Large-scale high-dimensional nearest neighbor search using flash memory with in-store processing

Sang-Woo Jun, Chanwoo Chung and Arvind
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
{wjun, cwchung, arvind}@csail.mit.edu

***Abstract*—Modern datasets of importance such as images, videos, protein sequences or text, usually contain very high dimensional information from the search point of view. Nearest neighbor search is one of the most fundamental building blocks in dealing with large amounts of data. It is the problem of finding points in a database that are most similar to a query data point by some distance metric. There is a large body of work in algorithms for nearest-neighbor search on large high-dimensional datasets. Since these algorithms invariably involve random access to data, most existing implementations ensure that the data is stored in DRAM, and does not spill into secondary storage such as hard disks. However, the immense size of modern datasets often requires hundreds of computers to accommodate the dataset in DRAM. An alternative to such a system is a much smaller cluster that stores the dataset in flash memory (instead of DRAM) and has in-store computing capability. In this paper, we build and demonstrate the performance of high-dimensional nearest-neighbor search on a flash-based system with FPGA acceleration and show that it sometimes exceeds the performance of a DRAM-based solution. We chose two example applications, images and documents, for this demonstration. Since flash storage, in comparison to DRAM, is an order of magnitude cheaper and consumes an order of magnitude less power, a flash-based solution for nearest-neighbor searches offers a viable and attractive alternative.**

I. Introduction

In our current "Big Data" era, there is an ever-increasing amount of high-dimensional data, such as text, images, videos and time series information that are collected and available for analysis. In order for this data to be valuable, there must be an affordable way to quickly analyze and gain insight into a large amount of data. The sheer size of modern datasets makes this challenging for conventional computer systems.

One of the fundamental applications in extracting insight from Big Data is nearest-neighbor search, where the system receives a query data point and finds data points in its database that are closest to the query, according to some distance metric. Nearest neighbor search is widely used on many different types of high-dimensional data such as images, text, time-series or other custom data types to implement useful applications such as image retrieval, plagiarism detection or time series prediction. Because the distance between two data points can often be determined only after performing distance calculation between the two, nearest neighbor search in a very large dataset is challenging, especially when the data type in question has high dimensionality and the distance metric is complex. Many existing systems employ clever indexing and sampling methods to reduce the search space and achieve higher performance. Such methods require additionally pre-processing the entire dataset to organize it into a better searchable structure such as trees, hashes or augmenting it with tags. The pre-processing has to be repeated when the pre-processing technique has to be modified.

Because modern datasets are often too large to be processed on a single computer, they are often stored and processed by a cluster of systems, often running distributed processing software. Due to the performance difference between main memory and secondary storage such as disk, the entire data needs to fit completely in distributed main memory to achieve the best performance [1]. This issue is exacerbated when sampling and indexing is employed, as random access performance of disks is dramatically lower than its sequential access. As datasets become larger, a completely DRAM-based system quickly becomes very expensive. Flash memory could present a cost-effective design, because flash memory is faster than disk and cheaper than DRAM, while consuming less power than either. However, using flash storage purely as a disk replacement offers limited improvement, because flash fabric has different characteristics than disk, and it is still slower than DRAM.

Application-specific hardware accelerators using reconfigurable hardware fabric such as Field Programmable Logic Arrays (FPGAs) are also gaining popularity, due to their good power-performance characteristics. FPGAs can be dynamically programmed to implement application specific hardware accelerators that can often perform much faster than software implementations while consuming a fraction of power. A popular way to deploy an accelerator is to plug it into a system bus such as PCIe [2]. For some class of applications, we think a more effective method is to embed it into the storage device. Such in-storage processing deployment has many advantages over the alternative method of a separate appliance. The accelerator avoids the overheads of going through system bus or software management to access storage. The accelerator does not need to include additional hardware IP to access the system bus, minimizing the extra cost for adding accelerators. MIT's BlueDBM cluster [3] is an example of such a system.

The major contributions of this paper are: the design and implementation of a high-dimensional nearest-neighbor search on a flash-based platform with in-store computing, and a demonstration that a flash-based solution shows performance that is comparable to a traditional DRAM-based solution, and sometimes even exceeds its performance. We have chosen images and documents examples of high-dimensional data, and implemented the distance metric for comparing the distance

978-1-4673-9406-2/15/$31.00

between a pair of images or documents in the FPGA. Our implementation is intended to be an example of a general high-dimensional nearest neighbor search, and not in any way limited to our selected applications.

