Compressive Sensing on Storage Data: An Effective Solution to Alleviate I/O Bottleneck in Data-Intensive Workloads

Hosein Mohammadi Makrani, Hossein Sayadi, Sai Manoj PD, Setareh Raftirad, and Houman Homayoun
Electrical and Computer Engineering Department
George Mason University
Fairfax, USA
{hmohamm8, hsayadi, spudukot, srafatir, and hhmoayou}@gmu.edu

***Abstract*- The gap between computation speed and I/O access on modern computing systems imposes processing limitations in data-intensive applications. Employing high-end memory has proven not to enhance the performance for I/O bound applications, given the low utilization of memory bandwidth in such applications, as highlighted in recent studies. Despite several solutions to improve the performance of storage, none of them is able to shift the bottleneck from the I/O access to the memory subsystem for I/O bound applications. In this paper, we show that in the case of data-intensive multimedia applications, by using Compressive Sensing (CS), a lossy data compression method, the bottleneck is lifted from the storage, increasing the bandwidth utilization of the memory to gain further performance improvement from a high-end memory. The reconstruction of compressed data is however time and memory consuming. To address this challenge, we employ and compare the hardware and software acceleration of Orthogonal Matching Pursuit (OMP), a greedy algorithm, which solves the problem by choosing the most significant variable to reduce the least square error. Our implementation results show that CS increases memory bandwidth utilization by 1.4x and using high bandwidth memory results in 24% performance improvement. Overall, the proposed solution of CS of storage data with FPGA accelerator achieves up to 45% speedup in an end-to-end implementation by only 4.6% accuracy degradation.**

Keywords— compressive sensing; hardware accelerator; memory; I/O access;

I. INTRODUCTION

Memory wall is one of the key challenges faced by the designers in devising high-performance big-data analysis platforms. Hybrid Memory Cube (HMC), High Bandwidth Memory (HBM), and Dual Data Rate (DDRx) memory technologies have been developed to cope with the memory bandwidth bottleneck [1, 26, 27]. In the era of Big-Data, machine learning [34], and Internet-of-Things (IoT), real-time decision making is becoming non-trivial for the envisaged autonomous systems along with processing large amounts of garnered data (social networks, e-commerce, and video streaming being the primary contributors in the expansion of the data volume). As such, this introduces new challenges to efficiently process data-intensive workloads on high-performance computing (HPC) machines. Furthermore, frequent storage access in I/O intensive applications and the gap between computational power and I/O performance exacerbates the problem [2, 28]. This leads to undesirable situations, where I/O bottleneck devastates the efficiency and scaling of big-data applications.

Moreover, a substantial amount of processing time is spent in writing/reading data to/from the storage [3] in data-intensive applications. This I/O latency prevents CPU to efficiently utilize the memory subsystem [29]. Hence, using high-end memory does not bring noticeable performance benefits for I/O intensive applications, as highlighted in recent studies [4, 30].

To improve the I/O performance, in-situ [5] and in-memory data [6] processing have emerged as an effective way to substitute the traditional data analytics and overcome the increasingly severe I/O bottleneck for scientific and big-data applications running at the petascale and beyond. However, these approaches could not shift the bottleneck from the storage to memory, as the access to data in the storage is inevitable.

Data compression has been recently proposed to minimize the storage utilization and decrease I/O time [7, 8]. While latest efforts have mainly focused on lossless data compression methods, their effectiveness is limited by the small achievable compression ratio, and high compression effort [7, 8]. To address these challenges and provide a higher degree of compression ratio, in this work we propose using Compressive Sensing (CS) [10], a lossy data compression method that enables the acquisition of a signal with fewer samples than the Nyquist rate, and offer significantly higher compression rate. As we focus on the data-intensive multimedia workloads, a certain degree of data loss is tolerable. This lossy method however, does not comes for

978-1-5386-7479-6/18/$31.00 ©2018 IEEE

free, as it requires reconstruction of data using computationally expensive methods such as Orthogonal Matching Pursuit (OMP) algorithm. Recent development in the hardware accelerators and the feasibility of utilizing them in conjunction with processors such as Microsoft Catapult project [9] motivates this study to use accelerators for expediting the decompression time in order to employ compressive sensing and alleviate I/O access.

CS incurs computational complexity for processing including delay and power overhead. It also incurs inaccuracy. The contribution of this work is to understand this trade-off and find out whether compressive sensing is effective in addressing the main challenge with storage subsystem to process data intensive applications. We find that CS, by identifying a right trade-off that realized by deploying off-the-shelf accelerator solutions can shift the bottleneck from the I/O access to the memory subsystem for I/O bound data intensive multimedia applications.

