayer, R. Fobes, R. Frazier, C. Ghabrous, T. Gorski, A. Gregerson, G. Hall, C. Hunt, G. Iles, J. Jones ,C. Lucas, R. Lucas, M. Magrans, D. Newbold, I. Oljavo, A. Perugupalli, M.Pioppi, A. Rose, I. Ross, D. Sankey, M. Schulte, D. Seemuth,W.H. Smith, J.Tikalsky, A. Tapper, and T. Williams, "CMS level-1 upgrade calorimeter trigger prototype development," *J. of Instrumentation*, vol. 8, no. 2, Feb. 2013.

7. K. Compton, S. Dasu, **A. Farmahini-Farahani**, S. Fayer, R.Fobes, R. Frazier, T. Gorski, G. Hall, G. Iles, J. Jones, P. Klabbers, D. Newbold, A. Perugupalli, S. Rader, A. Rose, D. Seemuth, W. Smith, and J. Tikalsky, "The MP7 and CTP-6: multi-hundred Gbps processing boards for calorimeter trigger upgrades at CMS," *J. of Instrumentation*, vol. 7, no. 12, Dec. 2012.
8. P. Klabbers, T.Gorski, M. Bachtis, K.Compton, S.Dasu, **A. Farmahini-Farahani**, R. Fobes, A. Gregerson, M. Grothe, I. Ross, D. Seemuth, M. Schulte, and W.H. Smith, "CMS Calorimeter Trigger Phase I upgrade," *J. of Instrumentation*, vol. 7, no. 1, Jan. 2012.
9. **A. Farmahini-Farahani**, A. Gregerson, M. Schulte, and K. Compton, "Modular High-throughput and Low-latency Sorting Units for FPGAs in the Large Hadron Collider," in *Proc. IEEE Intl. Conf. on Application Specific Processors (SASP)*, San Diego, CA, June 2011, pp. 38-45 (Nominated for best paper award).
10. **A. Farmahini-Farahani**, C. Tsen, and K. Compton, "FPGA Implementation of a 64-Bit BID-Based Decimal Floating-Point Adder/Subtractor," in *Proc. IEEE Intl. Conf. on Field-Programmable Technology (FPT)*, Sydney, Australia, Dec. 2009, pp. 518-521.
11. A. Gregerson, **A. Farmahini-Farahani**, W. Plishker, Z. Xie, K. Compton, S. Bhattacharyya, and M. Schulte, "Advances in Architectures and Tools for FPGAs and their Impact on the Design of Complex Systems for Particle Physics," in *Typical Workshop on Electronics for Particle Physics (TWEPP)*, Sep. 2009, Paris, France, pp. 617-626.
12. A. Gregerson, **A. Farmahini-Farahani**, B. Buchli, S. Naumov, M. Bachtis, K. Compton, M. Schulte, W. Smith, and S. Dasu, "FPGA Design Analysis of the Clustering Algorithm for the CERN Large Hadron Collider," in *Proc. IEEE Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, Apr. 2009, pp. 19-26.

8 References

[1] M. Taylor, “Is dark silicon useful?,” in *Design Automation Conference*, 2012, pp. 1131–1136.

[2] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, “QsCores: trading dark silicon for scalable energy efficiency with quasi-specific cores,” in *IEEE/ACM Intl. Symp. on Microarchitecture*, 2011, pp. 163–174.

[3] 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.

[4] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, “Understanding sources of inefficiency in general-purpose chips,” in *Intl. Symp. on Computer Architecture*, 2010, pp. 37–47.

[5] P. Garcia and K. Compton, “Shared memory cache organizations for reconfigurable computing systems,” in *Field Programmable Custom Computing Machines*, 2009, pp. 239–242.

[6] W. J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield, “Efficient Embedded Computing,” *Computer (Long. Beach. Calif).*, vol. 41, no. 7, pp. 27–32, 2008.

[7] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, “GPUs and the Future of Parallel Computing,” *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep. 2011.

