2018 IEEE International Parallel and Distributed Processing Symposium

Application Codesign of Near-Data Processing for Similarity Search

Vincent T. Lee, Amrita Mazumdar, Carlo C. del Mundo, Armin Alaghi, Luis Ceze, Mark Oskin
University of Washington
{vlee2, amrita, cdel, armin, luisceze, oskin}@cs.washington.edu

***Abstract*—Similarity search is key to a variety of applications including content-based search for images and video, recommendation systems, data deduplication, natural language processing, computer vision, databases, computational biology, and computer graphics. At its core, similarity search manifests as k-nearest neighbors (kNN), a computationally simple primitive consisting of highly parallel distance calculations and a global top-k sort. However, kNN is poorly supported by today's architectures because of its high memory bandwidth requirements.**

This paper proposes an application codesign of a near-data processing accelerator for similarity search: the Similarity Search Associative Memory (SSAM). By instantiating compute units close to memory, SSAM benefits from the higher memory bandwidth and density exposed by emerging memory technologies. We evaluate the SSAM design down to layout on top of the Micron hybrid memory cube (HMC), and show that SSAM can achieve up to two orders of magnitude area-normalized throughput and energy efficiency improvement over multicore CPUs. We also show SSAM has higher throughput and is more energy efficient than competing GPUs and FPGAs.

***Keywords*-similarity search, k-nearest-neighbors, near-data processing**

I. Introduction

Similarity search is a key computational primitive found in a wide range of applications, such as computer graphics [1], image and video retrieval [2], [3], data mining [4], and computer vision [5]. While much attention has been directed towards accelerating feature extraction techniques like convolutional neural networks, there has been relatively little work focused on accelerating the task that follows: taking the resulting feature vectors and searching the vast corpus of data for similar content. In recent years, the importance and ubiquity of similarity search has increased dramatically with the explosive growth of visual content: users shared over 260 billion images on Facebook in 2010 [6], and uploaded over 300 hours of video on YouTube every minute in 2014 [7]. This volume of visual data is only expected to continue growing exponentially [8], and has motivated many new search-based graphics and vision applications.

Similarity search manifests as a simple algorithm: k-nearest neighbors (kNN). At a high level, kNN is an approximate associative computation which tries to find the most similar content with respect to the query content. At its core, kNN consists of many parallelizable distance calculations and a single global top-k sort, and is often supplemented with indexing techniques to reduce the volume of data that must be processed. While computationally simple, kNN is notoriously memory intensive on modern CPUs and heterogeneous computing substrates making it challenging to scale to large datasets. In kNN, distance calculations are cheap and abundantly parallelizable across the dataset, but moving data from memory to the computing device is a significant bottleneck. Moreover, this data is used only once per kNN query and discarded since the result of a kNN query is only a small set of identifiers. Batching requests to amortize this data movement has limited benefits as time-sensitive applications have stringent latency budgets. Indexing techniques such as kd-trees [9], hierarchical k-means clustering [10], and locality sensitive hashing [11] are often employed to reduce the search space but trade reduced search accuracy for enhanced throughput. Indexing techniques also suffer from the *curse of dimensionality* [12]. In the context of kNN, this means indexing structures effectively degrade to linear search for increasing accuracy targets.

Because of its significance, generality, parallelism, underlying simplicity, and small result set, kNN is an ideal candidate for near-data processing. The key insight is that a small accelerator can reduce the traditional bottlenecks of kNN by applying orders of magnitude data reduction near memory, substantially reducing the need for data movement. While there have been many attempts at processing-in-memory (PIM) in the past [13]–[15], much of prior work suffered from DRAM technology limitations. Logic created in DRAM processes was too slow, while DRAM implemented in logic processes suffered from poor retention and high power demands; attempts at hybrid processes [16] resulted in the worst of both. PIM architectures are more appealing today with the advent of die-stacked memory technology which enables the co-existence of an efficient DRAM layer *and* efficient logic layer [17].

We propose Similarity Search Associative Memory (SSAM) which integrates a programmable accelerator on top of a Hybrid Memory Cube (HMC) which is a recent high bandwidth, die-stacked memory module. Semantically, a SSAM takes a query as input and returns the top-k closest neighbors stored in memory as output. We evaluate the performance and energy efficiency gains of SSAM by implementing, synthesizing, and simulating the design down to layout. We then compare SSAM against current CPUs, GPUs, and FPGAs, and show that it achieves better area-normalized throughput and energy efficiency.

Our paper makes the following contributions: (1) A characterization of state-of-the-art k-nearest neighbors including

1530-2075/18/$31.00 ©2018 IEEE
DOI 10.1109/IPDPS.2018.00099

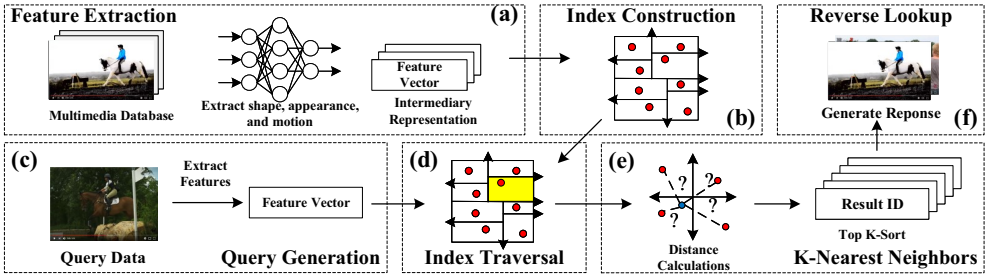


Fig. 1: Software application pipeline for similarity search: (a) feature extraction, (b) feature indexing, (c), query generation, (d) index traversal (e) k-nearest neighbor search, (f) reverse lookup. Feature extraction and indexing is done offline.

both application-level and architectural opportunities that justify acceleration. (2) An application-driven codesign of a near-memory, vector processor accelerator architecture with hardware support for similarity search on top of Hybrid Memory Cube (HMC). (3) Instruction extensions to leverage hardware units to accelerate similarity search.

The rest of the paper is organized as follows. Section II introduces and characterizes the kNN algorithm. Section III describes the SSAM architecture and the hardware/software interface. Section IV outlines evaluation methodology, and Section V presents results. Section VI discusses the cost of specialization, SSAM's utility beyond kNN, and comparison with other near-data processing technologies. Section VII discusses related work.

II. Characterization of kNN

This section characterizes the kNN algorithm pipeline and indexing techniques, and highlights the application-level and architectural opportunities for acceleration.

A. Case Study: Content-based Search

A typical kNN software application pipeline for content-based search (Fig. 1) has five stages: feature extraction, feature indexing, query generation, k-nearest neighbors search, and reverse lookup. In *feature extraction* (Fig. 1a), the raw multimedia corpus is converted into an intermediary feature vector representation. Feature vectors may represent pixel trajectories in a video, word embeddings of a document [18], or shapes in an image [19], and are extracted using feature descriptors or convolutional neural networks [20]. While feature extraction is important, it only needs to be performed once per dataset and can be done offline; prior work has also shown feature extraction can be achieved efficiently [21], [22]. In *indexing* (Fig. 1b), feature vectors from feature extraction are organized into data structures (discussed in Section II-C). At query time, these data structures are used to quickly prune the search space; intuitively, these data structures should be able to reduce the search time from linear to logarithmic in the size of the data. Indexing can also be performed offline and away from the critical path of the query.

