Reconfigurable Accelerator Compute Hierarchy: A Case Study Using Content-Based Image Retrieval

Nazanin Farahpour*, Yuchen Hao*, Zhenman Fang†, Glenn Reinman*
*University of California, Los Angeles †Simon Fraser University, Canada
{nazanin, haoyc, reinman}@cs.ucla.edu zhenman@sfu.ca

***Abstract*—The recent adoption of reconfigurable hardware accelerators in data centers has significantly improved their computational power and energy efficiency for compute-intensive applications. However, for common communication-bound analytics workloads, these benefits 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. While prior studies focus on the benefits of Near Data Processing (NDP) solely on one level of the memory hierarchy (one of cache, main memory or storage), we focus on the collaboration of NDP accelerators at all levels and their collective benefits in accelerating an application pipeline.**

In this paper, we present a Reconfigurable Accelerator Compute Hierarchy (ReACH) that combines on-chip, near-memory, and near-storage accelerators. 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. 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 achieving asynchronous task flow control. To minimize the programming efforts of using the compute hierarchy, 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 (CBIR) system on ReACH. Simulation results demonstrate that a proper application mapping eliminates unnecessary data movement, and ReACH achieves 4.5x throughput gain while reducing energy consumption by 52% compared to conventional on-chip acceleration.

I. Introduction

Since power and energy efficiency become the primary motivators in today's data centers, the leading industry companies are shifting toward incorporating FPGAs (Field-Programmable Gate Array) into their servers. Microsoft and Amazon have pioneered the large-scale deployment of FPGAs in cloud services [1], [2] and nowadays FPGA is becoming a 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 and 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 [3], [4]. 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 [5].

As we present later in section VI, 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 8 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. Modern commodity servers have relatively sparse off-chip memory bandwidth and 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 exists in the IO interface. The host-side PCIe Gen 3 x16 bandwidth to the storage hierarchy is theoretically 16GB/s, which downgrades to about 12GB/s due to the inefficiency in the IO software stack [6]. 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 disaggregated storage servers that communicates volumes of data through the network between the 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 to deploy accelerators near the data medium to distribute the computation into memory or disk modules, so as to both exploit the available internal bandwidth and avoid the data movement across chip boundaries [7], [8], [9], [10], [11], [12]. 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 **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 acceleration level provides distinct compute and memory capabilities, offering a broad spectrum of acceleration options. As characterized in the prior work [13], on-chip cache-coherent accelerators are more suitable for compute-intensive workloads or applications that require frequent interactions with the host CPU. Near-memory accelerators are more suitable for parallelizable tasks with a large memory footprint and a high bandwidth requirement. Near-storage accelerators are more 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 levels. 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 CBIR system on ReACH as a case study. Based on our simulation results, a proper application mapping significantly reduces the data movement and achieves 4.5x throughput and 52% energy improvements. The main contributions of this paper are:

1. Propose a compute hierarchy that combines on-chip, near-memory, and near-storage accelerators (Section II).
2. Design a hardware-based global accelerator manager for coordinating compute levels, reducing inter-task memory access interference, and providing asynchronous task flow control (Section II-D).
3. 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 III).
4. Demonstrate a real-world application example that could benefit from the hierarchy and quantify its performance and energy gains based on cycle-accurate simulation (Section VI).

II. Overview of ReACH Architecture

A high-level overview of our compute hierarchy is depicted in Figure 1. ReACH includes multiple cores, an on-chip global accelerator manager (GAM) and reconfigurable accelerators attached to each level of the memory hierarchy. While our compute engines are based on FPGA accelerators, the compute hierarchy is not dependent on a specific type of accelerator logic: they could be, for example, special processor cores, ASIC accelerators, or CGRAs (coarse-grained reconfigurable architectures). This work focuses on FPGAs since they provide more flexibility than ASICs and CGRAs.

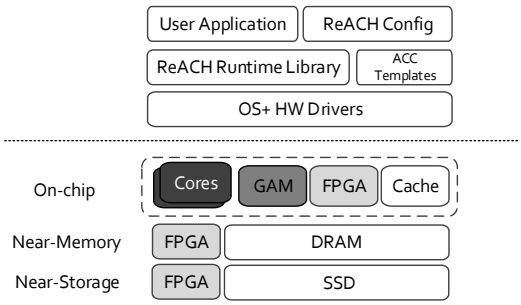

Fig. 1. Overview of ReACH hardware and software stack.

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 the 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 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 bandwidth from main memory to accelerators. The near-storage FPGA module has similar power/thermal constraints, 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 parameters' reuse ratio.

With careful hardware/software co-design in the GAM, ReACH can automatically handle the execution flow in the compute hierarchy and asynchronously pipeline the data transfer and execution between compute levels, while allowing users to write host applications with traditional synchronous coding style. The detailed description of each compute level and GAM is presented in the following subsections. The software infrastructure will be presented in Section III.

A. On-chip Accelerator

Figure 2 illustrates our on-chip 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 [14].

On-chip FPGA accelerators often have higher clock frequency 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 [15]. Therefore, it can achieve higher performance by designing larger 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.