[8] S. Borkar, “Role of Interconnects in the Future of Computing,” *J. Light. Technol.*, vol. 31, no. 24, pp. 3927–3933, Dec. 2013.

[9] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store: A shared memory framework for accelerator-based systems," *ACM Trans. Arch. Code Optim.*, vol. 8, no. 4, pp. 48:1–48:22, 2012.

[10] R. Hou, L. Zhang, M. Huang, K. Wang, H. Franke, Y. Ge, and X. Chang, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *High Performance Computer Architecture*, 2011, pp. 312–320.

[11] M. F. Deering, S. A. Schlapp, and M. G. Lavelle, "FBRAM: a new form of memory optimized for 3D graphics," in *Computer Graphics and Interactive Techniques*, 1994, pp. 167–174.

[12] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, and G. Daglikoca, "The architecture of the DIVA processing-in-memory chip," in *Intl. Conf. on Supercomputing*, 2002, pp. 14–25.

[13] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "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.

[14] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. Mckenzie, "Computational RAM: Implementing processors in memory," *IEEE Des. Test Comput.*, vol. 16, no. 1, pp. 32–41, 1999.

[15] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park, "Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture," in *Conf. on Supercomputing*, 1999.

[16] D. Keen, V. Lam, P. Pattnaik, and J. Torrellas, “FlexRAM: Toward an advanced intelligent memory system,” in *Intl. Conf. on Computer Design: VLSI in Computers and Processors*, 1999, pp. 192–201.

[17] P. M. Kogge, S. C. Bass, J. B. Brockman, D. Z. Chen, and E. Sha, “Pursuing a petaflop: point designs for 100 TF computers using PIM technologies,” in *Symp. on the Frontiers of Massively Parallel Computation*, 1996, pp. 88–97.

[18] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick, “Scalable processors in the billion-transistor era: IRAM,” *Computer (Long. Beach. Calif).*, vol. 30, no. 9, pp. 75–78, 1997.

[19] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, “Smart Memories: a modular reconfigurable architecture,” in *Intl. Symp. on Computer Architecture*, 2000, pp. 161–171.

[20] M. Oskin, F. T. Chong, and T. Sherwood, “Active Pages: A computation model for intelligent memory,” in *Intl. Symp. on Computer Architecture*, 1998, pp. 192–203.

[21] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, “A case for intelligent RAM,” *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.

[22] T. L. Sterling and H. P. Zima, “Gilgamesh: A multithreaded processor-in-memory architecture for petaflops computing,” in *Supercomputing*, 2002, pp. 1–23.

[23] G. H. Loh, N. Jayasena, M. Oskin, M. Nutter, D. Roberts, M. Meswani, D. P. Zhang, and M. Ignatowski, “A processing in memory taxonomy and a case for studying fixed-function PIM,” in *Workshop on Near-Data Processing*, 2013.

[24] D. Patterson, K. Asanovic, A. Brown, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, C. Kozyrakis, D. Martin, S. Perissakis, R. Thomas, N. Treuhaft, and K. Yelick, “Intelligent

RAM (IRAM): the industrial setting, applications, and architectures," in *Intl. Conf. on Computer Design*, 1997, pp. 2–7.

[25] M. Chu, N. Jayasena, D. P. Zhang, and M. Ignatowski, "High-level programming model abstractions for processing in memory," in *Workshop on Near-Data Processing*, 2013.

[26] R. G. Dreslinski, D. Fick, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Centip3De: A 64-Core, 3D Stacked Near-Threshold System," *IEEE Micro*, vol. 33, no. 2, pp. 8–16, Mar. 2013.

[27] D. H. Kim, S. Mukhopadhyay, and S. K. Lim, "Through-silicon-via aware interconnect prediction and optimization for 3D stacked ICs," in *Intl. workshop on System level interconnect prediction*, 2009, pp. 85–92.

[28] B. Black, "Die stacking is happening (keynote)," in *Intl. Symp. on Microarchitecture*, 2013.