While feature extraction and indexing can be performed offline, the *query generation* stage (Fig. 1c) of the search pipeline occurs online. In query generation, a user uploads a multimedia file (image, video, etc.) and requests similar content back. The query runs through the same feature extractor used to create the database before being passed to the search phase. Once a query is generated, the search traverses the indexing structures to prune away portions of the search space (Fig. 1d). Indexing structures reduce the search space size but trade accuracy for performance. The *k-nearest neighbors* stage (Fig. 1e) then attempts to search through the remaining database candidates for the most similar content in the database. The kNN algorithm consists of many highly parallelizable distance calculations and a global top-k sort. The similarity metric employed by the distance calculation often depends on the application, but common distance metrics include Euclidean distance, Hamming distance [23], cosine similarity, and learned distance metrics [24]. The final step in the pipeline is *reverse lookup* (Fig. 1f) where the resulting k nearest neighbors are mapped to their original database content. The resulting media is then returned to the user.

B. Typical Workload Parameters

Prior work shows the feature dimensionality for descriptors such as word embeddings [25], GIST descriptors [19], and AlexNet [20] ranges from 100 to 4096 dimensions. For higher dimensional feature vectors, it is common to apply techniques such as principal component analysis to reduce feature dimensionality to tractable lengths [26]. The number of nearest neighbors k for search applications has been shown to range from 1 (nearest neighbor) up to 20. Each kNN algorithm variant also has a number of additional parameters such as indexing technique, distance function, bucket size, index-specific hyperparameters, and hardware specific optimizations.

To simplify the characterization, we limit our initial evaluation to Euclidean distance and three real world datasets: the Global Vectors for Word Representations (GloVe) dataset [25], the GIST dataset [27], and the AlexNet dataset [20]. The GloVe dataset consists of 1.2 million word embeddings extracted from Twitter tweets and the GIST dataset consists of 1 million GIST feature vectors extracted from images. We constructed the AlexNet dataset by taking 1 million images from the Flickr dataset [28] and applying AlexNet [20] to extract the feature vectors. For each dataset, we separate a "training" set to build the search index, and a "test" set of 1000 vectors used as the queries when measuring application accuracy. We set the number of neighbors to 6, 10, and 16 for GloVe, GIST, and AlexNet respectively.

C. Approximate kNN Algorithms Tradeoffs

We characterize three canonical indexing techniques employed by approximate kNN algorithms: kd-trees, hierarchical k-means, and multi-probe locality sensitive hashing (MPLSH). Indexing techniques employ hierarchical data structures which are traversed at query time to prune the search space. In kd-trees, the index is constructed by randomly cutting the dataset by the N vector dimensions with highest variance [29]. The resulting index is a tree data structure where each leaf in the tree contains a *bucket* of similar vectors and the depth of the bucket depends on how tall the tree is limited to be. Queries which traverse the index and end up in the same bucket should be similar; multiple trees are often used in parallel with different cut orders. Multiple leaves in the tree can be visited to improve the quality of the search. To do this, the traversal employs *backtracking* to check additional "close by" buckets in a depth first search-like fashion. A user-specified bound typically limits the number of additional buckets visited when backtracking.

Similarly, in hierarchical k-means the dataset is partitioned recursively based on k-means cluster assignments to form a tree data structure [10]. Like kd-tree indices, the height of the tree is restricted, and each leaf in the tree holds a bucket of similar vectors which are searched when a query reaches that bucket. Backtracking is also used to expand the search space and search "close by" buckets.

Finally, MPLSH constructs a set of hash tables where each hash location is associated with a bucket of similar vectors [30]. In MPLSH, hash functions are designed to intentionally cause hash collisions to map similar vectors to the same bucket. To improve accuracy, MPLSH applies small perturbations to the hash result to create additional probes into the same hash table to search "close by" hash partitions. In our evaluation, we use hyperplane MPLSH (HP-MPLSH) which cuts the space into random hyperplanes and set the number of hash bits or hyperplane cuts to 20.

Each of these approximate kNN algorithms trade accuracy for enhanced throughput. In kNN, accuracy is defined as $S_E \cap S_A/|S_E|$, where S_E is the true set of neighbors returned by exact floating point linear kNN search, and S_A is the set of neighbors returned by approximate kNN. In general, searching more of the dataset improves search accuracy for indexing techniques. To quantify the accuracy of indexing structures, we benchmark the accuracy and throughput of indexing techniques for the GloVe, GIST, and AlexNet datasets. We use the Fast Library for Approximate Nearest Neighbors (FLANN) [10] to benchmark kd-trees and hierarchical k-means, and Fast Lookups for Cosine and Other Nearest Neighbors Library (FALCONN) [31] to benchmark HP-MPLSH. For kd-trees and hierarchical k-means we vary the number of leaf nodes or buckets in the tree that backtracking will check, while for HP-MPLSH we increase the number of probes used per hash table. Each of these modifications effectively increases the fraction of the dataset searched per query and lowers overall throughput.

The resulting throughput versus accuracy curves are shown in Fig. 2, for single threaded implementations. In general, our results show indexing techniques can provide up to 170× throughput improvement over linear search while still maintaining at least 50% search accuracy, but only up to 13× in order to achieve 90% accuracy. Past 95-99% accuracy, we find that *indexing techniques effectively degrade to linear search* (blue solid line). More importantly, our results show there is a significant opportunity for also accelerating approximate kNN techniques. Hardware acceleration of approximate kNN search can *either increase throughput at iso-accuracy* by simply speeding up the computation or *increase search accuracy at iso-latency* by searching larger volumes of data.

D. Alternative Representations and Distance Metrics

We now discuss the impact of numerical representation and distance metrics for kNN.

Fixed-Point Representations: Fixed-point arithmetic is much cheaper to implement in hardware than floating point units. To evaluate whether floating point is necessary for kNN, we converted each dataset to a 32-bit fixed-point representation and repeated the throughput versus accuracy experiments. Overall, we find *there is negligible accuracy loss* between 32-bit floating-point and 32-bit fixed-point data representations.

Hamming Space Representations: Recent work has shown that Hamming codes can be an effective alternative for Euclidean space representations [23], [32]. Binarization techniques trade accuracy for higher throughput since precision is lost by binarizing floating point values but increases throughput since the dataset is smaller. Binarization also enables Hamming distance calculations which are cheaper to implement in hardware. In practice, carefully constructed Hamming codes have been shown to achieve excellent results [32].

Alternative Distance Metrics: While the canonical distance metric for kNN is the Euclidean distance, there are a number of alternative distance metrics. Such alternative metrics include Manhattan distance, cosine similarity, Chi squared distance, Jaccard similarity, and learned distance metrics [24].

