2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

Scaling Datacenter Accelerators With Compute-Reuse Architectures

Adi Fuchs, David Wentzlaff
Department of Electrical Engineering
Princeton University
{adif,wentzlaf}@princeton.edu

***Abstract*—Hardware specialization is commonly used in datacenters to ameliorate the nearing end of CMOS technology scaling. While offering superior performance and energy-efficiency returns compared to general-purpose processors, specialized accelerators are bound to the same device technology constraints, and are thus prone to similar limitations in the future. Once technology scaling plateaus, accelerator and application tuning will reach a point of near-optimum, with no clear direction for further improvements.**

Emerging non-volatile memory (NVM) technologies follow different scaling trends due to different physical properties and manufacturing techniques. NVMs have inspired recent efforts of innovation in computer systems, as they possess appealing qualities such as high capacity and low energy.

We present the COmpute-REuse Accelerators (COREx) architecture that shifts computations from the scalability-hindered transistor-based logic towards the continuing-to-scale storage domain. COREx leverages datacenter redundancy by integrating a storage layer together with the accelerator processing layer. The added layer stores the outcomes of previous accelerated computations. The previously computed results are reused in the case of recurring computations, thus eliminating the need to re-compute them.

We designed COREx as a combination of an accelerator and specialized storage layer using emerging memory technologies, and evaluated it on a set of datacenter workloads. Our results show that, when integrated with a well-tuned accelerator, COREx achieves an average speedup of 6.4× and average savings of 50% in energy and 68% in energy-delay product. We expect further increase in gains in the future, as memory technologies continue to improve steadily.

***Keywords*-accelerators; memoization; emerging memories;**

I. Introduction

Specialized hardware such as GPUs and domain-specific accelerators have become increasingly popular over the past decade. Hardware specialization represents the shift towards alternative computing models, which is motivated by the end of Moore's Law for CMOS technology scaling, currently projected by the ITRS to take place in 2021 [1]. Hardware specialization couples domain-specific accelerators and applications to deliver higher performance within a given power envelope or for the same energy rates. This positions accelerators as an appealing building block for future datacenters and has sparked the emergence of datacenter accelerators in both industry [2]–[4] and academia [5]–[7] which was made possible by changes to the software stack [3,8].

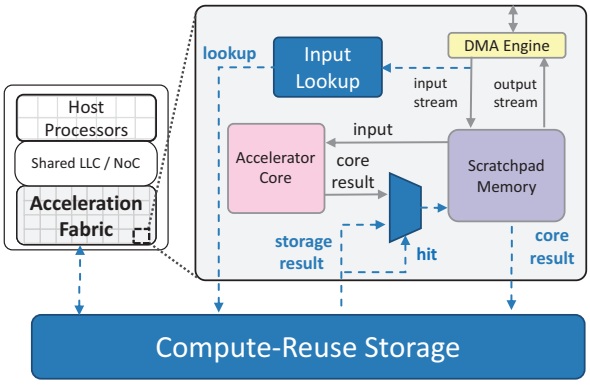


Figure 1: Accelerator with Compute-Reuse Storage.

Unfortunately, the implications of the end of Moore's Law are not solely restricted to general-purpose processors, as accelerators are also implemented using transistors. Since hardware specialization is limited by the number of ways that exist to map computation problems to a fixed hardware budget, accelerator gains cannot improve beyond a certain point. When transistors have stopped scaling, and all available transistors are specifically tailored to a given computation, additional optimizations will not be possible. The pressing question at this point should be: *can systems keep improving their computing capabilities, even when the number of transistors has stopped increasing?*

In contrast to the nearing end of CMOS scaling, recent advancements in emerging memory technologies have shown promise for future scaling. Compared to traditional DRAMs, Phase-Change Memories (PCMs) have superior storage due to the ability of storing multiple bits in a single memory cell [9]–[12], Resistive-RAM (ReRAM) crosspoint structures have superior density [13,14] since cells scale to small feature sizes [15,16], Spin-Transfer Torque RAM (STT-RAM) has been shown to have superior performance while reducing energy costs [17,18], and the density and energy efficiency of Magnetic Domain-Wall Racetrack memories motivated its use in on-chip memories [19].

An important observation for datacenter workloads is that in many scenarios datacenters process the same computed data. Internet traffic has been empirically shown to follow skewed Zipfian distributions [20,21]; video encoders in

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00038

services like YouTube compress identical blocks stemming from spatial redundancy in the encoded video frames; search engines fetch the same popular pages for different search terms, and access patterns were also shown to have skewed distributions in cloud services such as media [22] and social networks [23].

The re-manifestation of the same computed data introduces redundancies in accelerator execution, which could (and should) be avoided, by leveraging mechanisms for *memoization* of redundant computations [24]. Given the correct storage to compute cost ratio, it is possible to record an accelerated kernel's inputs and results and store them. In the case that a previously encountered input is fed into an accelerator, the pre-computed result is fetched from a storage device instead of being re-computed by the accelerator, therefore saving transistor-based compute and energy.

We present COmpute-REuse Accelerators (COREx) that employ emerging memory technologies as an auxiliary storage layer for the purpose of memoization. The COREx architecture builds a customized memory hierarchy, optimized for the accelerator. By shifting recurring computations from accelerators to the memory domain, **COREx essentially trades technologically non-scaling transistor logic for still-scaling storage.** Figure 1 shows how compute-reuse storage is embedded as an extension layer to the acceleration fabric, consisting of loosely-coupled accelerators (LCAs) [25,26]. LCAs and host processors have isolated memory spaces and explicitly communicate via DMA transactions, making LCA designs a natural target for memoization.

We explore the use of ReRAMs, PCMs, Racetrack memories, and STT-RAMs for memoization tables. We show how COREx can be optimized for different goals: runtime, energy, and energy-delay product (EDP). The contributions of this paper are as follows:

- We present a discussion on workload redundancy in datacenter accelerators, quantitatively focusing on two case studies: video encoding and traffic compression in search engines.
- We develop a framework for accelerator memoization, discuss tradeoffs, and present an end-to-end flow that specializes compute-reuse memory layers for a variety of accelerated computations and optimization goals.
- We design the COREx prototype, and tune it separately for performance, energy, and EDP. We show that compared to well-tuned accelerators COREx achieves an average speedup of 6.4×, consumes 50% less energy, and reduces EDP by an average of 68%.

II. Background and Motivation

A. Specialization and Scalability

The inevitable end of Moore's Law and Dennard scaling renders traditional processor performance and energy scaling models obsolete and motivates a shift away from existing computing paradigms to achieve sustainable technological growth [27]. Specialized hardware became the focus of recent studies that integrate new programming models with domain-specific accelerators to deliver more efficient compute capabilities compared to general-purpose processors under a given hardware budget.

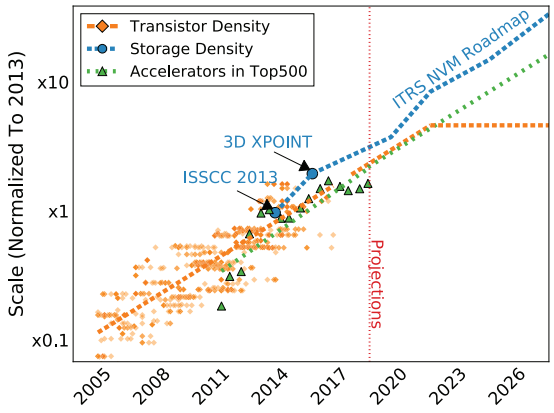


Figure 2: Transistor, Accelerator, and NVM Trends. Marked Points are Existing Systems; Dashed Lines are Projections.

Unfortunately, the end of CMOS scaling that motivated specialization also implies that specialization gains cannot continue growing. Specialization relies on co-optimization of computation problems and hardware to achieve an efficient mapping of the computation problem to a given transistor budget. The gains from mapping algorithms to given hardware are capped since the optimization space is not unlimited, and when hardware stops scaling co-optimization gains cannot exceed the point of near-optimum. This will necessitate a shift away from direct dependence on the non-scaling transistor technology in future accelerators.

In contrast to CMOS process technology, modern memory and storage systems continue to scale. Figure 2 depicts traditional scaling trends, and uses contemporary projections to predict future technology scaling curves. For transistor density scaling, we used existing CPU datasheets from past databases [28,29] and extrapolated the curve following the current predictions of the ITRS roadmap [1] for the end of Moore's law in 2021. For accelerator adoption in HPCs, we use the statistics of six recent years from the Top500 list [30], when it started listing accelerators and co-processors. For storage density scaling we used the ITRS "More Moore" technology roadmap for NVMs, and data from existing academic [31] and commercial devices [32].

B. Memoization Requirements: Three C's

Proper memoization designs must meet several criteria. We term them as the *Three C's for Memoization*:

❶ **Correct Results**: Memoization must preserve computation semantics by always producing the same result as the non-memoized computation, a property known as *"referential transparency"* [33]. This property requires the construction

of a *generalized* input set that consists of every input and element in the system state that can affect the computation outcome. For example, the search results for the term: *"where am I"* fed as input to a search engine depend not only on the input itself, but also on additional state such as the inquirer's location. This suggests that search engine memoization solely based on the search term might not produce correct results, as the correct result is different for distinct locations. As we show in this work, memoization can be naturally done for loosely-coupled accelerators. Since the relation between computation results and input is deterministic and well-defined, correctness is guaranteed by design. This makes memoization of accelerated kernels natural and appealing compared to datacenter memoization which is software-based and requires ad-hoc program adaptations or employing particular software frameworks [34]–[36].

