UNIVERSITY OF CALIFORNIA

Los Angeles

Modeling and Optimization of Accelerator-Rich Architectures for Near Data Processing

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Nazanin Farahpour

2020

ABSTRACT OF THE DISSERTATION

Modeling and Optimization of Accelerator-Rich Architectures
for Near Data Processing

by

Nazanin Farahpour
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2020
Professor Glenn Reinman, Chair

The exponential growth of the dataset size demanded by modern big data applications requires innovative server architecture design for data centers. A server workload comprise of a diverse set of data-intensive and compute-intensive applications. These applications often interact with in-memory or in-storage datasets and their throughput is crucial to the user experience. But today's commodity hardware solutions which serve such applications, often follow a compute-centric approach which lacks the adequate interconnection bandwidth for many of these use-cases and leads to substantial data transfer latency and energy consumption. One promising solution has been to offload part of the computation closer to the data medium. Various Near data processing (NDP) techniques, while targeting data at different levels of the memory hierarchy, share common benefits: higher aggregate bandwidth, concurrency in access and mitigating the energy cost of moving data.

While prior work have shown many benefits of NDP accelerators, they have not solved three of the main challenges: 1) Existing NDP accelerators are mostly early-stage prototypes with limited capability of system configuration. Thus, researchers lack a practical way to explore the design space and show the benefits of NDP techniques under different system parameters and applications. 2) Focusing solely on one level of the memory hierarchy (one of cache, main memory or storage) cannot provide a satisfying solution for the data center servers which serve a diverse range of applications. A good understanding of application characteristics and exploring their suitability for NDP acceleration is missing. 3) It is common to have variation in compute and memory requirements, even within different execution phases of a single application. Thus, a single application could benefit from the collaboration of different NDP accelerators, but the collective benefits has not been explored.

In this dissertation, We try to address these three challenges, by presenting a multi-level acceleration platform that combines on-chip, near-memory and near-storage accelerators, spanning all levels of the conventional memory hierarchy. Our simulation platform features compute levels with adjustable memory/accelerator parameters, thus offering a broad spectrum of acceleration options. To enable effective acceleration on various application pipelines, we propose a holistic approach to coordinate between the compute levels, reducing inter-level data access interference and asynchronous task flow control. To minimize the programming efforts of using the platform, a uniform programming interface is designed to decouple the configuration from the user application source code and allow runtime adjustments without modifying the deployed application. We use our simulation platform to quantify the collective performance and energy benefits of NDP in all levels of the memory hierarchy. We also present an in-depth study of workload characteristics on various class of applications including visual retrieval, database, finance and security to demonstrate that a proper application mapping could avoid unnecessary data movements and achieve significant improvements to performance and energy efficiency.

The dissertation of Nazanin Farahpour is approved.

University of California, Los Angeles

2020

To my husband Cameron . . .

and our parents

—Mahin, Ahad, Fariba and Hossein—

for their endless love, support, and encouragement

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I wish to thank my advisor, Professor Glenn Reinman, for giving me the freedom to persue interesting research whilst providing continued guidance and support throughout my time at UCLA. His mentorship was instrumental in helping me to improve and develop not only my research but myself.

I am grateful for the support and thoughtful feedback from my other dissertation committee members: Professor Milos Ercegovac, Professor Anthony Nowatzki, and Professor Zhenman Fang. I would like to say special thanks to Dr. Fang for his continued guidance and research collaboration during my graduate school experience.

To the members of our research group Yuchen Hao, Beayna Grigorian and Farnoosh Javadi for their collaborations and camaraderie over the years. I am truly grateful to my fellow PhD students, who have become a second family over the past several years, as we pursue our dreams thousands of miles from our homeland. Their presence and support has been invaluable. This includes Shaghayegh Mardani, Homa Esfahanizadeh, Hanieh Hashemi, Neda Derakhshani, Reihane Boghrati, Karthika Mohan, Noor Abani, Laila Aryan, Farhad Shahmohammadi, Hossein Jahandideh and Ehsan Abasi.

Last but not least, I would like to express my deep appreciation to my family: to my husband, Cameron, my parents, Mahin and Ahad, my siblings, Negin and Amin, my family in the US, Hossein, Fariba, Tim, Jahan and Donna. Their support, patience, and love have been the foundation upon which I have relied during difficult times.

This work is partially supported by the Center for Domain-Specific Computing; funding from our industrial partners including Samsung Semiconductor INC; and C-FAR, one of the six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

VITA

2012	Undergraduate Researcher, Processor Architecture Laboratory Ecole polytechnique fédérale de Lausanne, Switzerland.
2013	Bachelor of Science, Computer Engineering, Hardware, University of Tehran, Iran.
2013 - 2014	Department Fellowship, Computer Science Department, University of California, Los Angeles.
2015	Hardware Engineering Intern, Advanced Technology Group, Dolby Laboratories, Sunnyvale, CA.
2016	Master of Science, Computer Science, University of California, Los Angeles.
2016 , 2017	Machine Learning Intern, Memory Solutions Lab, Samsung Semiconductor INC, San Jose, CA.
2016 - 2018	Teaching Assistant, University of California, Los Angeles.
2018	Ph.D. Candidate, Computer Science, Computer Architecture University of California, Los Angeles.

SELECTED PUBLICATIONS

N. Farahpour, Y. Hao, Z. Fang and G. Reinman, “ReACH: reconfigurable accelerator compute hierarchy,” *ACM Transactions on Architecture and Code Optimization* (under review), 2020.

N. Farahpour, Z. Fang, and G. Reinman, “FPGA-based near data processing platform selection using fast performance modeling,” *21st ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems* (under review), 2020.

S. P. Olarig, F. Worley, and **N. Farahpour**, “Method and apparatus for supporting machine learning algorithms and data pattern matching in ethernet ssd,” Jul. 5 2018, US Patent App. 15/472,061

B. Grigorian, **N. Farahpour**, and G. Reinman,“BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing,” *21st IEEEInternational Symposium on High Performance Computer Architecture*, pp. 615-626, February 2015.

D. Novo, **N. Farahpour**, P. Ienne, U. Ahmad, and F. Catthoor,“Energy efficient MIMO processing: A case study of opportunistic run-time approximations,” *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pp. 1-6, March 2014.

CHAPTER 1

Introduction

Energy consumption has always been a major concern in designing systems from mobile devices to servers in a data center. But, the recent growth in cloud computing and the advent of hyper-scale data centers have made energy consumption an even more pressing issue. The unprecedented amount of data being generated by people and IoT devices, are constantly transferred, analyzed and maintained in these facilities. By 2020, data center electricity consumption in US is projected to be around 140 billion KW-hours annually, the equivalent annual output of 50 power plants [4].

As general-purpose multi-core scaling cannot be sustained due to the end of Dennard scaling, industry-leading companies are incorporating specialized hardware accelerators into their servers. Thus, the research on accelerator-rich architectures and general-purpose accelerators are rapidly growing. There are several types of accelerators that are used in data centers today such as ASICs (Application Specific Integrated Circuits), GPUs (Graphics Processing Units) and FPGAs (Field Programmable Gate Arrays).

In particular, FPGAs have shown great potential in performance/energy efficiency of clusters and data centers. Amazon has deployed F1 compute instances in its Elastic Compute Cloud (EC2) [5]. F1 instances could feature multiple FPGAs and used for time-sensitive cloud services. Microsoft's Catapult [6] is another example of an FPGA system built for accelerating large-scale workloads such as search engine and neural network. Since the acquisition of Altera [7], Intel has introduced various CPU-FPGA acceleration platforms (AgileX [8], XeonSP [9]) that connect the xeon processor and FPGA fabric in the same package through a cache-coherent interface. Xilinx have also shifted their priority to data center market and introduced Xilinx Versal ACAP [10] as an on-chip accelerator.

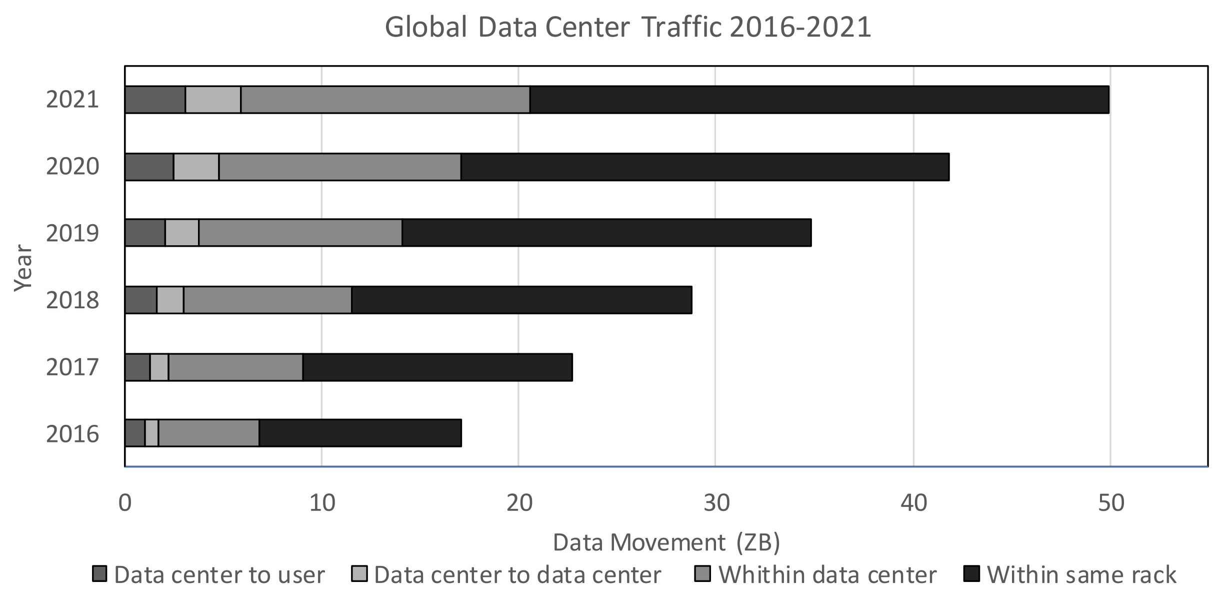


Figure 1.1: Global data center traffic forecast (2016-2021) [1]

Despite energy efficiency of accelerators, a major contributor to the overall server energy consumption is the off-chip memory hierarchy (40-50%) [11]. Figure 1.1 shows the estimated global data center traffic for 2016-2021 [1]. The data is divided into four categories:

- Data center to user: Traffic that flows from the data center to end users through the Internet or IP WAN (e.g. Web, email, streaming video).
- Data center to data: Traffic that flows from data center to data center (e.g. moving data between clouds, or copying content to multiple data centers as part of a content distribution network).
- Within data center: Traffic that remains within the data center, excludes traffic within the rack (e.g. reading and writing data to storage, moving data from a development environment to a production).
- the rack-local traffic: traffic that remains within a given server rack. Rack-local traffic is approximately twice the size of the “within data center” volume.

The figure 1.1 shows that more than 90% of traffic is remaining local to the data center and mostly comprise of read/write accesses to the storage. Data movement dominates the energy consumption in today's servers and it will continue to do so unless we change the traditional compute-centric server model to a more data-centric one. In compute-centric architectures, there are three major problems with memory and storage subsystem:

1. Slow scaling of chip pin-out and relatively weak memory/IO interfaces which provide insufficient bandwidth for data movement. For instance in a typical DRAM, there is often a reduction in available bandwidth of six orders of magnitude between the sense amplifiers in RAS/CAS and the CPU edge. The same problem exist in storage level as well.

2. The lack of access concurrency across memory and IO interfaces, which degrades the overall system throughput. The physical mismatch between internal and external bandwidth of the disk, as well as the lack of access concurrency in the host file system prevents the server from fully utilizing the disk performance.

3. Energy cost of moving data. Figure 1.2 shows how data movement dominates the energy consumption in modern computing systems. The data is collected from a 20 mm CMOS processor chip with 28 nm technology [2]. The figure shows that it takes about 20 pJ to do a double-precision floating point operation such as multiply-accumulate. But it takes more energy (26 pJ) to move the operands for the same operation for even 1 mm. Reading and Writing from SRAM takes 50 pJ and accessing off-chip dram take 500 pJ. The reading and writing from DRAM itself is in order of 1000x more than the arithmetic operation.

In contrast to a compute-centric model, in a data-centric compute model, data resides in different storage levels within the memory hierarchy and compute engines surround the data. Thus, compute engines can process the data without the need to move it across the system. By mitigating the data movement cost, a data-centric system could benefit from higher aggregate bandwidth and a more parallel access to memory hierarchy.

The potential benefits of data-centric computing have led to renewed research interest in near data processing (NDP) architectures in recent years. A large number of near-memory and near-storage accelerator models have been proposed and studied (detailed in chapter 2). However, there are three main challenges remaining: First, most existing NDP accelerators

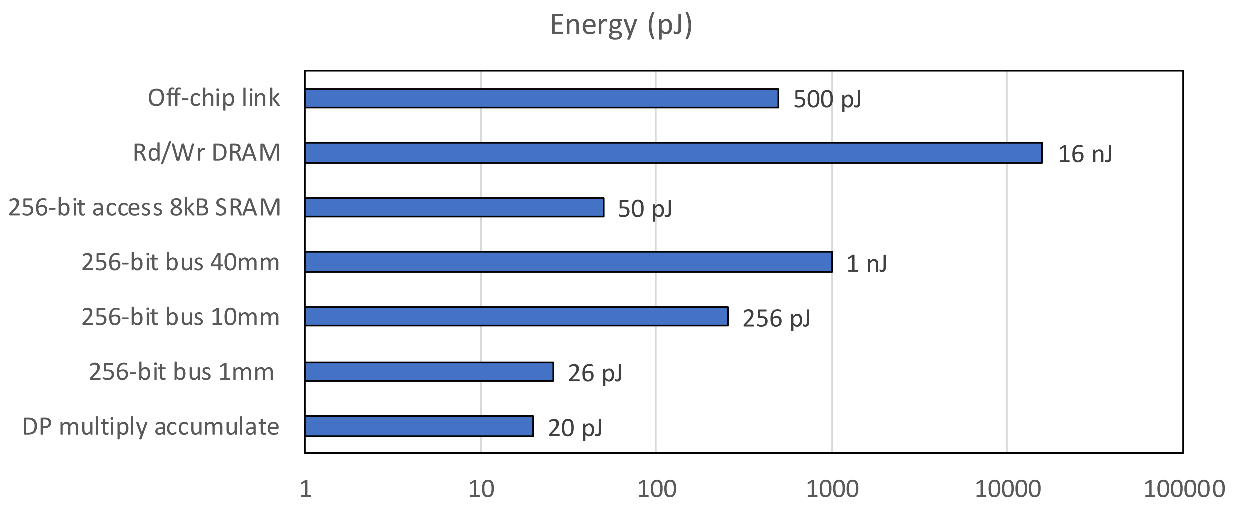


Figure 1.2: Energy consumption for a double-precision multiply-accumulate instruction running on a 28nm CMOS chip (as reported in [2])

are early-stage prototypes, and there is limited NDP devices on the market and they are bound by a specific system configuration. This makes it harder for researcher to explore the benefit of NDP systems under different configurations.

Second, there is not a clear understanding of type of applications that could benefit from these systems. We expect to see heterogeneous server racks with one or more types of NDP accelerators, being available for the server workload in the future and it is crucial to have a methodology for accelerator selection.

Third, the interaction of the NDP resources and combining them to accelerate one application has not been explored. A typical data analytic workload in a server consists of multiple stages of computation, generally including a data filtering stage followed by a more detailed analysis of an extracted subset of the data. Unfortunately, a large set of existing research has focused on NDP solely on one level of the memory hierarchy (cache, main memory or storage), while disregarding the potential benefits of having multiple NDP levels for data-intensive application pipelines.

1.1 Contributions

In this dissertation, we try to address the above challenges with following contributions:

In Chapter 3, we study the impact of near storage acceleration on a server workload, using a hardware experimental board. We demonstrate that the host-side accelerator scheduling schemes plays a key role in the performance benefits of near-storage accelerators. But we also face the first challenge described above: limited system configuration and limited number of hardware instances. This motivate us to present a cycle-accurate full-system simulation platform for analyzing various accelerator and their attached memory modules. We extend the PARADE [3] simulator (described in Section 2.6) to features compute levels with adjustable memory/accelerator parameters, thus offering a broad spectrum of acceleration options.

In Chapter 4, we introduce *ReACH*, a multi-level computing platform that combines on-chip, near-memory and near-storage accelerators, spanning all levels of the conventional memory hierarchy. The compute hierarchy offers both distributed computational power which potentially eliminates data movement, and the flexibility to adjust to the compute and memory requirements of different applications. We propose software and hardware infrastructure to coordinate between each compute level and managing resources. We use our extended simulation platform to quantify the collective performance and energy benefits of NDP in all levels of the memory hierarchy.

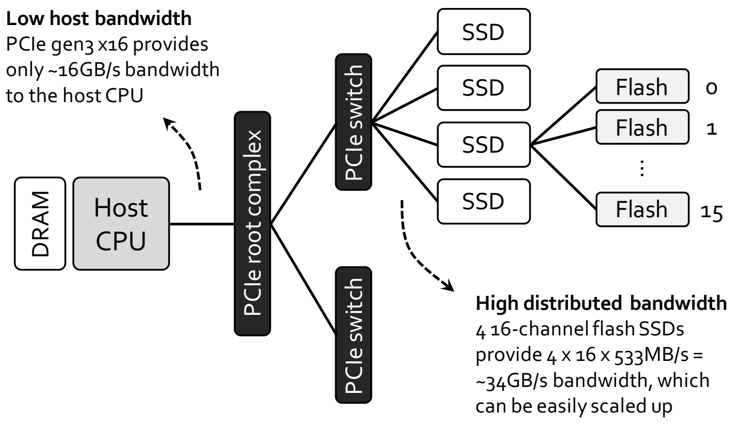
In Chapter 5, we characterize the modern data center applications and present a quantitative model to determine when an application is more suitable for each acceleration hierarchy. We analyze 18 benchmarks from 6 domains and create a guideline for both application and hardware developers. We demonstrate that a proper application mapping could avoid unnecessary data movements and achieve higher performance.

CHAPTER 2

Background and Related Work

2.1 Conventional Server Architecture

Figure 2.1 depicts an example of a modern data center server. Its main components include one or multiple CPUs, DRAM, PCIe switch, PCIe bus, multiple SSDs and Host/IO interface. The CPU cores could access multiple SSDs concurrently through Host interface and PCIe switch. The main limitation of the current server systems is the bandwidth mismatch between the array of SSDs and the Host interface. The gap increases as we scale the number of SSDs. Hence, big data applications that interact with large volumes of data resident in storage level will incur a performance bottleneck.