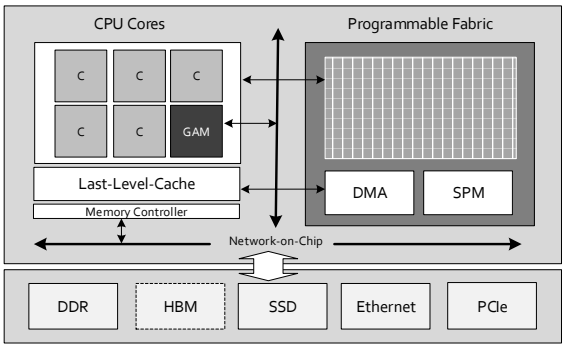

Fig. 2. Overview of on-chip accelerator and GAM.

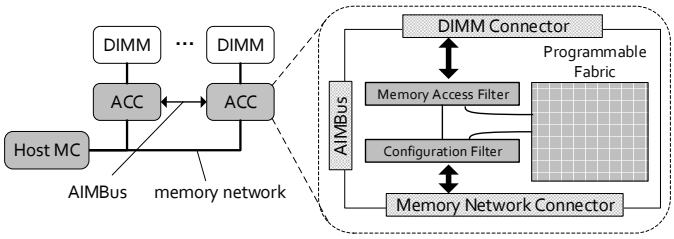

Fig. 3. The accelerator-interposed memory (AIM) architecture with the AIMbus which enables inter-DIMM communication.

B. Near-Memory Accelerator

The performance of on-chip accelerators are bounded by the main memory bandwidth when data have 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. Our design is based on accelerator-interposed memory (AIM) [10]. As shown in Figure 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 1) an embedded FPGA accelerator that can be customized and controlled by the GAM, 2) an AIMbus interface that allows inter-DIMM communication, 3) a configuration filter for accelerator commands coming from the memory channel, and 4) a memory access filter to forward memory responses to the local accelerator, a 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 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 channels are typically designed to offer high capacity and 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 near-memory 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.

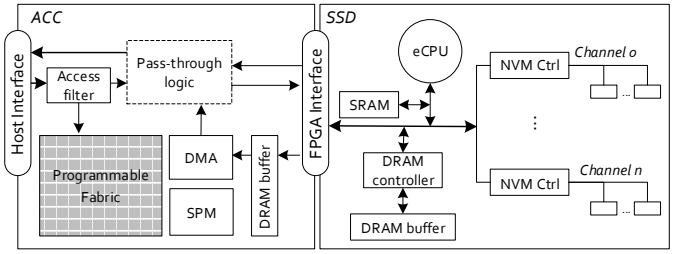

Fig. 4. Internal architecture of the near-storage accelerator.

C. Near-Storage Accelerator

The internal bandwidth and latency of storage (disks) have been improved more than two orders of magnitudes in recent years and emerging non-volatile memory technologies have the potential to achieve near DDR bandwidth and latency. However, the host/disk IO interconnect throughput has not been improved at the same rate and the system bottleneck is moved from disk to IO interconnect. This gap becomes even larger as we scale up the number of SSD units in a system, each with more flash channels.

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 PCIe link. Figure 4 illustrate the internal architecture of our near-storage accelerator. In addition to the user accelerator function unit—the programmable logic, DMA and SPM (scratchpad memory) 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 the attached storage unit. Near-storage accelerators work the best for applications with reduction operations, where the data size is ideally reduced by orders of magnitude before data are transferred to the upper levels of the memory hierarchy.

D. Global Accelerator Manager

To efficiently coordinate hardware accelerators in the compute hierarchy, free CPU cores from managing resources, and avoid costly operating system context switches, we propose

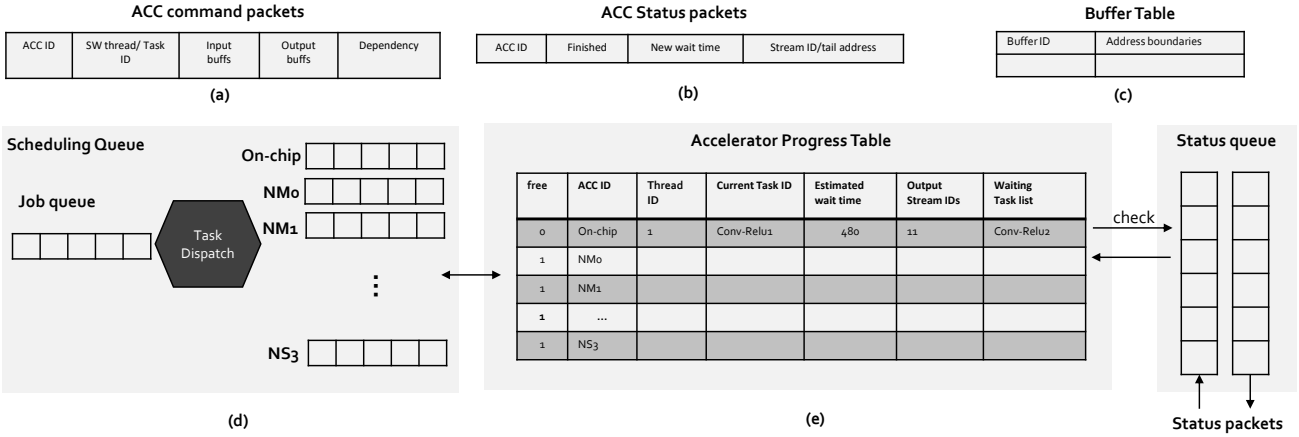

Fig. 5. Global Accelerator Manager (GAM) micro-architecture.

to use an on-chip hardware-based global 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 on or waiting for 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 5 shows the micro-architecture of GAM that enables these capabilities.