❷ **Constrained Storage**: While storage scaling rates show promise for the coming years, we strive to keep storage tables constrained to ensure a feasible implementation of the storage layer. In order to achieve confined memoization storage, one must understand how computation input values vary. If the input variance is high, the amount of information that needs to be stored might exceed the available storage.

❸ **Costworthy Reuse**: The last requirement is the profitability of memoization. The cost of the memoized computation should be greater than the costs of storing, looking up, and fetching computation results. This requires an understanding of tradeoffs in the compute and compute-reuse design spaces. For instance, large reuse tables yield better coverage, however, as table size increases, access times and energy increase accordingly. Depending on memoization granularity, input behavior, and the mapping of computation to reuse tables, memoization might, or might not, be of gain.

We formulate the prerequisites for memoization profitability using a general cost model. The cost function (e.g., runtime) for each operation x is marked as: C_x. For simplicity, we assume that lookups are always done, and the original computation is done only in case of a miss.

$$C_{compute} \geq C_{lookup} + P_{hit} \cdot C_{hit} + (1 - P_{hit}) \cdot C_{miss}$$

$$C_{miss} = C_{compute} + C_{table\_update} \quad ; \quad C_{hit} = C_{table\_read}$$

Memoization is therefore profitable if the following holds:

$$\boxed{P_{hit} \geq \frac{C_{lookup} + C_{table\_update}}{C_{compute} + C_{table\_update} - C_{table\_read}}; 1 \geq P_{hit} \geq 0} \quad (1)$$

C. Reuse Opportunities in Datacenters

The increase in datacenter energy consumption has sparked changes in every layer of the datacenter stack in search of efficient means of processing. This trend leads to growing efforts of designing and integrating accelerators in commercial datacenters [2]–[4].

While accelerators enable energy-efficient datacenter processing capabilities by specializing for common computation templates (e.g., machine-learning computations and graph-processing), recurring *data patterns* in datacenter workloads provide an additional opportunity for further efficiency.

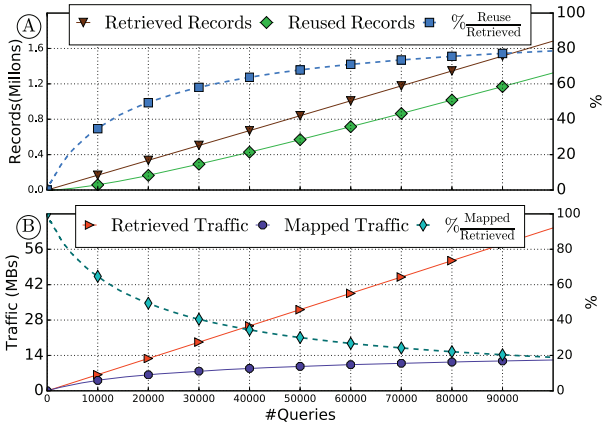


Figure 3: Traffic Reuse and Storage Requirements Across 100K Web Searches; Ⓐ Retrieved versus Reused Records, Ⓑ Retrieved versus Uniquely Stored Traffic Size.

Case Study I: Web-Server Traffic Compression: The steady increase in client-server traffic has brought traffic compression to the frontline of web-server designs. Recent surveys report that roughly 70% of all websites use gzip or one of its variants for traffic compression [37]. Gzip is supported by Apache and has been used as the baseline for *Snappy* [38], which is Google's compression library for its datacenters, big data processing machines, and internal RPCs.

Recently there have been several proposals for embedding datacenters with hardware implementations of gzip in ASIC and FPGA [39,40]. We present a case where memoization support for compression accelerators in datacenters is useful. In this scenario, we demonstrate the behavior of a web-based search engine that compresses result traffic. We collected a trace of database server traffic using the TailBench client-server benchmark suite [41]. The trace consists of the traffic generated by 100k search queries to a Xapian server [42] simulating a Wikipedia search engine. The generated traffic consists of records containing Wikipedia abstracts; each search query returns up to 25 records. In this scenario, it is likely that different terms will hit on the same records as part of their query result (e.g., "sign" and "language" can both retrieve the Wikipedia abstract for "sign-language"). As the number of tracked search terms grows, the fraction of previously retrieved abstracts grows accordingly.

Figure 3 depicts the timely mapping of: Ⓐ the records retrieved by search queries and the number of recurring records, i.e., records fetched by more than one query, and the fraction of recurring records from overall retrieved records; Ⓑ server traffic generated by retrieved records and the increase in capacity for a theoretical storage device that uniquely maps the traffic by records, i.e., without storing any record more than once. The figure highlights two interesting trends: (i) since the searched database is finite, the capacity required

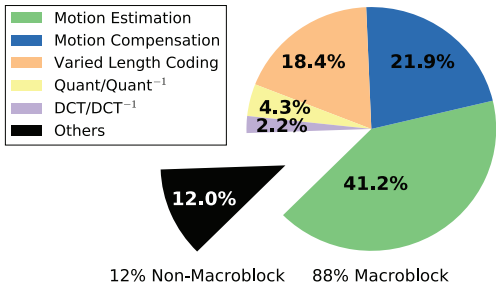


Figure 4: Video Encoding Execution Time Breakdown.

to construct the irredundant storage device saturates when the more popular records are already mapped, therefore mapping retrieved traffic to a confined storage device is possible. (ii) Record reuse rate increases to ∼80%, and is expected to increase further as the number of searches grows. From this experiment, we conclude that traffic compression accelerators for web search servers can gain from reuse storage.

Case Study II: Video Encoding: We present a different compute-reuse scenario that is not caused by recurring server traffic, but by redundant computations in video encoding. Video services have become increasingly ubiquitous over the past decade, and in specific, video uploading and sharing among different users. In 2015, the reported upload rate for YouTube servers was estimated at 400 hours of video every minute [43]. These rates establish the popularity of video encoders making them a target for hardware acceleration [7].

Modern video encoding techniques rely on the spatio-temporal locality in distinct parts of the encoded frames; each frame is split into groups of pixels, referred to as: "macroblocks". The locality assumption for video macroblock handling is that many videos consist of a few continuously moving subjects and a stationary background. Therefore, macroblocks that comprise the background tend to have fewer changes, while the macroblocks corresponding to moving objects can be tracked across frames in a process called: "motion estimation". Under the assumption of continuously moving objects, video encoders predict the future location of the moving objects' macroblocks in a process called: "motion compensation prediction".

Figure 4 shows an execution time breakdown of the main tasks in the x264 video encoder [44], for a sequence of the ten longest videos from the YouTube Faces video dataset [45]. The breakdown statistics were gathered from profiles of multiple x264 executions on a 40-core x86 machine, using 32 threads. From the depicted breakdown, it is apparent that macroblock handling accounts for a large ∼88% fraction of overall runtime, and specifically, more than 60% is attributed to motion estimation and motion prediction alone.

Interestingly, the stationary nature of macroblocks that video encoders leverage can be exploited in the context of memoization as well due to the computational redundancy made by processing recurring stationary macroblocks. Figure 5 demonstrates a visualization of macroblock reuse in a video, for different macroblock sizes. As expected for small macroblock sizes (8×8), after four seconds of video 77% of

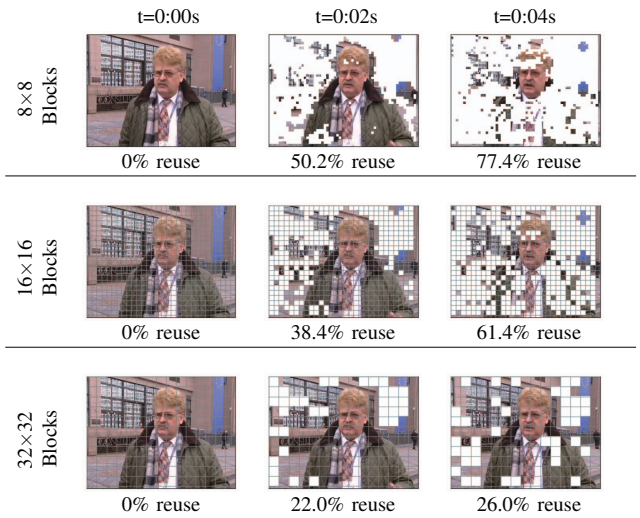


Figure 5: Macroblock Reuse Granularity in Video Frames. Reused Blocks Are Marked in White.

a frame's blocks were already recorded in previous frames. As macroblocks size increases, it is harder to match recurrent macroblocks. This figure demonstrates a fundamental tradeoff for memoization, which can be applied to equation (1): while potential memoization gains are higher for expensive computations with larger inputs ($C_{compute} \uparrow$), larger inputs are often harder to match, since they are less likely to exhibit high recurrence rates ($P_{hit} \downarrow$).

III. The COREx Architecture

COREx adopts a storage-centric approach to reduce the use of transistor-based accelerators by using memoization to fast-forward redundant computations. It relies on the integration of an auxiliary storage layer to back the computation layer. In this section, we present the fundamental elements of the COREx architecture. We describe the integration of COREx in the accelerator flow, present the main hardware components in the COREx scheme, and show the different stages of the hardware/software co-design flow.

A. Execution Flow

Accelerator Coupling: Accelerator models can be classified as Tightly-Coupled Accelerators (TCAs) and Loosely-Coupled Accelerators (LCAs) [46]. While TCAs are integrated with the CPU's pipeline and share its caches, LCAs are equipped with private scratchpad memories and communicate with the CPU via DMA transfers. LCA designs are considered more flexible, as they are not confined to a specific CPU architecture, they do not compete with CPUs over caches, and their use of heavily-banked scratchpad memories makes them a better fit for data-intensive workloads.