Low host bandwidth
PCIe gen3 x16 provides only ~16GB/s bandwidth to the host CPU
DRAM
Host CPU
PCIe root complex
PCIe switch
PCIe switch
SSD
SSD
SSD
SSD
Flash 0
Flash 1
Flash 15
High distributed bandwidth
4 16-channel flash SSDs provide 4 x 16 x 533MB/s = ~34GB/s bandwidth, which can be easily scaled up

Figure 2.1: Conventional server architecture

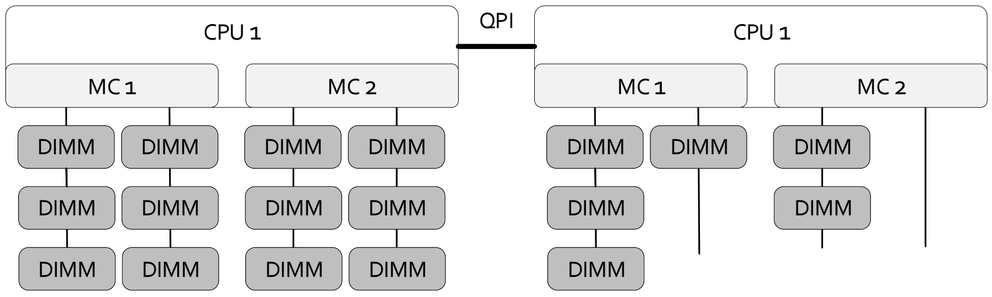


Figure 2.2: Memory architecture, the right socket has an imbalanced memory configuration

2.2 Server Memory Architecture

Similar to IO interface, today's commodity servers have relatively sparse off-chip memory bandwidth too. Figure 2.2 depicts a typical Xeon-based 2U sized chassis which features 4 memory channel per socket and each channel is shared between up to 3 dual inline memory modules (DIMMs). The DIMMs on the same channel share the bandwidth to CPU and increasing the number of DIMMs per channel (DPC) has significant impact on memory frequency and overall bandwidth. However, performance is sometimes sacrificed for reaching higher capacity for in-memory databases. Memory controllers decide how to interleave the data among DIMMs, but usually they interleave the data in cache granularity (64 bytes) between DIMMs of the same socket and the same row. This way, they improve the off-chip bandwidth of the main memory to CPU. The right-side socket in Figure 2.2 depicts an imbalanced DIMM configuration that has a low off-chip bandwidth to CPU. The single DIMM in the last row of the memory channels has one-way interleave with 25% of the peak bandwidth. This limitations are part of our motivations to analyze near-memory accelerators.

2.3 SSD Architecture

Figure 2.3 shows the hardware architecture of a modern SSD. Most modern SSDs contain several packages of NAND flash memory for non-volatile data storage media. Each flash package can be accessed in parallel and flash chips within the same package can be multiplexed on the shared channel. Both chip and channel level parallelism allows an SSD to acheive a

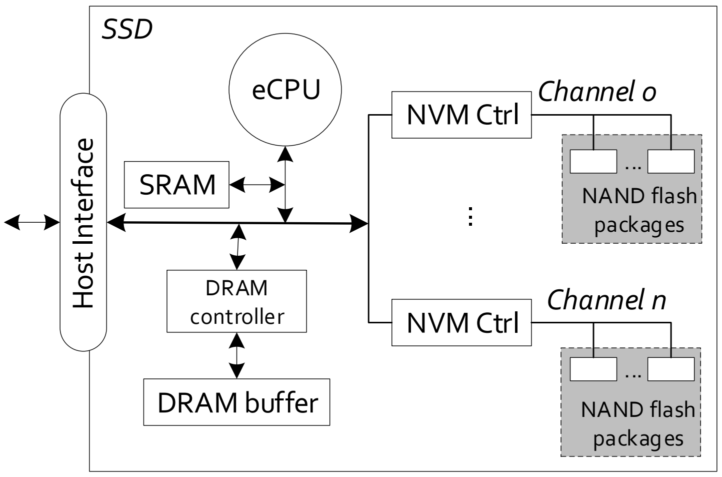


Figure 2.3: The architecture of a modern SSD

very high internal bandwidth (e.g. 16 GB/s). To communicate with the host system, SSDs can use NVM Express (NVMe) protocol via the underlying system interconnect, like PCI Express (PCIe) [12]. One PCIe lane can provide up to 1 GB/s bandwidth. High-end SSDs commonly use 4 or 8 PCIe Gen.3 lanes [13], which implies that they could obtain about 4 to 8 GB/s of external bandwidth, which is lower than the internal bandwidth. The SSD employs a large DRAM buffer (2GB) and embedded processors to handle the SSD firmware, NVMe commands and flash translation layer table. The smallest granularity for accessing flash memory is one page, which is around 4–16KB. Each NAND flash chip can read data lower than $10\mu s$ [13], but the host side I/O stack latency is over $22\mu s$ [14]. This shows that the latency of host-side I/O stack is the major bottleneck in performance of data-dependent applications.

2.4 Compute-Centric Acceleration Platforms

2.4.1 PCIe-Attached Private-Memory Accelerators

The most common FPGA integration is to connect an FPGA board equipped with private memory, to a CPU via PCIe (Figure 2.4a). Amazon EC2 F1 instances [5] and Microsoft Catapult boards [6] use this way of integration because of its flexibility and easy plug-

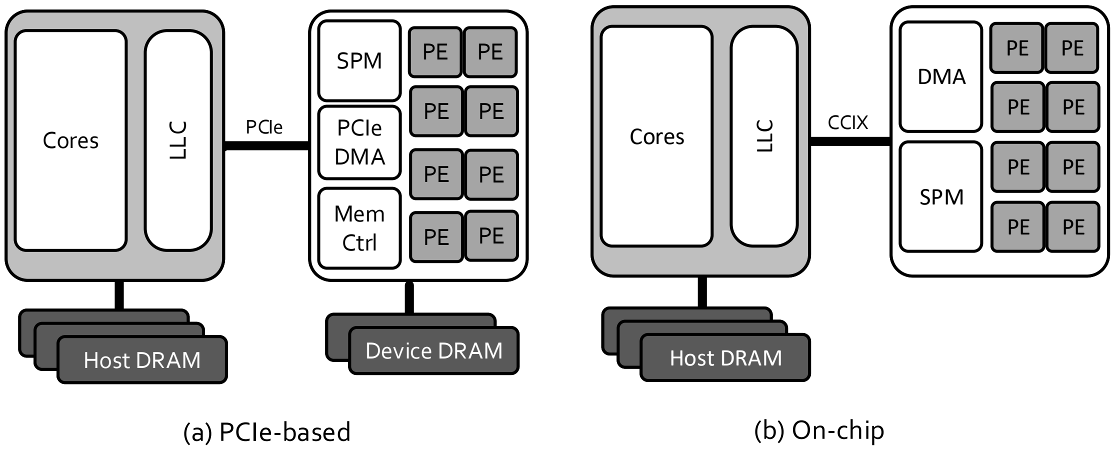


Figure 2.4: Compute-centric platforms

in. While these system offer high performance and energy efficiency for compute-intensive applications, they are limited by the effective PCIe bandwidth and communication latency between host CPU and FPGA. Any data transfer would require a PCIe-based direct memory access from host memory to device private memory. If the data transfer size is small, PCIe communication incurs significant overhead. These limitation hinder their performance on real-time applications or application that require a lot of interaction with CPU. They are best suitable for coarse-grained task which have an initial large payloads data transfers to the device private memory, followed by data-reuse. The Xilinx Virtex Ultrascale+ FPGA family have high-end FPGA boards with 346Mb on chip memory and 64GB on board DDR4 DIMM memory. If a database is small enough to fit in on-chip memory or the on-board DRAM of a PCIe-attached accelerator, it could significantly benefit from this accelerator. However, most databases exceed these memory requirements.

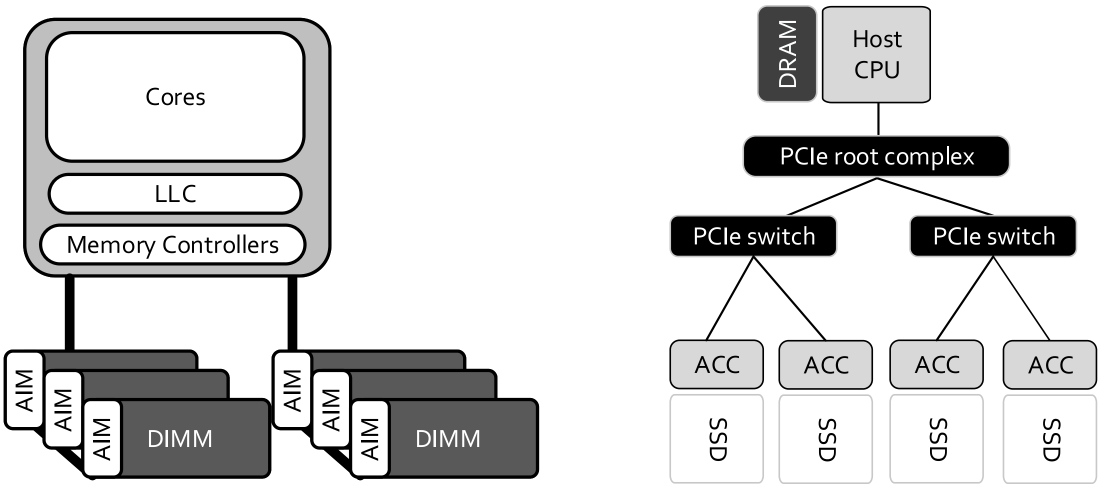
2.4.2 Shared-memory On-chip Accelerators

Unlike FPGA DRAM, the host-side DRAM capacity could be much higher than 64 GB. To leverage the higher memory bandwidth and capacity, a closer server-FPGA integration has been proposed that couples the CPU and FPGA into a single package and provides a shared memory, cache-coherent interface to allow seamless data access to CPU cache or memory without the redundant host to device memory copy or DMA (Figure 2.4b). Recently, Intel has introduced two such on-chip FPGA platforms (AgileX [8], XeonSP [9]) that connect the Xeon processor and FPGA fabric in the same package through a cache-coherent interface. This is possible thanks to Intel's Embedded Multi-die Interconnect Bridge (EMIB) and UltraPath Interconnect (UPI) links that could provide 41.2 GB/s to CPU cache. Xilinx have also introduced Versal ACAP [10] as an FPGA-based SoC. Versal ACAP connects to host CPU using Cache Coherent Interconnect for Accelerators (CCIX) link that could achieve up to 100 GB/s to CPU cache [15]. Moreover, with capability of swapping partial bitstreams in sub-millisecond, the Versal ACAP could be utilized by multiple real-time applications simultaneously. However, once the working set exceeds the on-chip cache capacity, the acceleration is limited by memory access latency and off-chip bandwidth.

2.5 Data-Centric Acceleration Platforms

2.5.1 Near-memory Accelerators

Emerging memory technology and advancements in 3D stacking are considered as the true enabler of processing close to the memory. The stacking of logic die and memory using through-silicon via (TSV) allows lower memory access latency and higher bandwidth. High bandwidth Memory (HBM) [16] from AMD and Hynix, and Samsung's Wide I/O [17] are the memory industries competing 3D memory products. The logic die which contains the dedicated memory controller could encompass simple SIMD cores or an embedded FPGA chip for data analysis. However, WideIO is used for Mobile SoC systems and HBM is costly to populate the server memory and replace conventional DDR4. Thus, We focus on near-memory accelerators for conventional DRAM architecture. Today's high-end servers have limited memory channels per socket and multiple DIMMs share the same memory channel which limits the overall bandwidth to the CPU. Near memory accelerators help to achieve a lower latency and a higher bandwidth to DIMMs sharing the same memory channel. For instance, Copacobana [18] builds FPGA modules directly into DIMMs. AIM [19] places FPGA modules between the DIMM and the memory network, making the design noninvasive to the existing memory controller, memory bus and DIMMs. Contutto [20] prototypes such idea by



(a) Near-memory (b) Near-storage

Figure 2.5: Data-centric platforms

plugging accelerators in DIMM slots in a POWER8 machine and shows acceleration with end-to-end experiments. RAMON [21] provide the hardware support for data partitioning at the memory controller to balance the bandwidth load in heterogeneous systems. Our work is most similar to AIM (Figure 2.5a). We also use a similar technique to RAMON to reduce access interference in our compute hierarchy (Chapter 4).

2.5.2 In-Storage Accelerators

In Section 2.1 and 2.3, we discuss two main limitations of the modern storage systems: the bandwidth gap between Host interface and array of SSDs, and the gap between internal and external bandwidth of a single SSD. While near-storage accelerators could tackle the former problem, the solution to latter one is in-storage acceleration. The idea of in-storage acceleration has recently redrawn considerable attentions [22–25]. This is mainly because an NVM-based SSD normally has a very high internal bandwidth, exceeding its external bandwidth to host by factors of 2x to 4x. Therefore, processing data in storage might be able to achieve a higher performance and save more energy than transferring them all the way to the host CPU. Most existing work such as SmartSSD [22], Active Disk [23], Biscuit [25] and Summariser [24] utilize the embedded cores in a modern SSD to avoid data movement.

However, this embedded cores are not able to saturate the high internal bandwidth of the storage. To achieve a higher performance, work like Minerva [26] utilize FPGA as in-storage accelerators. But integrating FPGA into an SSD controller comes with its own challenges: FPGA chip with limited resource, data fragmentation, interacting with FTL for block address translation. The near-storage accelerators, on the other hand, are more practical and require no internal SSD modifications. While our work focus on near-storage acceleration, the presented techniques and simulation platform can be extended to in-storage accelerators as well.

2.5.3 Near-Storage Accelerators

Unlike in-storage accelerators, near-storage accelerators are more practical and non-invasive to existing server architectures. They also integrate better with conventional dis-aggregated storage servers where front-end computing nodes are separated from network-attached storage systems. Employing near-storage acceleration offloads the CPU load into array of SSDs and provides a higher aggregated host interface (or network) bandwidth. Several studies in NVM-based near-storage accelerators have been reported recently in academia and industry. Projects such as IBM Netezza [27], Mobiveil [28], Willow [29] and BlueDBM [30] place FPGA units between the flash controller and the host IO interface, providing reconfigurabity while avoiding unnecessary energy consumption on data movement. In Chapter 3, we study an FPGA-based near-storage experimental board (Fig 2.5b).

2.6 Simulation Infrastructure

Studying near-storage accelerators using hardware experimental boards in Chapter 3 gives us a glimpse into potential of NDP accelerator-rich architectures, but limited accelerator instances and system configurations makes the design space exploration challenging. Therefore, we employ a cycle-accurate full-system simulation platform in Chapter 4 as a tool for designing and modeling our NDP accelerators. Figure 2.6 illustrates the PARADE simulator [3] which is a cycle-accurate simulator for modeling accelerator-rich architectures, built upon gem5 [31].

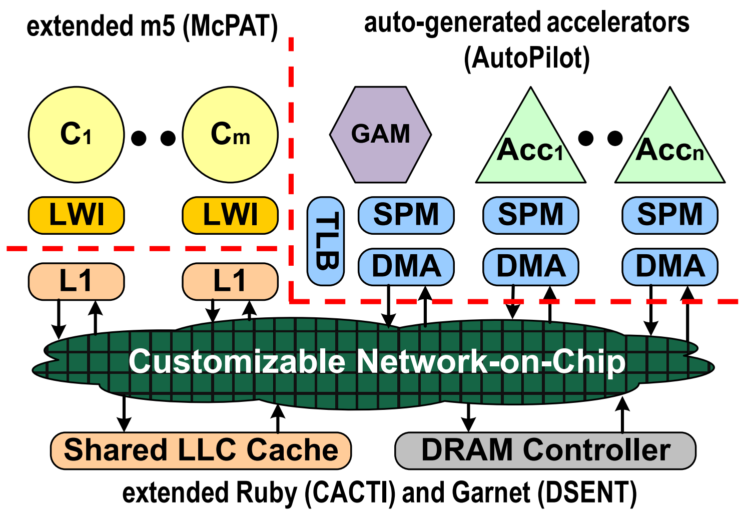

extended m5 (McPAT)
auto-generated accelerators (AutoPilot)
C1
Cm
GAM
Acc1
Accn
LWI
LWI
TLB
SPM
SPM
SPM
L1
L1
DMA
DMA
DMA
Customizable Network-on-Chip
Shared LLC Cache
DRAM Controller
extended Ruby (CACTI) and Garnet (DSENT)

Figure 2.6: Overview of PARADE simulator [3]

The legacy PARADE simulator introduces the accelerator simulation modules that can be automatically generated through a high-level synthesis (HLS) description of the accelerator. Each accelerator in PARADE has a scratchpad-memory (SPM) and Direct-Memory Access (DMA) module and adopts the three stage accelerator execution model (load-compute-store) and double-buffering. Moreover, a global accelerator manager (GAM) and light-weight interrupt is supported to enable efficient interactions between cores and accelerators. A customized network-on-chip (NoC) is used to loosely couple all the cores, accelerator modules, LLC and memory interface.

To design a new accelerator, users only need to provide a HLS C description with clock frequency, pipeline initiation interval and pipeline depth. The automation tool chain in PARADE will encode the information into an accelerator simulation module. Meanwhile, a set of software APIs will be generated so that the new accelerator can be invoked from simulator benchmarks.

After extensive modification in accelerator interface, GAM scheduling and profiling tools, we extend the legacy PARADE simulator to support metered memory modules. The metered memory module has configurable characteristics (bandwidth, access latency and granularity) and can be connected to different ports of the NoC router, similar to Figure 2.7. The accelerator SPM interface has the option to go through the normal cache hierarchy or bypass the cache and interact directly with the attached metered-memory. This way, we could simulate the behaviour of near-memory and near-storage accelerators. During system initialization, various accelerator modules get connected to NoC ports and interface with either L2 cache modules or metered memory. The GAM keeps track of the near-memory and near-storage ports and distribute the task based on the benchmark description.

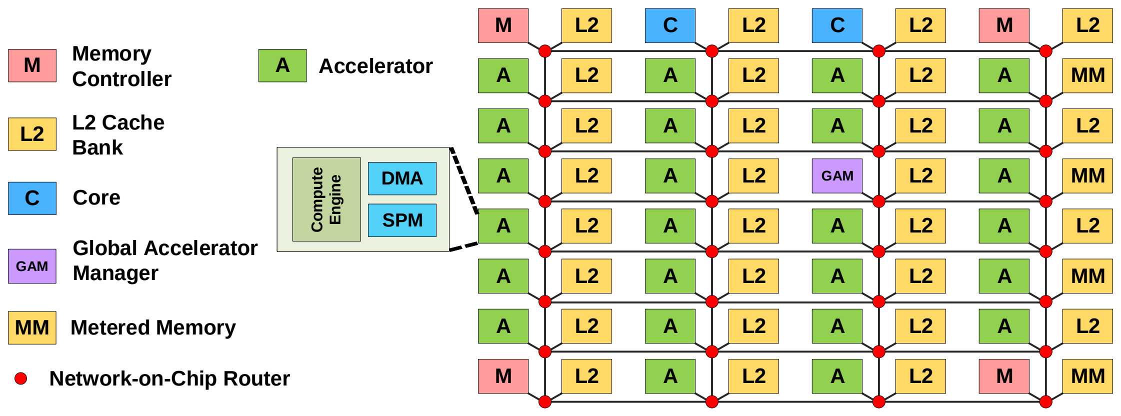


Figure 2.7: Overview of the accelerator-rich architecture [3]

CHAPTER 3

Performance Analysis of FPGA-based Near-Storage Acceleration

3.1 Introduction

The increasing demand on power-efficient computing in today's data centers, has sparked a growing number of CPU-FPGA acceleration platforms that can be reconfigured to accelerate a broad class of applications with orders-of-magnitude performance/watt gains. Attaching FPGAs as an IO-attached accelerator (section 2.4.1) is the most conventional way to deploy accelerators in the systems, especially for compute-intensive applications in which the CPUs are the bottleneck of these systems. While performance gain and energy efficiency of these architectures are promising, their applicability is constrained by the working set size not exceeding the accelerator's private DRAM capacity.

A typical server workload could comprise a diverse set of compute- and data-intensive applications [32]. These applications often interact with in-memory or in-storage datasets and their throughput is crucial to the user experience. The benefits of conventional accelerators are limited by the efficiency of data movement in the IO stack. For this reason, server architects are proposing a more data-centric acceleration scheme by moving the compute elements closer to the data. In this work, we employ a near-storage accelerator to study the impact of near data processing on server workloads.

We use a near-SSD FPGA experimental board provided by our industry collaborators. [1] We target content-based image retrieval (CBIR) as a proof-of-concept application where

[1]The experimental board is provided by Memory Solutions Lab, Samsung Semiconductor INC.