In our evaluation, the flash-based solution achieves over 10X performance compared to a disk-based system, and sometimes outperforms even a DRAM-based multicore system. Our system consumes about half of power per node compared to a DRAM-based system, and less than half in terms of power consumption per amount of processed data. It also requires a much fewer number of machines to accommodate the entire dataset.

The rest of the paper is organized as follows: In Section II we present some existing work related to large-scale nearest neighbor search on high-dimensional data. In Section III we describe the detailed architecture of our system. In Section IV we demonstrate the performance of our system using a large image dataset and a document dataset. We conclude in Section V.

II. Background and Related Work

A. High-Dimensional Nearest Neighbor Search

Nearest neighbor search on high-dimensional data suffer from a so-called "Curse of dimensionality", in which it becomes very difficult to pre-process data and organize them into a structure that is easily queried. For example, kd-trees [4] are often used for low-dimensional nearest neighbor queries, but become less effective in a high dimensional setting because the relative importance of any single dimension becomes low. In the worst case, a query has to iteratively compute the distance function between the query point and all points in the dataset to determine the nearest neighbor. This becomes especially difficult since distance calculation often does not benefit from preprocessing and requires comparing two raw data points when the dimensionality is high. A good example of this is time series comparisons.

Many clever indexing and sampling schemes have attempted to solve this problem [5], [6]. A prominent work in this area is Locality Sensitive Hashing (LSH) [7], where the dataset is pre-processed using a set of hash functions and organized into a few of many buckets. The same hash functions are applied to a query, and only the data already in the corresponding bucket has to be compared. LSH is an approximate nearest neighbor algorithm, where it provides some statistical guarantee of finding the nearest neighbor. An interesting aspect of such algorithms is that because data is no longer read in a sequential manner, random access performance of the storage device becomes a limiting factor of performance. Because random access performance of hard disk drives is very bad due to its high seek time, performance often drops sharply when data does not fit completely in DRAM. Due to the large size of modern datasets, data and computation are often distributed across many machines in a cluster, using distributed processing platforms such as Hadoop [8] or Spark [9].

B. Comparing Images

A popular application of nearest neighbor search is content-based image retrieval, where a system looks for images that are most similar to a query image, according to some distance metric. The usefulness of a content-based image retrieval system is determined not only by how fast the result can be obtained, but also by the usefulness of the distance metric. There exist image comparison techniques optimized for various applications, including feature extraction [10], [11], facial recognition [12] and scene recognition [13]. A popular distance metric is comparing color histograms of images, where a multidimensional histogram is constructed using pixel values from an image and compared using euclidean distance or Earth Mover's Distance [14].

Histograms using just color values of individual pixels (RGB or HSV) to construct a histogram has been useful in the past, as they often correctly represent the color profile of the images while being robust in regards to movements in the image such as shifting. However, color histograms begin to fail when the image corpus becomes large, when there are different images that have similar color compositions. Joint histograms [15] overcome this limitation by adding more features into consideration. Joint histograms of an image is a multidimensional histogram built using color values of each pixel and other local pixel features, such as edge density, texturedness and gradient magnitude of a pixel location. In constructing joint histograms, an image is first processed using various filters that generate new images that emphasize different local features of the image. A histogram constructed using these images in addition to color values is much more representative of the features of the image.

A modern way of image retrieval that has quickly gained popularity is using deep neural networks [16]. Each image is processed using a neural network and are given a set of tags that are determined relevant to the image, reducing the dimensionality by a great amount. Using deep neural networks to classify images have shown enormous accuracy, and is becoming the de facto method of image recognition. However in this work, we have chosen to implement a joint-histogram based image comparison, for its simplicity of implementation and to not detract the focus of the work away from nearest-neighbor search.

In a real world deployment of content based image retrieval, images in a dataset would be pre-processed to be augmented with meta data, such as descriptive tags or histogram values, in order to make search more efficient. In this work, we have not implemented such pre-processing and do all distance comparisons on the fly on raw image data, in order to demonstrate an example of a complex distance metric. The results obtained from this system will translate well to other high dimensional search applications, such as time series and protein sequences.