Accelerator Flow Modifications: One of the main appeals of the LCA model is that by decoupling DMA transfers from accelerator executions, they can be orchestrated independently and benefit from DMA optimizations such as double-buffering. The decoupling of input, execution, and

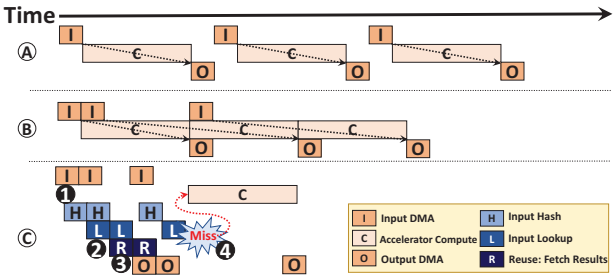


Figure 6: Accelerator Execution Flavors.

output makes LCAs ideal for memoization, since accelerators are relatively stateless and have no side effects on the system memory. Referential transparency is, therefore, guaranteed by-design without additional workload assumptions or support from the software stack. To support memoization, we modified the original execution model. Figure 6 shows a sequence of three kernel computations under different execution flows: Ⓐ the traditional model, which corresponds to a serial input → compute → output execution, Ⓑ a double-buffering model by Cota et al. [46], which improves utilization by overlapping execution and DMA transfers, and Ⓒ *the "HLR" model*: the new COREx flow of: "Hash→ Lookup→ Reuse", which is integrated with the accelerator compute stage. In the example given, the first two executed kernel inputs were already recorded by previous executions, while the third kernel input is a new input. Numbers signify the stages of different HLR executions. ❶ The first input is incrementally hashed as DMA transfers are fed to the accelerator (therefore the "H" and "I" stages overlap), and then the hash result is looked up in the table at the "L" stage. ❷ When the input is fully hashed, the hash lookup "hits", i.e., the hashed input matches the input fed to the accelerator, therefore the output can be fetched from the table instead of recomputing the input. ❸ Once the output is fetched, in the "R" stage the computation is reused i.e., sent back to the CPU. The input of the third kernel was not previously recorded, therefore, in ❹ it results in a "miss" in the "L" phase, and the accelerator subsequently executes. We also examined a policy in which accelerator execution speculatively overlaps with the lookup and aborts on lookup hit, but we found that the runtime gains of this approach were insignificant compared to the added energy costs.

B. Top-Level Design

To support the HLR execution model, we integrate new hardware structures with the existing LCA hierarchy, as shown in Figure 7. Since storage tables might be large, table accesses may have high timing and energy costs. Therefore, we designed the COREx architecture to have a two-phase lookup. A lookup of the hashed input in a small cache, and a full input comparison using entries stored in the large, RAM-like table. Without the first lookup phase, the flow still works correctly but might have more unnecessary accesses to the large table. We added three structures to the baseline accelerator; (i) the Input Hash Unit (IHU): a

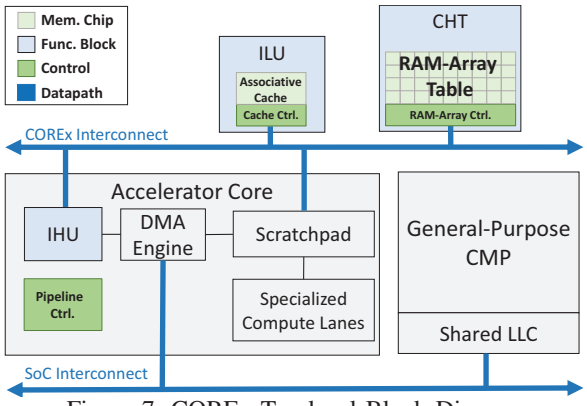


Figure 7: COREx Top-level Block Diagram.

computational unit handling DMA transfers of input, which constructs "index" and "pointer" hashes for incoming input blocks. (ii) the Input Lookup Unit (ILU): a cache indexed by the index hash value and tagged by the pointer hash, and: (iii) the Computation History Table (CHT): a large table addressed by the pointer hash. The CHT records past accelerator executions. Each CHT entry contains a tuple of: {computation input, computation output}. When fetching the entry, COREx does a full comparison between the fetched input and the input stored in the scratchpad and returns the fetched output in case the inputs match. As efficient table accessing is critical for timing and energy [47], we designed the CHT as a RAM array aggregating multiple memory banks that are organized in correspondence to two layout parameters: (i) **level of bank interleaving:** dividing the CHT space addressed by CHT pointer to distinct groups of smaller banks, and (ii): **level of bank parallelism:** distributing a single CHT entry across multiple parallel banks with smaller word size, such that the aggregation of the words read from the parallel banks constitutes the CHT entry.

The execution across all structures is orchestrated by the pipeline controller. All pipeline stages are shown in Figure 8. As the figure shows, there are three possible scenarios for the COREx flow: ❶ **hit**: the pointer hash is matched in the ILU, and computation input matches the input of the fetched CHT entry, and the CHT output is used instead of computing the kernel via the accelerator, ❷ **miss:** the pointer is not matched by the ILU, therefore the pre-computed result is not in the CHT, ❸ **false-hit**: the pointer hash is matched in the ILU, but the fetched entry contains a different input, therefore the accelerator needs to compute the kernel results.

C. Input Hashing

The first execution stage is the input hashing stage. This stage is done by the IHU, which incrementally constructs hashes as input blocks are fed to the accelerator. In our experiments, we found that a tree-based bitwise xor is a good fit for incremental construction, and generates a near-uniform distribution. The N-bit hash function is described as follows: given the N-bit hash value currently stored in the register file, $R_{current}^{\{0,..,N-1\}}$, an M-bit input block, $I^{\{0,..,M-1\}}$ (without loss of generality, we assume: $M = \ell \cdot N$),

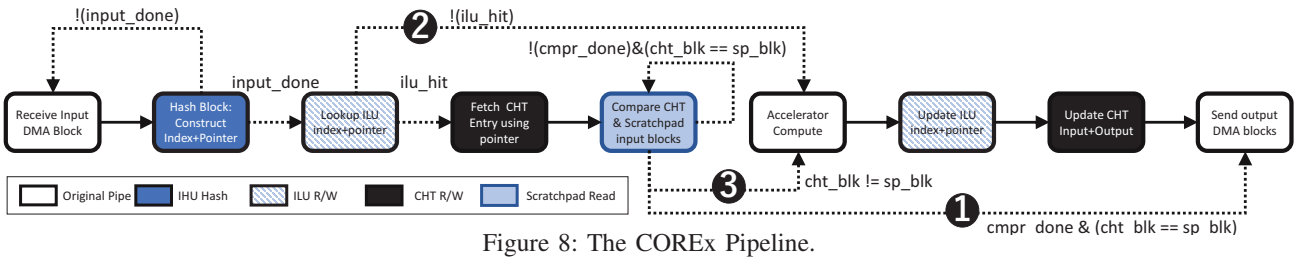
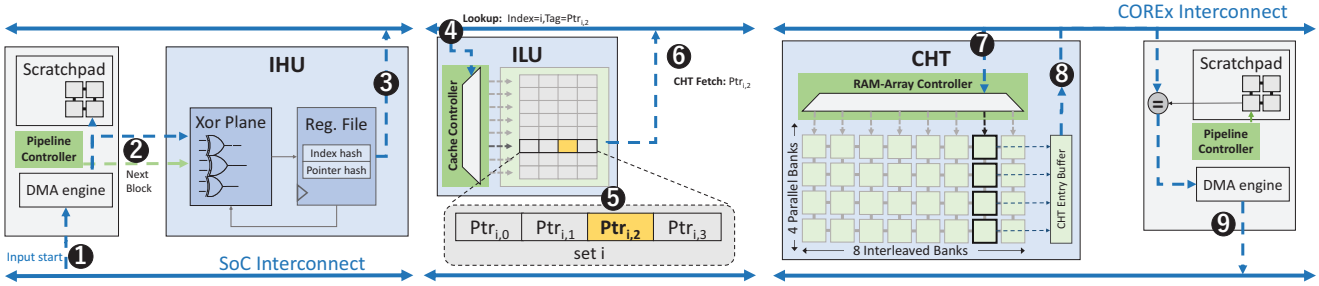


Figure 8: The COREx Pipeline.



(a) Input Hashing (b) Input Lookup (c) Compute-Reuse

Figure 9: COREx Execution Across The Different COREx Components.

the hash value calculated in the next cycle, $R_{next}^{\{0,\ldots,N-1\}}$, is:

$$R_{next}^{0} \leftarrow R_{current}^{0} \oplus I^{0} \oplus I^{N} \oplus I^{2N} \ldots \oplus I^{(\ell-1)\cdot N}$$
$$R_{next}^{1} \leftarrow R_{current}^{1} \oplus I^{1} \oplus I^{N+1} \oplus I^{2N+1} \ldots \oplus I^{(\ell-1)\cdot N+1}$$
$$\ldots$$
$$R_{next}^{N-1} \leftarrow R_{current}^{N-1} \oplus I^{N-1} \oplus I^{2N-1} \oplus I^{3N-1} \ldots \oplus I^{M-1}$$

Figure 9a demonstrates the input hashing operation: ❶ an "input start" signal is received, marking the beginning of an input DMA sequence. ❷ the IHU tracks the DMA transfers of input blocks and uses the xor-tree to calculate hashes of the input. The IHU concurrently constructs two hashes to use in the lookup stage: (a) the "index hash", an N-bit hash used as index to the ILU, and (b) the "pointer hash", an $\tilde{N}$-bit hash ($\tilde{N} > N$) used as a pointer to the CHT, and is stored in the ILU as tag. An "input done" signal received finalizes the incremental hash values, and therefore in ❸ , the IHU initiates a lookup to the ILU via the COREx interconnect.