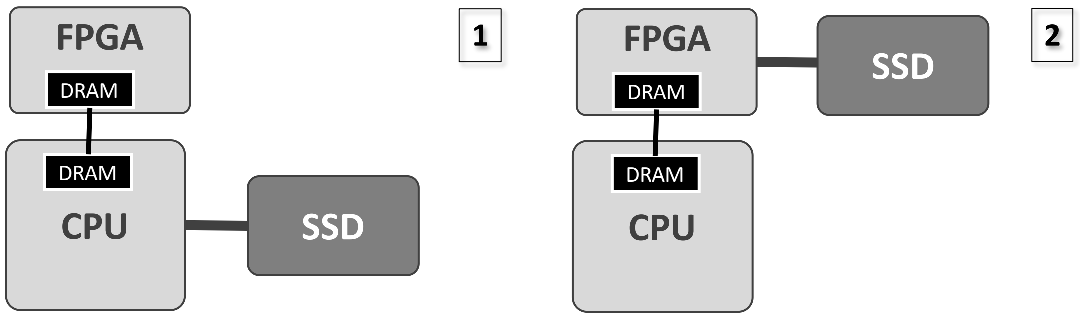


Figure 3.1: (1) Data flow in OpenCL framework and (2) Revised data flow in near-SSD platform with P2P connection of SSD to FPGA's device DRAM

moving computation closer to data is vitally important. We analyze the bottleneck of CBIR application and the performance of near-storage acceleration platform. We also discuss what is the best way to schedule consecutive accelerator calls on this platform.

3.2 Near-SSD Experimental Board

Our Near-SSD custom board includes a Xilinx VCU1525 FPGA board [33], an NVMe SSD [13], a PCIex4 connection to host Xeon CPU and a PCIex4 connection to SSD. It also includes an on-board switch that implements a peer to peer (P2P) connection between FPGA device DRAM and SSD. The on-board switch is controlled through host application. It allows the host CPU to access the SSD directly through direct memory access (DMA) requests or have a P2P connection between SSD and device DRAM.

We used SDAccel development environment and OpenCL framework to implement our accelerator [33]. Our OpenCL framework is slightly enhanced by Xilinx to include programming interface for P2P connection between device and SSD. Therefore, our custom board is capable of acting as a general PCIe-attached accelerator as well as a near-storage accelerator. This behavior could easily be controlled through the host software interface.

Figure 3.1 illustrate the two versions of data flow that is available to our experimental board through on-chip switch and enhanced OpenCL software interface. In a traditional OpenCL framework, the data for FPGA processing has to be completely copied from SSD to

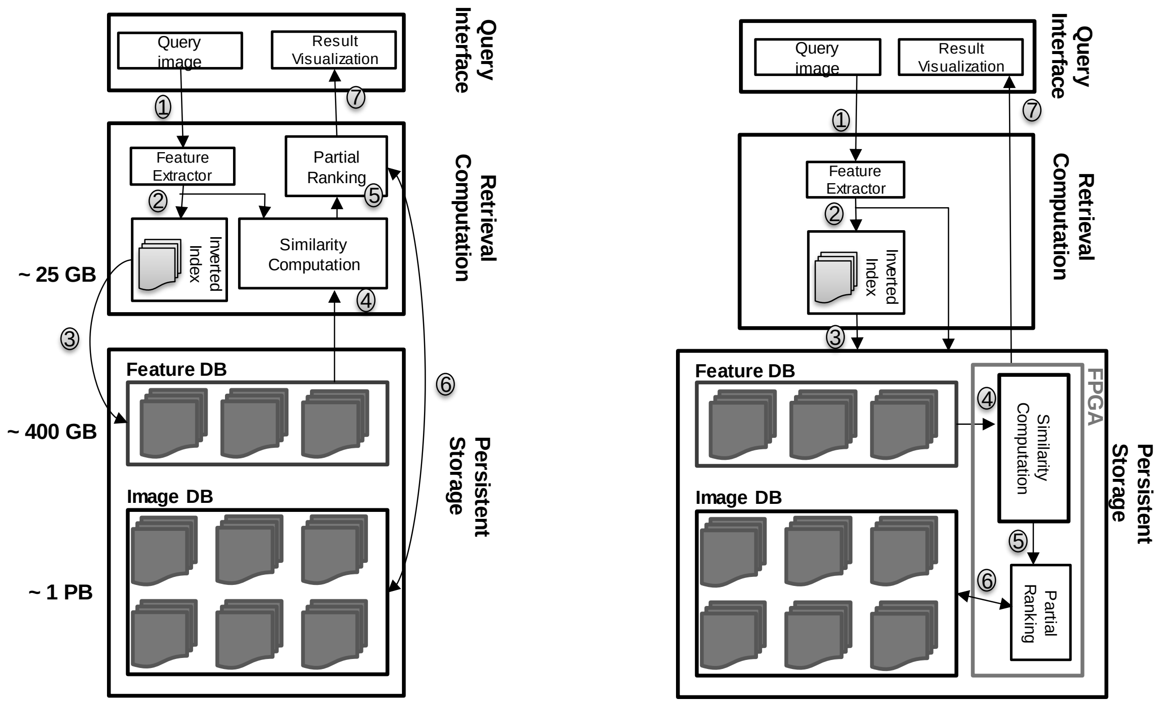
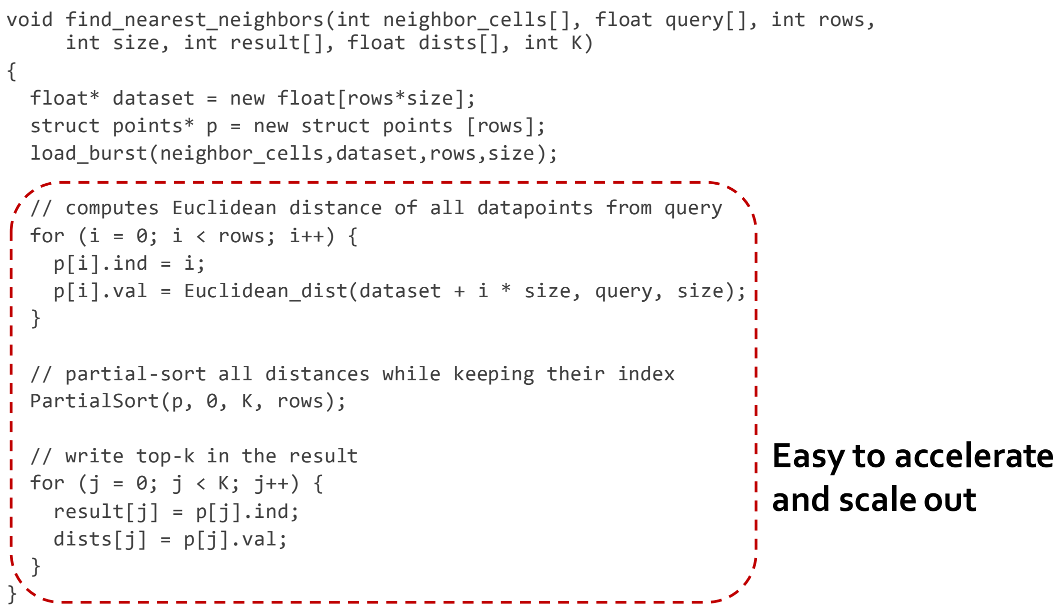


Figure 3.2: The general pipeline of billion-scale CBIR systems. We offload stages 4 and 5 to near-SSD FPGA.

host DRAM and then to device DRAM. But, in our revised framework, there is no implicit copying. The data buffer for device DRAM is assigned to a memory-mapped virtual buffer that is associated with a file in SSD device and during the FPGA processing the data buffer accesses the storage as needed and bypasses the host DRAM completely.

3.3 Case Study: Content-based Image Retrieval

Content-based image retrieval (CBIR), which identifies relevant images from large-scale image databases based on the representation of visual content, has attracted increasing attention in recent two decades. Modern image retrieval systems [34, 35] have to search through billion-scale databases in several milliseconds to respond to user queries. This imposes strict demands on the efficiency of the algorithm as well as the underlying system architecture.



```
void find_nearest_neighbors(int neighbor_cells[], float query[], int rows,
     int size, int result[], float dists[], int K)
{
  float* dataset = new float[rows*size];
  struct points* p = new struct points [rows];
  load_burst(neighbor_cells,dataset,rows,size);

  // computes Euclidean distance of all datapoints from query
  for (i = 0; i < rows; i++) {
    p[i].ind = i;
    p[i].val = Euclidean_dist(dataset + i * size, query, size);
  }

  // partial-sort all distances while keeping their index
  PartialSort(p, 0, K, rows);

  // write top-k in the result
  for (j = 0; j < K; j++) {
    result[j] = p[j].ind;
    dists[j] = p[j].val;
  }
}
```

Figure 3.3: The software algorithm for step 4 and 5 of CBIR

A typical billion-scale CBIR system consists of an off-line and an on-line stage. During an offline stage, CBIR system converts all images into features and clusters them using methods such as kmeans and kdtree. Each cluster is then represented by a centroid or hierarchy of centroids. Figure 3.2 illustrates the major steps in online stage of the CBIR system. In the on-line stage, user query images are batched in the front-end interface (❶) and transformed into feature vector using the same feature extraction method as in the off-line stage(❷). The indexing structure generated in the off-line stage is leveraged to retrieve a *short-list* of candidate feature vectors that are likely to be close to the user query(❸). These candidate vectors are then retrieved from the storage (❹) and are reranked based on their computed distance to the user query(❺). Lastly, the original images corresponding to the k-nearest feature vectors are retrieved from the database(❻). We describe the CBIR stages in more details in Chapter 4. But in this work, we focus on steps 4 and 5.

The Figure 3.3 shows the high-level function written for this part of the application. The *neighbor_cells* is the ID short-list of closest centroids to the query. The function first loads the candidate vectors from storage and then computes their similarity distance and partially sort the result to find the top K closest vectors. For a billion-scale CBIR system, the feature

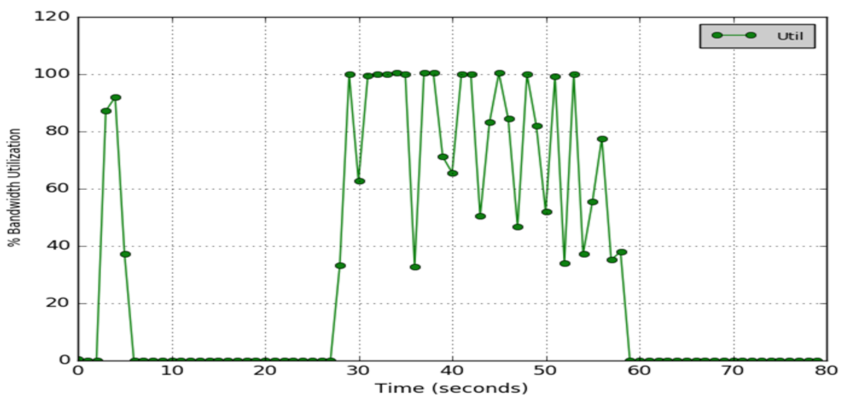


Figure 3.4: The host bandwidth utilization during run time of multi-threaded CBIR

database can be distributed among an array of SSDs and multiple threads can call *load_burst* function concurrently, causing a contention in host IO-interface. However, the part of the function in red box is easy to offload to near-storage accelerator and linearly scale out as the number of SSDs increases.

3.3.1 Software Profiling

In this section, we study whether nearest-neghbor function in Figure 3.3 could saturate the bandwidth of a single attached SSD. We run a multi-threaded CBIR application on host CPU (16 threads and 1024 queries). We eliminated the feature extraction part and focus on the rest of the pipeline. Figure 3.4 illustrates the host IO bandwidth. The initial delay in access to the storage is due to batch-based short-list retrieval. The figure shows that 16 thread could easily saturate the host-side bandwidth. The limitation would be worse in case of multiple SSDs. We offload the nearest-neighbor (KNN) function of the application to near-SSD experimental board. We also decided that short-list retrieval needs to be accelerated, but we leave the discussion to Chapter 4.

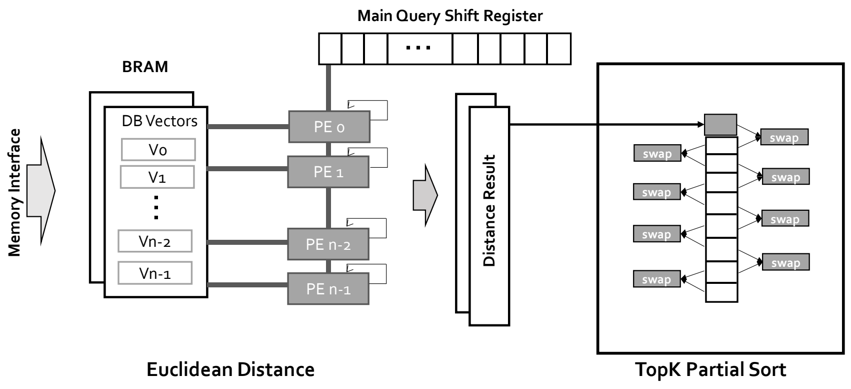


Figure 3.5: Nearest-neighbor (KNN) kernel design

3.4 Hardware Accelerator Design

Our KNN application is implemented in C++ and then compiled into binary files using Vivado High-Level Synthesis (HLS) tool chain. The OpenCL framework is used for writing the host which includes programming the device, setting the P2P switch, setting arguments for the kernel, and launching the kernel.

A high-level design of our KNN kernel is depicted in Figure 3.5. We use maximum bit-width, PE duplication and double-buffering techniques to improve the throughput of our kernel. The query vector is placed in a shift-register and shared among n processing elements (PEs). The database vectors are interleaved among multiple block RAMs (BRAMs) for concurrent access in each cycle. The main difference of the this KNN kernel from previous implementations is having partial sort functionality. The main reason we could have a partial sort module is because the CBIR feature vectors have between 64-1024 elements. So, distance computation can surpass the run time of partial sorting. Thus, sorting could run concurrent to distance computation of next batch. Our partial sort module receives the distance values and their indices and every two cycle, push one of them to the top of the sort queue. There are two arrays of swap modules in the sort queue. In odd cycles, the right-side swap modules make comparison and in the even cycles, the left-side swap modules make comparison. Alternatively, we could have two instances of partial sort module that each keep track of K nearest neighbour. This way we double the throughput of partial sort. In our

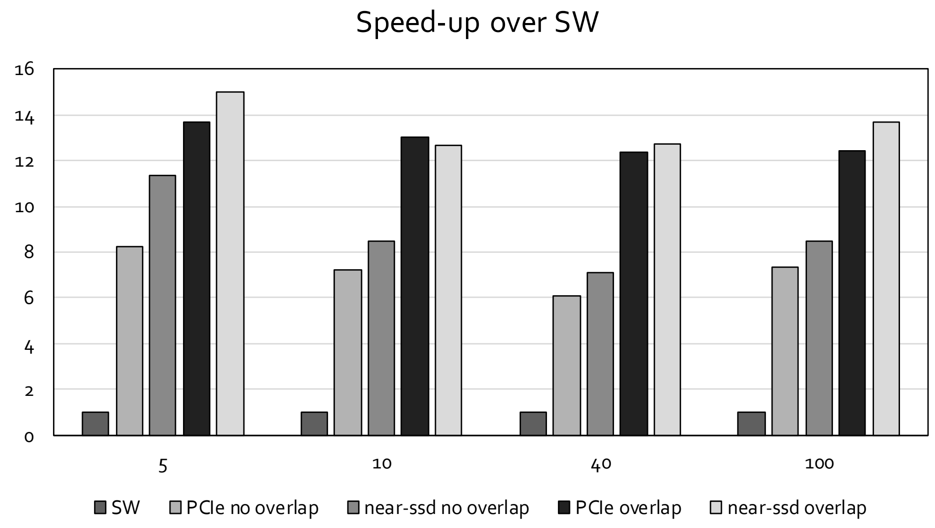


Figure 3.6: Speed-up comparison of 4 acceleration options for KNN kernel

experiment, we use vectors with 128 dimensions and 128 PEs and 2 partial sort modules.

3.5 Evaluation

In this section, we compare the performance of our FPGA kernel under four different conditions and software execution. We keep the kernel binary file fixed, but we modify the host interface code to emulate PCIe-attached and near-storage accelerators. For each acceleration option, we also change the accelerator scheduling scheme between in-order and out-of-order executions. When we use in-order scheduling, the host will wait until the current kernel execution task is done, before loading the next chunk of data. The in-order scheduling is useful for applications with data-dependency (e.g. hash look-up and graph processing) and application like KNN could always use out-of-order scheduling to overlap the data movement for current query from SSD with kernel execution of a previous query. But, we wanted to compare both scheduling schemes. In PCIe-attached version, the candidate points are first loaded from SSD to host DRAM and then Device DRAM. In Near-SSD version, a virtual buffer is memory-mapped from SSD to Device DRAM directly.

Figure 3.6 shows the KNN speed-up as we increase the number of queries. The single instance of near-storage accelerator has 15$\times$ speed-up over software and $\sim 2\times$ over simple PCIe-attached acceleration. But it has limited performance benefit compared to out-of-order PCIe-attached acceleration ($\sim$ 15%), because overlapping kernel execution and data movement masks a portion of the benefits that single-instance near-storage accelerator provides. One observation from the result is that out-of-order scheduling benefits PCIe-attached acceleration more than near-storage acceleration, because it could overlap the delay of loading data from SSD to host DRAM with previous kernel execution. Another observation is that DRAM to DRAM DMA bandwidth is actually higher than the SSD to DRAM bandwidth. So, in some cases the PCIe-attached accelerator surpass the speed-up of near-storage accelerator, when there is locality in the accessed chunks of database.

3.6 Conclusion

With single instance of near-storage accelerator, the performance benefits of KNN kernel is limited regardless of the kernel scheduling scheme. Because the overhead of host DRAM to device DRAM data movement is partially overlapped with execution. If we increase the number of SSDs, near-storage accelerators and the queries, the gap between the performance of PCIe-attached accelerator and near-storage accelerator will grow. Moreover, the KNN software execution is only 40%-60% of the overall CBIR execution time and we should consider accelerating other portions of the application as well. Therefore, in Chapter 4 we introduce a cycle-accurate simulation platform that could simulate multiple number of near-storage accelerators and help us study the end-to-end performance of CBIR application.

CHAPTER 4

ReACH: A Reconfigurable Accelerator Compute Hierarchy

4.1 Introduction

As power and energy efficiency become the primary motivators in today's data centers, industry-leading companies are shifting toward incorporating FPGAs into their servers. Microsoft and Amazon have pioneered the large-scale deployment of FPGAs in cloud services [5, 6] and nowadays FPGA is becoming an standard component of the cloud infrastructures. Therefore, it is crucial for server architects to choose the best server-FPGA integration that matches the server workload, memory/IO requirements and power budget.

A server workload can be comprised of both compute-intensive and memory-bound applications. In fact, it is common to have variations in compute and memory requirements, even within different execution phases of a single application. Content-Based Image Retrieval (CBIR), which identifies relative images from large-scale image databases using visual content, is one example of such an application [34,35]. A large number of data analytics applications like CBIR have working sets that span several hundreds of gigabytes of data. These applications often scan, join, and summarize large volumes of data, and their throughput is crucial to user experience [36].

As we show later in section 4.6, using conventional CPU-side FPGA acceleration on these applications would reduce the run time and compute energy, but the total energy savings would be limited by data movement cost (energy spent on the memory hierarchy and interconnects). To give an example, Figure 4.11 shows the energy distribution of a CBIR system using on-chip FPGA acceleration. The energy breakdown shows that after FPGA acceleration, 79% of the total remaining energy cost is due to data movement.

Another important limitation in conventional systems is the bandwidth gap between the host and attached memory/disk modules. Today's commodity servers have relatively sparse off-chip memory bandwidth, as well as limited IO interface bandwidth. A typical Xeon-based, 2U sized chassis features 4 memory channels per socket and each channel is shared between up to 3 dual inline memory modules (DIMMs). The same problem exist in the IO interface. The host-side PCIe bandwidth to storage hierarchy is theoretically 16GB/s which downgrades to about 12GB/s due to inefficiency in the host IO software stack [37]. Since the number of SSD slots in a typical host server could be up to 16 and a single disk latency and bandwidth is continuously improving, the performance bottleneck is moving towards the host-side IO interface. The bandwidth gap is even wider in data centers with dis-aggregated storage servers that communicates volumes of data through the network between host server and storage servers. These constraints hinder the effectiveness of on-chip accelerators on communication-bound data analytics workload, where these accelerators are only suitable to perform part of the work and must coordinate with CPU, memory modules, IO stack and network interface in the system for data movement.