C. Image Search and Accelerators

Because of the high complexity of many image comparison algorithms, there has been a large body of work on accelerating them. One popular type of accelerator is the General Purpose Graphic Processing Unit (GPGPU) [17], [18]. Modern GPUs have thousands of cores, and can run thousands of threads in parallel and speed up execution. However, running that many general purpose cores incur a large power consumption. Adding a GPU accelerator often doubles the total power consumption of a single machine. Furthermore, a GPU accelerator

can fully demonstrate its performance only when data fits in its graphic memory, which is usually much smaller than system DRAM.

There has also been attempts in using FPGAs to accelerate image retrieval, in order to benefit from its high performance and very low power consumption. Some work has focused on accelerating image comparison functions [19]. At least one work has explored coupling FPGA with flash storage to accelerate image retrieval [20]. This work pre-computed descriptors for the image dataset and focused on querying images based on the pre-computed descriptors. There has also been a great amount of work on low power acceleration of image processing using FPGAs [21], [22], [23].

D. Document Similarity Search

Another important application where nearest neighbor search is employed is document similarity search, where a database of documents is searched to retrieve those that are similar to a query document. Document similarity search is used in applications such as finding similar web pages on the internet, plagiarism detection or searching for relevant legal documentation. A popular method of measuring similarity between documents is term similarity, where documents are deemed similar if they share the same terms, or words. In term similarity comparison, a document is viewed as a "Bag of words", where a document is characterized by the orderless set of words and the number of occurrence of each word. The simplest way to compare two bags of words is to count the occurrence of shared words. However, this simple method often does not perform well in terms of accuracy. A simple yet effective method is using *Cosine Similarity*, where each bag of words is considered a multidimensional vector, where each term corresponds to a dimension and its occurrence corresponds to the magnitude. Similarity between vectors is determined by calculating the cosine between the two vectors. In this work, we have created a synthetic database of documents pre-processed into bags of words, and performed nearest neighbor search using cosine similarity as the distance metric.

E. Flash Storage

NAND flash based storage is gaining popularity in consumer devices and datacenters, due to its low power consumption and superior random access performance compared to traditional hard disks. Flash storage consumes less power than disk or DRAM, while having an order of magnitude lower price per space compared to DRAM.

A single flash chip can deliver a fixed amount of bandwidth and latency, and more bandwidth can be obtained by organizing multiple flash chips into busses and accessing them in parallel. The performance of flash-based SSDs have become so fast, there is active research in developing a next-generation non-volatile memory interface method that overcomes the limitations of slow SATA interface [24]. Multiple vendors have also developed extremely high performance flash devices that plug into the PCIe port, which provides much higher bandwidth [25], [26].

Flash storage is usually organized into units of 4 kilobyte pages, and a larger unit consisting of hundreds of pages called blocks. Reading and writing can happen at page granularity, but a page has to be erased before being written, and erases can happen only at block granularity. If page modifications were to be done always in-place, each write will require whole block erases. Because erases take much longer time than read and writes, such a write method will incur unnecessary overhead. Most flash devices are managed by firmware software called FTL (Flash Translation Layer) so that overwrites are done to a new page that has been pre-erased, and a mapping table maintains a logical address space to physical address space mapping.

There is active research in combining flash storage and FPGAs for efficient big data analytics. MIT's BlueDBM [3] is a platform for accelerating big data analytics using 20 TBs of distributed 20-node flash storage with FPGA-based in-store processing. The BlueDBM system provides an abstract interface into the flash, host server and the storage area network to the FPGA accelerator. This allows the application developer to focus on developing the application logic. We have conducted our research on this platform.

III. System Architecture and Implementation

Figure 1 shows the overall architecture of the nearest neighbor analytics system. The whole system is distributed across multiple computer nodes. Each node is composed of a software daemon running on a host CPU, and a flash storage device augmented with a FPGA-based in-storage processor. Each in-storage processor implements an array of application-specific distance comparators and a controller than manages them. For different applications, a different controller and comparators are programmed to the in-storage processor. A client software connects to each node's software daemon separately via TCP. The software daemon sends commands to the controller and receives nearest neighbor results. The controller sends read commands to the flash array to read individual data points, and routes the data to idle comparators. The BlueDBM architecture includes the BlueDBM storage device, which is a flash storage device with FPGA-based in-store processing capabilities, as well as interface to the flash array and the PCIe interface to the software daemon.