COREx uses two hash values to achieve both coverage and accuracy. Since the lookup determines whether or not to access the large CHT, its decision is performance and power critical. We use the index hash as an index to the ILU for low latency and power efficiency in case of a miss. The use of a pointer hash as tag increases confidence by reducing the number of "false-hits", i.e., redundant accesses to the CHT that end up as a mismatch following a full input comparison.

D. Input Lookup

Once the input hashes are constructed, they are used to access the ILU, a set-associative cache indexed by the index hash, and tagged by the pointer hash. The ILU's data array contains pointers used to access the CHT. Figure 9b shows an example of a successful lookup; ❹ the lookup is initiated by the IHU, which accesses the i^{th} set using index hash = $Ptr_{i,2}$; ❺ the pointer hash is compared with all of the set tags, matching $\text{Ptr}_{i,2}$ in way 2; ❻ the matched pointer hash, $\text{Ptr}_{i,2}$, is used to access the CHT.

E. Compute-Reuse

The final stage is fetching and reusing results from the CHT. Figure 9c shows an example of a 32 banked CHT, in which each CHT entry, i.e., {kernel_input, kernel_output} is distributed across four banks, and the CHT address space is divided to eight equally sized spaces. Compute-reuse is done in the following steps: ❼ a fetch request is sent from the ILU (using the address of $Ptr_{i,2}$), matching a group of parallel banks. The fetched CHT entry is stored in a buffer and forwarded via the interconnect to the accelerator, which, in ❽ COREx performs a full input comparison by comparing the kernel input stored in the scratchpad memory, with the {kernel_input} from the fetched entry. In ❾ , comparison is completed, verifying that the requested input matches the CHT entry, and the result can be used instead of using the accelerator, and is sent via DMA on the SoC interconnect back to the host processor, as the final computation result.

F. End-to-End Design Flow Example

We demonstrate the end-to-end design flow of the accelerated architecture co-designed with specialized compute-reuse memory layers. This process involves an understanding of the costs of accelerated kernels and underlying memory technology, and an exploration of the memoization tradeoff space to maximize the effectiveness of the memory modules, depending on the optimization goal.

We used the Sum of Absolute Differences (SAD) computation as a case-study. In video encoding, SAD is used as a metric for differences between blocks of pixels referred to as: "macroblocks". Using SAD, the encoder constructs a motion vector for objects in the video. Figure 10 provides an auxiliary visualization for the motion vector construction.

Application Programming: Memoization requires a high-level understanding of the program. Fortunately, the required

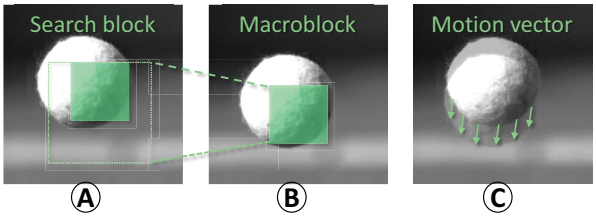


Figure 10: Motion Vector Construction; Ⓐ the previous frame. Ⓑ current frame Ⓒ the constructed motion vector.

Parameter	Explored Values
Clock Cycle Time	2ns, 4ns, 6ns, 10ns
Loop Unrolling Factor	1, 2, 4, 8, 16, 32
Loop Pipelining	True/False
Array partitioning (=Mem. Ports)	2, 4, 8, 16, 32, 64

Table I: Accelerator Design Sweep Parameters.

reasoning involves defining the memoized kernel and decoupling it from its inputs and outputs, all of which are already done by programmers when writing kernels and programs. Snippet 1 demonstrates an implementation of a SAD kernel using an API compatible with heterogeneous programming frameworks such as CUDA or OpenCL. Since the kernel, its inputs, and its outputs are already defined by the programmer no additional programming effort is needed since memoization correctness is guaranteed by the compiler.

```
// Definitions: search an 8x8 pixel macroblock
// in a 20x20 pixel search block
#define  MX 8
#define  MY 8
#define  SX 20
#define  SY 20
#define  dX (SX/2)
#define  dY (SY/2)
__kernel__ void SAD(rgb** mb,rgb** sb,int** res) {
  for (int i = 0; i< SX-MX; i++) {
    for (int j = 0; j< SY-MY; j++) {
      res[i][j]=0;
      for (int x = 0; x< MX; x++) {
        for (int y = 0; y< MY; y++) {
          res[i][j]+=abs(sb[i+x][j+y]-mb[x][j]);
        }
      }
    }
  }
}
..
// Main program
for (int x = 0; x< IMAGE_WIDTH; x+=MX) {
  for (int y = 0; y< IMAGE_HEIGHT; y+=MY) {
    id = x*IMAGE_WIDTH + y;
    // dma2DCpyFromHost/Acc(dest,src,x1,x2,y1,y2):
    // DMA copies of a 2D array, src[x1:x2][y1:y2],
    // to a flat array: dest[0..(x2-x1)*(y2-y1)]
    // from the host/accelerator to the accelerator/host
    dma2DCpyFromHost(mb[id],img,x,x+MX,y,y+MX);
    dma2DCpyFromHost(sb[id],ref,x-dX,x+dX,y-dY,y+dY);
    SAD(mb[id],sb[id],acc_res[id]);
    dma2DCpyFromAcc(res[id],acc_res[id],0,SX-MX,0,SX-MX);
  }
}
```

Snippet 1: SAD Computation in Video Motion Estimation

Accelerator Design: Since the end of Moore's law dictates that accelerator gains will be limited by the design points achievable under a fixed transistor budget, we wish to estimate the characteristics of highly-tuned accelerators that are tailored for three different optimization goals: (i) minimal execution time, (ii) minimal energy spent, and: (iii)

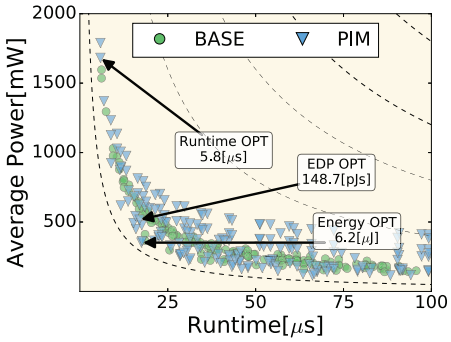


Figure 11: SAD Design Sweep Across Traditional ("BASE") and Processing-In-Memory ("PIM") Accelerators.

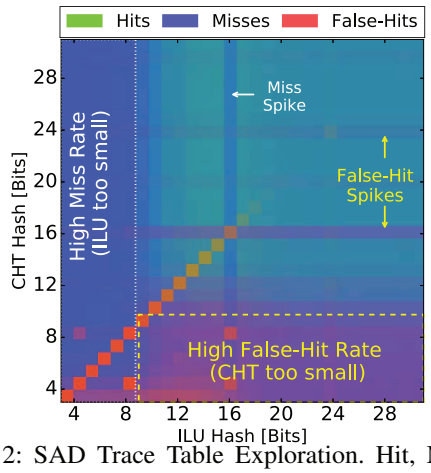


Figure 12: SAD Trace Table Exploration. Hit, Miss, and False-Hit Rates As a Function of ILU+CHT Space Sizes.

minimal energy-delay product (EDP). We used Aladdin [48], a performance, power, and area estimation framework for accelerators that operates in a similar fashion to an HLS tool. We tested a range of parameters, detailed in Table I to find the optimal levels of software pipelining and parallelism, RTL clock cycles, and degrees of memory level parallelism. We explored two accelerator designs: a "BASE" design which refers to an accelerator with SRAM scratchpad memory banks connected to a logic chip with registers and specialized computation lanes, and a "PIM" design which refers to a Resistive-RAM scratchpad packaged with an in-memory logic layer that carries out the arithmetic operations. Figure 11 shows all obtained design points (highlighting the per-goal optimal points) and dashed Pareto curves of fixed EDPs.

Trace Mapping Exploration: We quantify the level of compute-reuse in the SAD kernel, in order to estimate the effectiveness of memoization tables for the application. We constructed a trace simulator that collects workload input and output traces and characterizes their distribution with respect to different table sizing, from which we derive the table hash width. As input we used traces extracted from ten videos from the YouTubeFaces dataset [45] and classified the table performance according to the number of hits, misses, and false-hits which were defined in Section III-B. Compute-reuse is expected due to spatial-redundancies in video frame, similar to those discussed in Section II-C.