[29] D. H. Kim, K. Athikulwongse, M. Healy, M. Hossain, M. Jung, I. Khorosh, G. Kumar, Y.-J. Lee, D. Lewis, T.-W. Lin, C. Liu, S. Panth, M. Pathak, M. Ren, G. Shen, T. Song, D. H. Woo, X. Zhao, J. Kim, H. Choi, G. Loh, H.-H. Lee, and S. K. Lim, "3D-MAPS: 3D massively parallel processor with stacked memory," in *IEEE Intl. Solid-State Circuits Conference*, 2012, pp. 188–190.

[30] D. H. Woo, H.-H. S. Lee, J. B. Fryman, A. D. Knies, and M. Eng, "POD: A 3D-Integrated Broad-Purpose Acceleration Layer," *IEEE Micro*, vol. 28, no. 4, pp. 28–40, Jul. 2008.

[31] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: throughput-oriented programmable processing in memory," in *Intl. Symp. on High Performance Parallel and Distributed Computing*, 2014.

[32] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti, "A 3D-stacked logic-in-memory accelerator for application-specific data intensive computing," in *IEEE Intl. 3D Systems Integration Conf.*, 2013, pp. 1–7.

[33] R. Sampson, M. Yang, S. Wei, C. Chakrabarti, and T. F. Wenisch, "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.

[34] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating sparse matrix-matrix multiplication with 3D-stacked logic-in-memory hardware," in *IEEE High Performance Extreme Computing Conf.*, 2013, pp. 1–6.

[35] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep. 2012.

[36] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An Overview of Reconfigurable Hardware in Embedded Systems," *EURASIP J. Embed. Syst.*, vol. 2006, no. 1, pp. 1–19, Jan. 2006.

[37] B. Mei, S. Vernalde, D. Verkest, H. Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *Field Programmable Logic and Application*, 2003, pp. 61–70.

[38] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Reed Taylor, "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.

[39] E. Mirsky and A. DeHon, “MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources,” in *FPGAs for Custom Computing Machines*, 1996, pp. 157–166.

[40] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, “Dynamically specialized datapaths for energy efficient computing,” in *High Performance Computer Architecture*, 2011, pp. 503–514.

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

[42] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, “CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit,” in *Intl. Symp. on Computer Architecture*, 2000, pp. 225–235.

[43] M. Mishra, T. J. Callahan, T. Chelcea, G. Venkataramani, S. C. Goldstein, and M. Budiu, “Tartan: evaluating spatial computation for whole program execution,” in *Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 163–174.

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

[45] T. Callahan, J. Hauser, and J. Wawrzynek, “The Garp architecture and C compiler,” *Computer (Long. Beach. Calif).*, vol. 33, no. 4, pp. 62–69, 2000.

[46] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, “A general constraint-centric scheduling framework for spatial architectures,” in *Conf. on Programming Language Design and Implementation*, 2013, pp. 495–506.

[47] 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.

[48] A. Parashar, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lustig, V. Pavlov, A. Zhai, M. Gambhir, A. Jaleel, R. Allmon, R. Rayess, S. Maresh, and J. Emer, "Efficient Spatial Processing Element Control via Triggered Instructions," *IEEE Micro*, vol. 34, no. 3, pp. 120–137, May 2014.

[49] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu, "Elastic CGRAs," in *Intl. Symp. on Field Programmable Gate Arrays*, 2013, pp. 171–180.

[50] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications," in *Intl. Conf. on Field-Programmable Technology*, 2012, pp. 329–334.

[51] F. Veredas, M. Scheppler, and W. Moffat, "Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes," in *Intl. Conf. on Field Programmable Logic and Applications*, 2005, pp. 106–111.

[52] W.-J. Lee, S.-O. Woo, K.-T. Kwon, S.-J. Son, K.-J. Min, G.-J. Jang, C.-H. Lee, S.-Y. Jung, C.-M. Park, and S.-H. Lee, "A scalable GPU architecture based on dynamically reconfigurable embedded processor," in *High Performance Graphics, Posters*, 2011.

[53] N. R. Miniskar, P. S. Gode, S. Kohli, and D. Yoo, "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.