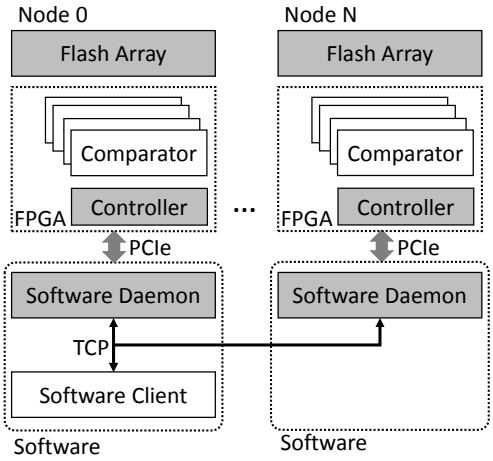
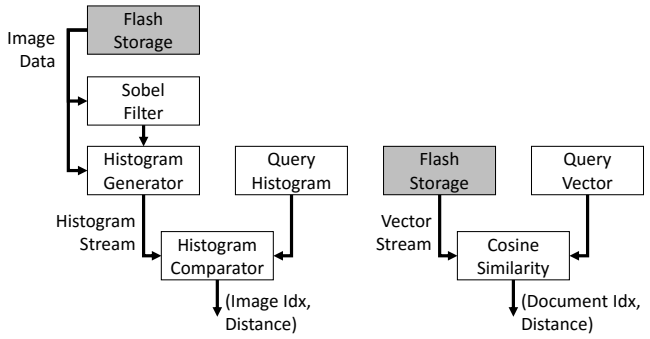


Fig. 1: System Architecture

A. Comparator Architecture

Each node's FPGA accelerator implements an array of application-specific distance comparators. In this work we demonstrate accelerators for two applications, image and document search. Image search implements joint histogram based image comparators, and document search implements a cosine similarity based document comparators. The image search example depicts a system with a complex distance metric and large data points, while the document search example depicts a system with a much simpler distance metric and smaller data points. Each comparator takes a stream of data points from the dataset and emits a list of tuples, consisting of an index and its similarity to the query data point.



(a) Image Comparator (b) Document Comparator

Fig. 2: Application-Specific Comparator Architecture

1) Image Comparator: Each node's FPGA accelerator implements an array of image comparators, as the bandwidth of the flash array is faster than the performance of a single image comparator. We chose to compare the raw images themselves, instead of pre-processing the image and storing histogram values in order to demonstrate a complex high-dimensional distance function. The internal structure of the comparator can be seen in Figure 2a.

The comparator streams the image into a sobel filter, which generates a greyscale image where the edges are emphasized. The resulting edge-emphasized image, along with the original image is streamed into the histogram generator. The constructed histogram is compared with the query histogram, and the resulting distance is returned to the controller along with the index of the image that is being compared.

Sobel Filter : The sobel filter implementation works in a streaming fashion, so that the whole image does not have to be stored in a cache until the sobel operator is done generating the image. The structure of the sobel filter can be seen in Figure 3. Since a sobel filter operates on three rows at a time, the pixels are streamed through three different row buffers consecutively. This effectively scans the image three rows at a time and applies the sobel coefficients that emphasize edges in the image. Because coefficients cannot be correctly applied on edge pixels, a mux at then end of the stream keeps track of the pixel coordinates in the image and emits a zero if the pixel is an edge pixel.

Histogram Generator : The histogram generator populates a four-dimensional histogram, where the axes are RGB values,

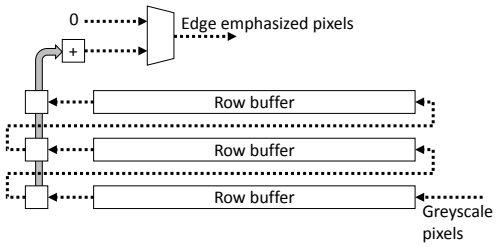


Fig. 3: Sobel Filter Implementation

along with the edgeness of each pixel location. Once the whole image is streamed into the histogram generator, it starts streaming out the contents of the constructed histogram while resetting the histogram bin values, so that it can start receiving data for the next image. In our implementation the histogram generator builds a histogram with 8 bins per dimension, but the granularity of the histogram is parameterized and can be changed easily.