The remainder of this paper is organized as follows: Section II describes the background. Section III presents the experimental setup and implementation of accelerator on FPGA. Results are presented in Section IV. Section V concludes the paper.

II. BACKGROUND

In this section, we briefly introduce data compression methods, and the reconstruction of CS.

A. Data Compression

The rationale to use compression is that the compressed data could be transmitted faster, thereby improving the performance of I/O intensive workloads, such as multimedia applications. However, decompression is a compute-intensive operation that imposes an extra processing cost and resource contention on compute node [11, 12]. Therefore, the design and choice of data compression algorithms involve trade-offs among various factors, including the degree of compression, the amount of introduced distortion (using lossy data compression), and the utilized computational resources for decompression [13].

Lossless data compression algorithms usually exploit the statistical redundancy to represent data more concisely without losing information, and can precisely reconstruct the original data from the compressed data [14]. Lossy data compression is contrasted with lossless compression and can reconstruct an approximation of the original data. In these algorithms, some loss of information is acceptable [15]. The data compression can be evaluated by compression ratio and decompression speed. The results in [8] show that a lossy method can achieve 2.27x and 5.36x more compression and decompression throughput, respectively, compared to two best known lossless methods at the same compression ratio. Compression accuracy is a new metric involved in lossy compression. Compression accuracy is a metric used to judge the difference between original data and decompressed data. The inherited deficiency of information loss limits the application area of lossy compression on the lossless requirement. Here, as we focus on the multimedia big data applications, a certain degree of data loss is tolerable.

Compressive sensing (CS) is a lossy compression technique that enables the acquisition of a signal with fewer samples than the Nyquist rate. Hence, it has a high compression ratio. While CS reduces the hardware requirements of signal acquisition [16], digital signal processing necessary to recover the original signal becomes much more involved. Though CS has several advantages, reconstruction of CS is complex and computationally intensive [17]. Recently, several proposed reconstruction algorithms show the trade-off between complexity and accuracy. The most popular reconstruction algorithm for compressive sensing is Orthogonal Matching Pursuit (OMP) [18].

It is important to note that the traditional image/video codec techniques while are effective in reducing the data size, they are limited in terms of their applicability and the functionality i.e., objective-specific. For example, the video codec is developed to keep the quality at the particular level or to reduce the decompression processing time. However, CS is a general technique that enables us to have an arbitrary compression ratio and also to have an arbitrary decompression quality [22]. It also can be used on any form of data (text, objects, images, and videos) [24].

B. Reconstruction

OMP is a greedy algorithm of less complexity that searches for closely correlated values in each iteration, shown in Figure 1 [19]. The complexity of the design increases with data length and sparsity number. OMP has two inputs: the measured signal (y) and the measurement matrix (ϕ). At each iteration (t), a column of ϕ which has strong correlation with

Table 1: Variables' definition

Variable	Definition	Example
N × N	Images Size	128 * 128
M	Measurements	42…252
k	Sparsity	32
R	Residual Matrix	M * 1
ϕ	Measurement Matrix	M * N
λ	Maximum Index after Dot Product	180
t	No. of iterations	k

OMP Reconstruction Algorithm

1: Initialize $R_0 = y$, $\phi_0 = \emptyset$, $\Lambda_0 = \emptyset$, $\phi_0 = \emptyset$ and t= 0
2: Find Index $\lambda_t = \max_{j=1...n}$ subject to $| < \phi_j R_{t-1} > |$
3: Update $\Lambda_t = \Lambda_{t-1} \cup \lambda_t$
4: Update $\phi_t = [\phi_{t-1} \ \ \Lambda_t]$
5: Solve the Least Squares Problem
$X_t = \min_x ||y - \phi_y x||^2$
6: Calculate new approximation: $\alpha_t = \phi_t x_t$
7: Calculate new residual: $R_t = y - \alpha_t$
8: Increment t, and repeat from step 2 if t<k
After all the iterations, we can find correct sparse vector

Fig. 1. OMP Algorithm

y is chosen. Least squares algorithm is used to obtain a new signal estimate. In the next step, the amount of contribution that column y provides is subtracted to obtain a residue which is used for the next iteration. Finally, after k iterations, correct set of columns are determined. The variables used in the algorithm are defined in Table 1.

The reconstruction of images through OMP is time and memory consuming. If the reconstruction issue is unanswered, the achievable performance benefits with the reduction of I/O access time are very minimal or none. Hence, software and hardware based accelerator are employed to reduce the reconstruction time. This solution also addresses the high-end memory underutilization problem highlighted in recent work for big-data applications in high-end servers [31]. In this paper, we will show that a compression technique with a high compression ratio such as compressive sensing in conjunction with FPGA [32, 33] or GPU [25] accelerator can be used to reduce the I/O access time and lift the bottleneck from the storage.