[54] A. Parashar, A. Jaleel, R. Allmon, R. Rayess, S. Maresh, J. Emer, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lustig, V. Pavlov, A. Zhai, and M. Gambhir, "Triggered instructions: a control paradigm for spatially-programmed architectures," in *Intl. Symp. on Computer Architecture*, 2013, pp. 142–153.

[55] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, “Continuous real-world inputs can open up alternative accelerator designs,” in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 1–12.

[56] S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, *Handbook of Signal Processing Systems, Volume 2*. Springer Science & Business, 2013, p. 1420.

[57] A. Carroll, S. Friedman, B. Van Essen, A. Wood, B. Ylvisaker, C. Ebeling, and S. Hauck, “Designing a Coarse-grained Reconfigurable Architecture for Power Efficiency,” *Dep. Energy NA-22 Univ. Inf. Tech. Interchang. Rev. Meet.*, 2007.

[58] J. L. Tripp, J. Frigo, and P. Graham, “A Survey of Multi-Core Coarse-Grained Reconfigurable Arrays for Embedded Applications,” 2008.

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

[60] J. Kelm and S. Lumetta, “HybridOS: Runtime support for reconfigurable accelerators,” in *Field Programmable Gate Arrays*, 2008, pp. 212–221.

[61] E. Caspi, M. Chu, Y. Huang, J. Yeh, Y. Markovskiy, A. Dehon, and J. Wawrzynek, “Stream computations organized for reconfigurable execution (SCORE): Introduction and tutorial,” in *Field-Programmable Logic and Applications*, 2000, pp. 605–614.

[62] P. Garcia and K. Compton, “A scalable memory interface for multicore reconfigurable computing systems,” in *Field-Programmable Technology*, 2011, pp. 1–8.

[63] P. Garcia and K. Compton, “A reconfigurable hardware interface for a modern computing system,” in *Field-Programmable Custom Computing Machines*, 2007, pp. 73–84.

[64] T. Oguntebi, S. Hong, J. Casper, N. G. Bronson, C. Kozyrakis, and K. Olukotun, “FARM: A prototyping environment for tightly-coupled, heterogeneous architectures,” in *Field-Programmable Custom Computing Machines*, 2010, pp. 221–228.

[65] A. Putnam, S. Eggers, D. Bennett, E. Dellinger, J. Mason, H. Styles, P. Sundararajan, and R. Wittig, "Performance and power of cache-based reconfigurable computing," in *Intl. Symp. on Computer Architecture*, 2009, pp. 395–405.

[66] J. Choi, K. Nam, A. Canis, J. Anderson, S. Brown, and T. Czajkowski, "Impact of cache architecture and interface on performance and area of FPGA-based processor/parallel-accelerator systems," in *Field-Programmable Custom Computing Machines*, 2012, pp. 17–24.

[67] K. Wilson and K. Olukotun, "Designing high bandwidth on-chip caches," in *Intl. Symp. on Computer Architecture*, 1997, pp. 121–132.

[68] J. Rivers, G. Tyson, E. Davidson, and T. Austin, "On high-bandwidth data cache design for multi-issue processors," in *Microarchitecture*, 1997, pp. 46–56.

[69] T. Austin and G. Sohi, "High-bandwidth address translation for multiple-issue processors," *SIGARCH Comput. Arch. News*, vol. 24, no. 2, pp. 158–167, 1996.

[70] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," *Microarchitecture*, pp. 248–259, 1999.

[71] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Intl. Symp. on Computer Architecture*, 2003, pp. 136–146.

[72] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache: Demand Based Associativity via Global Replacement," in *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 544–555.

[73] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches," in *High-Performance Computer Architecture*, 2001, pp. 147–157.

[74] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *Intl. Symp. on Low Power Electronics and Design*, 2000, pp. 90–95.

[75] S.-H. Yang, M. Powell, B. Falsafi, and T. Vijaykumar, "Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay," in *High-Performance Computer Architecture*, 2002, pp. 151–161.

[76] P. Ranganathan, S. Adve, and N. Jouppi, "Reconfigurable caches and their application to media processing," in *Intl. Symp. on Computer Architecture*, 2000, pp. 214–224.