E. Architectural Characterization

To more concretely quantify the architectural behaviors of kNN variants, we instrumented the baselines presented earlier using the Pin [33] instruction mix tool on an Intel i7-4790K CPU. Table I shows the instruction profile for linear, kd-tree, k-means, and MPLSH based algorithms respectively. Recall that linear search performance is still valuable since higher accuracy targets reduce to linear search. In addition, approximate algorithms still use linear search to scan buckets of vectors at the end of their traversals. As expected, the instruction profile shows that vector operations and extensions are important for kNN workloads due to the many vector-parallel distance calculations. In addition, the high percentage of memory reads confirms that the computation has high data movement demands. Approximate kNN techniques like KD-trees and MPLSH exhibit less skew towards vectorized

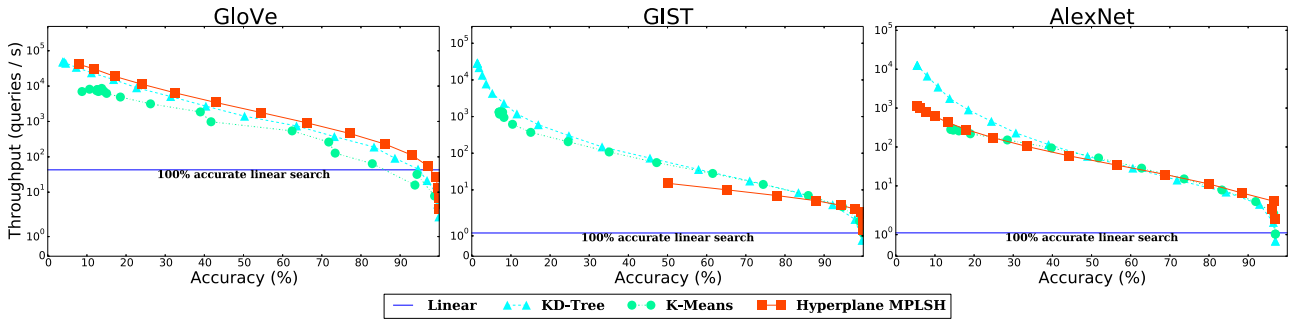


Fig. 2: Approximate kNN algorithms tradeoff accuracy for throughput (up and to the right is better).

TABLE I: Instruction mix profiles of kNN algorithms for GloVe dataset.

Algorithm	AVX/SSE Inst. (%)	Mem. Reads (%)	Mem. Writes (%)
Linear	54.75	45.23	0.44
KD-Tree	28.75	31.60	10.21
K-Means	51.63	44.96	1.12
MPLSH	18.69	31.53	14.16

instructions but still exhibit similar memory intensive behavior and show vectorization is valuable.

III. SSAM Architecture

Based on the characterization results in Section II, it is clear that similarity search algorithms (1) are an ideal match for vectorized processing units, and (2) can benefit from higher memory bandwidth to support data-intensive execution phases. We now present our SSAM module and accelerator architecture which exploits near-data processing and specialized vector compute units to address these bottlenecks.

A. System Integration and Software Interface

SSAM is a memory module that integrates into a typical system similar to how existing DRAM are integrated as shown in Fig. 3. A host processor interfaces with a SSAM module similar to how it interacts with a DRAM memory module. The host processor is connected to each SSAM module over a communication bus; additional communication links are used if multiple SSAM-enabled modules are instantiated. Since HMC modules can be composed together, these additional links and SSAM modules allows us to scale up the capacity of the system. A typical system may also have multiple host processors (not shown) as the number of SSAM modules that the system must maintain increases.

To abstract the lower level details of SSAMs away from the programmer, we assume a driver stack exposes a minimal memory allocation API which manages user interaction with SSAM-enabled memory regions. A SSAM-enabled memory region is defined as a special part of the memory space which is physically backed by a SSAM instead of a standard DRAM module. A sample programming interface of how one would use SSAM-enabled memory regions is shown in Fig. 4. SSAM-enabled memory regions would be tracked and stored in a free list similar to how standard memory allocation is implemented in modern systems. Allocated SSAM memory

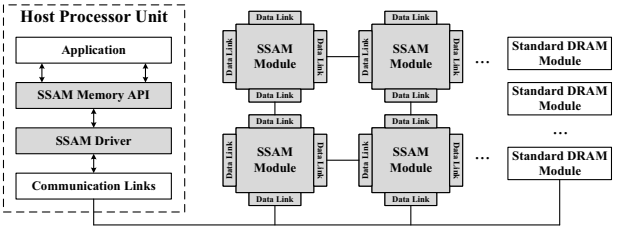


Fig. 3: SSAM system integration (modifications in grey). SSAM modules replace or coexist with standard DRAM modules.

```
// Example program using SSAM
int * knn(int * query, int *dataset,
    size_t length, size_t dims, int k) {
  //allocate buffer of SSAM memory
  int * nbuf = nmalloc(length * dims);
  nmode(nbuf, LINEAR);
  nmemcpy(nbuf, dataset, length * dims
       * sizeof(int));
  nbuild_index(nbuf, params = NULL);
  nwrite_query(nbuf, query);
  //execute kNN search
  nexec(nbuf);
  int * result = nread_result(nbuf);
  nfree(nbuf);
  return result;
}
```

Fig. 4: Example program using SSAM-enabled memory regions. Lower level hardware configuration details are abstracted away from the programmer.

regions come with a set of special operations that allow the user to set the indexing mode, in additional to handling standard memory manipulation operations like `memcpy`. Similar to the CUDA programming model, we propose analogous memory and execution operations to use SSAM-enabled memory. Pages with data subject to SSAM queries are pinned (not subject to swapping by the OS).

B. SSAM Architecture and Hybrid Memory Cube

The SSAM architecture is built on top of a Hybrid Memory Cube 2.0 (HMC) [34] to capitalize on enhanced memory

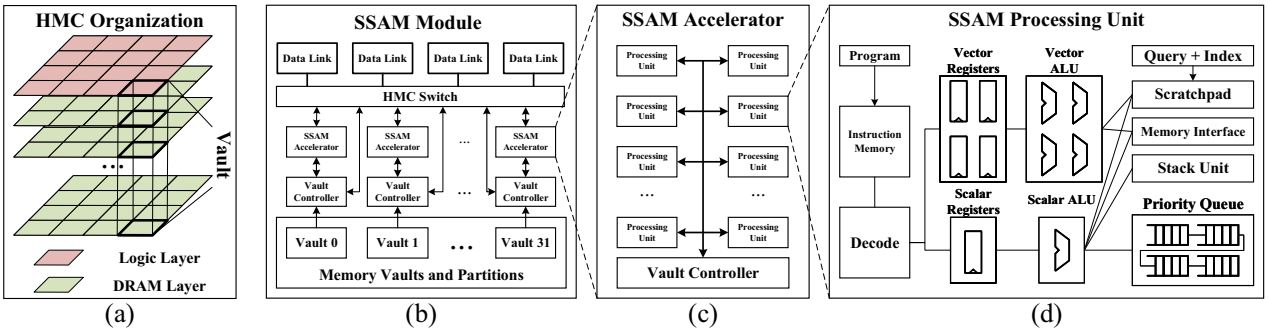


Fig. 5: SSAM accelerator architecture. (a) HMC die organization (only 16 vaults shown, HMC 2.0 has 32), (b) SSAM logic layer organization, (c) SSAM accelerator organization, (d) processing unit microarchitecture.

bandwidth. The HMC is a die-stacked memory architecture composed of multiple DRAM layers and a compute layer. The DRAM layers are vertically partitioned into a number of *vaults* (Fig. 5a). Vaults are each accessed via a vault controller which reside on a top-level compute layer. In HMC 2.0, the module is partitioned into a maximum of 32 vaults (only 16 shown), where each vault controller operates at 10 GB/s yielding an aggregate internal memory bandwidth of 320 GB/s. The HMC architecture also is composed of four external data links (240 GB/s external bandwidth) which send and receive information to the host processor or other HMC modules. These external data links allow one or more HMC modules to be composed to effectively form a larger network of SSAMs if data exceeds the capacity of a single SSAM module.