2) Document Comparator: The cosine similarity based document comparator is much simpler and less computation-heavy than the image comparator, so a single comparator can sustain the bandwidth of the flash storage. Each document in the dataset is pre-processed and stored as a vector of tuples consisting of a word and the number of its occurrences. The vocabulary is determined beforehand, so each document is represented using a fixed-length vector of numbers representing the occurrence of each word. This also significantly reduces the memory footprint of document data. The comparator reads a stream of encoded document vectors and calculates the cosine similarity against the query vector. The resulting distance is returned to the controller along with the index of the document that is being compared. The internal structure of the document comparator can be seen in Figure 2b.

B. Software Daemon

The software daemon exists as a process on every worker node's host server. The software daemon talks to the hardware controller through a PCIe interface, and listens to a TCP socket for the user client to connect to. The daemon can be configured to handle images or documents. When the software daemon receives an image query, it creates a query histogram from it and populates the query histogram buffers in the hardware comparators. When it receives a document query, it creates a query vector from it and populates the query vector buffer. The daemon then sends a stream of read requests to the flash array for items in the dataset to be read and sent to idle hardware comparators. The query can be configured to do an exhaustive search sequentially through the entire dataset, or to only compare a subset of the dataset, either by random sampling or by using a list of data indices. The daemon also provides a TCP interface for the user software to read images or documents.

The software daemon can be configured at runtime to either use the hardware comparators, or to do the comparisons in software. The software comparator can spawn a variable number of threads to process data in parallel. We usually spawned as many threads as there are cores in the CPU, as

performance tapered off after that many threads. The daemon can also be configured to load data from and off-the shelf flash SSD, BlueDBM, disk, or from DRAM. The hardware comparators can only be used when the data was being loaded from the custom flash device.

In order to maintain a simple hardware interface between the array of comparators and the flash array, the software daemon manages the logical to physical mapping of pages in the host server's memory space. When comparing the software daemon wants to compare the query with a particular data point in the dataset, it consults its mapping table to determine the location of the required pages, and sends a string of commands to the hardware side controller to read the required pages and forward it to the appropriate comparator. The mapping is done at block granularity, because this workload required no random overwrites. This resulted in a much smaller mapping table as compared to using a page granularity mapping table.

C. Software Client

The software client connects to the individual worker node's software daemons simultaneously via a TCP connection. The query is broadcast from the software client to the software daemons of all worker nodes at the same time, and the client listens to all TCP sockets at the same time for their responses. Because the software daemons return only its best candidates for nearest neighbors, the required data rate between the client and software daemons is low enough that even using simultaneous TCP does not become a bottleneck.

A special GUI was created for the image query workload to make the dataset navigation easier. The users can browse the dataset to select a query image using a simple navigational interface. Once a query image is selected, the query is broadcast to the software daemons. The user interface displays information such as the nearest neighbors returned so far, sorted according to distance, total progress of the query, and the performance of the system. This interface can be seen in Figure 4. The nearest images discovered so far are displayed in a grid near the top, and user interface information such as a progress bar and performance graph is displayed at the bottom.

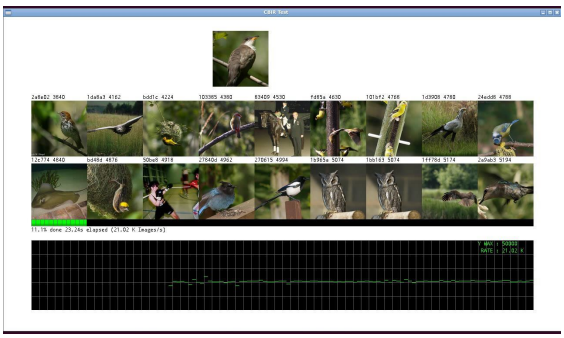


Fig. 4: Software Client Interface

IV. Performance Evaluation

We have implemented our solution on MIT's BlueDBM cluster. BlueDBM consists of 20 identical nodes. Each node is a Xeon server machine with a BlueDBM storage device plugged into its PCIe port. A BlueDBM storage device consists of a Xilinx VC707 FPGA development board and two flash expansion cards, each with 512GBs of capacity and 1.2GB/s of bandwidth. The Xeon server has a 24-core processor and 50GBs of DRAM. The server also could be configured with an off-the-shelf M.2 PCIe SSD. Our implementation used only one of the two flash expansion cards available to each BlueDBM nodes.