[77] S. Srikantaiah, E. Kultursay, T. Zhang, M. Kandemir, M. J. Irwin, and Y. Xie, "MorphCache: A reconfigurable adaptive multi-level Cache hierarchy," *High-Performance Comput. Archit.*, pp. 231–242, Feb. 2011.

[78] R. Balasubramonian, D. Albones, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," in *Intl. Symp. on Microarchitecture*, 2000, pp. 245–257.

[79] S. Kumar, "Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy," in *Micro 2012*, 2012.

[80] F. Vahid and W. Najjar, "Energy benefits of a configurable line size cache for embedded systems," in *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.*, pp. 87–91.

[81] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007, p. 900.

[82] J. Lee, Y. Solihin, and J. Torrellas, "Automatic code mapping on an intelligent memory architecture," *IEEE Trans. Comput.*, vol. 50, no. 11, pp. 1248–1266, 2001.

[83] A. Firoozshahian, A. Solomatnikov, O. Shacham, Z. Asgar, S. Richardson, C. Kozyrakis, and M. Horowitz, “A memory system design framework: Creating smart memories,” in *Intl. Symp. on Computer Architecture*, 2009, vol. 37, no. 3, pp. 406–417.

[84] Q. Guo, X. Guo, Y. Bai, and E. Ipek, “A resistive TCAM accelerator for data-intensive computing,” in *Intl. Symp. on Microarchitecture*, 2011, pp. 339–350.

[85] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, “AC-DIMM: associative computing with STT-MRAM,” in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 189–200.

[86] J. Yoo, S. Yoo, and K. Choi, “Active Memory Processor for Network-on-Chip-Based Architecture,” *IEEE Trans. Comput.*, vol. 61, no. 5, pp. 622–635, May 2012.

[87] J. Ahn, M. Erez, and W. J. Dally, “Scatter-Add in Data Parallel Architectures,” in *Intl. Symp. on High-Performance Computer Architecture*, 2005, pp. 132–142.

[88] 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.

[89] Z. Fang, L. Zhang, J. B. Carter, S. A. McKee, A. Ibrahim, M. A. Parker, and X. Jiang, “Active memory controller,” *J. Supercomput.*, vol. 62, no. 1, pp. 510–549, Jan. 2012.

[90] J. T. Pawlowski, “Hybrid memory cube (HMC),” in *Hot Chips 23*, 2011.

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

[92] S. Pugsley, R. Balasubramonian, J. Jestes, F. Li, A. Davis, V. Srinivasan, and A. Buyuktosunoglu, “Comparing Different Implementations of Near Data Computing with In-Memory MapReduce Workloads,” *IEEE Micro*, vol. 34, no. 4, pp. 44–52, 2014.

[93] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and

D. A. Wood, “The gem5 simulator,” *SIGARCH Comput. Arch. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[94] A. Hansson, N. Agarwal, A. Kolli, T. Wenisch, and A. Udipi, “Simulating DRAM controllers for future system architecture exploration,” in *Intl. Symp. on Performance Analysis of Systems and Software*, 2014.

[95] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “The McPAT Framework for Multicore and Manycore Architectures,” *ACM Trans. Archit. Code Optim.*, vol. 10, no. 1, pp. 1–29, Apr. 2013.

[96] ARM, “Cortex-A9 Processor.” [Online]. Available: http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[97] R. Singhal, “Inside Intel Next Generation Nehalem Microarchitecture,” *Hot Chips*, vol. 20, 2008.

[98] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “CACTI 6.0: A tool to model large caches,” 2009.

[99] K. Chandrasekar, C. Weis, Y. Li, B. Akesson, N. Wehn, and K. Goossens, “DRAMPower: Open-source DRAM power & energy estimation tool.” [Online]. Available: http://www.drampower.info.

[100] Micron, “TN-41-01: Calculating Memory System Power for DDR3,” 2007.