Our SSAM architecture leverages the existing HMC substrate and introduces a number of SSAM accelerators to handle the kNN search. These SSAM accelerators are instantiated on the compute layer next to existing vault controllers as shown in Fig. 5b. SSAM accelerators are further decomposed into processing units (Fig. 5d). To fully harness the bandwidth available, we replicate processing units to fully use the memory bandwidth by measuring the peak bandwidth needs of each processing unit across all indexing techniques. For kNN, we expect to achieve near optimal memory bandwidth since most data accesses to memory are large contiguously allocated blocks such as bucket scans and data structures. Our modifications are made orthogonal to the functionality of the HMC control logic so that the HMC module can still operate as a standard memory module (i.e. acceleration logic can be bypassed). Our processing units do not implement a full cache hierarchy since there is little data reuse outside of the query vector and indexing data structure. Unlike GPUs cores, processing units are not restricted to operating in lockstep and multiple different indexing kernels can coexist on each SSAM module. Finally, we do not expect external data links to become a bottleneck as a vast majority of the data movement occurs within SSAM modules themselves. As a result, we only expect the communication network between the host processors and SSAM units to consist of kNN results which are a fraction of the original dataset size and configuration data.

C. Processing Unit Architecture

Each processing unit consists of a fully integrated scalar and vector processing unit similar to [35] but are augmented with several instructions and hardware units to better support kNN. Fully-integrated vector processing units are naturally well-suited for accelerating kNN distance calculations because they are (1) able to exploit the abundant data parallelism in kNN and (2) well-suited for streaming computations. Using vector processing units also introduces flexibility in the types of distance calculations that can be executed. The scalar unit is better suited for executing index traversals which are sequential in nature, and provides flexibility in the types of indexing techniques that can be employed. Vector units on the other hand are better suited for high throughput, data parallel distance calculations in kNN. We use a single instruction stream to drive both the scalar and vector processing units since at any given time a processing unit will only be performing either distance calculations or index traversals in kNN. For our evaluation, we perform a design sweep over several different vector lengths: 2, 4, 8, and 16. We find that 32 scalar registers, and 8 vector registers are sufficient to support our kNN workloads. Finally, we use forwarding paths between pipeline stages to implement chaining of vector operations.

We also integrate several hardware units that are useful for accelerating similarity search. First, we introduce a *priority queue unit*, implemented using the shift register architecture proposed in [36], and use it to perform the sort and global top-k calculations. For our SSAM design, priority queues are 16 entries deep. We opt to provide a hardware priority queue instead of a software one since the overhead of a priority queue insert becomes non-trivial for shorter vectors. Because of its modular design, the priority queues can be chained to support larger k values. Likewise, priority queues in the chain can also be disabled if they are not needed. Second, we introduce a small hardware stack unit instantiated on the scalar datapath to aid kNN index traversals. The stack unit is a natural choice to facilitate backtracking when traversing hierarchical index structures. Finally, we integrate a 32 KB scratchpad to hold frequently accessed data structures, such as the query vector and indexing structures. We find that a modestly sized scratchpad memory is sufficient for kNN since the only heavily reused

TABLE II: Processing unit instruction set. (S/V) are scalar and vector instructions. (S) instructions are scalar only.

Type	Instruction
Arithmetic (S/V)	ADD, SUB, MULT, POPCOUNT, ADDI, SUBI, MULTI
Bitwise/Shift (S/V)	OR, AND, NOT, XOR, ANDI, ORI, XORI, SR, SL, SRA
Control (S)	BNE, BGT, BLT, BE, J
Stack Unit (S)	POP, PUSH
Register Move/Memory Instructions (S/V)	SVMOVE, VSMOVE, MEM_FETCH, LOAD, LOAD, STORE
New SSAM Instructions	**(S)PQUEUE_INSERT, (S)PQUEUE_LOAD, (S)PQUEUE_RESET, (S/V)FXP**

data are the query vectors and indices (data vectors are scanned and immediately discarded).

Unlike conventional scalar-vector architectures, we introduce several new instructions to exercise new hardware units for similarity search. The full instruction set is shown in Table **??**. First, we introduce priority queue insert (PQUEUE_INSERT), load (PQUEUE_LOAD), and reset (PQUEUE_RESET) instructions which are used to manipulate the hardware priority queue. The PQUEUE_INSERT instruction takes two registers and inserts them into the hardware priority queue as an (id, value) tuple. The PQUEUE_LOAD instruction reads either the id or value of a tuple in the priority queue at a designated queue position, while the PQUEUE_RESET clears the priority queue. We also introduce a scalar and vector 32-bit *fused xor-population count* instruction (SFXP and VFXP) which is similar to a fused multiply-add instruction. The FXP instruction is useful for cheaply implementing Hamming distance calculations and assumes that each 32-bit word is 32 dimensions of a binary vector. The FXP instruction is also cheap to implement in hardware since the XOR only adds one additional layer of logic to the population count hardware. Finally, we introduce a data prefetch instruction MEM_FETCH since the linear scans through buckets of vectors exhibit predictable contiguous memory access patterns.

D. SSAM Configuration

We assume that the host processor driver stack is able to communicate with each SSAM to initialize and bring up SSAM devices using a special address region dedicated to operating SSAMs. Execution binaries are written to instruction memories on each processing unit and can be recompiled to support different distance metrics, indexing techniques, and kNN parameters. In addition, any indexing data structures are also written to the scratchpad memory or larger DRAM prior to executing any queries on SSAMs. Any large data structures such as hash function weights in MPLSH or centroids in hierarchical k-means are stored in SSAM memory since they are larger and experience limited reuse. If hierarchical indexing structures such as kd-trees or hierarchical k-means do not fit in the scratchpad, they are partitioned such that the top half of the hierarchy resides in scratchpad, and the bottom halves are dynamically loaded to the scratchpad from DRAM as needed during execution. A small portion of the scratchpad is also allocated for holding the query vector; this region is continuously rewritten as a SSAM services queries. If a kNN query must touch multiple vaults, the host processor broadcasts the search across SSAM processing units and performs the final set of global top-k reductions on the host processor. Finally, if SSAM capabilities are not needed, the host processor can disable the SSAM accelerator logic so that it operates simply as a standard memory.

IV. Evaluation Methodology

This section outlines evaluation methodology to compare and contrast SSAMs with competing CPUs, GPUs, and FPGAs. To provide fair energy efficiency and performance measurements, we normalize each platform to a 28 nm technology process.

SSAM ASIC: To evaluate SSAM, we implemented, synthesized, and place-and-routed our design in Verilog with the Synopsys Design Compiler and IC Compiler using a TSMC 65 nm standard cell library. SRAM memories were generated using the ARM Memory Compiler. We also built an assembler and simulator to generate program binaries, benchmark assembly programs, and validate the correctness of our design. To measure throughput, we use post-placement and route frequency estimates and simulate the time it takes to process each of the workloads in Section II-B. Each benchmark is handwritten using our instruction set defined in Table II. For power and energy efficiency estimates, we generate traces from real datasets to measure realistic activity factors. We then use the PrimeTime power analyzer to estimate power and multiply by the simulated run time to obtain energy efficiency estimates. Finally, we report the area estimates provided in the post-placement and route reports normalized to a 28 nm technology.

Xeon E5-2620 CPU: We evaluate a six core Xeon E5-2620 as our CPU baseline. For each platform, we benchmark wall-clock time using the implementations of kNN provided by the FLANN library [10] for linear, kd-tree, and k-means based search, and the FALCONN library for hyperplane MPLSH [31]. For power and energy efficiency measurements, we use an external power meter to measure dynamic compute power. Dynamic compute power is measured by taking the difference between the load and idle power when running each benchmark. Energy efficiency is then calculated as the product of the run time and dynamic power. Estimates of the CPU die size is taken from [37].

Titan X GPU: For our GPU comparison, we use an NVIDIA Titan X Geforce GPU using a well-optimized, off-the-shelf implementation provided by Garcia et al. [38]. We again record wall-clock time, and measure idle and load power using a power meter to measure run time and energy efficiency. We estimate the die size of the Titan X from [39].