The implementation of the system involved development of the software component and hardware component. The software component includes the client and software daemon, and were written in C++. The baseline system only requires the software component and has complete functionality. The hardware component was written in Bluespec [27], and implements the distance comparator. The distance comparator uses the interfaces provided by the BlueDBM framework to access flash storage and to communicate with the software component over PCIe. The software daemon can be dynamically configured to either perform distance comparison in software or to offload it to the hardware component. BlueDBM also provides a high-performance network directly between the FPGAs, but we have chosen not to use it. Instead, the client software communicates with each software daemon via separate TCP connections. This structure fit our application because the data rate between the client and software daemon is low. The FPGA resource utilization of the image search system can be seen in Table I.

For the image dataset, we used the ImageNet [28] dataset, resized to have width and height of 128 pixels. The resized dataset had a capacity of roughly 1TB, and fit completely in two nodes. We used a synthetic document dataset using English words for the document dataset. Each document was pre-processed into a vector of 8,000 8-bit values, each representing the occurrence of a word in the document.

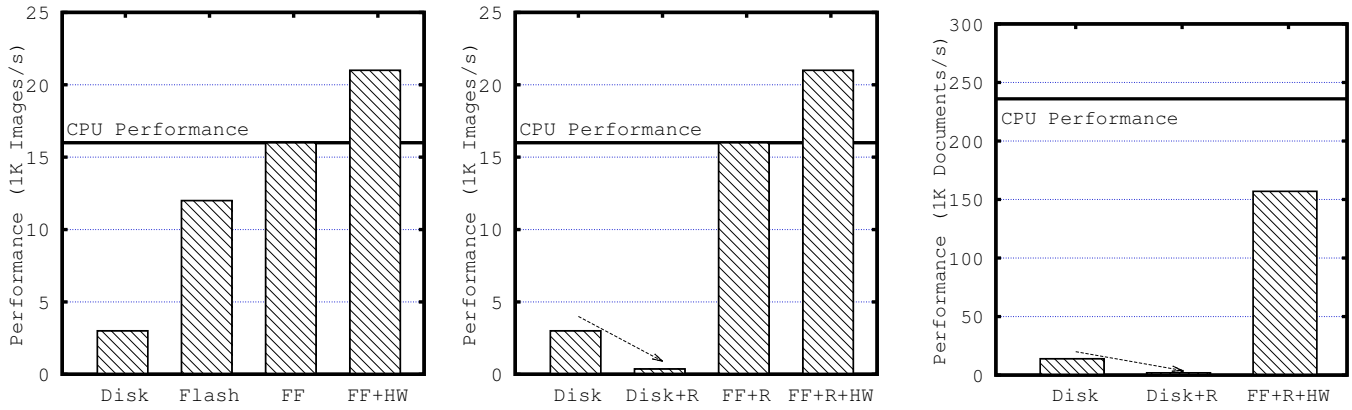
LUTS	Registers	BRAM
148974 (49.06%)	106823 (17.59%)	325 (31.55%)

TABLE I: FPGA Resource Utilization

A. Baseline Image Search Performance Without Sampling

First, we measured the baseline performance of nearest neighbor search with images without any clever indexing or sampling. Each query was done by comparing the distance of a query image against all images in the dataset. We measured the performance of each node in the system while running a nearest neighbor query, by counting the number of images the array of comparators processed per second. The system was configured to load the data from either disk, off-the-shelf flash or the BlueDBM storage device. The following describes the configurations that were explored:

- Disk : Data read from disk, processed using software
- Flash : Data read from off-the-shelf flash, processed using software
- FF : Data read from BlueDBM storage device, processed using software
- FF+HW : Data read from BlueDBM storage device, processed using hardware comparators



(a) Image Search Performance Without Sampling (b) Image Search With Sampling Factor of 30 (c) Document Search Performance

Fig. 5: Nearest Neighbor Search Performance

The results of the experiments can be seen in Figure 5a. The horizontal line reading "CPU Performance" shows the maximum performance a multicore software implementation can reach, where the entirety of the data could fit in DRAM. When this is not the case, the system will start thrashing the secondary storage, and start to show the performance characteristic of the disk or flash based system. Because disk and off-the-shelf flash was much slower than DRAM, the configurations using them fail to reach maximum performance deliverable by the CPU. However, because the CPU work required in comparing the images is not too trivial either, using a faster flash card allows storage bandwidth to saturate the CPU performance. In such a scenario where the storage device can deliver more performance than the processor can handle, using an application specific accelerator can achieve higher performance while consuming even less power.