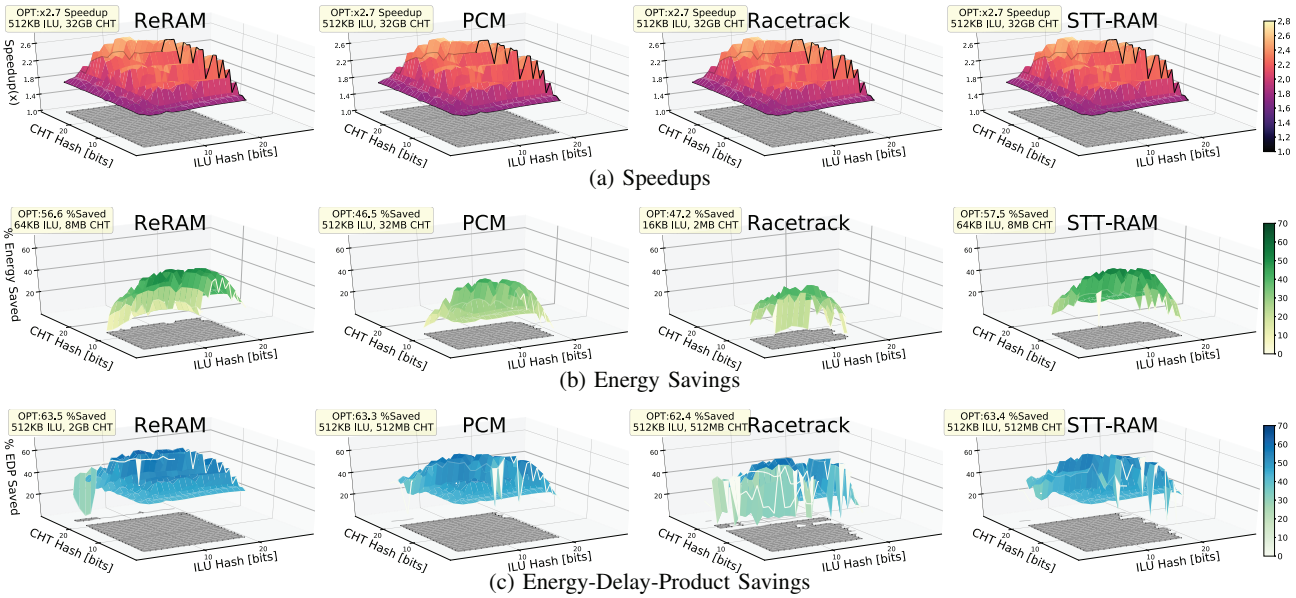


Figure 13: COREx Gain Space Exploration for SAD. Optimal Gains and Design Points Are Detailed in Each Graph.

Figure 12 shows the distribution of hit, miss, and false-hit rates for different ILU and CHT hash widths. The figure reflects the following trends: (i) as expected, higher miss rates and false-hit rates are observed for small ILU and CHT sizes, accordingly, and: (ii) the hashing function experiences higher collision rates for hash sizes that are multiples of eight, resulting in the periodic patterns shown in the figure; this is the result of aliasing introduced by the 8-bit RGB channel representation of the video images in the input trace.

Memory Layer Specialization: Following the construction of the map of hits, misses, and false hits, we estimate the gains achieved by COREx for each design goal. We integrate the memory layers with the goal-optimized accelerator, and conduct a design sweep for each memory technology and pair of ILU+CHT hash sizes under different CHT layouts. We simulate the tables using Destiny [49], a contemporary emerging memory modeling tool. Figure 13 shows the sweeps across the space of possible gains, highlighting the configuration that achieved the best gain. For presentation purposes, we removed non-beneficial configurations (e.g., ILU and CHT sizes that did not achieve speedups). As the figure reflects, the best speedup for all memory technologies was around 2.7×, while energy and EDP savings depend on the characteristics of each technology's resistive memory cell and layout. For energy and EDP savings, the beneficial space (i.e., configurations that produce gains) is smaller than the space for runtime. This is due to the use of banking which parallelizes accesses to small aggregated memory modules with low access latency. The use of a large CHT is sub-optimal when optimizing for energy, due to high energy costs of parallel accessing, and static power dissipated by large arrays. Finally, for all design goals, we saw that gains improve as table space increases up to the point of diminishing returns.

IV. Evaluation

In this section, we demonstrate the benefits of using COREx with ReRAMs, PCMs, Racetrack memories, and STT-RAMs. We compare COREx against highly-tuned accelerators. We explore accelerators with SRAM-based scratchpads connected to a CMOS-based logic chip, similar to previous academic proposals [5,46,50]–[52]. We also processing-in-memory accelerators that consist of logic cores which are packaged with Resistive-RAM based memory arrays on the same die to reduce data movement costs. We tailor each accelerator to a specific application, and extract input traces to specialize the memory-based tables, following the end-to-end COREx design flow discussed in Section III-F.

A. Methodology

Evaluated Workloads and Inputs: We outline all evaluated datacenter workloads in Table II. For the video encoding kernels, i.e., *"DCT"* and *"SAD"*, we used videos from the YouTube Faces dataset [45], and expect gains from block reuse in videos, as described in Section II-C. *"Snappy"* is the compression algorithm used by Google, and we used it as an example for traffic compression of Wikipedia query responses. We used 13 million queries generated by a Xapian DB server [41,42] containing abstract dumps published by the Wikipedia foundation [54]. As we described in Section II-C, web query results tend to hit the same pages, therefore we expect to see recurring results even for different queries, resulting in multiple kernel executions for the same blocks. The growing popularity of graph-centric social networks and content recommendation systems has driven the emergence of graph processing accelerators. We included several graph processing algorithms in our experiments: (i) the single-source shortest-path (*"SSSP"*) algorithm used by map navigation

Kernel	Domain	Use-Case	Kernel Source	Input Source and Description
DCT	Video Encoding	Video Server	x264 [44]	YouTube Faces [45]. 10 Videos, 10 Seconds, 24 FPS, Warmup 1^{st} Sec., 8×8 Macroblocks.
SAD	Video Encoding	Video Server	PARBOIL [53]	YouTube Faces [45]. 10 Videos, 10 Seconds, 24 FPS, Warmup 1^{st} Sec., 8×8 Macroblocks,20×20 Searchblocks.
SNAPPY ("SNP")	Compression	Web-Server Traffic Compression	TailBench [41][1] Snappy-C [55]	Wikipedia Abstracts [54]. 13 Million Search Queries, Up to 20 Retrieved Abstract Per Query, 4KB Abstracts.
SSSP ("SSP")	Graph Processing	Maps Service: Shortest Walking Route	Internal	DIMACS, Graphs of NYC Streets [56], 10 Million Zipfian Transactions, 264K Nodes, 733K Edges.
BFS	Graph Processing	Online Retail	MachSuite [57]	Amazon Co-Purchasing Network [58], 10 Million Zipfian Transactions, 403K Nodes, 3M Edges.
RBM	Machine Learning	Collaborative Filtering	CortexSuite [59]	Netflix Prize: Movie Ratings [60]. 10 Million Zipfian Transactions, 150K Floating Point Weights.

Table II: Evaluated Workloads, Input Sources, and Use-Cases.

Kernel	DCT	SAD	SNP	SSP	RBM	BFS
Runtime[μs]	7.32	5.85	0.88	1.95	50.15	2.69
Energy[μJ]	8.98	6.16	10.17	2.98	1.5K	15.03
EDP[pJs]	396.88	148.75	8.99	8.05	78K	64.06

Table III: Kernels and Optimal Design Sweep Points.

COREx Interconnect			Bank Muxing	
Flit-Size	Latency	Energy	Latency	Energy
64-bit	1ns	30pJ	$\sqrt{\#Banks}\times$20ps	# Banks×2fJ

Table IV: General System Parameters.

services such as Google Maps; we used the roads of New York as a case study. In this scenario, a datacenter serves navigation requests to find the shortest walking path between two addresses. We used a graph containing five-hundred adjacent streets as accelerator input, as most walking paths are limited to a few blocks. (ii) The Breadth-First Search (*"BFS"*) graph traversal algorithm which we used in the context of product co-purchasing recommendations in online retailer services. In our experiments, we used the Amazon product co-purchasing network taken from data gathered by Leskovec et al. [58]. (iii) The Restricted-Boltzmann Machine *("RBM")* is a machine-learning algorithm used by online content services for collaborative filtering, i.e., to detect user-content correlations. We used the Netflix prize dataset [60] to construct an accelerator for a movie recommendation system.

Zipfian traffic: Datacenters and web-servers are designed to satisfy biased traffic. More specifically, traffic has been shown to abide by Zipf's Law; a statistical linguistic law, formulated in 1929 [61] stating that given a corpus of words (i.e., a large dictionary), words are distributed in an inversely proportional manner to their popularity, i.e., the i^{th} most common word is distributed proportionally to i^{-1}. Interestingly, Zipf's Law was shown to apply for internet traffic [20,21], and it was empirically shown to be the key component for enterprise traffic compression [62]. Therefore, as some datacenter requests are more popular than others, some computations have high repetition rates, and this repetition might manifest as recurrent (and redundant) accelerator computations. For "SSSP", "BFS", and "RBM" we generated traces that follow a Zipfian distribution with 75%, 90%, and 95% skew rates. These skew rates are conservative compared to the rates used by recent studies [5,63,64].

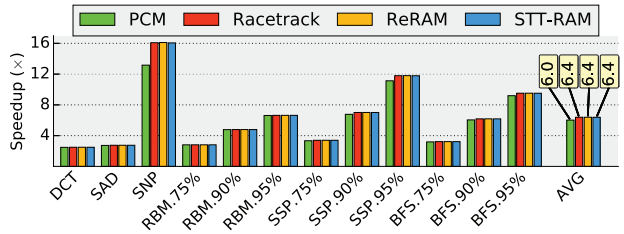
Accelerator, COREx Tables, and Interconnection Modeling Framework: For each workload, we ran a design sweep for the accelerator core using the Aladdin modeling framework [48] and the Destiny memory modeling tool [49] to simulate both scratchpad and the ReRAMs for PIM-based accelerators. We extracted the best results for each design goal. We also used Destiny to model the ILU and CHT memory layers in COREx as ReRAM, PCM, STT-RAM, and Racetrack memories. We used these technologies as it is likely for some of them to scale post the end of Moore's Law. We used heavy memory banking and block-level parallelism to achieve low latency and low dynamic energy accesses to the large CHT storage layer. Since the hashing scheme is coupled with the table organization and banking mapping, it is bank aware, which enables the selection of the appropriate bank via muxing done outside the CHT (i.e., lookup phase) to further reduce energy. Since Aladdin models 45nm CMOS logic and scratchpads, we use 45nm memories for fair comparison. We set interconnection energy and latency following recent NoC studies [65,66]. For each workload, we swept across both traditional and PIM accelerators and picked the optimal points as the baseline accelerator. Table III summarizes the optimal accelerator design sweep points. The parameters for interconnect and muxing are shown in Table IV.