As shown in Figure 5, 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 (accelerator) command packets through the GAM driver (5a). GAM breaks each job into multiple tasks (called task groups) that may or may not be assigned to the same compute level. For instance, for a CNN (convolutional neural network) 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 (convolution layer and rectified linear unit), Pool (max pooling layer) or FCN (fully connected layer) 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 (5d) and gets assigned to its target platform's dedicated queue. And its input/output buffers are allocated and the addresses are stored in the buffer table (5c).

With multiple levels of accelerators, the GAM keeps track of the running accelerators using a progress table (5e). When a target hardware is set free from a previous task, GAM invokes the next task from the queue by sending the command request to the target hardware. GAM acts as the master to all accelerators. Since memory/storage modules cannot send acknowledgement to GAM upon finishing a task, it is GAM's responsibility to send status request packets (5b) 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 to be forwarded to input buffers of all dependent tasks through DMA requests. If the task is not finished, a 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 when there is no data dependency. This reduces idle time and improves the pipeline efficiency.

III. Programming Support for ReACH

A. Software Infrastructure

The top part of Figure 1 presents the software stack of ReACH. To minimize the programming effort of using the compute hierarchy, a library-based accelerator programming model inspired by [16] is 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 general accelerator APIs that show a uniform view of all FPGA resources regardless of their compute-level.

Listing 1 shows a snippet of ReACH host APIs written in C++. An application developer could use these APIs 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.

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

Listing 1. A Snippet of ReACH Host APIs (ReACH.h)

```
1 #include <ReACH.h>
2 struct ImgData {...};
3 struct TopK {...};
4 ReACH::Buffer<float> vgg_param =
      CreateFixedBuffer("./vgg16_param", OnChip, size);
5 ReACH::Buffer<float> db0 =
      CreateFixedBuffer("./feature_db0",NearStor, DB0_size);
6 ReACH::Buffer<float> db1 =
      CreateFixedBuffer("./feature_db1",NearStor, DB1_size);
7
8 ReACH::Stream<ImgData> Input =
9    CreateStream(CPU, Onchip, Pair, Img_size*Batch, depth);
10 ReACH::Stream<float> features =
11    CreateStream(OnChip, NearStor, Broadcast,
          feat_size*batch, depth);
12 ReACH::Stream<TopK> Result =
13    CreateStream(NearStor, CPU, Collect,
          batch*K*sizeof(int),depth);
14
15 ReACH::ACC cnn = RegisterAcc("VGG16-VU9P", OnChip);
16 cnn.setArgs(0,Input);
17 cnn.setArgs(1,vgg_param);
18 cnn.setArgs(2,Features);
19 ReACH::ACC knn0 = RegisterAcc("KNN-ZCU9", NearStor);
20 knn0.setArgs(0,Features);
21 knn0,setArgs(1,db0);
22 knn0.setArgs(2,Result);
23 ReACH::ACC knn1 = RegisterAcc("KNN-ZCU9", NearStor);
24 knn1.setArgs(0,Features);
25 knn1.setArgs(1,db1);
26 knn1.setArgs(2,Result);
```

Listing 2. ReACH Configuration (config.h)

Listing 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 registration of physical accelerators (*ReACH::ACC*, lines 15, 19, and 23), allocation of each accelerator's memory region (*ReACH::Buffer*, lines 4-6), communication buffers between compute levels (*ReACH::Stream*, lines 8-13), and binding between the accelerator and its buffers (*setArgs* API, lines 16-18, 20-22, and 24-26). 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 3 shows the host code to describe the flow of the application during run time using accelerator-specific APIs. 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 the application, the accelerators could be invoked using the function *execute* from the API. The core would offload the work to GAM to be scheduled for an available accelerator.

```
1 #include <config.h>
2 while (Input.enqueue(new_query_batch)){
3   cnn.execute(threadId);
4   Features.broadcast();
5   knn0.execute(threadId);
6   knn1.execute(threadId);
7   Result.collect();
8   //process(Result.dequeue());
9 }
```

Listing 3. Host Application Accelerator Calls (host.cpp)

B. GAM Scheduling

Based on the ReACH config file, the runtime library calls the GAM driver to set up 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 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.

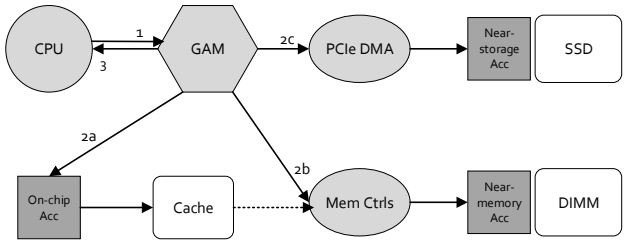

Fig. 6. Scheduling of ReACH through GAM.

Figure 6 describes 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 based on the compute level.
   a) **On-chip Acc:** GAM launches the required kernel into on-chip accelerator, 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. The on-chip accelerator later accesses the data via the cache hierarchy.
   b) **Near-memory Acc:** GAM launches kernel into near-memory modules by writing into their configuration filter. 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 a 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 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.