B. Image Search Performance With Sampling

The next experiment of interest was when random access was introduced by techniques such as indexing or sampling. We ran an experiment with similar configurations, but instead of doing an exhaustive search of the entire dataset, we sampled the dataset so that each image only has a 1/30 chance of being examined. If the throughput of the system does not change, the query should finish 30 times faster. The result of this experiment can be seen in Figure 5b. The following describes the configurations that were explored:

- Disk : Data read from disk, processed using software. Exhaustive search
- Disk+R : Data read from disk, processed using software. Sampling ratio of 30
- FF+R : Data read from BlueDBM storage device, processed using software. Sampling ratio of 30
- FF+R+HW : Data read from BlueDBM storage device, processed using hardware comparators. Sampling ratio of 30

While the performance of systems that use flash maintained the same, a disk-based system's performance dropped drastically when random access patterns were introduced via sampling. This shows that random access performance of the storage device is important for an analytics system, and shows that flash storage is a good platform for exploring analytics problems such as nearest neighbor search.

C. Document Search Performance

We measured the performance of nearest neighbor search with documents with various configurations. The performance of document search was also calculated by running a nearest neighbor query, and counting the number of documents the system was able to process per second. The system was configured to either load the data sequentially from disk, randomly from disk, or randomly from the BlueDBM storage device. The following describes the configurations that were explored.

- Disk : Data read sequentially from disk, processed using software.
- Disk+R : Data read randomly from disk, processed using software.
- FF+R+HW : Data read randomly from BlueDBM storage device, processed using hardware comparators.

The results of the experiments can be seen in Figure 5c. The horizontal line reading "CPU Performance" shows the maximum performance a multicore software implementation can reach, when the entirety of the data can fit in DRAM. Because the computational overhead of the document comparator was low, the performance of the system becomes more bound to the storage system performance. As a result, the BlueDBM system performance is slower than the maximum software performance, because DRAM bandwidth is much faster than the BlueDBM storage device. However, considering that a machine with enough DRAM to accommodate a large dataset is an order of magnitude more expensive than one with the same capacity in flash storage, achieving a comparable performance to a DRAM based system is still an important achievement. Also, the relative performance of a disk based system is even slower than in the image search example.

D. Power Consumption

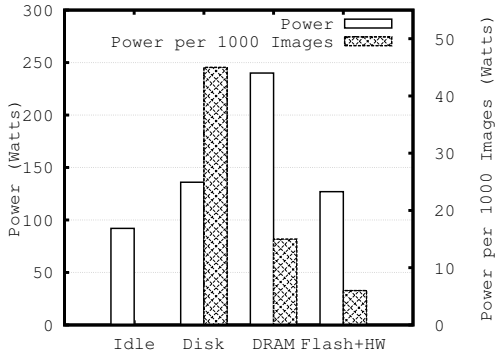


Fig. 6: Power Consumption

We have measured the power consumption to determine the power performance characteristic of our system. We measured the power consumption of a node while running a nearest neighbor query with images, and also calculated the power consumption per amount of work done by dividing it by the performance numbers obtained from the earlier experiments. The results can be seen in Figure 6.

The figure shows that while running everything in DRAM consumes the most amount of power, power consumption per amount of processed data of the DRAM based system is lower than the disk based system because the disk based system is too slow. Our system, which uses flash with hardware comparators consume less power than both disk and DRAM based configurations, and also shows much better power per performance numbers. Our system consumes about half of power compared to the DRAM based system per node, and less than half of power per data processed. Our experimental system also had an order of magnitude smaller DRAM capacity than what would be required if we were to store the entire dataset in DRAM. If we had used such a large system, its power consumption numbers would also be much higher.

E. Expanding the System

A single node in our configuration can store up to 512GBs of data. Assuming a DRAM-based system with 128GBs of DRAM, we would need four DRAM nodes to accommodate the same amount of data. Although the performance would also scale with more DRAM nodes, the price of purchase and power consumption will also increase. Our implementation used only one of the two flash expansion cards installed on each BlueDBM storage device. Using both cards will increase the capacity and bandwidth by two times. The FPGA accelerator has been shown to be capable of accommodating 2.4GB/s or bandwidth, and an additional flash card only adds 5 watts of power to each node. In effect, augmenting the flash device will double the performance of the system, while reducing the power per data processed by half.