To overcome these issues, prior studies have proposed deploying accelerators near the data medium (memory or disk modules) to distribute computation into memory or disk modules as a way of both exploiting the available internal bandwidth and avoiding the movement of data across chip boundaries [19, 20, 27, 30, 38, 39]. Unfortunately, a large set of existing research has solely focused on only one level of the memory hierarchy (cache, main memory or storage), while disregarding the potential benefits of having multiple levels of near data processing for communication-bound application pipelines.

In this work, we present **ReACH**, a reconfigurable accelerator compute hierarchy that combines on-chip, near-memory, and near-storage accelerators that cross all levels of the conventional memory hierarchy. Each acceleration hierarchy provides distinct compute and memory capabilities, offering a broad spectrum of acceleration options. On-chip cache-coherent accelerators are most suitable for compute-intensive workloads or applications that require frequent interactions with the host CPU. Near-memory accelerators are suitable for parallelizable tasks with a large memory footprint and a high bandwidth requirement. Near-storage accelerators are suitable for streaming-like applications with simple tasks that are IO-intensive and heavily rely on storage bandwidth.

To enable effective acceleration on various application pipelines, we propose a hardware-based global accelerator manager (GAM) that serves as a master to all accelerator hierarchies. Rather than relying on CPU cores to interact with accelerators directly, ReACH opts for a more centralized task scheduling and control flow, assuming the accelerators across the memory hierarchy are coarse-grained and can run for a long time. The hardware and software co-design of GAM enables programmers to use conventional synchronous programming, while handling asynchronous task flow in the compute hierarchy. ReACH provides hardware support for data partitioning to limit the inter-task memory interference and pipelines the data movement and processing in all levels of the compute hierarchy.

We experimentally deploy a billion-scale content-based image retrieval (CBIR) system on ReACH. Based on our simulation results, a proper application mapping eliminates data movement and achieves 4.5× throughput and 51% energy improvements. To show the flexibility of the compute hierarchy, we further implement two alternative CBIR pipelines in our platform and evaluate their performance and accuracy. The main contributions of this paper are:

- Propose a compute hierarchy that combines on-chip, near-memory, and near-storage accelerators (Section 4.2).
- Design a hardware-based global accelerator manager for coordinating between compute levels, reducing inter-task memory access interference and providing asynchronous task flow control (Section 4.2.4).
- Introduce a uniformed library-based programming interface that decouples the user application from the system configurations of the ReACH for seamless use of the hierarchy (Section 4.3).

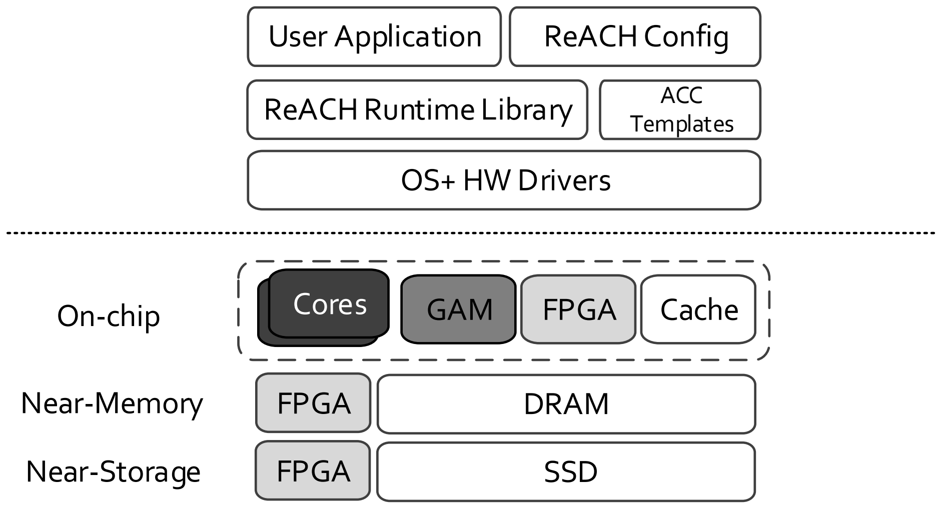


Figure 4.1: Overview of ReACH

- Demonstrate real-world application examples that could benefit from the hierarchy and quantify the performance and energy gains based on cycle-accurate simulation (Section 4.6).

4.2 Overview of ReACH

In this section, we describe the architecture of the ReACH system. While our compute engines are based on FPGA accelerators, the compute hierarchy is not dependent on a specific type of accelerator logic: they could, for example, be general-purpose cores, GPUs, ASIC accelerators, or CGRAs. This work focuses on FPGAs due to their improved performance per watt ratio versus CPUs and GPUs, since low power is critical for near-memory and near-storage acceleration where power/thermal constraints are more stringent.

A high-level organization of our compute hierarchy is depicted in Figure 4.1. ReACH includes multiple cores, an on-chip global accelerator manager (GAM) and reconfigurable accelerators attached to each level of the memory hierarchy. FPGA modules in each compute level have their own specific set of resources based on the power/area constraints of the attached memory level. While on-chip accelerator has the largest area and most resources (e.g. DSP, FF, BRAM, LUT), near-memory accelerators have to use a more power-efficient

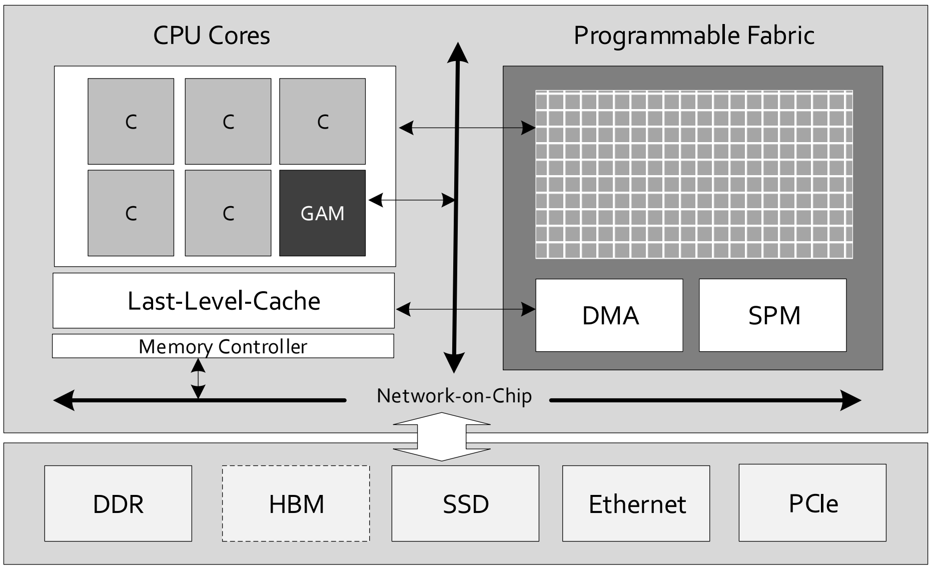


Figure 4.2: Detailed view of on-chip accelerator and GAM

embedded FPGA module. Even though a single near-memory accelerator is not as powerful as an on-chip accelerator, the shortcomings are balanced by each DIMM having its own dedicated near-memory FPGA module. As a result, near-memory accelerators can provide higher aggregated computation capability and higher aggregated bandwidth from main memory to accelerators. The near-storage FPGA module has power/thermal constraints similar to near-memory one, but in addition to an embedded FPGA module, it also requires a small dedicated DRAM buffer to act as a cache for accelerator parameters, to limit disk accesses and exploit the parameter's reuse ratio.

With careful hardware/software co-design in the global accelerator manager (GAM), ReACH can automatically handle the execution flow in the compute hierarchy and asynchronously pipeline the data transferring and execution between compute levels, while allowing user to write host applications with traditional synchronous coding style. The detailed description of each compute level and GAM is presented in this section. The software infrastructure is described in Section 4.3.

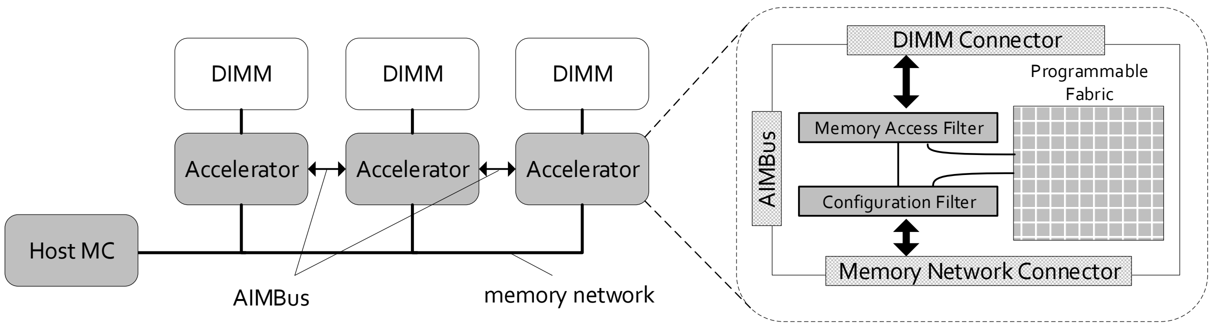


Figure 4.3: The accelerator-interposed memory architecture with the AIMbus which enables inter-DIMM communication (left). The internal architecture of each AIM module (right).

4.2.1 On-chip Accelerator

Figure 4.2 illustrates our on-chip hardware accelerator with CPU cores and GAM, all tied together with a high-bandwidth network-on-chip (NoC), which provides a cache-coherent interface between all elements and main memory. To enable closer logical integration to the host cores, virtual memory capabilities are supported by implementing TLBs and page table walkers for the accelerator [40].

On-chip FPGA accelerators often have higher clock frequencies and larger area to work with in order to keep up with the host cores and cache hierarchy, as is the case in Xilinx Versal ACAP [10]. Therefore, it can achieve high performance improvements by designing better FPGA kernels and utilizing the high-bandwidth access to cache. However, once the working set exceeds the on-chip cache capacity, the acceleration is limited by memory access latency and off-chip bandwidth.

4.2.2 Near-Memory Accelerator

While the performance of on-chip accelerators are bounded by the main memory bandwidth when data has to be fetched from off-chip, and little locality can be exploited, near-memory accelerators overcome the limit of narrow memory channels and achieve low-latency and high-bandwidth memory access by moving the compute engine closer to the main memory.

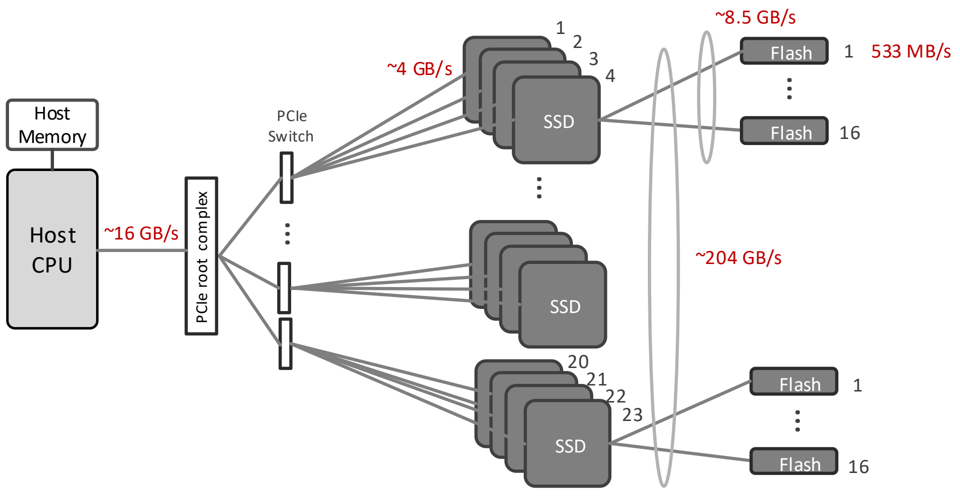


Figure 4.4: The throughput gap between the host PCIe bandwidth and the distributed SSD bandwidth. This gap grows as we scale-up the SSD units in the system.

Our design is based on accelerator-interposed memory (AIM) [19]. As shown in Figure 4.3, an AIM module is introduced to interface with the memory network and the commodity DRAM DIMM, making it a noninvasive design to existing components in the system. Each AIM module contains an embedded FPGA accelerator that can be customized and controlled by the GAM, an AIMbus interface that allows inter-DIMM communication, a configuration filter for accelerator commands coming from the memory channel, and a memory access filter to forward memory responses to local accelerator, remote accelerator via AIMbus or the host CPU via the memory channel.

This near-memory accelerator is generally treated as a co-processor and executes only when an application kernel is launched by the host CPU on the AIM module. Once a kernel is launched, the host memory controller hands over the control of a DIMM to the AIM module connected to it. The AIM module effectively enforces a *closed-row* policy when accessing the DRAM, so that the host memory controller can assume all rows are in *precharge* state when the control is handed back. This minimizes the amount of synchronization and data sharing that takes place between the host CPU and AIM modules, which is vital to the efficiency of near-memory acceleration.

As DRAMs and the memory channel are typically designed to offer high capacity and

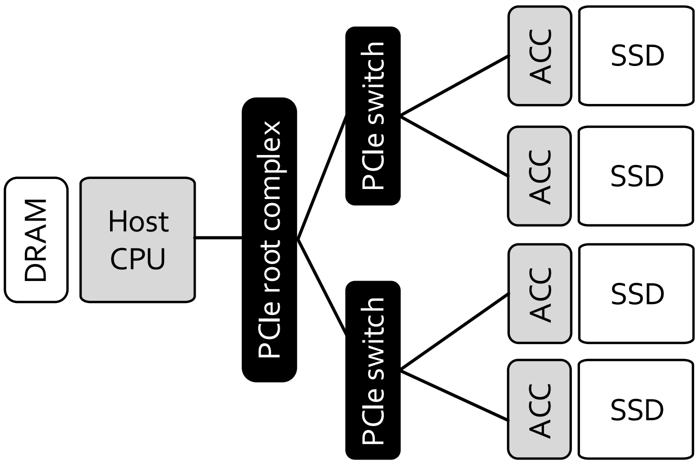


Figure 4.5: FPGA accelerator attached to each SSD unit in order to match the distributed bandwidth and utilize the large aggregated bandwidth.

low-latency, near-memory accelerators must avoid complex designs to work with the tight timing and power requirements of the DRAM standard. As a result, the FPGA accelerators are less powerful than the on-chip ones, but have lower memory access latency and can achieve higher aggregated bandwidth through parallel processing using multiple instances.

4.2.3 Near-Storage Accelerator

The disk internal bandwidth and latency has improved more than two orders of magnitudes in recent years and emerging non-volatile memory technologies have the potential to achieve near-memory bandwidth and latency. However, the host/disk IO interconnect throughput has not improved at the same rate and the system bottleneck is moved from Disk to IO interconnect. In an example system shown in Figure 4.4, the internal storage bandwidth can be as high as 204GB/s with 24 SSDs (solid state disks) attached through PCIe switch, while the host-side PCIe bandwidth is only 16GB/s. This bandwidth gap becomes even more substantial as we scale up the SSD units in the system. Therefore, the collective bandwidth of the array of SSD is underutilized.

Instead of improving the I/O interconnect throughput to bridge the gap, we move compute engines closer to the storage by connecting an FPGA accelerator to each SSD unit via a local

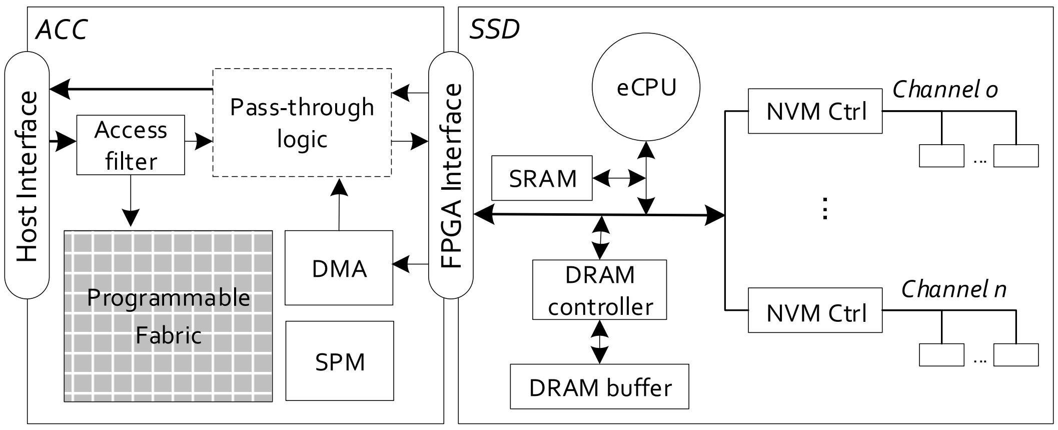


Figure 4.6: Internal architecture of the near-storage accelerator connected to a conventional SSD

PCIe link (Figure 4.5). Figure 4.6 illustrate the internal architecture of our near-storage accelerator. In addition to the user accelerator function unit (the prgrammable logic, DMA and SPM units), the FPGA accelerator has a host interface to receive accelerator commands, an FPGA-SSD interface to transfer data from/to the local SSD unit, and a control logic that filters the conventional disk access from accelerator commands. The pass-through logic allows the IO requests intended for the disk to pass with minimal overhead.

Similar to near-memory accelerators, near-storage accelerators are treated as coprocessors attached to the storage. They are designed to handle an entire computation kernel, eliminating the need for costly synchronizations with the host CPU. The FPGA chosen here must work with the cost and power budget of the server with respect to the number of SSD units in the system. The FPGA requires a private DRAM buffer to limit the number of IO access to attached storage unit. Near-storage accelerators work best for applications with reduction operations, where the data size is ideally reduced by orders of magnitude before data is transferred to the upper levels of the memory hierarchy.

4.2.4 Global Accelerator Manager

To efficiently coordinate between hardware accelerators in the compute hierarchy and free CPU cores from managing resources, we propose using an on-chip hardware-based global

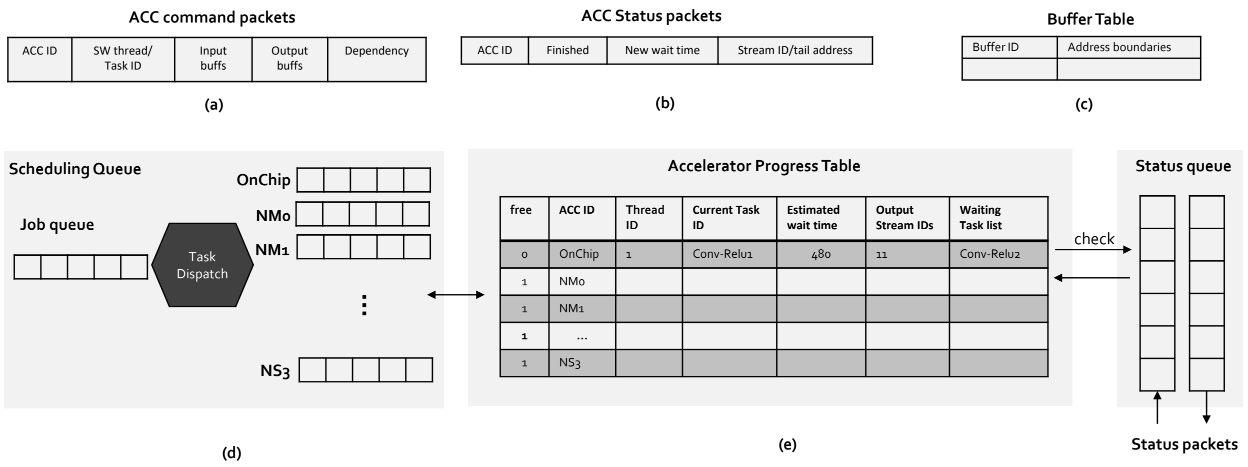

ACC command packets
ACC ID | SW thread/ Task ID | Input buffs | Output buffs | Dependency
(a)
ACC Status packets
ACC ID | Finished | New wait time | Stream ID/tail address
(b)
Buffer Table
Buffer ID | Address boundaries
(c)
Scheduling Queue
Job queue
Task Dispatch
OnChip
NM0
NM1
NS3
(d)
Accelerator Progress Table
free | ACC ID | Thread ID | Current Task ID | Estimated wait time | Output Stream IDs | Waiting Task list
0 | OnChip | 1 | Conv-Relu1 | 480 | 11 | Conv-Relu2
1 | NM0
1 | NM1
1 | ...
1 | NS3
(e)
check
Status queue
Status packets

Figure 4.7: GAM micro-architecture

accelerator manager (GAM) to 1) receive job requests for accelerators from cores; 2) distribute tasks within each job to available accelerators; 3) track the tasks currently running or waiting to run on accelerators, their start time and estimated execution time; 4) initiate data transfers between dependent tasks; and 5) interrupt the host core when the requested job is completed.