IV. Case Study: CBIR Overview

Content-based image retrieval (CBIR) identifies relevant images from large-scale image databases based on the representation of visual content, and has attracted increasing attention in recent two decades. We use CBIR as an application example where moving computation closer to data is vitally important. In this section, we present the challenges of implementing the online pipeline of CBIR and how mapping it to compute hierarchy overcomes those challenges.

A. Application Description

The application pipeline of CBIR is shown in Figure 7.

Feature extraction. Visual features within an image are often extracted and packed into a fixed-length vector for image representation. While SIFT and SURF based representations are commonly studied and adopted in early CBIR systems [17], the focus of recent studies have shifted to representations produced by deep neural network (DNN). Several studies [18], [19], [20] have shown that DNN-based representations can

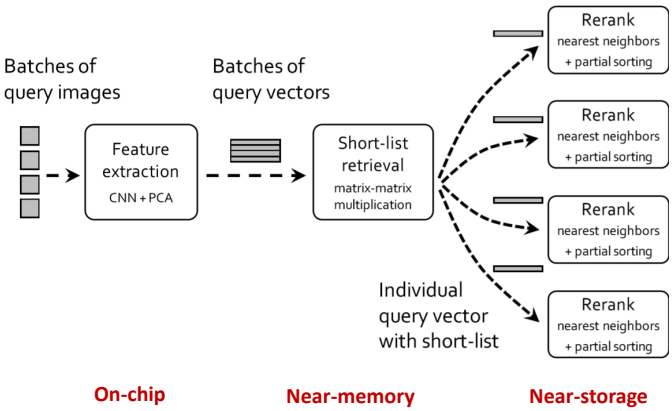

Fig. 7. The CBIR pipeline and its optimized deployment on ReACH.

significantly improve the recall accuracy on image retrieval benchmarks. In our implementation, we extract the feature vector from images using the VGGNet [21] neural network and PCA (principal component analysis) compression with a dimensionality (D) of 96. We assume user query inputs are sufficiently frequent for batched processing in order to improve the throughput of the system.

Short-list retrieval. To avoid exhaustive search on billion-scale databases, state-of-the-art CBIR systems extract a short-list of clusters 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. Specifically, we compute the distances $\|q - C_m\|$ between the query and the centroids. Let us denote by $m_1, m_2, ..., m_M$ indices of M centroids that are produced by the indexing step during the off-line stage.

$$dist[m] = \|q - C_m\|^2 = \|q\|^2 + \|C_m\|^2 - 2\langle q, C_m\rangle, m = 1, ..., M \quad (1)$$

Note that the term $\|C_m\|^2$ in the decomposition can be pre-computed, stored in DRAM and reused for all queries. Also, the term $\|q\|^2$ can be reused within each query. Therefore, the bottleneck of this step is the evaluation of matrix-matrix multiplication $\langle Q_{B\times D}, C_{D\times M}\rangle$, where Q is the batch ($B$) of queries and C is the matrix representation of centroids pre-loaded in columnar fashion. We then perform addition of $\|q\|^2$ and $\|C_m\|^2$, and partial sorting of the *dist* array to produce the short-list for the query q.

Rerank. After short-list retrieval, we traverse the clusters and collect data points from these clusters to form a candidate list. Rerank computes the distances between the query and the data points within this candidate list. For both short-list retrieval and rerank steps, we use the square of Euclidean distance to measure the similarity between data points defined as:

$$\|p - q\|^2 = (p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2 \quad (2)$$

Since short-lists are produced per query, the likelihood of different queries sharing the same clusters is small and thus batched processing is not applicable for most cases. So, the major computation in this step is vector-matrix multiplications. Lastly, a partial sorting on the computed distances is required

TABLE I. AN OVERVIEW OF MEMORY AND COMPUTE REQUIREMENTS OF EACH CBIR PIPELINE STAGE

	Memory Requirement	**Computation Requirement**
Feature extraction	552MB, 11.3MB if compressed [23] Neural network model parameters	*High* Convolutional neural network
Short-list retrieval	∼2.2GB Cluster centroids and cell info	*Medium* Non-square matrix multiplication
Rerank	∼355GB 1 billion feature vectors	*Low* K Nearest Neighbors
Reverse lookup	200TB - 2PB 1 billion images	*Very low* Database access

to produce the K-nearest data points. To return the corresponding K-nearest images to the user, a reverse lookup in the image database is performed using the indices from the rerank results.

Billion-scale CBIR applications pose serious challenges on conventional CPU architectures and memory hierarchies. On one hand, exhaustive search on huge dataset that far exceeds the capacity of DRAM is impractical because (1) comparing against all data points stresses the overall throughput and latency; (2) the time and energy spent on bringing data to CPU outweigh those spent on computation. On the other hand, the performance of clustering methods is degraded by increasing dimensionality in the feature vector – a large portion of data blocks must be read in order to determine nearest neighbors, which is known as the *curse of dimensionality* [22]. As a consequence, a large body of work focuses on compression methods such as binary codes and product quantization which reduces the dimensionality of feature vectors, leading to orders of magnitude reduction in data visited. However, these methods significantly penalize the recall accuracy of the CBIR system. In this paper, we focus on hierarchical near data acceleration to improve the performance of CBIR while preserving the recall accuracy.

B. Application Mapping onto ReACH

Table I summarizes the memory and compute requirements of the major kernels within each CBIR pipeline stage.