[101] E. Cooper-Balis and B. Jacob, “Fine-Grained Activation for Power Reduction in DRAM,” *IEEE Micro*, vol. 30, no. 3, pp. 34–47, May 2010.

[102] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. Ahn, “Reducing memory access latency with asymmetric DRAM bank organizations,” in *Intl. Symp. on Computer Architecture*, 2013, vol. 41, no. 3, pp. 380–391.

[103] T. Vogelsang, “Understanding the Energy Consumption of Dynamic Random Access Memories,” in *Intl. Symp. on Microarchitecture*, 2010, pp. 363–374.

[104] R. Gonzalez and M. Horowitz, “Energy dissipation in general purpose microprocessors,” *IEEE J. Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.

[105] Synopsys, “DesignWare Library - Datapath and Building Block IP,” 2013. [Online]. Available: http://www.synopsys.com/dw/buildingblock.php.

[106] M. Vuletic, L. Pozzi, and P. Ienne, “Seamless hardware-software integration in reconfigurable computing systems,” *IEEE Des. Test Comput.*, vol. 22, no. 2, pp. 102–113, 2005.

[107] D. Kroft, “Lockup-free instruction fetch/prefetch cache organization,” in *Intl. Symp. on Computer Architecture*, 1981, pp. 81–87.

[108] J. Tuck, L. Ceze, and J. Torrellas, “Scalable cache miss handling for high memory-level parallelism,” in *Microarchitecture*, 2006, pp. 409–422.

[109] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems,” in *Microarchitecture*, 1997, pp. 330–335.

[110] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, “MediaBench II video: Expediting the next generation of video systems research,” *Microprocess. Microsyst.*, vol. 33, no. 4, pp. 301–318, 2009.

[111] J. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu, “Parboil: A revised benchmark suite for scientific and commercial throughput computing,” 2012.

[112] D. Chang, C. Jenkins, P. Garcia, S. Gilani, P. Aguilera, A. Nagarajan, M. Anderson, M. Kenny, S. Bauer, M. Schulte, and K. Compton, “ERCBench: An open-source benchmark

suite for embedded and reconfigurable computing," in *Field Programmable Logic and Applications*, 2010, pp. 408–413.

[113] A. L. Shimpi, "NVIDIA's Tegra 3 launched: Architecture revealed," 2011. [Online]. Available: http://www.anandtech.com/show/5072/nvidias-tegra-3-launched-architecture-revealed.

[114] D. Burgess, E. Gieske, J. Holt, T. Hoy, and G. Whisenhunt, "e6500: Freescale's low-power, high-performance multithreaded embedded processor," *IEEE Micro*, vol. 32, no. 5, pp. 26–36, 2012.

[115] M. Khellah, N. Kim, J. Howard, G. Ruhl, J. Tschanz, D. Somasekhar, N. Borkar, F. Hamzaoglu, G. Pandya, A. Farhang, K. Zhang, and V. De, "A 4.2GHz 0.3mm2 256kb dual-V/sub cc/ SRAM building block in 65nm CMOS," in *IEEE Intl. Solid State Circuits Conference*, 2006, pp. 2572–2581.

[116] C. Park, H. Chung, Y.-S. Lee, J. Kim, J. Lee, M.-S. Chae, D.-H. Jung, S.-H. Choi, S. Seo, T.-S. Park, J.-H. Shin, J.-H. Cho, S. Lee, K.-W. Song, K.-H. Kim, J.-B. Lee, C. Kim, and S.-I. Cho, "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.

[117] J.-S. Kim, C. S. Oh, H. Lee, D. Lee, H. R. Hwang, S. Hwang, B. Na, J. Moon, J.-G. Kim, H. Park, J.-W. Ryu, K. Park, S. K. Kang, S.-Y. Kim, H. Kim, J.-M. Bang, H. Cho, M. Jang, C. Han, J.-B. LeeLee, J. S. Choi, and Y.-H. Jun, "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.

[118] T. Sekiguchi, K. Ono, A. Kotabe, and Y. Yanagawa, "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.