Figure 4.7 shows the micro-architecture of GAM that enables this capability. GAM features a simple scheduling queue, a progress tracking table, a small buffer table and TLB, and a status queue interfacing with rest of the system. The ReACH host-side runtime environment creates a series of job requests according to the job description of the host application. These job requests are sent to GAM in form of ACC command packets through the GAM driver (7a). GAM breaks each job into multiple tasks (called task groups) that may or may not assigned to same compute level. For instance, for a CNN inference job on a batch of query images, the job request is actually a series of tasks, each associated with one of the Conv-ReLu, Pool or FCN layers of the CNN model; all tasks in a task group share the same software thread id, but not necessarily the same target compute level. As an example, all layers of the inference job could be assigned to on-chip accelerator, or it could be divided to run all convolution layers using on-chip accelerator and all fully-connected layers using the near-memory accelerators. Each job goes through the GAM scheduling queue (7d) and get assigned to their target platform's dedicated queue and their input/output buffers are allocated and their address is stored in the buffer table (7c).

With multiple levels of accelerators, the GAM keeps track of the running accelerators using a progress table (7e). When a target hardware is set free from previous task, GAM invokes the next task from the queue by sending the command request to the target hardware. GAM acts as master to all accelerators and since memory/storage modules cannot send acknowledgement to GAM upon finishing of a task, it is GAM's responsibility to send status request packets (7b) to each running accelerator when the estimated runtime of a task finishes. If a task is done, the returned status packet will have the memory region address for the output of the task that could be forwarded to input buffers of all dependent tasks through DMA requests. If the task is not finished, new wait time value will be updated in the progress table.

If a job involves multiple compute levels, the GAM breaks the job into tasks to be assigned to different levels and makes sure results produced by one compute level can be fed to other levels. For near-memory accelerators, the GAM forces a write back in order to send the input data that were previously cached to the accelerators in memory. For near-storage accelerators, the GAM initiates PCIe transfer to send input data to the SSD-attached accelerators.

The GAM enables coarse-grained pipelining between different compute levels by allowing accelerators at different levels to work on different tasks at the same time. The accelerator tasks are intentionally designed to be small enough to exploit task-level parallelism but large enough to amortize the data transfer overhead. Also, the GAM assigns tasks from the next job to accelerators without waiting for all the tasks in the previous job to complete. This reduces idle time and improves the pipeline efficiency.

4.3 Programming ReACH

4.3.1 Software Infrastructure

Figure 4.8 presents the software stack of ReACH. To minimize the programming effort of using the compute hierarchy, a library-based accelerator programming model inspired by [3] is

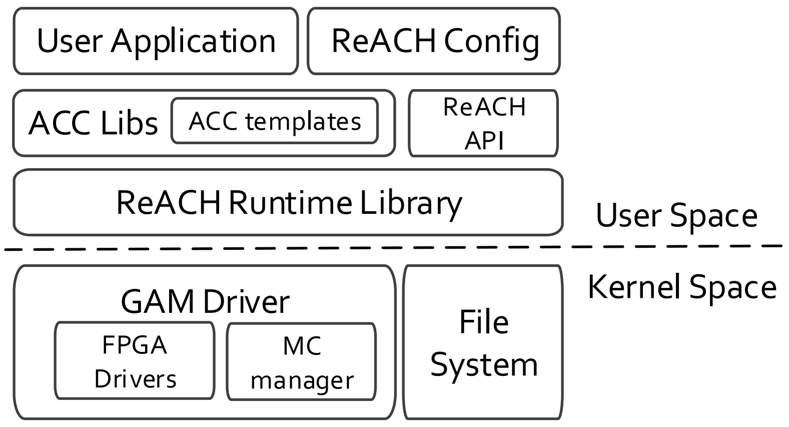


Figure 4.8: Software stack of ReACH

provided. For any new accelerator, once a compute kernel is carefully designed and generated for a specific compute level, the FPGA bitstream alongside a kernel-specific driver and data flow graph would be stored as an ***accelerator template***. The ***ReACH runtime library*** provides a comprehensive set of pre-optimized templates that are ready to deploy on FPGA devices. The library also has a general accelerator API that shows a uniform view of all FPGA resources regardless of their compute-level.

Listing 4.1: A Snippet of ReACH Host API (ReACH.h)

```
enum Level {OnChip, NearMem, NearStor, CPU}
enum StreamType {BroadCast, Collect, Pair}
ReACH::ACC RegisterAcc(string template, Level l)
ReACH::Buffer<typename T> CreateFixedBuffer(string real_path, Level dst,
    int size)
ReACH::Stream<typename T> CreateStream(Level src, Level dst, StreamType
    type, int size, int depth)
```

Listing 4.1 shows a snippet of ReACH host API written in C++. An application developer could use this API and optimized templates to write a ***ReACH config file*** and instantiate a meta accelerator, consisting of on-chip, near-memory and near-storage accelerators and all the required buffers and communication streams for it. These APIs enable users to 1) register an accelerator at each compute level, 2) create buffers at each level with data initialized from the file system, and 3) create buffers that can be transferred from source level to destination level using broadcast (one to all), collect (all to one), and pair (one to one) patterns.

Listing 4.2: ReACH Configuration (config.h)

```
#include <ReACH.h>
struct ImgData {...};
struct TopK {...};
ReACH::Buffer<float> vgg_param = CreateFixedBuffer("./vgg16_param", OnChip
    , size);
ReACH::Buffer<float> db0 = CreateFixedBuffer("./feature_db0",NearStor,
    DB0_size);
ReACH::Buffer<float> db1 = CreateFixedBuffer("./feature_db1",NearStor,
    DB1_size);

ReACH::Stream<ImgData> Input =
    CreateStream(CPU, Onchip, Pair, Img_size*Batch, depth);
ReACH::Stream<float> features =
    CreateStream(OnChip, NearStor, Broadcast, feat_size*batch, depth);
ReACH::Stream<TopK> Result =
    CreateStream(NearStor, CPU, Collect, batch*K*sizeof(int),depth);

ReACH::ACC cnn = RegisterAcc("VGG16-VU9P", OnChip);
cnn.setArgs(0,Input);
cnn.setArgs(1,vgg_param);
cnn.setArgs(2,Features);
ReACH::ACC knn0 = RegisterAcc("KNN-ZCU9", NearStor);
knn0.setArgs(0,Features);
knn0,setArgs(1,db0);
knn0.setArgs(2,Result);
ReACH::ACC knn1 = RegisterAcc("KNN-ZCU9", NearStor);
knn1.setArgs(0,Features);
knn1.setArgs(1,db1);
knn1.setArgs(2,Result);
```

Listing 4.2 shows an example config file for ReACH that instantiates a simplified CBIR meta accelerator using only on-chip and near-storage accelerators. ReACH config file contains the initial setup of the application for ReACH, such as initialization of physical accelerators (*ReACH:ACC*), allocation of each accelerator's memory region (*ReACH:Buffer*) and communication buffers between compute levels (*ReACH::Stream*). The main goal in ReACH is to limit data movement across the hierarchy during runtime of an application pipeline. So, it is important to formally define the fixed regions of the memory space where data would be sedentary and regions of the address space to be defined as communication buffer, so the intermediate result could be stored. The programming style of the config file is similar to OpenCL standard, where users can create buffers and streams, register accelerators, and associate the buffers and streams with accelerator kernel arguments.

Listing 4.3: Host Application Accelerator Calls (host.cpp)

```
#include <config.h>
while (Input.enqueue(new_query_batch)){
  cnn.execute(threadId);
  Features.broadcast();
  knn0.execute(threadId);
  knn1.execute(threadId);
  Result.collect();
  //process(Result.dequeue());
}
```

Listing 4.3 shows the host code to describe the flow of the application during run time using accelerator-specific API. Users simply need to call corresponding APIs to execute the accelerator and initiate the data transfer. We decided to separate ReACH configurations from the application host source code, because it allows the user application to (1) be as abstract as possible, (2) be portable across different ReACH systems and (3) allow GAM to balance the hardware resources during runtime. The ReACH runtime library also contains a GAM driver and a user interface that can translate the template execute function into communication packets to be sent to GAM. The kernel synthesis report—which includes pipeline initiation interval, depth and iterations, and frequency—is also used to update the GAM accelerator table with timing estimates of the new kernel. For an arbitrary application pipeline, the programmer could utilize the APIs to write codes with software pipelining of accelerators. During runtime of application, the accelerators could be invoked using function *execute* from the API. The core would offload the work to GAM to be scheduled for an available accelerator.

4.3.2 GAM Scheduling

Based on the ReACH config file, the runtime library calls the GAM driver to setup the memory regions for on-chip and near-memory accelerators. Initially the physical address space range is shared between CPU, on-chip accelerator and the near-memory accelerators. Since near-memory accelerators block accesses to the their attached DIMMs during kernel execution time, GAM reorganizes the memory space between the three components, by modifying the memory controller (MC) registers. The MCs that are connected to near-memory channels will divide the data in tile granularity specified by accelerator template, while the MCs that are connected to CPU/on-chip accelerator will interleave data with cache granularity for higher aggregated bandwidth to chip. The reorganization and isolation of the memory space for both compute levels helps decrease the access interference and give both accelerators their bandwidth requirements. The stream buffer defined for communication between two compute levels is actually a pair of queues allocated in the memory space of both source and destination compute levels. If the stream is broadcast type, the destination queue needs to be duplicated for each accelerator instance of that compute level. If the stream is collect type, all the source accelerators need to have a copy of the queue. Only GAM could request enqueue and dequeue operations for the stream buffers.

Figure 4.9 describe what happens in ReACH configuration:

1. ReACH runtime library sends the accelerator templates and buffer descriptions to GAM.

2. GAM detects the accelerator type and performs corresponding operations.

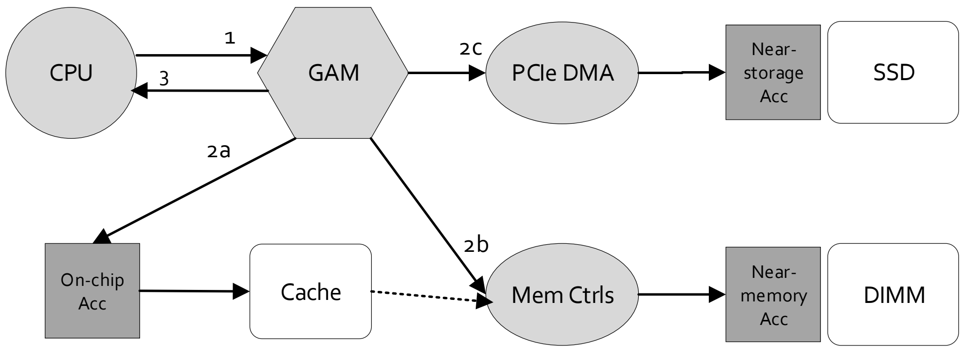


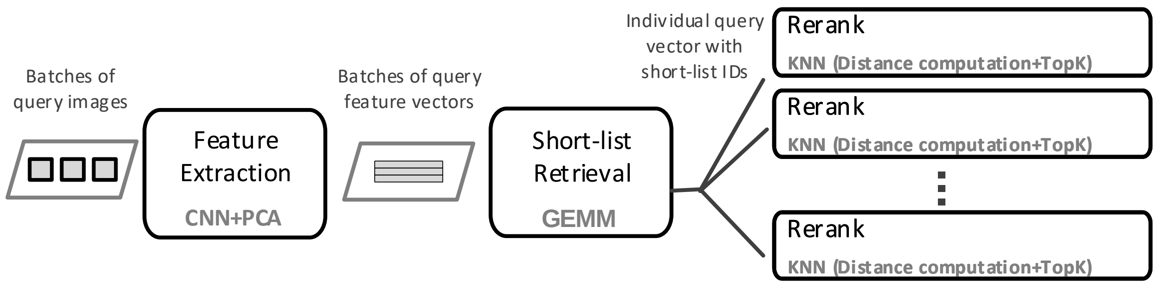
Figure 4.9: Scheduling of ReACH through GAM

(a) **On-chip Acc:** GAM launches the required kernel into on-chip Acc, update its table with the acc ID. It also sends DMA requests to load the required data (in buffers or streams) by on-chip accelerator into DRAM region for CPU, with high interleaving among channels.

(b) **Near-memory Acc:** GAM launches kernel into near-memory modules by writing into configuration filter of the them. It also updates the memory interleaving based on the tile size of the kernels. Finally, GAM sends DMA requests to load the required data by near-memory accelerators and divides it between DIMMs; if the source data of a stream is in cache, GAM further forces the cache write back to memory.

(c) **Near-storage Acc:** GAM launches kernel into near-storage modules by a user-defined NVMe command. It also receives the meta-data of storage files that are defined as fixed buffers of the accelerator; for those stream data that are in cache or memory, GAM forces the data write back to storage.

3. After updating the ACC table with empty task queues, GAM acknowledges the CPU that the data and accelerators are ready to receive tasks.



Batches of query images
Feature Extraction
CNN+PCA
Batches of query feature vectors
Short-list Retrieval
GEMM
Individual query vector with short-list IDs
Rerank
KNN (Distance computation+TopK)
Rerank
KNN (Distance computation+TopK)
Rerank
KNN (Distance computation+TopK)

Figure 4.10: Content-based image retrieval system pipeline

4.4 Case Study: CBIR Overview

We introduce the CBIR application briefly in Chapter 3. We use CBIR as a target application where data movement plays a vital role in application performance. In this section, we describe the stages of CBIR in more details. Figure 4.10 illustrates the application pipeline with three main kernels.

1. Feature extraction: User query images are first transformed into fixed-length feature vectors using a pre-trained convolutional neural network (CNN) followed by a PCA compression. CNN is the most compute-intensive kernel of the pipeline while requiring the least amount of memory among all steps. So, the on-chip accelerator can potentially achieve higher performance because 1) it has more resources to exploit parallelism and efficient routing and 2) on-chip SRAM provides low access to on-chip PEs.

Short list retrieval: To avoid exhaustive search on billion-scale databases, CBIR extract a short-list of cluster IDs that correspond to the centroids that are the closest to the query. These centroids are data points produced using clustering methods such as kd-trees or k-means during the off-line stage, so that the search space can be pruned at query time. At query time, the distances between the query and the centroids is computed using matrix-matrix operations.

Rerank: After short-list retrieval, we traverse the clusters and collects data points from

Table 4.1: The overview of memory/compute requirements of each CBIR pipeline stage and their corresponding mapping to ReACH levels

Stage	***Memory Requirement***	***Compute Requirement***	***ReACH Mapping***
Feature extraction	552 MB, 12.5 MB CNN model parameter, peak intermediate layer	*High* Convolutional neural network	On-chip
Short-list retrieval	~2.2 GB Cluster centroids and cell info	*Medium* Matrix-matrix multiplication	Near-memory
Rerank	~335 GB 1 billion feature vectors	*Low* Matrix-vector multiplication	Near-storage
Revers lookup	200 TB – 2 PB 1 billion images	*High* Database access	N/A

these clusters to form a candidate list. Rerank computes the similarity distances between the query and the data points within this candidate list. We use euclidean distance computation and partial sorting for this step.

To meet the strict timing constraints, some prior work adopt approximated methods such as hashing and product quantization in order to fit the working set into main memory of a single server node. While these methods could still benefit from multiple level of compute hierarchy (on-chip and near-memory), the overall accuracy of the result degrades significantly and therefore, it is not the focus of our design. But we will discuss the performance accuracy trade-off of these methods in Section 4.6.5.

4.4.1 Application Mapping

We target a CBIR system with 1 billion images. Each image is converted to a 96-dimension feature vector using VGG-16 [41] neural network and PCA compression. All feature vectors are preprocessed using k-means algorithm, generating a list of cluster centroids. The incoming user query images are first batched in the feature extraction and short-list retrieval steps, and then distributed according to their short-lists.

Table 4.1 summarizes the memory/compute requirement of the major kernel within each CBIR pipeline stage, as well as our proposed kernel-accelerator mapping. The user query image batch is first cached on-chip and then, converted to a batch of feature vectors by the on-chip feature extraction accelerators using the parameters entirely in on-chip SRAM. After the GAM is notified of the completion of the first step, it hands the feature vector batch to the near-memory short-list retrieval accelerators. These accelerators perform matrix-matrix multiplications between the incoming feature vectors and cluster centroids distributed between accelerator-attached DIMMs. Then, the GAM transfers individual query vector along with its retrieved short-list to the near-storage rerank accelerators. The rerank accelerators gathers dataset vectors from SSDs, computes distance to the query and perform a partial sort. Finally, the top K images are retrieved from the original image database and returned to the host. As we can see from the figure, the only data movement required is the user query vector and retrieved short-list. Other than that, the pipeline potentially touches several hundreds of data without actually moving any of them across the memory hierarchy.

4.5 Experimental Setup

Performance evaluation. To evaluate the performance of ReACH, we extend the open source cycle-accurate accelerator-rich architecture simulator PARADE [3] to model accelerators near the DRAM and SSD. We evaluate a system with on-chip accelerators attached to coherent shared cache, accelerators attached to the main memory, and accelerators attached to SSDs. Our on-chip accelerator is modeled based on Xilinx Virtex Ultrascale+ VU9P FPGA [42] and is coherently attached to CPU using a cache-coherent interconnect. Our near-memory accelerator is modeled after AIM [19] where an embedded Zynq Ultrascale+ ZCU9EQ FPGA is placed between each DRAM DIMM and the memory bus. An AIMbus connects each AIM module to enable inter-DIMM communication. The near-storage accelerator is configured as the same Zynq Ultrascale+ ZCU9EQ FPGA with a 1GB DRAM buffer and a PCIe link connecting to the SSD instance. Table 4.2 summarizes the configuration of the system.