Feature extraction. The major algorithm involved in the feature extraction step is convolutional neural network (CNN), which is the most compute-intensive kernel while requiring the least amount of memory among all steps. The compressed CNN model parameters are only 11.3 MB using techniques proposed in [23], which can fit in on-chip SRAM. In light of this, the on-chip accelerator can potentially achieve higher performance because 1) it has higher frequency and more resources to exploit parallelism, 2) on-chip SRAM provides low access latency so that PEs can quickly be unblocked and resume processing. Our implementation is based upon [24], where both computation and memory access are optimized for FPGA implementation to achieve high throughput across all layers.

Short-list retrieval. For short-list retrieval, we use the decomposition in Equation 1 to turn this step to a matrix-matrix multiplication and a broadcast addition. This kernel is ideally implemented using the on-chip accelerator due to its higher frequency and larger resources. However, since the input $\|C_m\|$ and precomputed $\|C_m\|^2$ takes ∼2.2GB of memory to store which exceeds the on-chip SRAM capacity, on-chip accelerators would need frequent access of off-chip DRAM and contend for the shared cache, leading to a degraded performance. To this end, we support short-list retrieval by developing customized accelerators near memory. Although the compute power of near-memory accelerators is lower, the lower DRAM access latency and the ability to scale to multiple instances can offset this issue. Note that near-storage accelerators can potentially provide even better scalability, but the storage access latency is much higher and undesirable for compute-intensive workloads like matrix-matrix multiplication.

Rerank (and reverse lookup). The memory requirement of the rerank and the reverse lookup steps strongly favors near-storage computation – using either on-chip or near-memory acceleration would inevitably require moving large amount of data across the memory hierarchy which is slow and energy-inefficient. Moreover, the performance of matrix-vector multiplication kernel is sensitive to memory bandwidth rather than computation power. But the limited bandwidth of host IO interface cannot fully utilize the aggregated bandwidth of the SSD array. Offloading the task to near-storage accelerators would expose the full bandwidth of SSDs and can potentially achieve linear speedup as we scale-up the number of FPGA-SSD units in the system.

Overall Flow. The overall deployment strategy of the CBIR pipeline on the ReACH system is summarized in Figure 7. 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 broadcasts 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 lists 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 can be retrieved from the original image database and returned to the host; note that we do not include this final step of reverse lookup in our experiments due to its huge storage requirements. As we can see from the figure, the only data movement required is the user query vector and retrieved short-list.

V. EXPERIMENTAL SETUP

Performance evaluation. To evaluate the performance of ReACH, we extend the open source cycle-accurate accelerator-rich architecture simulator PARADE [16] 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 [25] and is coherently attached to CPU using a cache-coherent interconnect.

Our near-memory accelerator is modeled after AIM [10] 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 II summarizes the configuration of the system.

TABLE II. 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	8 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+ [25], 100 GB/s to shared cache
Near-Memory Accelerator	Zynq Ultrascale+ [25], 18 GB/s bandwidth to DDR4
Near-Storage Accelerator	Zynq Ultrascale+ [25] with 1GB DRAM, 12GB/s effective bandwidth to NVMe SSD

TABLE III. FPGA UTILIZATION FOR EACH ACCELERATOR

FPGA	Kernel	Utilization (ff,lut,dsp,bram)	Kernel Freq	Power (W)
Xilinx Virtex Ultrascale+ XCVU9P	CNN	(36%,81%,78%,42%)	273 MHz	25
	GeMM	(24%,27%,56%,77%)	273 MHz	22.13
	KNN	(10%,10%,10%22%)	200 MHz	11.14
Xilinx Zynq Ultrascale+ ZCU9EQ	CNN	(11%,31%,38%,36%)	200 MHz	5.19/6.13
	GeMM	(36%,27%,76%,92%)	150 MHz	5.3/8
	KNN	(23%,20%,30%,22%)	150 MHz	1.8/2.4

Table III shows the list of kernels designed for our experiment: including convoluational neural network (CNN), matrix multiplication (GeMM), and k nearest neighbors (KNN). It 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 [26] 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 [27] 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 IV shows a summary of tools and references we 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.

TABLE IV. ENERGY MODEL TOOLS AND REFERENCES

Component	Reference
FPGA Accelerators	Xilinx SDAccel 2019.1 [26] and XPE power calculator [27]
Cache	CACTI 6.5 [28]
DRAM	Micron DDR4 Power Calculator [29]
Storage	NVMe SSDs [30] with PCIe Gen3x16 interfaces
Interconnect	Host/IO interface switch [31], PCIe links [32] and Memory channels [33]

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.

VI. EVALUATION RESULTS

To illustrate the limitations of the compute-centric acceleration approach, we analyze the energy distribution between components of the system during on-chip acceleration of CBIR pipeline in Section VI-A. We discuss the performance and energy cost of each CBIR kernel at various compute levels of ReACH in Section VI-B. Then we implement an end-to-end CBIR pipeline only using accelerators at one compute level at a time and compare their performance and energy efficiency in Section VI-C. To demonstrate the effectiveness of ReACH and the proposed application mapping, in Section VI-D, 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.

A. 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 [15]. Figure 8 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, around 79% of the total energy cost is due to the data movement across the memory hierarchy. In fact, around 52% of the total cost is for data movements of Rerank step. The 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 while comparing various acceleration options.