TABLE III: SSAM accelerator power (uW) by module.

Module	Priority Queue	Stack Unit	ALUs	Scratchpad	Reg. Files	Ins. Memory	Pipeline/ Control	Total
SSAM-2	1.63	1.02	0.33	1.92	2.52	0.45	2.28	8.52
SSAM-4	1.56	1.00	0.32	2.16	3.24	0.44	2.82	9.98
SSAM-8	1.42	1.02	0.32	2.58	4.68	0.44	4.28	13.32
SSAM-16	1.45	0.84	0.51	3.80	6.97	0.41	7.09	19.62

TABLE IV: SSAM accelerator area (mm^2) by module

Module	Priority Queue	Stack Unit	ALUs	Scratchpad	Reg. Files	Ins. Memory	Pipeline/ Control	Total
SSAM-2	1.07	0.52	1.20	20.70	1.35	4.76	0.92	30.52
SSAM-4	1.06	0.52	1.65	27.28	1.78	4.76	1.29	38.34
SSAM-8	1.04	0.51	3.55	43.53	2.64	4.76	2.18	58.21
SSAM-16	1.04	0.51	6.79	76.26	4.33	4.76	3.79	97.48

Kintex-7 FPGA: We measure the performance and energy efficiency of our implementation on a Xilinx Kintex-7 FPGA using Vivado 2014.5. We use post-placement and route frequency estimates and simulated run times to estimate the throughput of kNN on the FPGA fabric. For power measurements, we use the Vivado Power Analyzer tool and refer to [40] for device area estimates.

V. Evaluation Results

This section presents evaluation results for SSAM. For brevity, we evaluate Euclidean distance kNN, then separately evaluate different indexing techniques and distance metrics.

A. Accelerator Power and Area

Our post-placement and route power and area results are shown in Table III and Table IV respectively for different processing unit vector lengths and different submodules in the design. Area and power measurements are normalized to 28 nm technology using linear scaling factors. In terms of area, a large portion of the accelerator design is devoted to the SRAMs composing the scratchpad memory. However, relative to the CPU or GPU, the SSAM acceleration logic is still significantly smaller. Compared to the Xeon E5-2620, a SSAM is 6.23-15.62× smaller, while, compared to the Titan X a SSAM is 9.84-24.66× smaller. For comparison, the die size for HMC 1.0 in a 90 nm process is 729 mm^2 [17]; normalized to a 28 nm process, the die size would be $\approx$ 70.6 mm^2 which is roughly the same or larger than our SSAM accelerator design[1]. In terms of power, SSAM logic uses no more than a typical memory module which makes it compatible with the power consumption of die stacked memories. Prior work by Puttaswamy et al. [41] shows temperature increases from integrating logic on die-stacked memory are not fatal to the design even for a general purpose core. Since SSAM consumes less power than general purpose cores, we do not expect thermal issues to be fatal.

[1]Die size for HMC 2.1 are not publicly available.

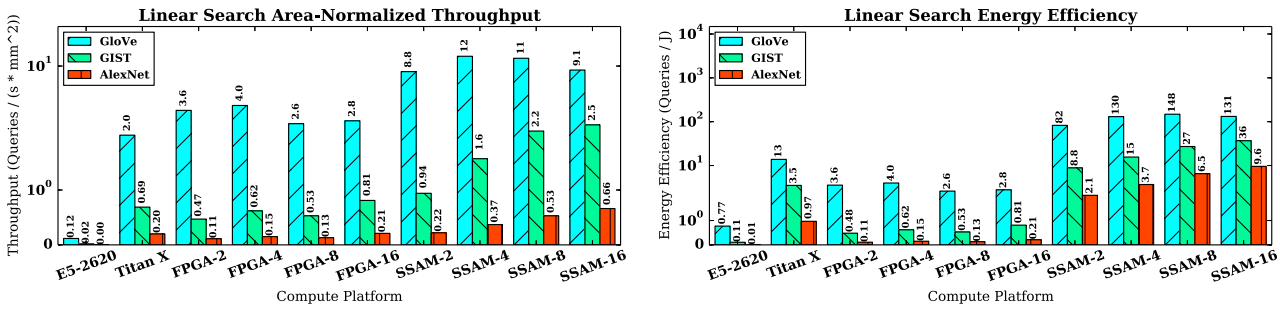
B. Throughput and Energy Efficiency

We now report area-normalized throughput and energy efficiency gains across each platform for exact linear search, which is agnostic to dataset composition and index traversal overheads. This quantifies the gains attributed to different heterogeneous computing technologies. Fig.s 6a and 6b show the area-normalized throughput and energy efficiency of a SSAM against competing heterogeneous solutions. The FPGA and SSAM designs are suffixed by the design vector length; for instance, SSAM-4 refers to a SSAM design with processing units that have vector length 4. We observe SSAM achieves area-normalized throughput improvements of up to 426×, and energy efficiency gains of up to 934× over multi-threaded Xeon E5-2620 CPU results. We also observe that GPUs and the FPGA implementation of the SSAM acceleration logic exhibit comparable throughput and energy efficiency. The FPGA in some cases underperforms the GPU since it effectively implements a soft vector core instead of a fixed-function unit; we expect that a fixed-function FPGA core would fare better.

In terms of the enhanced bandwidth, we attribute roughly one order of magnitude run time improvement to the higher internal bandwidth of HMC 2.0. Optimistically, standard DRAM modules provide up to 25 GB/s of memory bandwidth whereas HMC 2.0 provides 320 GB/s. For similarity search, the difference in available bandwidth directly translates to raw performance. The remaining gains in energy efficiency and performance can be attributed mostly to accelerator specialization. To quantify the impact of the priority queue, we simulate the performance of SSAM using a software priority queue instead of leveraging the hardware queue. At a high level, the hardware queue improves performance by up to 9.2% for wider vector processing units.

C. Approximate kNN Search

We now evaluate the impact of approximate indexing structures and specialization on throughput and energy efficiency. Fig. 7 compares the throughput versus accuracy curves for a SSAM and Xeon E5-2620 CPU for each dataset. In general, at a 50% accuracy target we observe up to two orders of magnitude throughput improvement for kd-tree, k-means, and HP-MPLSH over CPU baselines. The kd-tree and k-means indexing structures are still dominated by distance calculations and benefit greatly from augmented bandwidth when sequentially scanning through buckets for neighbors. HP-MPLSH on the other hand is composed of a combination of many hash function calculations and bucket traversals; we find that for the parameter sets used in our characterization, the performance of HP-MPLSH is dominated mostly by hashing rate. However, the parameters for HP-MPLSH can be adjusted to reduce the dependence on hash performance by reducing the number of hash bits; this would increase the number of vectors hashed to the same bucket and shift the performance bottleneck from hashing performance back to linear bucket scans.



(a) Area-normalized throughput

(b) Energy efficiency

Fig. 6: Area-normalized throughput and energy efficiency for exact linear search using Euclidean distance (higher is better).