Table 4.3 shows the list of kernels designed for our experiment: including convoluational neural network (CNN), matrix multiplication (GeMM), and k nearest neighbors (KNN). It

Table 4.2: Experimental setup of the compute hierarchy system

Component	**Parameters**
CPU	1 X86-64 OoO core @ 2GHz 8-wide issue, 32KB L1, 2MB shared L2
Memory Controller	2 MCs, 64/64-entry read/write request queue, FR-FCFS
Memory System	DDR4 DIMMs, 4 for near-memory accelerators and 4 for on-chip accelerator
Storage System	4 NVMe SSD attached with PCIe gen3x16
On-chip Accelerator	Virtex Ultrascale+ [42] with 100 GB/s connection to shared cache
Near-Memory Accelerator	Zynq Ultrascale+ [42] , 18 GB/s bandwidth to DDR4 DRAM and AIMbus
Near-Storage Accelerator	Zynq Ultrascale+ [42] with 1GB DRAM , 12GB/s effective bandwidth to NVMe SSD

Table 4.3: FPGA utilization, frequency and power for each accelerator

FPGA	Kernel	Utilization (FF,LUT,DSP,BRAM)	Kernel Freq	Power (W)
Xilinx Virtex Ultrascale+ XCVU9P	CNN	85E4(36%), 95E4(81%), 3078(45%), 1814(42%)	273 MHz	25
	GeMM	56E4(24%), 32E4(27%), 2188(32%), 3326(77%)	273 MHz	22.13
	KNN	23E4(10%), 73E4(10%), 128(2%), 3628(84%)	200 MHz	14.14
Xilinx Zynq Ultrascale+ ZCU9EQ	CNN	19E4(36%), 16E4(27%), 1916(76%), 1674(92%)	200 MHz	5.19/6.13
	GeMM	6E4(11%), 18E4(31%), 958(38%), 656(36%)	150 MHz	5.3/8
	KNN	8E4(16%), 35E4(60%), 64(3%), 1456(80%)	150 MHz	2.4/3.8

also lists the FPGA utilization, kernel frequency and the estimated power for each kernel. There are two numbers for Zynq FPGA power which represent the power for near-memory and near-storage accelerator respectively. The near-storage accelerator has a small DRAM buffer and interface that increase the dynamic power compared to near-memory accelerator.

We compare the compute hierarchy against the baseline system which only has the on-chip FPGA accelerator. The three steps of the CBIR application is individually designed and optimized for both Virtex Ultrascale+ and Zynq Ultrascale+. Then, the required parameters for simulation such as kernel frequency, initiation interval, pipeline depth and iterations are extracted from the synthesis result and plugged into our ReACH simulator.

Energy evaluation. To estimate the energy consumption of ReACH, we use SDAccel [43] environment's post-routing power reports to estimate the power consumption of the on-chip accelerator. The reports are further used along with Xilinx Power Estimator (XPE) tool [44] to estimate the power for near-memory and near-storage accelerators, by adjusting the operating frequency, utilization and on-board resources. While on-chip accelerator has one instance, for near-memory and near-storage, we vary the number of DIMMs or SSD instances that are paired with FPGA modules. Table 4.4 shows a summary of tools and references we

Table 4.4: Energy model tools and references

Component	**Reference**
FPGA Accelerators	Xilinx SDAccel 2019.1 [43] and XPE power calculator [44]
Cache	CACTI 6.5 [45]
DRAM	Micron DDR4 Power Calculator [46]
Storage	NVMe SSDs [47] with PCIe Gen3x16 interfaces
Interconnect	Host/IO interface switch [48], PCIe links [12] and Memory channels [49]

used for our energy analysis. We estimate the energy consumption for all other components of the baseline system and ReACH architecture, except for the CPU power as it would be dependent on the workload running on the CPU and is almost idle in our case.

CBIR setup. For the CBIR pipeline simulation, we use a batch of 16 image queries. We preprocess the database image feature vectors with k-means to obtain 1000 cluster centroids. These centroids will later be used to retrieve the short-list for each query. In the final rerank step, we compare each query against 4096 data points based on the short-list to make the simulation time manageable. Overall retrieval accuracy is evaluated using a software pipeline that implements the same exact algorithms chosen by the hardware.

4.6 Evaluation Result

Table 4.1 summarizes the memory/compute requirement of the major kernel within each CBIR pipeline stage. The details of CBIR pipeline are in section 4.4. Each image batch is first converted to a batch of 96-dimensional feature vectors (*feature extraction*). These vectors are used as query to retrieve the closest images from the database. The query vectors are first compared with a list of cluster centroids to find a few candidate clusters in the database (*short-list retrieval*). Then, the query vectors needs to be compared with vectors in each candidate cluster and find the top closest matches (*rerank*).

To illustrate the limitations of compute-centric acceleration approach, we analyze the energy distribution between components of the system during on-chip acceleration of CBIR pipeline in Section 4.6.1. We discuss the performance and energy cost of each CBIR kernel at various compute levels of ReACH in Section 4.6.2. Then we implement an end-to-end CBIR

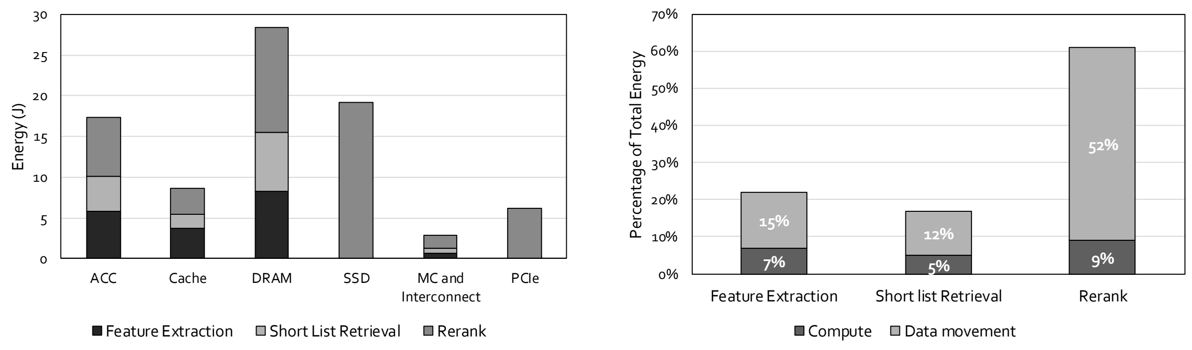


Figure 4.11: The energy consumption breakdown for accelerating CBIR pipeline using on-chip accelerator

pipeline only using accelerators at one compute level at a time and compare their performance and energy efficiency in Section 4.6.3. To demonstrate the effectiveness of ReACH and the proposed application mapping, in Section 4.6.4, we present the result for the end-to-end CBIR with proper application mapping to all compute levels of ReACH. We compare the latency/throughput of query response and give a detailed breakdown of energy spent on each system component while running CBIR on ReACH system. To demonstrate the flexibility of ReACH, we also implement three different CBIR pipelines using different levels of the compute hierarchy and discuss their performance and accuracy.

4.6.1 Energy Breakdown for On-chip Acceleration of CBIR Pipeline

To better understand the shortcomings of the on-chip accelerator, we implemented the end-to-end CBIR pipeline using only the on-chip FPGA. We used an optimized kernel for each step of the pipeline and do not account for the partial reprogramming delay, since today's FPGA technology can reduce this delay to sub-millisecond which is appropriate for latency-sensitive applications [10]. Figure 4.11 shows the total energy consumption for one batch of data, when accelerating the end-to-end pipeline on chip. It also shows a distribution of energy across system components, as well as different steps of the CBIR pipeline. In this implementation, 80% of the total energy cost is due to the data movement across the memory hierarchy. In fact, 52% of the total cost is for data movements of Rerank step. The

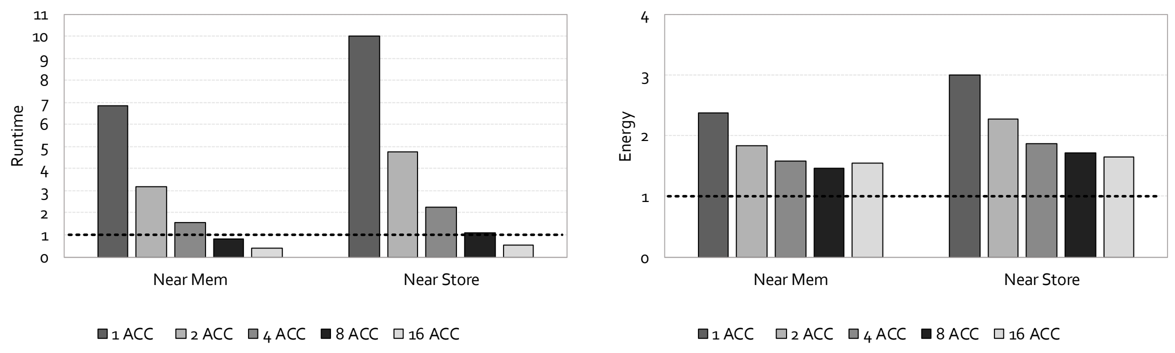
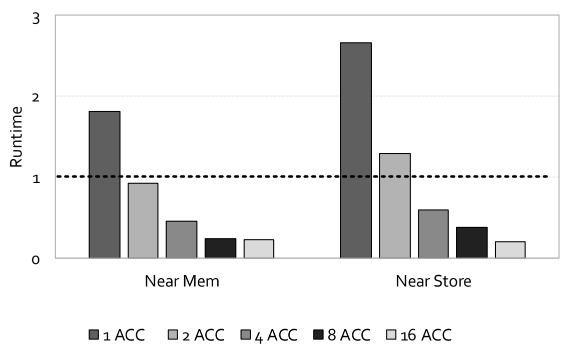


Figure 4.12: Runtime and energy consumption comparison of the feature extraction step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

data accessed in disk is only used once but incurs a large portion of DRAM and Disk energy consumption. In our subsequent experiments, we analyze if moving the computation near the data medium (DRAM or disk) will help reduce the energy cost and help with performance improvements. The on-chip implementation is our baseline model while comparing various acceleration options.

4.6.2 Performance and Energy Cost for Each Stage at Different Compute Level

Feature extraction. The main kernel of feature extraction step is a convolutional neural network, that is pre-optimized for each ReACH level. Figure 4.12 illustrates the runtime and energy consumption for CNN implemented using near-memory and near-storage accelerators. The result is normalized to on-chip accelerator's run time and energy cost. The initial network parameters are stored in DRAM for both on-chip and near-memory accelerators and for near-storage accelerators, the parameters are pre-loaded in private device DRAM and each instance has their own copy. For on-chip accelerator, the parameteres are interleaved in cache granularity between different memory channels for higher bandwidth to CPU and on-chip accelerator, But they need to be continuous and tiled for near-memory accelerators. The feature extraction workload has a high data-reuse ratio and could benefit from a large reconfigurable fabric for placements of PEs that share concurrent access to a large SPM. Thus,



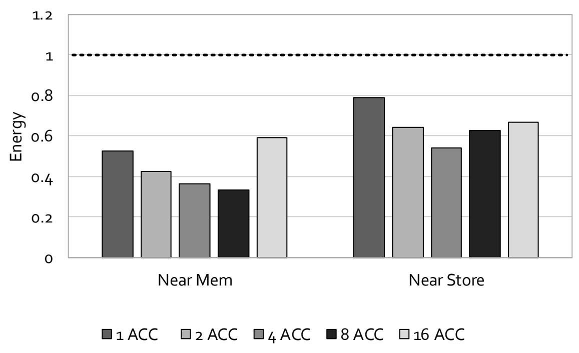


Figure 4.13: Runtime and energy consumption comparison of the short-list retrieval step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

when comparing a single instance of CNN in each compute level, the on-chip accelerator has a clear advantage over others thanks to larger area, operating frequency and high-bandwidth access to last-level cache (7-10x). As the number of instances grow, the collective performance of near-memory accelerators surpass the on-chip one. However, on-chip accelerator has the best overall energy.

Shortlist Retrieval. A similar runtime/energy comparison on the short-list retrieval step is illustrated in Figure 4.13. The on-chip accelerator performance is bounded by the bandwidth of loading data from DRAM as they do not fit in on-chip SRAM. The near-memory accelerator achieves better performance when there is 2 or more instances, due to the support of AIMbus and higher aggregated DRAM bandwidth. It also achieves up to 40-60% energy reduction compared to on-chip accelerator. For near-storage accelerator, accessing centroids through PCIe bus limits the performance and energy reduction. The near-storage accelerator has slightly higher runtime as the near-memory accelerator, because the latency of device DRAM access is more than the latency of DIMM access.

Rerank. Figure 4.14 shows the rerank stage runtime/energy comparison of all implementations. The rerank stage, uses KNN to find the closest images to query. KNN is an IO-intensive streaming application with a simple compute unit and no data reuse. As the input data can only be stored in the storage, on-chip and near-memory accelerators all have

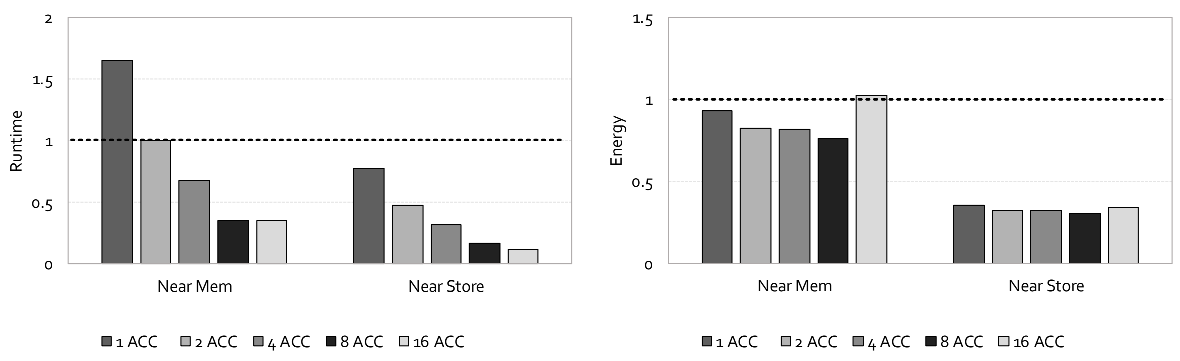


Figure 4.14: Runtime and energy consumption comparison of the Rerank step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

to fetch data from the SSDs via PCIe. As a consequence, the performance is heavily limited by the I/O bandwidth, which is quickly saturated as we add more near-memory accelerators. The near-memory accelerator achieves speedup but reaches a plateau when having more than 8 instances. This plateau is clearly due to high latency and limited bandwidth of host/IO interface. Moving data would also incur significant energy overhead on host/IO interface and host DRAM. On the contrary, the near-storage accelerators allow us to expose the full bandwidth of SSDs, so that higher aggregated bandwidth can be easily achieved by scaling up the number of FPGA-SSD units in the system and each accelerator can be kept busy. The Rerank step could save up to 60% of its energy cost by moving from on-chip to near-storage acceleration.

4.6.3 Overall Performance and Energy Efficiency using Single Compute Level

As described in Table 4.2, our experimental setup has 8 DRAM DIMMs and 4 NVMe SSDs. We use this setup for our energy estimations, so we scale the number of our near-memory accelerators from 1 to 2 and 4. The other half of the DIMMs are reserved for CPU and on-chip accelerator. Figure 4.15 shows the run time and energy cost for end-to-end implementation of CBIR pipeline using a single compute level at a time. The on-chip accelerator is our baseline and has one instance.

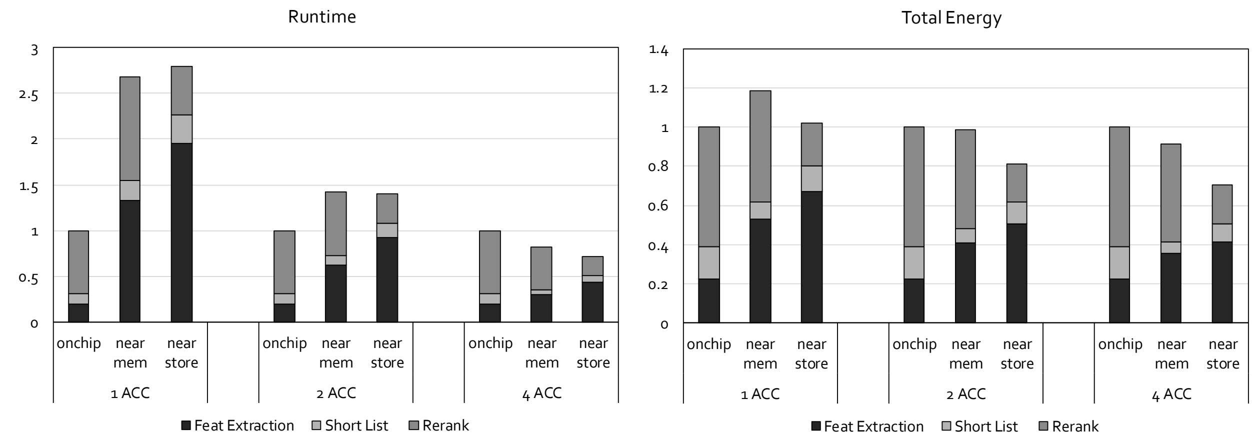


Figure 4.15: The total runtime and energy cost of CBIR normalized to the baseline on-chip accelerator

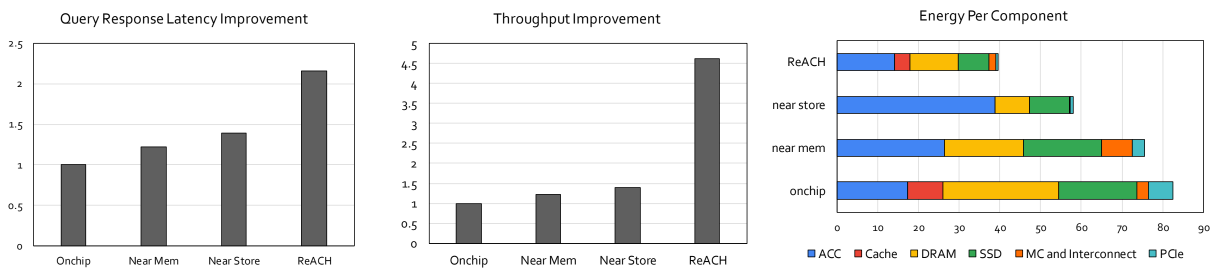


Figure 4.16: The performance and energy consumption of CBIR running on ReACH compared to onchip, near memory and near storage acceleration

When comparing on-chip with single instance of near-memory and near-storage accelerator for CBIR, on-chip accelerator performs better due to the powerful on-chip FPGA. But as we scale the number of near-data processing units to 4, we see higher performance and energy gains in CBIR pipeline. This shows that near-memory and near-storage accelerators are most effective when they benefit from aggregated bandwidth of their attached memory modules. While short-list retrieval and rerank stages benefit from offloading tasks to near-data processing units, feature extraction stage has to be modified to match the decentralized compute levels. Instead of working on batch of images, each near-data processing unit works on a single image and uses duplicated parameters. This is one of the limitations of moving an entire application pipeline near the storage; not all parts of an application could benefit from distribution and decentralization.

4.6.4 Performance/Energy efficiency after Proper Mapping

To reach even higher performance and energy efficiency, we propose to use a combination of on-chip, near-memory and near-storage accelerator in concert with each other, to do the computation. The optimized mapping is presented in Section 4.4.1: feature extraction uses on-chip accelerators, short-list retrieval uses near memory accelerators, and rerank uses near storage accelerators. As we discussed in Section 4.2.4, GAM assigns new tasks to each compute level as soon as the resources become available without waiting for the whole job to be finished, so the query processing throughput mainly depends on the longest stage of the pipeline, not the total run time of all stages. We compare the query latency, throughput and energy consumption of CBIR pipeline with 4 different acceleration options in Figure 4.16.

As shown in Figure 4.16, the throughput of ReACH improves 4.5x compared to on-chip acceleration only, when we map the application pipeline to multiple compute levels. We also see 2.2x improvement in query response latency due to proper mapping of CBIR stages. From the energy standpoint, the proper mapping achieves the highest energy efficiency, with 50% energy reduction compared to the baseline on-chip accelerator. The energy reduction mainly comes from lower SSD access time and lower usage of main memory to maintain the streaming data, as well as decrease in interconnect energy.

4.6.5 ReACH Flexibility

CBIR alternative 1: compressed vectors. In modern software CBIR systems, sometimes throughput is favored over recall accuracy. Some prior studies adopt techniques such as code book or product quantization to further compress the feature vector, so that the size of all database vectors can fit in the main memory of a single server-class machine and the rerank step becomes more efficient with a smaller vector size [34, 35]. The downside, however, is the overall retrieval accuracy is penalized by the feature vector compression.

To support a CBIR pipeline using approximate nearest neighbor methods, we map the feature extraction kernel to the on-chip accelerator, and map both short-list retrieval and rerank kernels to the near-memory accelerator, leaving the near-storage accelerators unused.