Evaluated Configurations: In our evaluation we use the tuned accelerators as the baseline for comparison, and tune COREx for three optimization goals: Runtime-OPT is the configuration that achieves the best speedup compared to the runtime-optimized accelerator; similarly, Energy-OPT and EDP-OPT are the configurations that save the most in energy and energy-delay-product, for the Energy-OPT and EDP-OPT accelerators, respectively. We explore different memory layer implementation alternatives for the ILU and CHT using emerging memory models that are used by researchers: Resistive-RAM (ReRAM) [67]–[69], Phase-Change Memory (PCM) [10,70], Spin-Transfer Torque RAM (STT-RAM) [18,71], and Domain-Wall Memories (DWM), commonly known as Racetrack memories [19,72]. We derive different configuration design points based on the optimization objectives and the relations between the given workload and underlying memory technology. Table V shows the optimal ILU and CHT configurations for each memory and optimization goal. For presentation purposes, we present

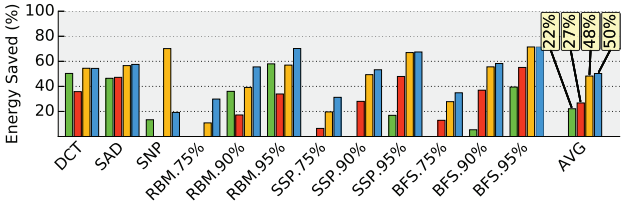
[1] TailBench is a client-server benchmark suite, in which the server provides the computational services for the client; therefore, we focused our evaluation on the server side only.

Workload	Optimization Target	PCM		Racetrack		ReRAM		STT-RAM	
		CHT	ILU	CHT	ILU	CHT	ILU	CHT	ILU
DCT	Runtime	4GB:2×64(12ns)	524KB	4GB:2×64(2ns)	524KB	4GB:2×64(2ns)	524KB	4GB:2×64(2ns)	524KB
	Energy	16MB:2×1(374ns)	524KB	2MB:2×1(50ns)	65KB	16MB:2×8(3ns)	524KB	16MB:2×1(321ns)	524KB
	EDP	536MB:2×8(374ns)	524KB	134MB:2×16(193ns)	262KB	536MB:2×32(7ns)	524KB	536MB:2×16(321ns)	524KB
SNP	Runtime	274GB:2×512(14ns)	8MB	1TB:2×512(13ns)	8MB	1TB:2×512(10ns)	8MB	1TB:8×512(6ns)	8MB
	Energy	2GB:2×32(374ns)	8MB	No Gains		8GB:2×256(9ns)	8MB	1GB:2×32(321ns)	4MB
	EDP	4GB:2×512(102ns)	1MB	1GB:2×512(50ns)	131KB	34GB:2×512(20ns)	8MB	2GB:2×512(82ns)	1MB
RBM.75%	Runtime	2TB:16×9K(11ns)	134MB	1TB:2×37K(2ns)	134MB	1TB:2×37K(1ns)	134MB	1TB:2×37K(2ns)	134MB
	Energy	No Gains		No Gains		614MB:2×586(6ns)	134MB	1GB:2×36(1μs)	262KB
	EDP	19GB:2×293(374ns)	2MB	1GB:2×73(764ns)	4KB	19GB:2×36(7ns)	2MB	19GB:2×146(1μs)	2MB
RBM.95%	Runtime	1TB:32×18K(11ns)	8MB	629GB:2×37K(1ns)	8MB	629GB:64×37K(1ns)	8MB	629GB:2×37K(1ns)	8MB
	Energy	4GB:2×36(1μs)	8MB	76MB:2×73(764ns)	65KB	1GB:2×586(13ns)	67MB	2GB:2×36(1μs)	16MB
	EDP	19GB:2×73(1μs)	65KB	1GB:2×73(764ns)	4KB	19GB:2×4K(3ns)	2MB	19GB:2×146(1μs)	65KB
BFS.75%	Runtime	8TB:16×2K(14ns)	16MB	8TB:4×2K(13ns)	16MB	8TB:4×2K(10ns)	16MB	8TB:4×2K(16ns)	16MB
	Energy	No Gains		8MB:2×32(50ns)	32KB	67MB:2×32(13ns)	524KB	67MB:2×8(320ns)	131KB
	EDP	1GB:2×259(34ns)	131KB	543MB:2×2K(13ns)	16KB	17GB:2×2K(7ns)	524KB	1GB:2×64(321ns)	131KB
BFS.95%	Runtime	8TB:4×2K(20ns)	8MB	8TB:4×2K(13ns)	8MB	8TB:4×2K(10ns)	8MB	8TB:4×2K(16ns)	8MB
	Energy	536MB:2×16(374ns)	524KB	33MB:2×32(50ns)	65KB	543MB:2×259(3ns)	16MB	134MB:2×8(321ns)	131KB
	EDP	1GB:2×129(102ns)	262KB	543MB:2×2K(13ns)	16KB	17GB:2×2K(7ns)	524KB	1GB:2×64(321ns)	131KB

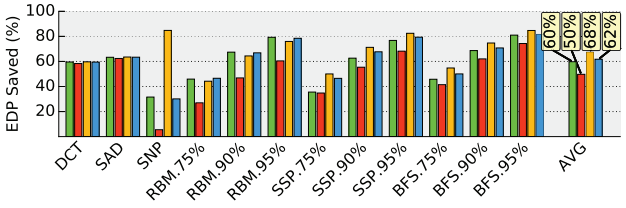
Table V: Optimal Table Configurations For a Subset Of Workloads, Following Memory-Layer Specialization; CHT Total Size and Layout ($N \times M = N$ Interleaved $\times$ M Parallel Banks), CHT Bank Latency, and ILU Size. ILUs are 4-Way Set-Associative.



(a) Speedups for Runtime-OPT COREx.



(b) Energy Savings for Energy-OPT COREx.



(c) EDP Savings for EDP-OPT COREx.

Figure 14: COREx Energetic and Runtime Gains.

the optimal table configurations for a subset of workloads. We used a 4-way associative organization for ILU, since we found that it provided a desirable hit-rate to cost tradeoff.

B. Results

Figure 14 shows the gains in runtime, energy, and EDP for the workloads evaluated using the design points in Table V.

Results Summary: Compared to the runtime optimized accelerator, COREx achieves an average speedup of 6.0× for PCM, and 6.4× for all other memory technologies. In energetic terms, COREX saves an average of 22%, 27%, 48%, and 50%, for PCM, Racetrack, Resistive-RAM, and STT-RAM, respectively. The EDP savings were 60%, 50%, 68%, and 62%, for COREx configurations using PCM, Racetrack, Resistive-RAM, and STT-RAM, respectively.

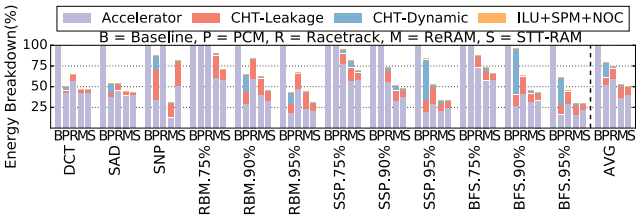


Figure 15: Energetic Breakdown for Energy-OPT Schemes.

Behavior Under Different Objectives and Skews: As reflected by Figure 14 and Table V, the increase in Zipfian traffic skew achieves higher speedups, but does not result in a change in the table size in Runtime-OPT. We found that the reason for this is that the workload configuration for these workloads was set to the largest CHT (determined by workload input and output sizes), and therefore achieved the highest hit rate. Furthermore, we found that CHT hit rate had a higher impact on the timely performance of COREx than lookup costs (i.e., Muxing, ILU, NoC, and CHT bank latencies). When energy was the goal, we saw that table sizes grew as skew increases. The reason for this behavior stems from the fact that for higher skews, the marginal leakage and access costs of increasing the table size were smaller than the energy gained from increasing the hit rate and using the tables instead of the accelerator.

Memory Technology Implications: In our experiments, we saw that all memory technologies achieved roughly the same speedups with minor differences, this is the result of COREx's parallel CHT banking structure, enabling shorter table access latencies compared to accelerator computation speeds, while the choice of underlying memory technology plays a less significant role. In general, when runtime is the primary objective, optimal CHT sizes are large, and are roughly the same size for all memory technologies, given the same workload. For PCM, we see that in three benchmarks there were no configurations that achieved energy gains, but we saw that gains were achieved for the same benchmarks

with higher skews. We saw that the reason for this is the high dynamic energy costs dominating phase-change memories compared to the other technologies explored. We also saw that STT-RAMs and Resistive-RAMs achieved superior and similar energy and EDP rates across most of the workloads.

Energetic Breakdowns: Figure 15 shows the energetic breakdown of different COREx components for the Energy-OPT configurations, compared to the baseline accelerator-only configuration. We present the energy spent by the computing accelerator core, the leakage energy from CHT arrays, the dynamic CHT access energy (reads/updates), and all other components (ILU leakage and dynamic energy, the scratchpad memory (SPM) reads for full-input comparison and NoC interconnect traffic). As expected, for workloads that have large traffic skews, tables can save the energy spent in accelerator core computations and reduce overall energy spent. The energy costs of COREx components are dominated by the accelerator core, followed by the rates of leakage energy of the CHT. In contrast to the other memory technologies, current phase-change memory models reflect relatively high write energy and latency, making CHT updates expensive. As reflected by Table V, since typical ILU storage size is orders-of-magnitude smaller than CHT size, static energy rates are negligible, this is also true for the energy spent by other components (e.g., NoCs and Scratchpad reads).