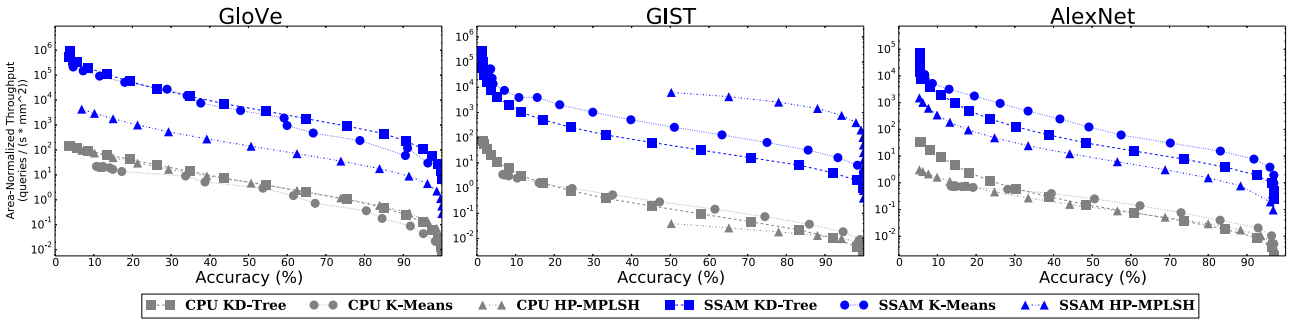


Fig. 7: Area-normalized throughput versus accuracy for Euclidean distance kNN using indexing structures (up and to the right is better).

TABLE V: Relative throughput of alternate distance metrics over Euclidean distance.

Distance Metric	GloVe	GIST	AlexNet
Euclidean	1×	1×	1×
Hamming	4.38×	7.98×	9.38×
Cosine similarity	0.46×	0.47×	0.47×
Manhattan	0.94×	0.99×	0.99×

D. Alternative Distance Metrics

We now briefly quantify the performance of alternative distance metrics on SSAM for three additional distance metrics: Hamming distance, Cosine similarity, and Manhattan distance. Unsurprisingly, binarizing data vectors and using Hamming distance provides good throughput improvement (up to 9.38×), since less data must be loaded to process a vector and Hamming distances using the `FXP` instruction on SSAMs are cheap. Manhattan distance and Euclidean distances have the roughly same throughput since they require similar numbers of operations. On the other hand, cosine similarity[2] is about twice as expensive as Euclidean distance because of the additional numerator and divisor terms. Fixed-point division for cosine similarity is performed in software using shifts and subtracts, however the software division is still much cheaper than the rest of the distance calculation.

[2]Cosine similarity is defined as $(\sum_i a_i b_i)^2/(\sum_i a_i^2 \sum_i b_i^2)$.

VI. Discussion

This section evaluates the cost of specialization, the generality of SSAM for other applications, and contrasts the SSAM against other emerging near-data processing technologies.

A. Cost of ASIC Specialization

As Dennard scaling nears its end, power limitations have accelerated the push towards specialization in domain-specific architectures, especially for important applications in the datacenter [42]–[45]. Recent work has shown that it is not unprecedented to build specialized accelerators when general purpose cores fail to meet viable peformance targets for important applications [43]–[45]. To evaluate the monetary cost viability of ASIC specialization for SSAM, we build an analytical model to estimate the aggregate throughput of a single CPU-based server, energy cost, and savings based on our CPU results. We assume that the NRE cost of ASIC specialization which includes mask and ASIC development costs for a 28 nm process is $88 million [46]. We use our multicore CPU implementation on an Intel Xeon E5-2620 as our cloud CPU baseline, and profile queries with medium sized GIST descriptors.

Recent data shows that Google handles in excess of 56,000 queries per second, of which up to 20% of all queries are new and unique [47], [48]; we assume the remaining 80% of queries are serviced by a front-end cache. While the internal implementations of search engines are significantly more complicated, the query rates serve as a good first order

estimate of throughput demands. Our CPU baseline can service the 11,200 unique queries per second with ∼1,800 machines, using an estimated 118 kW-hrs per second of dynamic compute power. Adapting the TCO analysis from [49], the baseline's total compute energy cost for servers alone would be $772 million over three years (assuming 6.9 cents / kWhr) [50][3]. SSAM-based servers would expend 0.7 kW-hrs per second on the same workload for computation yielding a total compute energy cost of $4.69 million over three years. As a result, we conclude a widespread deployment of our accelerator is potentially a cost-effective solution for similarity search, since the cost per year of computation on CPUs is significantly higher than the compute cost per year when using SSAM. Note that this savings estimate does not account for additional overheads that exist in a datacenter setting like mass storage devices, additional memories, networking apparatus, and power supply units but serves as a good first order estimate.

B. Index Construction and Other Applications

The SSAM is not limited to approximate kNN search and can also be used for kNN index construction and other data-intensive applications. In kNN, the overhead of building indexing structures are generally amortized over the number of queries executed; however, index construction is still three orders of magnitude slower than single query execution. Fortunately, SSAMs can be reprogrammed to also perform these data-intensive tasks. Index construction also benefits from near-data processing as techniques like k-means and kd-tree construction require multiple scans over the entire dataset. For instance, to train a hierarchical k-means indexing structure, we execute k-means by treating cluster centroids as the dataset and streaming the dataset in as kNN queries to determine the closest centroid. While a host processor must still handle the short serialized phases of k-means, SSAMs are able to accelerate the data-intensive scans in the k-means kernel by exploiting the enormous bandwidth exposed by performing the computation near memory. Similarly for kd-tree index construction, SSAMs can be used to quickly scan the dataset and compute the variance across all dimensions. The host processor can then quickly assign bifurcation points and generate the tree. In both cases, the host processor must provide some control and high level orchestration but the bulk of each application kernel can be offloaded and benefits from the augmented memory bandwidth.

SSAMs can also be used to accelerate other data-intensive applications that benefit from vectorization and enhanced memory bandwidth. Applications such as support vector machines, k-means, neural networks, and frequent itemset mining can all be implemented on SSAM. In particular, the vectorized FXP instruction is useful for evaluation classes of application which rely on many Hamming distance calculations such as binary neural networks [51], and binary hash functions.

[3]Based on 2015 7-month average retail industrial cost/kW-Hr.

TABLE VI: SSAM and AP throughput comparison for linear Hamming distance kNN.

Dataset	GloVe	GIST	AlexNet
SSAM-4 (queries/s)	2059.3	480.5	134.10
First Generation AP (queries/s)	288	2.64	0.553
Second Generation AP (queries/s)	1117.09	10.55	0.951

C. Alternative Near-Data Processing Architectures

Near-data processing manifests in many different shapes and forms. In this section, we briefly contrast our approach against alternative near-data processing architectures.

Micron Automata Processor (AP): The AP is a near-data processing architecture specialized for high speed non-deterministic finite automata (NFA) evaluation [52]. Unlike SSAM, the AP cannot efficiently implement arithmetic operations and is limited to distance metrics like Hamming distance or Jaccard similarity. At a high level, the automata design is composed of multiple parallel NFAs where each NFA encodes a distance calculation with a single dataset vector as shown in [53]. A query vector is streamed into the AP and compared against all NFAs in parallel and sorted. In order to scale to larger datasets, the AP can be reconfigured to process NFAs that do not fit on in one configuration much like reconfiguration on an FPGA. We briefly evaluate the AP by designing and compiling a design for each dataset, and use the results to estimate performance for an AP device against SSAM. Table VI highlights the AP's performance and energy efficiency results against the SSAM design presented in this paper.

In general, we find that for first generation APs the large datasets presented in this paper do not fit on one AP board configuration, and as a result the AP is bottlenecked by the high reconfiguration overheads compared to SSAM. With the generous 100× faster reconfiguration times proposed in [53], the AP is able to mitigate some these performance losses but still struggles for very high dimensional descriptors like AlexNet and GIST. For high dimensional vectors, each automata processor configuration can only fit a handful of vectors at a time, which severely reduces the available parallelism the automata processor is able to exploit. As a result, the automata processor platform is best suited for smaller dimensional feature vectors which allows it to better exploit available parallelism.