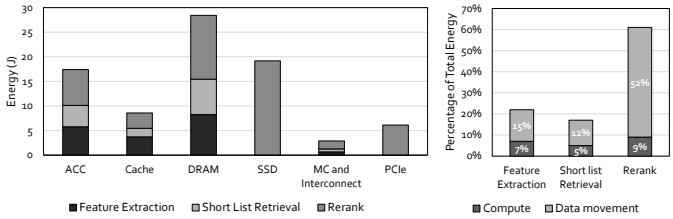

Fig. 8. The energy consumption breakdown for accelerating CBIR pipeline using on-chip accelerator.

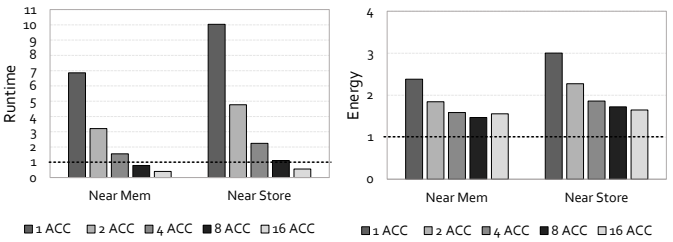

Fig. 9. Runtime and energy consumption comparison of the feature extraction step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

B. Performance and Energy Cost for Each Stage at Different Compute Levels

Feature extraction. The main kernel of feature extraction step is a convolutional neural network, which is pre-optimized for each ReACH level. While on-chip accelerator uses a batched implementation, the near-memory and near-storage accelerators are best optimized when using single image per task and keeping a duplicated copy of the CNN parameters. This way, they refrain from costly layer partitioning and data transfers to other accelerators. Figure 9 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; for near-storage accelerators, the parameters are pre-loaded in private device DRAM and each instance has its 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, tiled and duplicated 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, 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 (8 or 16 instances), the collective performance of near-memory and near-storage 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 10. The on-chip accelerator performance is bounded by the bandwidth of loading data from DRAM as they do not fit in on-chip

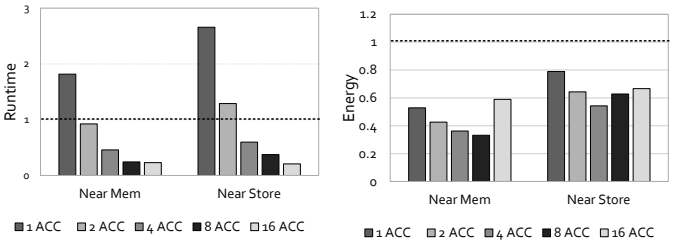

Fig. 10. Runtime and energy consumption comparison of the short-list retrieval step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

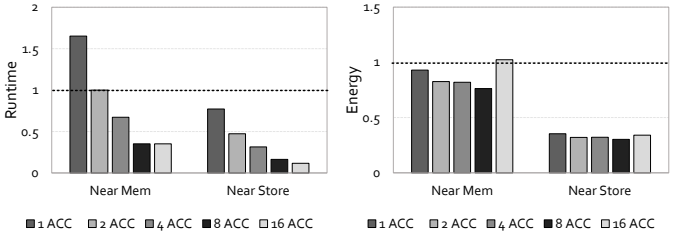

Fig. 11. Runtime and energy consumption comparison of the Rerank step using near memory and near storage accelerators. Numbers are normalized to that of the on-chip accelerator.

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 than the near-memory accelerator, because the latency of device SSD access is more than the latency of DRAM access.

Rerank. Figure 11 shows the rerank stage runtime/energy comparison of all implementations. The rerank stage uses KNN to find the closest images to the 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 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.

C. Overall Performance and Energy using Single Compute Level

As described in Table II, our experimental setup has 8 DIMMs and 4 NVMe SSDs. We use this setup for our energy

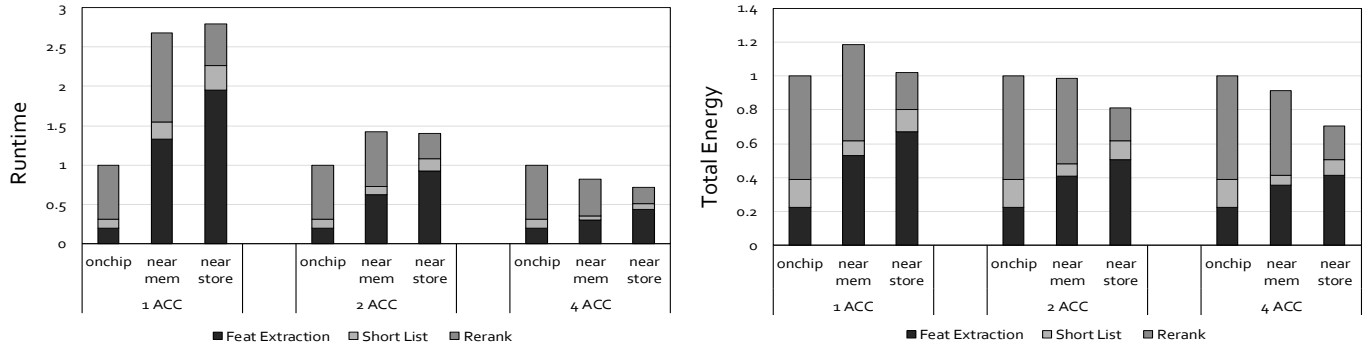

Fig. 12. The total runtime and energy cost of CBIR using a single compute level, normalized to the baseline on-chip accelerator.