An Outlook on Compute-Reuse for Accelerators: As the experiments conducted show, existing non-volatile memory technologies can deliver performance and energy returns for accelerated architectures, although the depend on traffic skews and other sources of computational redundancies for beneficial memoization. As discussed in Section II-A, since current projections show a steady increase in storage density, **architectures such as COREx which leverage storage as their primary source of gain will become more appealing.** We speculate that future technology roadmaps will enable the construction of denser tables with smaller arrays and expand the applicable table space for compute-reuse architectures, thereby delivering the same hit rates for smaller costs, or by improving accelerator returns for lower hit-rates.

V. Related Work

Table VI summarizes the different properties of previous studies and how they relate to this work. To the best of our knowledge, no previous hardware memoization study had explored coarse-grained memoization for accelerators nor the use of large NVM-based tables for memoization, no previous software memoization study had used memoization for accelerated applications, and no study had suggested memoization as a tool to alleviate the transistor-based scalability limitation is accelerated systems.

Datacenter Acceleration: The appealing performance efficiency of hardware specialization and recent ubiquity of ASIC and FPGAs motivated the integration of specialized hardware in datacenters and cloud servers. Academic studies

	Memoization	Accelerators/ GPUs/HSAs	Big-Data or Datacenter Apps	Emerging Memories	Massive Storage
[76]–[83]	✓	✗	✗	✗	✗
[84]	✓	✗	✓	✗	✗
[85]–[87]	✓	✓	✗	✗	✗
[88]	✓	✓	✓	✓	✗
[34]–[36]	✓	✗	✓	✗	✓
[2,5,7,74,75] [3,4,73]	✗	✓	✓	✗	✓
[71,89,90]	✗	✓	✗	✓	✗
[67,69,91,92]	✗	✓	✓	✓	✗
COREx	✓	✓	✓	✓	✓

Table VI: Taxonomy of Related Studies.

developed specialized hardware for Memcached [5], Apache and Hadoop [73], Bitcoin mining and video transcoding [7], DRAM-based reconfigurable datacenter fabrics and the effects of "dark-silicon" servers [74]. In industry, commercial datacenters have begun shifting towards heterogeneity with recent efforts of specialization using FPGAs [2,75], ASICs [4], and GPUs [3]. As shown in this work, we believe these schemes and future accelerated datacenters would benefit from memoization, as it would increase their computation scalability using emerging memories.

Memoization: Memoization has been extensively explored in past work and is supported by modern programming environments. To the best of our knowledge, it has not yet explored as a means to trade CMOS-based computations for scalable memory technologies, nor has it applied to entire kernels in accelerated systems.

Hardware memoization schemes were studied prior to the emergence of accelerators. In contrast to COREx, they were implemented using small tables and targeted fine-grained recurring computations such as instructions [77,78], math functions [76], floating-point ops [79], instruction scheduling sequences [81], and basic blocks [80].

Compiler-centric memoization studies employ optimizations to memoize different targets by exploiting redundancy in distinct code regions, such as loops [82], pure functions [83], and shared library code for datacenter applications (e.g., Memcached [84]). Recent efforts to integrate memoization into datacenters focus on changes to the software stack to improve storage [35], or eliminate redundant computations via iterative processing in MapReduce frameworks [34,36].

Compared to existing software and datacenter memoization, COREx is fit for accelerated architectures. Its flow does not require any specific workload assumptions or programming framework properties to guarantee memoization correctness. COREx gains from the low runtime overhead of hardware implementation. For example, the typical latencies measured in this work are on the order of a few microseconds, which are orders of magnitude less than commercial Memcached latencies [93]. Low overheads are critical to preserve the advantage of hardware acceleration over software in modern datacenters [2] and enable the deployment of large memoization tables to yield better coverage. Furthermore, not all memoization opportunities are reflected at the software layer;

in cases such as the video example in Section II-C, blocks are reusable only when constructed on-the-fly when videos are processed. In our experiments, we saw that macroblock reuse rates across videos were insignificant, therefore having a pre-constructed software database would not be useful.

Several memoization studies were explored in the context of GPUs targeting finer-grained computations, like rendering of pixels [85], approximate scatter/gather patterns [86], caching of texture computations [87], or using ReRAM based CAMs for fine-grained computations [88]. Compared to these works, COREx supports coarse-grained memoization by combining a comprehensive design flow of large tables and emerging memories that open up new opportunities for memoization. Since COREx does not rely on internal structures (e.g., GPU shader units) it offers a solution that applies to a wide range of accelerated applications.

PIM / NVM-Based Accelerators: The ubiquity of big-data workloads and progress made in die-stacking manufacturing technologies have motivated the re-emergence of processing-in-memory (PIM) architectures, originated in the 1990s [94,95]. Modern PIM architectures reduce the costs of data movement between logic and memory dies by vertically stacking memory and logic layers on the same die ("3D stacking") [96], or packaging multiple dies on the same silicon substrate ("2.5D stacking") [97].

The emergence of new memory models has motivated researchers to explore new avenues of using memory to accelerate big-data applications. Recent studies explored an in-situ acceleration of neuromorphic computations [67,90,91] combinatorial optimizations [69] and array-operations [92].

While some PIM accelerators rely on emerging memory technologies, they tackle the "memory wall" problem that is induced by the high costs of memory accesses in traditional architectures. In contrast, COREx uses scalable memory technologies to *replace* logic, rather than bringing memory closer to logic. As we compared with PIM accelerators in Section IV, the gains from memoization and PIM-based acceleration are independent and can be combined.

VI. Conclusions

We have presented COREx, a storage driven compute-reuse architecture for future datacenter accelerators. We discussed the implications of the end of transistor scaling in accelerators, while presenting still-scaling storage as means to make future improvement in computation capabilities possible. We integrated contemporary emerging memory technologies to construct COREx in order to compensate for non-scaling CMOS technology by transforming transistor-based computations to table storage which continues to scale. We presented a detailed overview of the COREx architecture, and apply it to accelerated kernels to improve runtime, energy, and EDP. Beyond the presented gains, the concept of transforming problems to the still-scaling storage domain shows promise for future architectures. We believe that these gains will improve as future memory technologies keep scaling, paving the way for storage-centric architectures.

Acknowledgment

We thank the anonymous reviewers for their valuable feedback. This work was partially supported by the NSF under Grants No. CCF-1453112 and CCF-1438980, AFOSR under Grant No. FA9550-14-1-0148, and DARPA under Grant No. N66001-14-1-4040. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] "International technology roadmap for semiconductors (ITRS)," http://public.itrs.net, 2015 executive summary, 2015.

[2] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24.

[3] "NVIDIA Tesla P100: The Most Advanced Data center GPU Ever Built," http://www.nvidia.com/object/tesla-p100.html.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2017, pp. 1–12.

[5] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin servers with smart pipes: Designing SoC accelerators for memcached," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2013, pp. 36–47.

[6] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2015, pp. 158–169.

[7] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "ASIC clouds: Specializing the datacenter," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 178–190.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283.

[9] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient data mapping and buffering techniques for multilevel cell phase-change memories," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 40:1–40:25, 2014.

[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2009, pp. 2–13.

[11] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[12] M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing performance and lifetime of MLC PCM by using a region retention monitor," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 385–396.

[13] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2015, pp. 476–488.

[14] R. Aluguri and T. Y. Tseng, "Overview of selector devices for 3-d stackable cross point RRAM arrays," *IEEE Journal of the Electron Devices Society*, vol. 4, no. 5, pp. 294–306, 2016.

[15] C. Liu and H. Li, "A weighted sensing scheme for reram-based cross-point memory array," in *IEEE Computer Society Annual Symposium on VLSI (ISLVLSI)*. Washington, DC, USA: IEEE Computer Society, 2014, pp. 65–70.

[16] J. S. Meena, S. M. Sze, U. Chand, and T.-Y. Tseng, "Overview of emerging nonvolatile memory technologies," *Nanoscale research letters*, vol. 9, no. 1, p. 526, 2014.

[17] "ISSCC 2017 trends," 2017. [Online]. Available: http://isscc.org/doc/2017/ISSCC2017_TechTrends.pdf

[18] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Intl. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.

[19] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "STAG: Spintronic-tape architecture for gpgpu cache hierarchies," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 253–264.

[20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM (1)*, 1999, pp. 126–134.

[21] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[22] T. R. G. Nair and P. Jayarekha, "A rank based replacement policy for multimedia server cache using zipf-like law," *CoRR*, vol. abs/1003.4062, 2010. [Online]. Available: http://arxiv.org/abs/1003.4062

[23] D. Mituzas, "Flashcache at facebook: From 2010 to 2013 and beyond," 2014.

[24] D. Michie, "Memo functions and machine learning," *Nature*, vol. 218, no. 5136, pp. 19–22, 1968.

[25] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "Accelerator-rich architectures: Opportunities and progresses," in *Intl. Design Automation Conference (DAC)*. New York, NY, USA: ACM, 2014, pp. 180:1–180:6.

[26] Y. S. Shao, S. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-Designing Accelerators and SoC Interfaces using gem5-Aladdin," in *Intl. Symp. on Microarchitecture (MICRO)*, 2016.

[27] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2011, pp. 365–376.

[28] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "CPU DB: Recording microprocessor history," *Queue*, vol. 10, no. 4, pp. 10:10–10:27, 2012.