Compute in Resistive RAM (ReRAM): Computation in ReRAM is an emerging near-data processing technique that can perform limited compute operations in-situ by directly exploiting the resistive properties of ReRAM memory cells [54]. This allows augmented computational capabilities beyond what are available to DRAM-based near-data processing techniques such as SSAM. Most notably, Chi et al. [55] has recently shown how in-situ ReRAM computation can be used to accelerate convolutional neural networks without moving data out of the ReRAM cells themselves. As the technology matures, it would not be unprecedented to replace DRAM in favor of ReRAM and its augmented computing capabilities.

In-Storage Processing: There has also been a renewed interest in instantiating computation near disk or SSD. Recent work

such as Intelligent SSD [56], [57] and Biscuit [58] have proposed adding computation near mass storage devices and shown promising improvements for applications like databases. However, compared to SSAM, in-storage processing architectures target a different bandwidth to storage capacity design point. Unlike SSAM, SSD-based near-data processing handles on the order of terabytes of data at lower bandwidth speeds, which is less ideal for latency critical applications like similarity search.

Die-Stacked HMC Architectures: Instantiating an accelerator adjacent to HMC is not a new proposal. Prior work has shown that such an architectural abstraction is useful for accelerating graph processing [59] and neural networks [60]. This architecture has several advantages over in-situ ReRAM computation and the automata processor. First, by abstracting the computation away from the memory substrate, the types of computation that can be instantiated in the compute layer are decoupled from the restrictions of underlying memory implementations. Second, by separating the computation from actual memory cells, this architectural abstraction achieves much higher compute and memory density. This is unlike substrates like the AP where compute and memory are both instantiated in the same memory process resulting in both lower memory density and suboptimal compute speeds.

VII. Related Work

Near-data processing for similarity search has been studied in the literature for decades, indicating ongoing interest.

CAMs: In the 1980s, near-memory accelerators were proposed to improve the performance of nearest neighbor search using CAMs [61]. Kohonen et al. [62] proposed using a combination of CAMs and hashing techniques to perform nearest neighbor search. Around the same time, Kanerva et al. [63] propose sparse distributed memory (SDM) and a "Best Match Machine" to implement nearest neighbor search. The ideas behind SDM were later employed by Roberts in PCAM [61] which is, to our knowledge, the first fabricated CAM-based accelerator capable of performing nearest neighbor search on its own.

Algorithms that exploit TCAMs to perform content addressable search such as ternary locality sensitive hashing (TLSH) [64] and binary-reflected Gray code [65] also exist. However, TCAMs suffer from lower memory density, higher power consumption, and smaller capacity than emerging memory technologies. While prior work [66] shows promising increases in performance, energy efficiency, and capacity, TCAM cells are less dense than DRAM cells. For the massive scale datasets in kNN workloads, the density disparity translates to an order of magnitude in cost. Despite these limitations, there is still active work in TCAMs for data-intensive applications to accelerator associative computations.

Multiprocessor and Vector PIMs: In the late 1990s, Patterson et al. [67] proposed IRAM, which introduced processing units integrated with DRAM. In particular, Gebis et al. [68] and Kozyarakis et al. [69] proposed VIRAM which used a vector processor architecture with embedded DRAM. Similar to our work, the intention of VIRAM was to capitalize on higher bandwidth and reduce energy consumption by co-locating MIPS cores with vector registers and compute units near DRAM. Unlike VIRAM, SSAM does not implement a full cache hierarchy, targets a different class of algorithms, and uses a 3D die-stacked solution.

Kogge et al. [70] propose the EXECUBE architecture which integrates general purpose cores with DRAM macros. Elliott et al. [71] propose C-RAM which add SIMD processing units adjacent to the sense amplifiers capable of bit serial operations. Active Pages [15] and FlexRAM [14] envisioned a programmable processing element near each DRAM macro block which could be programmed for kNN acceleration. However, none of these prior efforts directly addresses the kNN search problems we discuss. More recently, Active Memory Cube (AMC) [72] proposes a similar vector processing unit and cache-less system on top of HMC. While both SSAM and AMC arrive at the same architecture conclusion - that vector PIM on die-stacked DRAM is useful - our work provides a more application-centric approach which allows us to codesign architectural features.

Application-Driven PIM: Application-justified PIM design is not a new idea. Lipman and Yang [73] propose a DRAM-based architecture called smart access memory (SAM) for nearest-neighbor search targeting DB applications. Their design tightly integrates a k-nearest neighbor accelerator engine and has a microarchitecture that shares common elements with our design. Similarly, Tandon et al. [74] propose an all pairs similarity accelerator for NLP; however, their work integrates their accelerator with the last level cache instead of memory.

The emergence and maturity of die-stacked memory has also enabled a wide variety of PIM accelerator proposals [59], [72], [75]–[78]. Chi et al. [75], Kim et al. [79], and Gao et al. [80] all propose PIM solutions for accelerating neural networks. Ahn et al. [59] propose PIM on top of HMC for graph processing, and Hsieh et al. [76] and Zhang et al. [81] propose PIM-based GPU architectures. Imani et al. [82] propose MPIM for linear kNN search but their ReRAM model is limited to bitwise operations. MPIM also neither considers nor evaluates the quality versus accuracy tradeoffs in modern approximate kNN algorithms.

VIII. Conclusions

We presented SSAM, an application-driven near-data processing architecture for similarity search. We showed that by moving computation closer to memory, SSAM is able to address the data movement challenges of similarity search and exploit application codesign opportunities to accelerate similarity search. While we used HMC as our memory backend, the high-level accelerator design and insights still generalize to alternative memory technology and in-memory processing architectures. The PIM proposal presented in this paper is also relevant to other data-intensive workloads, where data movement is becoming an increasingly fundamental challenge in improving system efficiency.

IX. Acknowledgements

This work was supported in part by NSF under grant CCF-1518703, gifts by Oracle, a Qualcomm Innovation Fellowship, and by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. We also thank Ali Farhadi for his insights during the initial stages of this work.

References

[1] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan, "Photon mapping on programmable graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. Eurographics Association, 2003, pp. 41–50.

[2] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ser. ICCV '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 1470–.

[3] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1704–1716, Sep. 2012.

[4] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Dec. 2007.

[5] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[6] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in haystack: Facebook's photo storage," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 47–60.

[7] YouTube, "Statistics - YouTube," 2014.

[8] "Rebooting the it revolution: A call to action," 2015.

[9] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[10] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[11] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, ser. SCG '04. New York, NY, USA: ACM, 2004, pp. 253–262.

[12] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 604–613.

[13] 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, Mar. 1997.

[14] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "Flexram: toward an advanced intelligent memory system," in *Computer Design, 1999. (ICCD '99) International Conference on*, 1999, pp. 192–201.

[15] M. Oskin, F. T. Chong, and T. Sherwood, "Active pages: A computation model for intelligent memory," *SIGARCH Comput. Archit. News*, vol. 26, no. 3, pp. 192–203, Apr. 1998.

[16] IBM, ""Blue Logic SA-27E ASIC"," 1999, "http://www.ic72.com/pdffile/s/381279.pdf".

[17] J. T. Pawlowski, "Hybrid memory cube (hmc)," *Hot Chips*, vol. 23, 2011.

[18] M. Kusner, Y. Sun, N. Kolkin, and K. Q. Weinberger, ""From Word Embeddings To Document Distances"," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, D. Blei and F. Bach, Eds. JMLR Workshop and Conference Proceedings, 2015, pp. 957–966.

[19] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, "Evaluation of GIST Descriptors for Web-scale Image Search," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, ser. CIVR '09. New York, NY, USA: ACM, 2009, pp. 19:1–19:8.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014.