V. Conclusion

In the age of "Big Data", an affordable way to process large amounts of data is becoming increasingly important. Many of the important workloads on large datasets including nearest-neighbor search require high random access performance. This usually meant that high performance could only be achieved when all of the data fit in DRAM of the system. Flash based storage devices provide high random access performance while having much higher density at a cheaper price. We demonstrated that for for some high-dimensional nearest neighbor search applications, a flash-based system with an FPGA-based in-store processor could perform comparatively, or even exceed the performance of a DRAM-based system, while being much cheaper and using less power. Of course, this kind of architecture will most likely not be effective for all possible applications, but we think there is a large class of work where a flash based system with in-storage accelerators have a large advantage.

One interesting characteristic of this workload was that it is mostly a bandwidth-bound problem, and not very sensitive to the latency of the storage system. In the future, we plan to explore more complex problems that are latency-sensitive, such as graph analytics, on the BlueDBM platform. Since one of the major architectural benefits of BlueDBM is dramatically reducing the latency of distributed flash access, we think it might be a cheap and attractive alternative for very large latency-sensitive analytics systems.

Acknowledgment

This work was partially funded by Quanta (Agmt. Dtd. 04/01/05) and Lincoln Laboratory(PO7000261350). Author Sang-Woo Jun is partially supported by the Kwanjeong educational foundation, and author Chanwoo Chung is partially supported by the Samsung scholarship.

References

[1] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, "The case for ramclouds: Scalable high-performance storage entirely in dram," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1713254.1713276

[2] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, G. Jan, G. Michael, H. S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Yi, and X. D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Jun. 2014.

[3] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "Bluedbm: An appliance for big data analytics," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 1–13.

[4] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[5] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 11, pp. 2227–2240, 2014.

[6] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces," in *Similarity Search and Applications*. Springer, 2012, pp. 132–147.

[7] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, 1999, pp. 518–529.

[8] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[10] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[11] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[12] A. Samal and P. A. Iyengar, "Automatic recognition and analysis of human faces and facial expressions: A survey," *Pattern recognition*, vol. 25, no. 1, pp. 65–77, 1992.

[13] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 487–495.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[15] G. Pass and R. Zabih, "Comparing images using joint histograms," *Multimedia systems*, vol. 7, no. 3, pp. 234–240, 1999.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[17] V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud, "K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, Sept 2010, pp. 3757–3760.

[18] F. Zhu, P. Chen, D. Yang, W. Zhang, H. Chen, and B. Zang, "A gpu-based high-throughput image retrieval algorithm," in *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, ser. GPGPU-5. New York, NY, USA: ACM, 2012, pp. 30–37. [Online]. Available: http://doi.acm.org/10.1145/2159430.2159434

[19] K. Nakano and E. Takamichi, "An image retrieval system using fpgas," in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '03. New York, NY, USA: ACM, 2003, pp. 370–373.

[20] R. Chikhi, S. Derrien, A. Noumsi, and P. Quinton, "Combining flash memory and fpgas to efficiently implement a massively parallel algorithm for content-based image retrieval," in *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 2007, pp. 247–258.

[21] G. Knittel, "A pci-compatible fpga-coprocessor for 2d/3d image processing," in *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*. IEEE, 1996, pp. 136–145.

[22] D. Crookes, K. Benkrid, A. Bouridane, K. Alotaibi, and A. Benkrid, "Design and implementation of a high level programming environment for fpga-based image processing," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 147, no. 4, pp. 377–384, 2000.

[23] I. S. Uzun, A. Amira, and A. Bouridane, "Fpga implementations of fast fourier transforms for real-time signal and image processing," in *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 152, no. 3. IET, 2005, pp. 283–296.

[24] N. W. Group, *NVM Express*, 2015 (Accessed July 26, 2015). [Online]. Available: http://nvmexpress.org

[25] FusionIO, *FusionIO*, 2015 (Accessed July 26, 2015). [Online]. Available: http://www.fusionio.com

[26] V. Memory, *Violin Memory*, 2015 (Accessed July 26, 2015). [Online]. Available: http://www.violin-memory.com

[27] Bluespec, *Bluespec*. [Online]. Available: http://bluespec.com/

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.