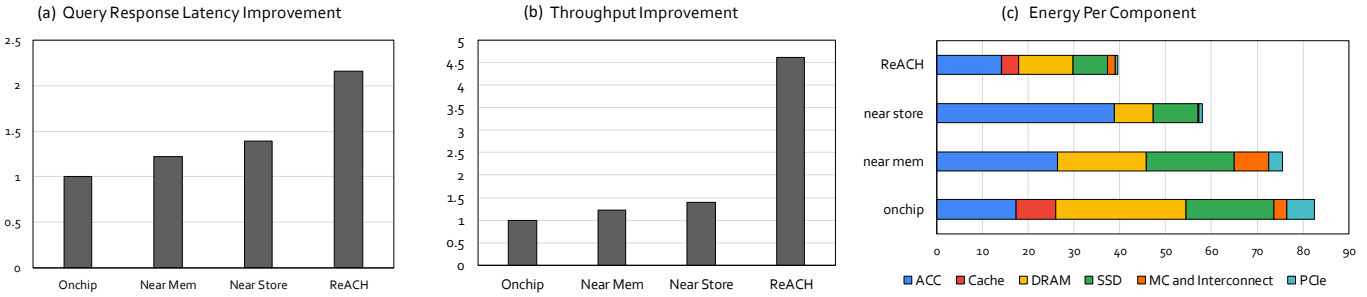

Fig. 13. The performance and energy consumption of CBIR running on ReACH compared to on-chip, near memory and near storage acceleration.

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 12 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.

When comparing on-chip accelerator 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.

D. Performance and Energy Cost after Proper Mapping on ReACH

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 IV-B: feature extraction uses on-chip accelerators, short-list retrieval uses near memory accelerators, and rerank uses near storage accelerators. As we discussed in Section II-D, 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 13.

As shown in Figure 13, compared to on-chip acceleration only, the throughput of ReACH improves 4.5x when we coordinately 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 52% energy reduction compared to the baseline on-chip accelerator (i.e., using less than half of the original energy). 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.

VII. Related Work

A. On-chip Acceleration

There is a large amount of prior work that implements a coprocessor or accelerator specialized for a single application domain through either ASIC or FPGA technology. These accelerators are often cache-attached, with the ability to access the shared coherent cache without the assistance from the host core, eliminating frequent interactions with the host [34], [35], [36], [37]. CHARM [35] explores the possibility of dynamically composing complex accelerators from accelerator

building blocks. CAMEL [36] extends CHARM by integrating FPGA logic on chip for better flexibility and longevity. Efficient support for a unified address space has also been explored by [14].

Since the acquisition of Altera, Intel has introduced various CPU-FPGA acceleration platforms (AgileX [38], XeonSP [39]) that connect the Xeon processor and FPGA fabric in the same package through a cache-coherent interface. Xilinx has also shifted its priority to data center market and introduced Xilinx Versal ACAP [15] as an on-chip accelerator. With the capability of swapping partial bitstreams in sub-millisecond, the Versal ACAP could be utilized by multiple real-time applications simultaneously.

Tools for designing and modeling accelerator-centric SoCs have also been developed. PARADE [16] provides accurate accelerator performance modeling by extending gem5 [40] with DMA, scratchpad memory and HLS-generated accelerator models. gem5-Aladdin [41], also based on gem5, focuses on the evaluation of design tradeoffs such as the choice of using cache or scratchpad for accelerators. Platforms such as Zynq [42] and Intel HARP [43], which provide from embedded to server-like setups, are widely adopted to prototype research ideas on on-chip accelerators.

B. Near-Memory Acceleration

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) [44] from AMD and Hynix, and Samsung's Wide I/O [45] 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 number of memory channels per socket and multiple DIMMs share the same memory channel which limits the overall bandwidth to the CPU. Near memory accelerators help achieve a lower latency and a higher bandwidth to DIMMs sharing the same memory channel. For instance, Copacobana [46] builds FPGA modules directly into DIMMs. AIM [10] places FPGA modules between the DIMM and the memory network, making the design noninvasive to the existing memory controller, memory bus and DIMMs. Contutto [12] prototypes such idea by plugging accelerators in DIMM slots in a POWER8 machine and shows acceleration with end-to-end experiments. Our near-memory acceleration part is most similar to AIM.

C. Near-Storage Acceleration

With the advancement of flash technology, near-SSD computing attracts increasing attention in recent years. Projects such as SmartSSD [47], Active Disk [48], Biscuit [49] and Summariser [50] propose to utilize the embedded cores in a modern SSD to avoid data movement and to free-up the host CPU and main memory. These studies demonstrate the versatility of software, but also show the limitation of compute complexity and parallelism.

Incorporating reconfigurable hardware accelerators to distributed SSDs is also being investigated actively. IBM Netezza [7] offloads operations such as filtering to the FPGA near storage. Willow [51] adds programmability to the SSD interface to allow programmers to easily customize accelerators. QuickSAN [52], BlueDBM [9], Caribou [53] and Blue-cache [54] deploy accelerators at the disaggregated storage servers to avoid data transmission over network. While most prior work focuses on accelerating big data analytics, ExtraV [55] and GraFBoost [56] demonstrate significant speedup on out-of-memory graph processing.