[119] Uksong Kang, Hoe-Ju Chung, Seongmoo Heo, Soon-Hong Ahn, Hoon Lee, Soo-Ho Cha, Jaesung Ahn, DukMin Kwon, Jin-Ho Kim, Jae-Wook Lee, Han-Sung Joo, Woo-Seop Kim, H.-K. Kim, Eun-Mi Lee, So-Ra Kim, Keum-Hee Ma, Dong-Hyun Jang, Nam-Seog Kim, Man-Sik Choi, Sae-Jang Oh, Jung-Bae Lee, Tae-Kyung Jung, Jei-Hwan Yoo, and Changhyun Kim, "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.

[120] K.-N. Lim, W.-J. Jang, H.-S. Won, K.-Y. Lee, H. Kim, D.-W. Kim, M.-H. Cho, S.-L. Kim, J.-H. Kang, K.-W. Park, and B.-T. Jeong, "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.

[121] "Looking Inside Samsung's 3x nm Process Generation DDR3 SDRAM," 2011. [Online]. Available: http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/looking-inside-samsungs-3x-nm-process-generation/.

[122] K. Sohn, T. Na, I. Song, Y. Shim, W. Bae, S. Kang, D. Lee, H. Jung, H. Jeoung, K.-W. Lee, J. Park, J. Lee, B. Lee, I. Jun, J. Park, J. Park, H. Choi, S. Kim, H. Chung, Y. Choi, D.-H. Jung, J. S. Choi, B. Moon, J.-H. Choi, B. Kim, S.-J. Jang, J. S. Choi, and K. S. Oh, "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.

[123] S. O, Y. H. Son, N. S. Kim, and J. Ahn, "Row-Buffer Decoupling: A Case for Low-Latency DRAM Microarchitecture," in *Intl. Symp. on Computer Architecture*, 2014.

[124] Micron, "Calculating Memory System Power for DDR SDRAM."

[125] M. Shevgoor, J.-S. Kim, N. Chatterjee, R. Balasubramonian, A. Davis, and A. N. Udipi, "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.

[126] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *Intl. Conf. on Architectural Support for programming Languages and Operating Systems*, 2000, vol. 28, no. 5, pp. 105–116.

[127] J. Mukundan, H. Hunter, K. Kim, J. Stuecheli, and J. F. Martínez, "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.

[128] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *Intl. Symp. on Computer Architecture*, 2012, vol. 40, no. 3, pp. 1–12.

[129] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," in *International Symposium on Computer Architecture*, 2013, vol. 41, no. 3, pp. 237–248.

[130] D. H. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," in *Architectural Support for Programming Languages and Operating Systems*, 2010, vol. 45, no. 3, pp. 397–408.

[131] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 210–221.

[132] A. N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. P. Jouppi, "LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems," in *Intl. Symp. on Computer Architecture*, 2012, pp. 285–296.

[133] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh, "Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric," in *Intl. Symp. on Microarchitecture*, 2010, pp. 137–148.

[134] P. Ranganathan, "From Microprocessors to Nanostores: Rethinking Data-Centric Systems," *Computer (Long. Beach. Calif).*, vol. 44, no. 1, pp. 39–48, Jan. 2011.

[135] J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for Data Intensive Scientific Analyses," in *IEEE Intl. Conf. on eScience*, 2008, pp. 277–284.

[136] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "SD-VBS: The San Diego Vision Benchmark Suite," in *International Symposium on Workload Characterization*, 2009, pp. 55–64.

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

[138] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in *Intl. Symp. on Computer Architecture*, 1995, pp. 24–36.

[139] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Intl. Symp. on Workload Characterization*, 2009, pp. 44–54.

[140] Micron, "DDR3 SDRAM," 2011. [Online]. Available: https://www.micron.com/~/media/Documents/Products/Data Sheet/DRAM/DDR3/4Gb_1_35V_DDR3L.pdf.

[141] H. Singh, F. J. Kurdahi, N. Bagherzadeh, and E. M. Chaves Filho, “MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications,” *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[142] A. Farmahini-Farahani, N. S. Kim, and K. Morrow, “Energy-Efficient Reconfigurable Cache Architectures for Accelerator-Enabled Embedded Systems,” in *Intl. Symp. on Performance Analysis of Systems and Software*, 2014, pp. 211–220.