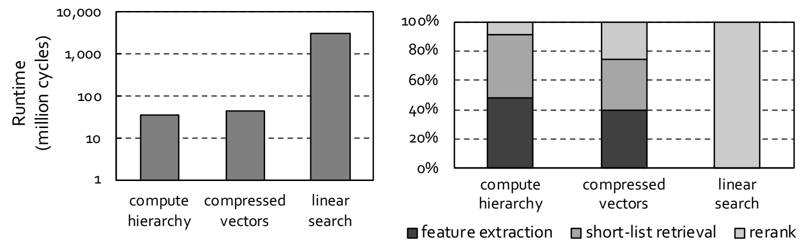
Note that these kernels need to implement or be aware of the compression method, making them different than those in our evaluated pipeline. During runtime, the GAM makes sure the data flow from on-chip to near-memory accelerators but not to near-storage ones by invoking proper accelerators with the location of the input data in the memory.

CBIR alternative 2: linear search. At the other end of the trade-off is the linear search approach, where each query vector is compared against the *entire list* of data points using full vectors representation. This approach provides 100% accuracy and often serves as the ground truth in prior research. However, the amount of comparison and data movement required in this approach makes it prohibitive for software system implementations with quality-of-service constraints.

To deploy a CBIR pipeline with linear search, we simply skip the short-list retrieval step. After the feature extraction is completed by the on-chip accelerators, the GAM initiates PCIe transfer to directly push query vectors to the near-storage accelerators for linear scan.

Results for different CBIR implementations. Figure 4.17(a) presents the execution time comparison between the three CBIR algorithms deployed. The linear search approach takes almost 1000x longer to finish due to it exhaustive computation on the entire database. The execution time of the exact vector approach (compute hierarchy) is on par with the compressed vector approach, even though compressed vector allows fitting all compressed vectors in DRAM and thus the computation is expected to be more efficient. The reason is that the compressed vector approach has to compare a lot more data points against each query in the rerank step in order to compensate for the overall retrieval accuracy, which offsets the saved execution time.

Figure 4.17(b) illustrates the execution time breakdown of the three algorithms deployed. The linear search approach is completely dominated by the rerank step as it does not have a short-list retrieval step and the data size of the rerank step is too big. The exact vector and compressed vector have more balanced distribution, while the compressed vector spends more time on rerank to improve accuracy. Overall, the retrieval accuracy of linear search, exact vector and compressed vector are 100%, 94% and 50% in the top 10 retrieved results.



(a) Execution time (b) Execution time breakdown

Figure 4.17: Execution time comparison of the three different algorithms deployed. From left to right is the compute hierarchy with exact vectors, compressed vectors and the linear search approach that skips the short-list retrieval step. The y-axis is in log scale.

4.7 Conclusion

In this work, we present ReACH, a reconfigurable accelerator compute hierarchy that combines on-chip, near-memory and near-storage accelerators, spanning all levels of the conventional memory hierarchy. We propose a holistic approach to coordinate between each compute level and managing resources. To minimize the programming efforts of using the compute hierarchies, a uniform programming interface is designed to decouple the ReACH configuration from the user application source code and allow runtime adjustments without modifying the deployed application. We experimentally deploy a billion-scale content-based Image Retrieval system on ReACH and demonstrate that based on user concern of the best throughput or most energy-efficient, a proper application mapping eliminates unnecessary data movement and achieves 4.5x throughput gain while reducing energy consumption by 52% compared to an on-chip acceleration. In summary, the compute hierarchy offers both distributed computational power which potentially eliminates data movement, and the flexibility to adjust to the compute and memory requirements of different applications and different trade-off points within the same application pipeline.

CHAPTER 5

Near Data Processing Platform Selection using Fast Performance Modeling

5.1 Introduction

In Chapter 4, we introduced ReACH, a new server architecture and a distributed computing platform comprise of on-chip, near-memory and near-storage accelerators. We demonstrate that the FPGA-based compute resources could collaborate together using a global accelerator manager (GAM) and a uniformed programming interface, in order to enhance the performance and energy efficiency of a data analytics pipeline. Each data center server could incorporate one or many of these FPGAs at different compute hierarchy levels to match its workload intensity. A server workload includes diverse set of data-intensive and compute-intensive application [32]. A number of these applications are presented in Table 5.1. These applications share the compute hierarchy and their throughput is crucial to the user experience. Thus, it is important to have a methodology to select the best compute level for each application.

In this Chapter, we present an in-depth analysis of applications' qualitative and quantitative attributes to derive an accurate performance model that could answer the following questions: (1) could all applications dealing with in-storage/in-memory datasets benefit from a data-centric acceleration? (2) How would the performance change for each application, as we change the accelerator type and system configuration? The model could be used by the programmer or GAM in order to load-balance the accelerators. Our benchmark suite includes widely-deployed accelerators from the Xilinx Vitis Library [50] and few of our own. We analyze 18 benchmarks from 6 domains and create a guideline for both application and hardware developers. Unlike prior work on performance estimation and workload characterization,

Table 5.1: Benchmark List

Domain	Kernel	Functionality	Freq (MHz)	bitwidth	Access pattern
Search	KMP	Find all occurrence of a pattern in each file	300	512	Sequential
	KNN	Find the closest vectors in a dataset to a given vector	250	1024	Sequential
Security	AES-ENC	File encryption using a 256-bit cipher code	273	512	Sequential
	AES-DEC	File decryption using a 256-bit cipher code	273	512	Sequential
Database	Select	Filtering rows of a table based on compare operator	400	512	Sequential
	PHJ	Partition Hash Join two table into one	182	512	Scatter-Sequential
	Aggregate	Grouping multiple rows of a table	193	512	Sequential
	Partition	Partition one table's rows into clusters	200	512	Scatter
	Combine-2Col	Merge two columns of a table into one column	200	1024	Gatter
	Snappy	High-throughput data compression method	380	256	Sequential
	Sort	Sort rows of table based on bitonic/merge tree sort	220	512	Scatter-Sequential
Vision	EqualizeHist	Improve contrast in the image or video	200	512	Tiled
	Viola-Jones	Face detection through Frames	273	512	Tiled
	FFT2D	Transform an Img between spatial/frequency domain	250	512	Sequential
	Conv	Convolution for a given 3D kernel	250	512	Tiled
	GeMM	Dense Matrix-Matrix multiplication	250	512	Tiled
Finance	Black Scholes	Stock option price prediction	344	256	Sequential
	Swaption	Swaption prices using Monte Carlo simulation	250	256	Sequential

[51–53], we take into account all kernel-specific characteristics, study three NDP acceleration schemes and target a broader range of application domains.

5.2 Workload Characterization

Table 5.1 shows the diverse set of application domains and FPGA kernels that we use for our evaluation. Selecting a compute level for an FPGA kernel is not trivial and depends on the application use-case as well as the kernel characteristics. To give a simple example, a GeMM kernel is used for matrix multiplication of A and B in two different use cases.

1. $Dim_A = Dim_B = 1024 \times 1024$ squared matrices.

2. $Dim_A = (65536 \times 128), Dim_B = (128 \times 128)$.

They both have same number of floating-point operations, but their working-set-size (12MB vs 64 MB) and tile reuse ratio (8 to 1) are different. These characteristics play a role in selecting their compute level.

Based on the goal of minimizing and amortizing data movement cost for a long period of time, there are a few application-specific and kernel-specific characteristics that help choose the best acceleration platform.

5.2.1 Application-Specific Characteristics

There could be multiple FPGA kernels designed for a given task. However, there are certain attributes that are more dependent on the use-case of an application rather than the FPGA kernel itself, which play a role in ruling out a compute level.

1) *Memory Capacity Requirement*: The data size that is potentially accessed, modified or generated by the same application's threads over a long period of time.

2) *Working Set Size (WSS)*: We define WSS as the subset of the memory capacity which is required to process a single application call. For instance, a query processing engine equipped with Join, Aggregate and Partition FPGA kernels, might receive consecutive queries that target dozens of SQL tables, while working on a single table (or two) for each query. We consider the entire database size as the memory capacity and size of the requested database tables per query as the working set size.

3) *Input/Output Data flow*: If an application data flow never requires access to the storage level, then we would certainly rule out a near-storage acceleration. Similarly, if the application requires an in-situ manipulation of data (read-modify-write), then we would favor the near-storage or near-memory accelerators.

4) *Data Reduction Ratio (γ)*: The main performance motivation of near-data processing is to leverage a higher internal bandwidth, so if an application output size is the same as its input, the application speed-up will be bound to the interconnection bandwidth for output. Figure 5.1 shows the reduction ratio γ of our applications. An applications with $\gamma = 1$, would be either compute-bounded or interconnect-bounded.

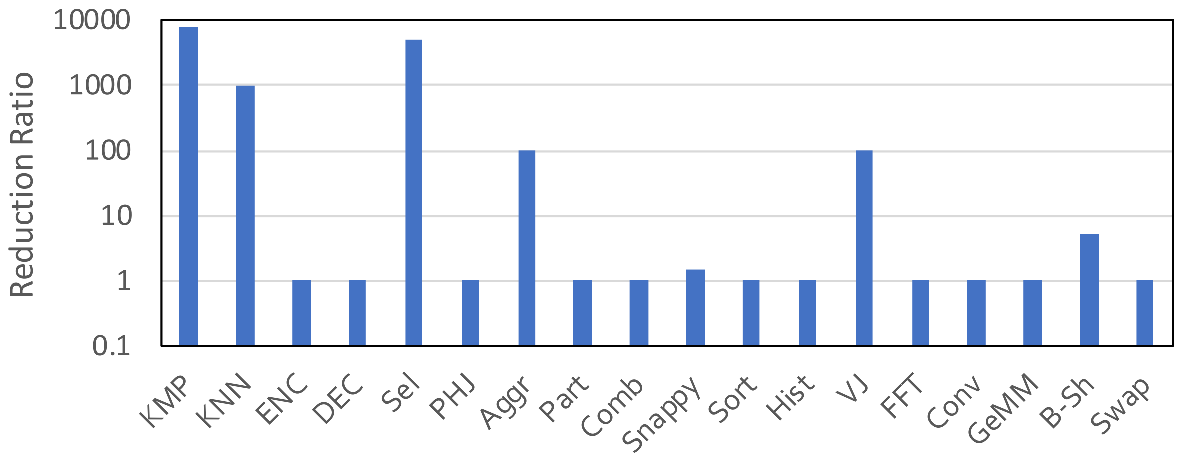


Figure 5.1: The ratio of the kernel's input to its output. The Y-axis is in log scale.

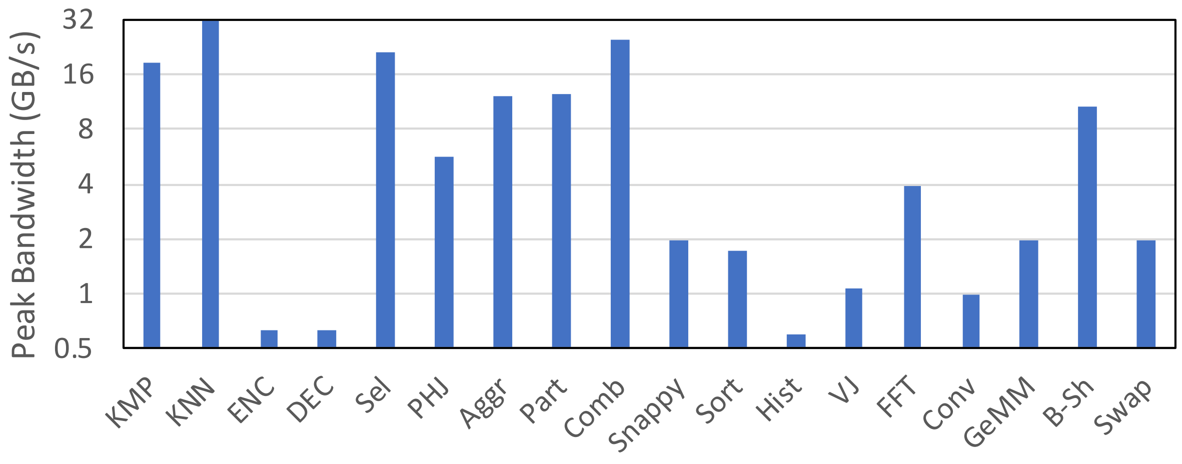


Figure 5.2: The analytical peak bandwidth of the kernel, if not bound by the attached memory bandwidth. The Y-axis is in log scale.

5.2.2 Kernel-Specific Characteristics

For a given FPGA kernel, there is plenty of useful information that can be extracted from the kernel and its high-level synthesis, which help us derive a performance model.

1) *Kernel Freq, Initiation Interval and input/output bit-width*: which help us compute the maximum analytical bandwidth. This way, we can determine if a kernel would ever saturate the bandwidth of the attached module. $bw_{peak} = datawidth \times Freq/II$. Figure 5.2 shows the peak bandwidth per FPGA kernel. Applications like ENC and Viola-Jones have a low

peak bandwidth and will not benefit from running near storage.

2) *Attached memory access pattern*: which helps us estimate the effective physical bandwidth utilization of the attached memory. Sequential access with large burst size result in higher effective bandwidth than scatter or gather patterns. The effective bandwidth of a tiled access depends on the tile row size.

3) *Multipass over the data* (α): Maximum analytical bandwidth does not take into account the number of passes over the input data. So, realistically the system throughput could be much lower than the kernel bandwidth. (α) is the number of extra passes over the input data.

4) *Intermediate data size (β)*: Similar to multiple passes over the input data, another contributing factor to the throughput is the generated intermediate data.

5.3 Performance Model

In this section, we present an execution model for in-storage datasets based on features from section 5.2.1 and 5.2.2. The assumption is that the WSS could fit in the main memory, but the initial input is resident in storage level. If a dataset is accessed regularly, the initial load from storage becomes amortized and insignificant. Regardless of the compute level, all of our FPGA-based accelerators have an internal scratchpad memory (SPM) and use the Xilinx AXI stream interface [54] to communicate with the attached memory level. The AXI stream interface supports a maximum bus bit-width of 4096. But using a built-in data-width and clock rate conversion, the AXI4 IP can support various DDR or flash module connections. Table 5.1 summarize some of the characteristics that we use in our performance model. We adopt a three stage accelerator execution model (load-compute-store) and double-buffering in SPM. The three stages can be scheduled to run concurrently. Therefore, each compute level l follows this equation:

$$T(l) = Max(T_{load}(l), T_{comp}(l), T_{store}(l)) \tag{5.1}$$

The kernel execution time is bounded by one of the terms T_{load}, T_{comp}, or T_{store} (Equation 5.1). For near-storage accelerator:

$$T_{load}(ns) = \frac{D_{in}}{bw_{nvm}} + \frac{\alpha D_{in}}{bw_{nvm}} + \frac{\beta D_{in}}{bw_{ddr}} \tag{5.2}$$

$$T_{comp}(ns) = \frac{(1+\alpha+\beta).D_{in}}{datawidth} \times \frac{II}{Freq} \tag{5.3}$$

$$T_{store}(ns) = \frac{D_{out}}{bw_{hostIO}} = \frac{D_{in}}{\gamma \times bw_{hostIO}} \tag{5.4}$$

where α is number of extra passes over the data, and β is the relative size of intermediate data generated compared to initial input. The α and β are algorithm-dependant and is collected manually. In future work, we plan to support automation through software profiling. The near-storage accelerator uses its own DRAM buffer as caching for intermediate variables. γ is the data reduction ratio, discussed in section 5.2.1. II is the initiation interval which is the number of cycles that the FPGA kernel takes to process one input of *datawidth* size. The II shows the computational intensity of a kernel. A kernel with $II = 1$ has a low computational intensity and adding more resources and PEs to the FPGA provides no extra benefit. But kerels with $II = n$ could benefit from having up to n PEs that share the data. bw_{nvm} is the collective bandwidth of the NVM channels that feed data to SSD controller. bw_{ddr} is the local DRAM buffer bandwidth. bw_{hostIO} is the effective bandwidth of the host/IO interface.

For near-memory accelerators, all data except the initial load from storage is accessed through the attached memory module. Therefore,

$$T_{load}(nm) = \frac{D_{in}}{bw_{hostIO}} + \frac{\alpha D_{in}}{PE \times bw_{ddr}} + \frac{\beta D_{in}}{PE \times bw_{ddr}} \tag{5.5}$$

$$T_{comp}(nm) = \frac{(1+\alpha+\beta)D_{in}}{datawidth} \times \frac{II}{Freq \times PE} \tag{5.6}$$

$$T_{store}(nm) = \frac{D_{in}}{\gamma \times bw_{ddr} \times PE} \tag{5.7}$$

If a workload consist of multiple subsequent queries accessing the same dataset (e.g.

multiple database select operations after a hash join operation on two tables), the α will increase.

The On-chip accelerator's bandwidth to shared memory is limited by number of memory channels and their effective bandwidth: $bw_{DRAM} = n_{ch} \times bw_{ch}$. For simplicity, we assume $bw_{ch} = bw_{ddr}$ The total execution time for on-chip accelerator is the maximum of the three stage:

$$T_{load}(oc) = \frac{D_{in}}{bw_{hostIO}} + \frac{\alpha D_{in}}{n_{ch} \times bw_{ddr}} + \frac{\beta D_{in}}{bw_{cc}} \tag{5.8}$$

$$T_{comp}(oc) = \frac{(1+\alpha+\beta)D_{in}}{datawidth} \times \frac{II'}{Freq} \tag{5.9}$$

$$T_{store}(oc) = \frac{D_{in}}{\gamma \times bw_{ddr} \times n_{ch}} \tag{5.10}$$

bw_{cc} is the interconnect bandwidth between CPU and the on-chip accelerator, which is considered the effective cache bandwidth. Please note that initiation interval of the kernel is defined as II', because on-chip accelerator has more FPGA resource compared to one instance of near-memory or near-storage accelerator. But more resource could only decrease the original II, if it was more than 1. This also means that on-chip accelerators disproportionately benefits the kernels with high computational intensity.

5.4 Evaluation

The system configuration consists of one on-chip FPGA, four DDR4 DIMMs and four NVMe SSDs (each connected to one near-memory/near-storage chip). There is only one near-storage PE working on the data at a given time. Due to their tight integration with memory/flash modules, the near-memory and near-storage accelerator have higher thermal/power constraints. Thus, they have less resources (e.g. BRAM, LUT) to implement large PEs. To keep the analysis consistent with this assumption, we consider an embedded FPGA chip from Xilinx Zynq ultrascale+ family for near-memory and near-storage accelerators. Please note that we are not using the whole SoC board and target the reconfigurable fabric only. For on-chip accelerator, we consider resources similar to a high-end Xilinx Virtex Ultrascale+ FPGA board. Based on resource utilization of our kernels, we assume the on-chip FPGA could fit up to 8 PEs of each kernel compared to one PE per near-memory and near-storage accelerator. The PEs could only help if the computational intensity is high enough. Table 5.2 shows the bandwidth parameters.

Table 5.2: System configuration

Parameter	bw_{hostIO}	bw_{nvm}	bw_{ddr}	bw_{cc}
Bandwidth (GB/s)	12.18	16	17.9	100

Throughput for in-storage dataset. Figure 5.3 shows the throughput of each application for a single input file based on the configuration in Table 5.2. Table 5.3 also shows which stage of the pipeline is the limiting factor during each kernel's execution.

Based on Table 5.3, there are 3 applications that are bound by load stage. This means that they can saturate the SSD bandwidth and benefit from near-storage acceleration: KMP, KNN and Select. They have a low computational intensity, high peak-bandwidth and high reduction ratio. There are two applications that are bound by store stage in near-storage acceleration: Partition and Combine. These applications have a low computational intensity, but since there is no reduction from input to output, they are bound by host/IO interconnect. There is no distinction between acceleration options for these applications. All other kernels are bound by compute stage. Among these applications, kernels with very high computational intensity (e.g. ENC, DEC, Hist, VJ, GeMM) favors the on-chip acceleration, while others could benefit from near-memory acceleration (e.g. PHJ, Aggr, Bsch). The factor that makes a distinction between on-chip and near-memory acceleration is data reuse ratio and computational intensity.