Compared to prior studies, we are the first to explore a compute hierarchy that combines hardware accelerators at all levels, including on-chip, near-memory, and neary-storage accelerators.

VIII. Conclusion

In this work, we are the first to 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 have explored hardware and software support to enable effective and easy use of the ReACH system. At the hardware level, we propose a global accelerator manager to coordinate between each compute level and manage resources. 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. To demonstrate the effectiveness of ReACH and validate the proof of concept, we conduct a case study to deploy a billion-scale content-based image retrieval system on ReACH to leverage accelerators at all levels in a coordinated way. Experimental results demonstrate that, compared to conventional on-chip acceleration, a proper application mapping for CBIR eliminates unnecessary data movement and achieves 4.5x throughput gain while reducing energy consumption by 52%.

Acknowledgements

We acknowledge the support from Center for Domain-Specific Computing; NSERC Discovery Grant RGPIN-2019-04613 and DGECR-2019-00120; Canada Foundation for Innovation John R. Evans Leaders Fund; Simon Fraser University New Faculty Start-up Grant; Samsung, Huawei, and Xilinx.

References

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

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

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

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

[5] 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.

[6] Z. Ruan, T. He, and J. Cong, "INSIDER: designing in-storage computing system for emerging high-performance drive," in *2019 USENIX Annual Technical Conference*, 2019, pp. 379–394.

[7] 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.

[8] 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.

[9] S.-W. Jun *et al.*, "Bluedbm: An appliance for big data analytics," in *ISCA-42*, 2015.

[10] 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.

[11] S. Yitbarek *et al.*, "Exploring specialized near-memory processing for data intensive operations," in *DATE, 2016*. IEEE, 2016, pp. 1449–1452.

[12] 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.

[13] N. Farahpour, Z. Fang, and G. Reinman, "Fpga-based near data processing platform selection using fast performance modeling (wip paper)," in *LCTES*, 2020, p. 151–155.

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

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

[16] 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.

[17] Z. Fang, D. Yang, W. Zhang, H. Chen, and B. Zang, "A comprehensive analysis and parallelization of an image retrieval algorithm," in *IEEE ISPASS*, April 2011, pp. 154–164.

[18] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *CVPR*, 2014.

[19] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *ECCV*, 2014.

[20] A. Babenko and V. Lempitsky, "Aggregating local deep features for image retrieval," in *ICCV*, 2015.

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

[22] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, 1998.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[24] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, 2016.

[25] X. Inc., "Ds890, ultrascale architecture and product data sheet," 2019.

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

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

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

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

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

[31] "64-lane 16-port pci express system interconnect switch," https://www.idt.com/document/dst/89pes64h16-data-sheet, 2017.

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

[33] S. Ghose *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.

[34] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Architecture support for accelerator-rich cmps," in *DAC-49*, 2012.

[35] ——, "Charm: A composable heterogeneous accelerator-rich microprocessor," in *ISLPED*, 2012.

[36] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, H. Huang, and G. Reinman, "Composable accelerator-rich microprocessor enhanced for adaptivity and longevity," in *ISLPED*, 2013.

[37] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "Accelerator-rich architectures: Opportunities and progresses," in *DAC-51*, 2014.

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

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

[40] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[41] Y. S. Shao, S. L. X. V. Srinivasan, and G.-Y. W. D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *MICRO-49*, 2016.

[42] S. Neuendorffer and F. Martinez-Vallina, "Building zynq accelerators with vivado high level synthesis." in *FPGA*, 2013.

[43] *Accelerator abstraction layer software programmer's guide*, Intel Corporation.

[44] D. U. Lee *et al.*, "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 *ISSCC*, Feb 2014.

[45] J. Kim *et al.*, "A 1.2 v 12.8 gb/s 2 gb mobile wide-i/o dram with 4x128 i/os using tsv based stacking," *IEEE Journal of Solid-State Circuits*, Jan 2012.

[46] 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.

[47] J. Do *et al.*, "Query processing on smart ssds: opportunities and challenges," in *SIGMOD*, 2013.

[48] 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.

[49] B. Gu *et al.*, "Biscuit: A framework for near-data processing of big data workloads," in *ISCA-43*, 2016.

[50] G. Koo *et al.*, "Summarizer: trading communication with computing near storage," in *MICRO-50*, 2017.

[51] S. Seshadri *et al.*, "Willow: A user-programmable ssd." in *OSDI*, 2014.

[52] A. M. Caulfield and S. Swanson, "Quicksan: A storage area network for fast, distributed, solid state disks," in *ISCA-40*, 2013.

[53] Z. István *et al.*, "Caribou: intelligent distributed storage," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1202–1213, 2017.

[54] S. Xu, S. Lee, S.-W. Jun, M. Liu, J. Hicks *et al.*, "Bluecache: A scalable distributed flash-based key-value store," *Proceedings of the VLDB Endowment*, vol. 10, no. 4, pp. 301–312, 2016.

[55] J. Lee, H. Kim, S. Yoo, K. Choi, H. P. Hofstee, G.-J. Nam, M. R. Nutter, and D. Jamsek, "Extrav: boosting graph processing near storage with a coherent accelerator," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1706–1717, 2017.

[56] S.-w. Jun, A. Wright, S. Zhang, S. Xu, and Arvind, "Grafboost: Using accelerated flash storage for external graph analytics," in *ISCA-45*, 2018.