[143] AMD, “AMD I/O Virtualization Technology (IOMMU) Specification Revision 2.0,” 2011. [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/48882.pdf.

[144] “JEDEC Publishes Wide I/O 2 Mobile DRAM Standard,” *JEDEC*, 2014. [Online]. Available: http://www.jedec.org/news/pressreleases/jedec-publishes-wide-io-2-mobile-dram-standard.

[145] R. Smith, “NVIDIA Updates GPU Roadmap; Unveils Pascal Architecture For 2016,” 2014. .

[146] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, “NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads,” in *Intl. Symp. on Performance Analysis of Systems and Software*, 2014, pp. 190–200.

[147] J. Menon, V. De Carli, Lorenzo hiruvengadam, K. Sankaralingam, and C. Estan, “Memory Processing Units,” in *Hot Chips 26*, 2014.

[148] Y. Eckert, N. Jayasena, and G. Loh, “Thermal Feasibility of Die-Stacked Processing in Memory,” in *Workshop on Near-Data Processing*, 2014.

[149] E. Azarkhish, I. Loi, D. Rossi, and L. Benini, “A Logic-base Interconnect for Supporting Near Memory Computation in the Hybrid Memory Cube,” in *Workshop on Near-Data Processing*, 2014.

[150] E. Azarkhish, I. Loi, D. Rossi, and L. Benini, “high performance AXI-4.0 based interconnect for extensible smart memory cubes,” in *Proc. Design, Automation, and Test in Europe*, 2015.

[151] G. Kim, J. Kim, J. H. Ahn, and J. Kim, “Memory-centric system interconnect design with Hybrid Memory Cubes,” in *Intl. Conf. on Parallel Architectures and Compilation Techniques*, 2013, pp. 145–155.

[152] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, “Query processing on smart SSDs,” in *Intl. Conf. on Management of Data*, 2013, pp. 1221–1230.

[153] R. M. Yoo, A. Romano, and C. Kozyrakis, “Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system,” in *IEEE Intl. Symp. on Workload Characterization*, 2009, pp. 198–207.

[154] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, “MineBench: A Benchmark Suite for Data Mining Workloads,” in *IEEE Intl. Symp. on Workload Characterization*, 2006, pp. 182–188.

[155] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. J. Desnoyers, and Y. Solihin, “Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines,” in *USENIX Conf. on File and Storage Technologies*, 2013, pp. 119–132.

[156] J. Ouyang, S. Lin, Z. Hou, P. Wang, Y. Wang, and G. Sun, “Active SSD design for energy-efficiency improvement of web-scale data analysis,” in *Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2013, pp. 286–291.

[157] S. Boboila, Y. Kim, S. S. Vazhkudai, P. Desnoyers, and G. M. Shipman, “Active Flash: Out-of-core data analytics on flash storage,” in *IEEE Symp. on Mass Storage Systems and Technologies (MSST)*, 2012, pp. 1–12.

[158] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: a case for intelligent SSDs," in *Int. ACM Conf. on Supercomputing*, 2013, pp. 91–102.

[159] Y. Kang, Y. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart SSD," in *IEEE Symp. on Mass Storage Systems and Technologies (MSST)*, 2013, pp. 1–12.

[160] S. Kim, H. Oh, C. Park, S. Cho, and S.-W. Li, "Fast, Energy Efficient Scan inside Flash Memory Solid-State Drives," in *Intl. Workshop on Accelerating Data Management Systems (ADMS)*, 2011.

[161] S. Borkar, "The future of computer architecture (keynote)," in *Fourth Workshop on Energy Efficient Design in conjection with Intl. Symp. on Computer Architecture*, 2012.

[162] P. Kogge and J. Shalf, "Exascale Computing Trends: Adjusting to the 'New Normal' for Computer Architecture," *Comput. Sci. Eng.*, vol. 15, no. 6, pp. 16–26, Nov. 2013.