Throughput for in-memory dataset. Figure 5.4 shows the throughput of each application when data is resident in main memory. Based on Table 5.3, there are 8 applications in near-memory compute level, that are bound by load stage when data is resident in storage. Figure 5.4 shows that when data is resident in main memory, these applications benefit greatly from the distributed accelerator in near-memory compute level. The other 10 applications in

Table 5.3: The performance bottleneck of three acceleration schemes

FPGA kernel	Near-storage	Near-memory	On-chip	File size (GiB)
KMP	load	load	load	10
KNN	load	load	load	10
AES-ENC	comp	comp	comp	4
AES-DEC	comp	comp	comp	4
Select	load	load	load	4
PHJ	comp	comp	comp	5
Aggr	comp	load	comp	4
Part	store	load	load	4
Combine	store	load	load	4
Snappy	comp	comp	comp	4
Sort	comp	comp	comp	8
Hist	comp	comp	comp	1
VJ	comp	comp	comp	1
FFT	comp	load	load	1
Conv	comp	comp	comp	1
GeMM	comp	comp	load	1
Black Scholes	comp	load	comp	1
Swaption	comp	comp	comp	1

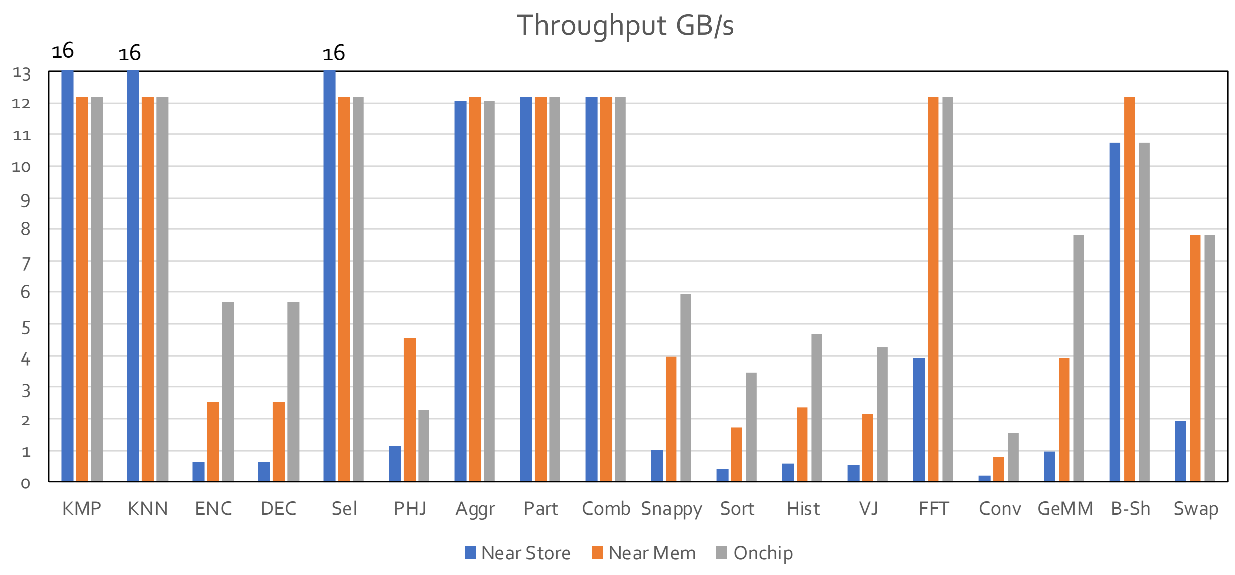


Figure 5.3: Throughput of applications for in-storage dataset

our benchmark list are compute-bound applications. With exception of PHJ and Swaption, the compute-bound applications benefit from on-chip acceleration more than the near-memory acceleration. Partition Hash Join application, requires multiple pass over the original dataset and even though it is compute-intensive, it benefits from distributed acceleration and in-situ access over the dataset. Kernels FFT and Swaption has similar throughput for on-chip and near-memory acceleration. This means that they have a medium computational intensity that does not benefit from more PEs in on-chip accelerator.

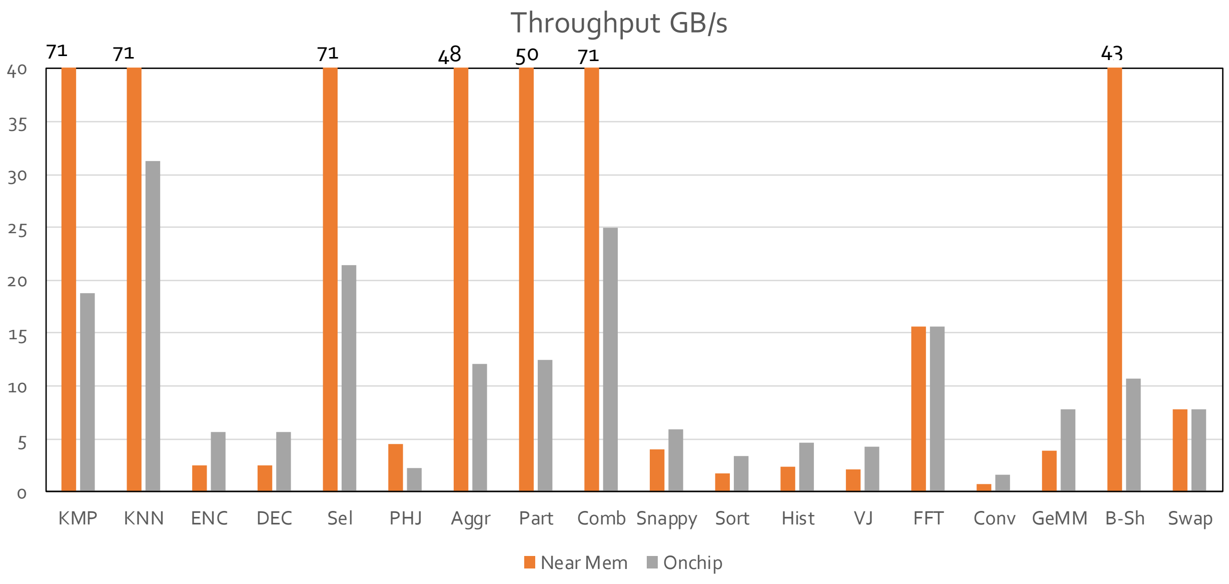


Figure 5.4: Throughput of applications for in-memory dataset

5.4.1 User Guideline

The important takeaway points from the model is discussed here.

For in-storage datasets, 1) Applications with high reduction ratio and high peak bandwidth, generally map well to a near-storage accelerator. 2) For applications that are interconnect-bound (e.g. partition, combine), the throughput is ultimately bound by the near-storage accelerator's performance. Therefore, there will be no benefit in processing the data out of storage. Meanwhile, by keeping the computation near storage, we could consolidate multiple kernels (e.g. Combine+Aggr, PHJ+select) which collectively result in a reduction in output.

3) For applications that are compute-bound, on-chip and near-memory accelerators are more effective. Partition Hash Join has irregular access patterns to the hash table. Therefore, it benefits more from near-memory acceleration.

For in-memory datasets, 1) Data-intensive applications map well to the distributed near-memory acceleration platform. 2) Compute-intensive applications map well to the on-chip accelerator, unless the computational intensity is not high enough to benefit from all the PEs on chip. 3) Applications with multiple pass over the data benefit from near-memory acceleration, whereas applications with intermediate data benefit from near-cache acceleration.

5.5 Conclusion

In this work, we present an in-depth analysis of applications' qualitative and quantitative attributes to derive an accurate performance model. The model could be used inside the Global Accelerator Manager (GAM) to make dynamic decisions on where to map new assigned applications. Based on our analysis, applications from the search domain and some of the database operators (simple kernels with no data reuse and a high reduction ratio), benefit the most from mapping to near-storage accelerators. For compute-bound applications such as in security, finance and vision domains, it helps to stream the data to a more powerful on-chip accelerator. For interconnect-bound applications, the throughput is ultimately bound by the near-storage accelerator's performance, but if the data is accessed through continuous queries, there is huge benefit in moving it to main memory for near-memory acceleration.

CHAPTER 6

Conclusion

Data movement has become a fundamental issue in today's compute-centric server architectures. One promising solution has been to integrate NDP accelerators near data medium to distribute computation into memory or disk modules, as a way of both exploiting the available internal bandwidth and avoiding the movement of data across chip boundaries. Therefore, FPGA-based near data processing has been a growing topic in both industry and academic research. We believe that in near future, data centers will feature racks of heterogeneous servers equipped with one or more types of NDP accelerators, to effectively enhance their system efficiency for the diverse set of common server workloads.

A server workload is comprised of both compute-intensive and data-intensive applications. In fact, it is common to have variations in compute and memory requirements, even within different execution phases of a single application. Multiple NDP accelerators within a system could be used to accelerate independent applications or coordinate to work on different kernels within the same application pipeline. Unfortunately, a large set of existing research has solely focused on only one level of the memory hierarchy (cache, main memory or storage), while disregarding the potential benefits of having multiple levels of near data processing for communication-bound application pipelines.

This dissertation is an effort towards addressing some of the challenges limiting the wide adoption of NDP accelerators and exploring their collective benefits for a common class of server applications. we present ReACH, a **Re**configurable **A**ccelerator **C**ompute **H**ierarchy that combines on-chip, near-memory, and near-storage accelerators, spanning all levels of the conventional memory hierarchy. Each memory level has a reconfigurable accelerator chip attached to it, which provides distinct compute and memory capabilities and offers a broad spectrum of acceleration options. Our simulation platform features fully adjustable memory/accelerator parameters, allowing effective design space exploration. We believe that the key to wide adoption of NDP accelerators lies in a proper runtime scheduilng support and well-defined programming interface. Rather than relying on CPU cores to interact with accelerators directly, we propose a hardware-based global accelerator manager (GAM) to coordinate between the compute levels, reducing inter-level data access interference and asynchronous task flow control. At the software level, we propose a holistic software stack to minimize the programming efforts of using the ReACH system: a uniform programming interface is designed to decouple the ReACH configuration from the user application source code and allow runtime adjustments without modifying the deployed application. We demonstrate that proper application mapping eliminates unnecessary data movement and achieves significant throughput gain and energy reduction. There still remains a number of challenges and research directions for widespread adoption of NDP accelerators:

First, our platform provides pre-optimized accelerator templates and a transparent programming interface that allows the programmer to easily conjoin various NDP accelerators through communication buffers. However, it still heavily relies on the application programmer to make the decision on which part of the application needs to be offloaded to which NDP accelerator. It would be beneficial to investigate automated profiling and compiler-based mechanisms that could decide what portion of software code needs to be replaced by NDP accelerator calls. In chapter 5, we proposed a performance model that could guide the programmer or the global accelerator manager for runtime adjustment of accelerator resources. The model requires application-specific parameters that could only be collected through application profiling. Integrating the performance model with the GAM and automating the parameter extraction would enable a better resource management.

Second, determining an effective data mapping mechanism for near-memory accelerators is another important research direction. Offloading task to near-storage accelerators are more straight-forward than near-memory accelerators, as the task gets scheduled for near-storage accelerator attached to the physical storage unit of the target data. But for near-memory accelerators, the initial data mapping plays an important role in efficiency of the computation.

Our GAM support a simple batch-based data mapping schemes in main memory level by utilizing the internal registers of the memory controller to divide the loaded data in contiguous regions and in batch granularity. For more sophisticated data mapping, we still require the programming effort and initial data manipulation. It is important to develop better static and adaptive data mapping mechanisms and utilize them in the memory controllers level.

Third, we focused on near-memory accelerators for conventional DRAM modules, but as the DRAM scaling becomes more difficult, there is a need for investigating near-memory accelerators for emerging non-volatile memory technologies that will replace DRAM in near future. This includes phase-change memory (PCM) [55], memristors [56], and 3D-stacked memory products such as high-bandwidth memory (HBM) [16] and WideIO [17]. The main memory level is shifting toward a more hybrid architecture that would contain combination of these modules. This opens the door for new NDP acceleration techniques.

In summary, the approaches described in this dissertation demonstrate the need for and benefits of accelerator-rich architectures with multiple levels of near data processing. The proposed hardware and software solutions are some initial steps towards the wide adoption of NDP accelerators and more efficient server architectures.

REFERENCES

[1] "Cisco global index, 2016-2021 white paper," https://newsroom.cisco.com/press-release-content?type=webcontentarticleId=1908858, 2018.

[2] W. Dally, "Challenges for future computing systems. keynote," 2015.

[3] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *ICCAD*, 2015.

[4] "Scaling up energy efficiency across the data center industry." https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf, 2014.

[5] "Amazon ec2 f1 instance," https://aws.amazon.com/ec2/instance-types/f1/, 2018.

[6] D. Chiou, "The microsoft catapult project," in *IISWC*, 2017.

[7] "Intel to start shipping xeons with fpgas in early 2016." www.eweek.com/servers/intel-to-start-shipping-xeons-with-fpgas-in-early-2016.html, 2016.

[8] "with agilex intel gets a coherent fpga strategy," https://www.nextplatform.com/2019/04/02/with-agilex-intel-gets-a-coherent-fpga-strategy, 2019.

[9] "Intel xeon scalable processor 6138p," https://www.eejournal.com/article/intel-delivers-xeon-scalable-processor-6138p-with-arria-10-gx-1150-fpga/, 2018.

[10] "Versal: The first adaptive compute acceleration platform (acap)." https://www.xilinx.com/support/documentation/white-papers/wp505-versal-acap.pdf, 2019.

[11] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, Dec 2003.

[12] B. Loop and Z. Yang, "Pcie nvme* ssd in smaller form factors," in *Flash Memory Summit*, 2016.

[13] "Samsung nvme ssd 960 pro," https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ssd-960-pro-m-2-512gb-mz-v6p512bw/, 2019.

[14] A. M. Caulfield, T. I. Mollov, L. A. Eisner, A. De, J. Coburn, and S. Swanson, "Providing safe, user space access to fast, solid state disks," *SIGPLAN Not.*, vol. 47, no. 4, p. 387–400, Mar. 2012. [Online]. Available: https://doi.org/10.1145/2248487.2151017

[15] "An introduction to ccix," https://www.synopsys.com/designware-ip/technical-bulletin/introduction-ccix-2017q3.html, 2017.

[16] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014.

[17] J. Kim, C. S. Oh, H. Lee, D. Lee, H. R. Hwang, S. Hwang, B. Na, J. Moon, J. Kim, H. Park, J. Ryu, K. Park, S. K. Kang, S. Kim, H. Kim, J. Bang, H. Cho, M. Jang, C. Han, J. LeeLee, J. S. Choi, and Y. Jun, "A 1.2 v 12.8 gb/s 2 gb mobile wide-i/o dram with 4×128 i/os using tsv based stacking," *IEEE Journal of Solid-State Circuits*, Jan 2012.

[18] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking ciphers with copacobana–a cost-optimized parallel code breaker," in *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 2006.

[19] J. Cong, Z. Fang, M. Gill, F. Javadi, and G. Reinman, "Aim: accelerating computational genomics through scalable and noninvasive accelerator-interposed memory," in *MEMSYS*, 2017.

[20] B. Sukhwani, T. Roewer, C. L. Haymes, K.-H. Kim, A. J. McPadden, D. M. Dreps, D. Sanner, J. Van Lunteren, and S. Asaad, "Contutto: a novel fpga-based prototyping platform enabling innovation in the memory subsystem of a server class processor," in *MICRO-50*, 2017.

[21] M. D. Marino and K. Li, "Ramon: Region-aware memory controller," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 4, pp. 697–710, 2018.

[22] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: opportunities and challenges," in *SIGMOD*, 2013.

[23] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: A case for intelligent ssds," in *ICS*, 2013.

[24] G. Koo, K. K. Matam, H. Narra, J. Li, H.-W. Tseng, S. Swanson, M. Annavaram *et al.*, "Summarizer: trading communication with computing near storage," in *MICRO-50*, 2017.

[25] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho *et al.*, "Biscuit: A framework for near-data processing of big data workloads," in *ISCA-43*, 2016.

[26] A. De, M. Gokhale, R. Gupta, and S. Swanson, "Minerva: Accelerating data analysis in next-generation ssds," *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 9–16, 2013.

[27] M. Singh and B. Leonhardi, "Introduction to the ibm netezza warehouse appliance," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research.* IBM Corp., 2011, pp. 385–386.

[28] “Mobiveil announces fpga-based ssd platform.” http://www.marketwired.com/press-release/- 2228973.htm.

[29] S. Seshadri, M. Gahagan, M. S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, “Willow: A user-programmable ssd.” in *OSDI*, 2014.

[30] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, “Bluedbm: An appliance for big data analytics,” in *ISCA-42*, 2015.

[31] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[32] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, “Towards characterizing cloud backend workloads: Insights from google compute clusters,” *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, p. 34–41, Mar. 2010. [Online]. Available: https://doi.org/10.1145/1773394.1773400

[33] X. Inc., “Xilinx virtex ultrascale+ fpga vcu1525,” https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html, 2017.

[34] A. Babenko and V. Lempitsky, “Efficient indexing of billion-scale datasets of deep descriptors,” in *CVPR*, 2016.

[35] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, “Searching in one billion vectors: re-rank with source coding,” in *ICASSP*, 2011.

[36] N. Elgendy and A. Elragal, “Big data analytics: A literature review paper,” in *Advances in Data Mining. Applications and Theoretical Aspects.* Springer International Publishing, 2014, pp. 214–227.

[37] Z. Ruan, T. He, and J. Cong, “INSIDER: designing in-storage computing system for emerging high-performance drive,” in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019.* USENIX Association, 2019, pp. 379–394.

[38] L. Woods, Z. István, and G. Alonso, “Ibex: an intelligent storage engine with support for advanced sql offloading,” *Proceedings of the VLDB Endowment*, vol. 7, no. 11, pp. 963–974, 2014.

[39] S. F. Yitbarek, T. Yang, R. Das, and T. Austin, “Exploring specialized near-memory processing for data intensive operations,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016.* IEEE, 2016, pp. 1449–1452.

[40] J. Cong, Z. Fang, Y. Hao, and G. Reinman, “Supporting address translation for accelerator-centric architectures,” in *HPCA-23*, 2017.

[41] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[42] X. Inc., "Ds890, ultrascale architecture and product data sheet, v 3.10." 2019.

[43] "Xilinx sdx." www.xilinx.com/products/design-tools/software-zone/sdaccel.html, 2019.

[44] X. Inc., "Xilinx power estimator user guide ug440," 2017.

[45] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *MICRO*, 2007.

[46] "Micron ddr4 sdram system-power calculator," https://www.micron.com/support/tools-and-utilities/power-calc, 2018.

[47] "Seagate nytro 5910 nvme ssd," https://www.seagate.com/enterprise-storage/nytro-drives/, 2017.

[48] "64-lane 16-port pci express system interconnect switch," https://www.idt.com/document/dst/89pes64h16-data-sheet, 2017.

[49] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, and et al., "What your dram power models are not telling you: Lessons from a detailed experimental study," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, 2018.

[50] "Xilinx vitis accelerated libraries," https://www.xilinx.com/products/design-tools/vitis/vitis-libraries.html, 2019.

[51] Y. Choi, P. Zhang, P. Li, and J. Cong, "Hlscope+,: Fast and accurate performance estimation for fpga hls," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 691–698.

[52] Z. Ruan, T. He, and J. Cong, "Analyzing and modeling in-storage computing workloads on eisc — an fpga-based system-level emulation platform," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[53] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

[54] M. Makni, M. Baklouti, S. Niar, and M. Abid, "Performance exploration of amba axi4 bus protocols for wireless sensor networks," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, 2017, pp. 1163–1169.

[55] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, pp. 143–143, 2010.

[56] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008. [Online]. Available: https://doi.org/10.1038/nature06932