[22] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, Dec 2014, pp. 609–622.

[23] Y. Gong and S. Lazebnik, "Iterative Quantization: A Procrustean Approach to Learning Binary Codes," in *CVPR '11*.

[24] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," in *Proceedings of the 15th International Conference on Neural Information Processing Systems*, ser. NIPS'02. Cambridge, MA, USA: MIT Press, 2002, pp. 521–528.

[25] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 1532–1543.

[26] I. Jolliffe, "Principal component analysis," 2014.

[27] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[28] Yahoo!, "Webscope — Yahoo Labs," 2014. [Online]. Available: http://webscope.sandbox.yahoo.com/catalog.php?datatype=i\&did=67

[29] C. Silpa-Anan and R. I. Hartley, "Optimised kd-trees for fast image descriptor matching." in *CVPR*. IEEE Computer Society, 2008.

[30] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07. VLDB Endowment, 2007, pp. 950–961.

[31] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1225–1233.

[32] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *CVPR*, 2016.

[33] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 190–200.

[34] H. M. C. Consortium, ""Hybrid Memory Cube Specification 2.1"," 2014. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.1_20151105.pdf

[35] R. M. Russell, "The cray-1 computer system," *Commun. ACM*, vol. 21, no. 1, pp. 63–72, Jan. 1978.

[36] S.-W. Moon, K. G. Shin, and J. Rexford, "Scalable hardware priority queue architectures for high-speed packet switches," *IEEE Trans. Comput.*, vol. 49, no. 11, pp. 1215–1227, Nov. 2000.

[37] R. S. Anand Lal Shimpi, "The intel ivy bridge (core i7 3770k) review," accessed: 2016-11-10. [Online]. Available: http://www.anandtech.com/show/5771/the-intel-ivy-bridge-core-i7-3770k-review/3

[38] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, June 2008, pp. 1–6.

[39] "Nvidia geforce gtx titan x," accessed: 2016-11-10. [Online]. Available: https://www.techpowerup.com/gpudb/2632/geforce-gtx-titan-x

[40] U. Technologies, "Logic detailed structural analysis of the xilinx kintex-7 28nm fpga (es)," 2011, accessed: 2016-11-10. [Online]. Available: http://dc.ee.ubm-us.com/i/64593-tech-insights-xilinx-report/10

[41] K. Puttaswamy and G. H. Loh, "Thermal analysis of a 3d die-stacked high-performance microprocessor," in *Proceedings of the 16th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '06. New York, NY, USA: ACM, 2006, pp. 19–24.

[42] M. B. Taylor, "Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012, pp. 1131–1136.

[43] N. Jouppi, "Google supercharges machine learning tasks with tpu custom chip," https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html, accessed: 10-01-16.

[44] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, "Asic clouds: Specializing the datacenter," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 178–190.

[45] A. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, October 2016.

[46] T. Austin, "How eda could save the world (of computing)," 2017.

[47] Intel, "What Happens in an Internet Minute?" Oct. 2014.

[48] "Google search statistics," http://www.internetlivestats.com/google-search-statistics/, accessed: August 31, 2016.

[49] L. A. Barroso and U. Hoelzle, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[50] "October 2016 monthly energy review," accessed: 2016-11-14. [Online]. Available: http://www.eia.gov/totalenergy/data/monthly/pdf/mer.pdf

[51] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016.

[52] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 3088–3098, 2014.

[53] V. T. Lee, J. Kotalik, C. C. d. Mundo, A. Alaghi, L. Ceze, and M. Oskin, "Similarity search on automata processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 523–534.

[54] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, stt-mram based computing," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 371–382, Jun. 2010.

[55] 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 *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 27–39.

[56] D.-H. Bae, J.-H. Kim, S.-W. Kim, H. Oh, and C. Park, "Intelligent ssd: a turbo for big data mining," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, pp. 1573–1576.

[57] S. Cho, C. Park, H. Oh, S. Kim, Y. Yi, and G. R. Ganger, "Active disk meets flash: A case for intelligent ssds," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13. New York, NY, USA: ACM, 2013, pp. 91–102.

[58] B. Gu *et al.*, "Biscuit: A framework for near-data processing of big data workloads," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 153–165.

[59] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 105–117.

[60] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 380–392.

[61] J. D. Roberts, "PROXIMITY CONTENT-ADDRESSABLE MEMORY:AN EFFICIENT EXTENSION TO k-NEAREST NEIGHBORS SEARCH (M.S. Thesis)," Santa Cruz, CA, USA, Tech. Rep., 1990.

[62] T. Kohonen, *Self-organization and Associative Memory: 3rd Edition*. New York, NY, USA: Springer-Verlag New York, Inc., 1989.

[63] P. Kanerva, *Sparse Distributed Memory*. Cambridge, MA, USA: MIT Press, 1988.

[64] R. Shinde, A. Goel, P. Gupta, and D. Dutta, "Similarity search and locality sensitive hashing using ternary content addressable memories," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 375–386.

[65] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Hel-Or, "Ultra-fast similarity search using ternary content addressable memory," in *Proceedings of the 11th International Workshop on Data Management on New Hardware*, ser. DaMoN'15. New York, NY, USA: ACM, 2015, pp. 12:1–12:10.

[66] Q. Guo, X. Guo, Y. Bai, and E. İpek, "A resistive tcam accelerator for data-intensive computing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 339–350.

[67] 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, Mar. 1997.

[68] J. Gebis, S. Williams, D. Patterson, and C. Kozyrakis, "Viram1: a media-oriented vector processor with embedded dram," 2004.

[69] C. Kozyrakis, J. Gebis, D. Martin, S. Williams, I. Mavroidis, S. Pope, D. Jones, and D. Patterson, "Vector iram: A media-oriented vector processor with embedded dram," 2000.

[70] P. M. Kogge, T. Sunaga, H. Miyataka, K. Kitamura, and E. Retter, "Combined DRAM and logic chip for massively parallel systems," in *16th Conference on Advanced Research in VLSI (ARVLSI '95), March 27-29, 1995, Chapel Hill, North Carolina, USA*, 1995, pp. 4–16.

[71] D. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational ram: Implementing processors in memory," *IEEE Des. Test*, vol. 16, no. 1, pp. 32–41, Jan. 1999.

[72] R. Nair *et al.*, "Active memory cube: A processing-in-memory architecture for exascale systems," *IBM Journal of Research and Development*, vol. 59, no. 2/3, pp. 17:1–17:14, March 2015.

[73] A. Lipman and W. Yang, "The Smart Access Memory: An Intelligent RAM for Nearest Neighbor Database Searching," in *In ISCA Workshop on Mixing Logic and DRAM*, 1997.

[74] P. Tandon, J. Chang, R. G. Dreslinski, V. Qazvinian, P. Ranganathan, and T. F. Wenisch, "Hardware acceleration for similarity measurement in natural language processing," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ser. ISLPED '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 409–414.

[75] 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 *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 27–39.

[76] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler, "Transparent offloading and mapping (tom): Enabling programmer-transparent near-data processing in gpu systems," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 204–216.

[77] G. H. Loh, N. Jayasena, M. H. Oskin, M. Nutter, D. Roberts, M. M. Dong, P. Zhang, and M. Ignatowski, "A processing-in-memory taxonomy and a case for studying fixed-function pim," 2013.

[78] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: ACM, 2014, pp. 85–98.

[79] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 380–392.

[80] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleratino with 3d memory," in *Proceedings of the Twenty Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017.

[81] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: ACM, 2014, pp. 85–98.

[82] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 757–763.