[29] "Cpu database." [Online]. Available: https://www.techpowerup.com/cpudb

[30] "Top 500 list," http://www.top500.org.

[31] T. y. Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. K. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, A. Al-Shamma, C. Chen, M. Gupta, G. Hilton, A. Kathuria, V. Lai, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, Y. Yin, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, H. Inoue, and L. Fasoli, "A 130.7-$hboxmm^2$ 2-layer 32-gb reram memory device in 24-nm technology," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 140–153, 2014.

[32] "Techinsights publishes preliminary analysis of 3D xpoint memory." [Online]. Available: https://www.anandtech.com/show/11454/techinsights-publishes-preliminary-analysis-of-3d-xpoint-memory

[33] C. Strachey, "Fundamental concepts in programming languages," *Higher-order and symbolic computation*, vol. 13, no. 1-2, pp. 11–49, 2000.

[34] H. Rito and J. a. Cachopo, "Memoization of methods using software transactional memory to track internal state dependencies," in *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*. New York, NY, USA: ACM, 2010, pp. 89–98.

[35] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, "Nectar: Automatic management of data and computation in datacenters." in *OSDI*, vol. 10, 2010, pp. 1–8.

[36] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 285–296, 2010.

[37] "Usage of Gzip Compression for websites," https://w3techs.com/technologies/details/ce-gzipcompression/all/all.

[38] "Snappy: a fast compressor/decompressor." https://github.com/google/snappy.

[39] J. Ouyang, H. Luo, Z. Wang, J. Tian, C. Liu, and K. Sheng, "FPGA implementation of gzip compression and decompression for idc services," in *Intl. Conference on Field-Programmable Technology (FPT)*, 2010, pp. 265–268.

[40] "Smashing Big Data Costs with GZIP Hardware," http://www.aha.com/Uploads/GZIP_Benefits_Whitepaper15.pdf.

[41] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *Intl. Symp. on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[42] "The Xapian Project," https://xapian.org.

[43] B. Brouwer, "Youtube now gets over 400 hours of content uploaded every minute," *URL: http://www.tubefilter.com/2015/07/26/youtube-400-hours-content-every-minute/, Abruf am*, vol. 15, p. 2016, 2015.

[44] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Intl. Conf. on Parallel Arch. and Compilation Techniques (PACT)*, 2008.

[45] L. Wolf, T. Hassner, and I. Maoz, "Face recognition in unconstrained videos with matched background similarity," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 529–534.

[46] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, "An analysis of accelerator coupling in heterogeneous architectures," in *Proceedings of the 52Nd Annual Design Automation Conference*. New York, NY, USA: ACM, 2015, pp. 202:1–202:6.

[47] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Intl. Symp. on Microarchitecture (MICRO)*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 468–479.

[48] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 97–108.

[49] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 1543–1546.

[50] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Intl. Symp. on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 449–460.

[51] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2013, pp. 24–35.

[52] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross, "Navigating big data with high-throughput, energy-efficient data partitioning," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 249–260, 2013.

[53] J. A. Stratton, C. Rodrigrues, I.-J. Sung, N. Obeid, L. Chang, G. Liu, and W.-M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," University of Illinois at Urbana-Champaign, Tech. Rep. IMPACT-12-01, 2012.

[54] "English Wikipedia dumps: Oct-2016," http://dumps.wikimedia.org/enwiki/20161001/.

[55] "C port of the snappy compressor," https://github.com/andikleen/snappy-c.

[56] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. American Mathematical Soc., 2009, vol. 74.

[57] B. Reagen, R. Adolf, Y. S. Shao, G. Y. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *Intl. Symp. on Workload Characterization (IISWC)*, 2014, pp. 110–119.

[58] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Trans. Web*, vol. 1, no. 1, 2007.

[59] S. Thomas, C. Gohkale, E. Tanuwidjaja, T. Chong, D. Lau, S. Garcia, and M. B. Taylor, "Cortexsuite: A synthetic brain

benchmark suite," in *Intl. Symp. on Workload Characterization (IISWC)*, 2014, pp. 76–79.
[60] J. Bennett, S. Lanning, and N. Netflix, "The netflix prize," in *In KDD Cup and Workshop in conjunction with KDD*, 2007.
[61] G. K. Zipf, "Relative frequency as a determinant of phonetic change," *Harvard Studies in Classical Philology*, vol. 40, pp. 1–95, 1929.
[62] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: Findings and implications," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*. New York, NY, USA: ACM, 2009, pp. 37–48.
[63] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey, "Architecting to achieve a billion requests per second throughput on a single key-value store server platform," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2015, pp. 476–488.
[64] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *ACM Symp. on Cloud Computing (SoCC)*. New York, NY, USA: ACM, 2010, pp. 143–154.
[65] M. K. Papamichael, J. C. Hoe, and O. Mutlu, "FIST: A fast, lightweight, FPGA-friendly packet latency estimator for noc modeling in full-system simulations," in *Intl. Symp. on Networks-on-Chip (NOCS)*. New York, NY, USA: ACM, 2011, pp. 137–144.
[66] M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, J. Balkind, A. Lavrov *et al.*, "Piton: A 25-core academic manycore processor," in *Symp. on High Performance Chips*, 2016.
[67] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 27–39.
[68] L. Zhang, B. Neely, D. Franklin, D. Strukov, Y. Xie, and F. T. Chong, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *Intl. Symp. on Computer Architecture (ISCA)*, 2016, pp. 519–531.
[69] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2016, pp. 1–13.
[70] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Intl. Symp. on Microarchitecture (MICRO)*. New York, NY, USA: ACM, 2013, pp. 25–36.
[71] J. Zhan, O. Kayiran, G. H. Loh, C. R. Das, and Y. Xie, "Oscar: Orchestrating STT-RAM cache traffic for heterogeneous CPU-GPU architectures," in *Intl. Symp. on Microarchitecture (MICRO)*, 2016, pp. 1–13.
[72] C. Zhang, G. Sun, X. Zhang, W. Zhang, W. Zhao, T. Wang *et al.*, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2015, pp. 694–706.
[73] M. Huang, D. Wu, C. H. Yu, Z. Fang, M. Interlandi, T. Condie, and J. Cong, "Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale," in *ACM Symp. on Cloud Computing (SoCC)*. New York, NY, USA: ACM, 2016, pp. 456–469.
[74] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, 2011.
[75] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *Intl. Symp. on Microarchitecture (MICRO)*. IEEE Computer Society, 2016.
[76] S. E. Richardson, "Caching function results: Faster arithmetic by avoiding unnecessary computation," Mountain View, CA, USA, Tech. Rep., 1992.
[77] A. Sodani and G. S. Sohi, "Dynamic instruction reuse," in *Intl. Symp. on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 1997, pp. 194–205.
[78] D. Citron, D. Feitelson, and L. Rudolph, "Accelerating multimedia processing by implementing memoing in multiplication and division units," *SIGOPS Oper. Syst. Rev.*, vol. 32, no. 5, pp. 252–261, 1998.
[79] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 922–927, 2005.
[80] J. Huang and D. J. Lilja, "Exploiting basic block value locality with block reuse," in *Symp. on High-Performance Computer Architecture (HPCA)*, 1999, pp. 106–114.
[81] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "Mirage cores: The illusion of many out-of-order cores using in-order hardware," in *Intl. Symp. on Microarchitecture (MICRO)*. New York, NY, USA: ACM, 2017, pp. 745–758.
[82] D. A. Connors and W.-m. W. Hwu, "Compiler-directed dynamic computation reuse: Rationale and initial results," in *Intl. Symp. on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 158–169.
[83] A. Suresh, B. N. Swamy, E. Rohou, and A. Seznec, "Intercepting functions for memoization: A case study using transcendental functions," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 2, pp. 18:18:1–18:18:23, 2015.
[84] V. Agrawal, A. Dabral, T. Palit, Y. Shen, and M. Ferdman, "Architectural support for dynamic linking," in *Intl. Conf. on Arch. Support for Programming Languages & Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2015, pp. 691–702.
[85] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis, "Eliminating redundant fragment shader executions on a mobile gpu via hardware memoization," in *Intl. Symp. on Computer Architecture (ISCA)*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 529–540.
[86] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *Intl. Conf. on Arch. Support for Programming Languages & Operating Systems (ASPLOS)*. New York, NY, USA: ACM, 2014, pp. 35–50.
[87] M. Sutherland, J. San Miguel, and N. E. Jerger, "Texture cache approximation on GPUs," in *Workshop on Approximate Computing Across the Stack*, 2015.
[88] M. Imani, D. Peroni, Y. Kim, A. Rahimi, and T. Rosing, "Efficient neural network acceleration on gpgpu using content addressable memory," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017, pp. 1026–1031.
[89] P. Wang, G. Sun, T. Wang, Y. Xie, and J. Cong, "Designing scratchpad memory architecture with emerging STT-RAM memory technologies," in *IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, 2013, pp. 1244–1247.
[90] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 541–552.
[91] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Intl. Symp. on Computer Architecture (ISCA)*, 2016, pp. 14–26.
[92] 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, pp. 1449–1452.
[93] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li *et al.*, "Scaling memcache at facebook," in *USENIX Conference on Networked Systems Design and Implementation (NDSI)*. Berkeley, CA, USA: USENIX Association, 2013, pp. 385–398.
[94] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, 1995.
[95] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.
[96] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Intl. Symp. on Computer Architecture (ISCA)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 453–464.
[97] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Intl. Symp. on Microarchitecture (MICRO)*. New York, NY, USA: ACM, 2015, pp. 546–558.