III. IMPLEMENTATION

In this section, we present a full end-to-end reconstruction of compressive sensing, a lossy data compression solution, to reduce data communication costs along the I/O path in order to shift the bottleneck of data intensive workloads from storage to memory for taking advantage of high-end memory. As the data decompression incurs computational resource contention with other analytics and given the rise of accelerators in data centers, the data decompression is offloaded to an accelerator. A detailed implementation and assessment of the accelerators in software (GPU) and hardware (FPGA) is presented here.

A. Experimental setup

To perform a comprehensive experiment and study the effect of memory subsystem on the performance of reconstruction, we use different SDRAM memory modules. Our experimental methodology is focused on the objective of understanding how memory affects the performance. For this goal, we used Intel Performance Counter Monitoring tool (PCM) [20] to understand memory as well as processor behavior. In our experiments, we collect OS-level performance information with DSTAT tool. We swept the processors' parameters when using memories with three different frequency setting of 1866 MHz, 2133 MHz, and 2400 MHz. We repeat each experiment for a different number of memory channels (1CH, 2CH, and 4CH).

For software accelerator implementation, we use two different GPU families (Tesla M2070 with 448 CUDA cores and 225W power draw, and GeForce 950 with 768 CUDA cores and 90Watt TDP) and CPU platforms. We employ CUDA 6.0 for our experiments. Moreover, cuBLAS library API is used for sorting and matrix operations. The reconstruction algorithm is also implemented in parallel on the Intel Xeon E5-2683 V4 using OpenMP. Table 2 shows our server configuration.

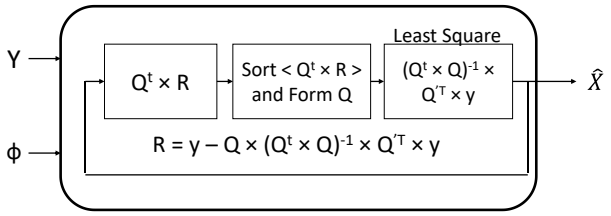


Fig. 2. Kernels of OMP

We used BigDataBench [21] for the choice of multimedia big-data applications. We emphasize that our proposed solution is effectively applicable to big-data domain applications where a certain level of loss in the data is tolerable. We therefore selected Image Segmentation (Partitioning an image into multiple segments), SIFT (Detect and describe local features in input images), and Face Detection (Detecting face in an image) workloads. We used 15000 images as a dataset for input of workloads. This data set is unstructured, organized according to the WordNet hierarchy, with 23 non-empty synsets, including categories of the plant, formation, natural object, sport, artifact, fun, guns, person, animal, and Misc. All images converted to grayscale of size 3072*2048 pixels. The average size of each image is ~5-6 MB.

B. Hardware accelerator

Based on the algorithm description, OMP is partitioned into three main kernels: dot product, sort and least square (which involves matrix inversion). These blocks are shown in Figure 2. Parallelization techniques proposed in [18, 19] are used for implementing these three kernels of OMP algorithm. The proposed accelerator can take different image sizes with sparsity up to 32.

For the implementation of hardware accelerator, Xilinx XC7VX485T Virtex-7 connected to the host processor through PCIe was used. The sparsity of k = 32 for different size of images is chosen. The high-end FPGA was chosen such that it can accommodate the design. For a sparsity of k = 32, OMP takes 6208 cycles to reconstruct each column and hence 10.12 µs for a 512 * 512 image size. Our FPGA accelerator is designed to process a vector with the size of 768 * 768 (this is the exact size of our compressed input vector file), the sparsity of 32. Table 3 shows resource utilization of Virtex-7. The following paragraphs present the architectural design information of OMP kernels:

1) Implementation of dot product kernel: OMP computes the dot product of ϕ^T (a N × M matrix) at each iteration, with a residual matrix R (a M×1 matrix) and has computational complexity of O(M×N). To leverage the parallel and pipeline architecture, a Tree multiplier is implemented, as shown in Figure 3. It is implemented for each 256 × 1 size array. Based on the matrix size, N multipliers and N - 1 adders are required

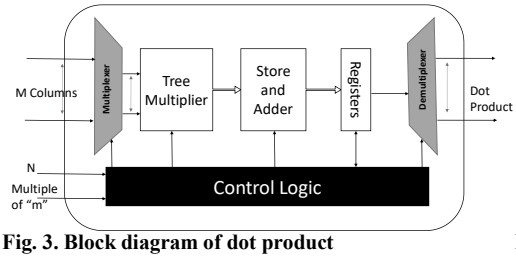


Fig. 3. Block diagram of dot product

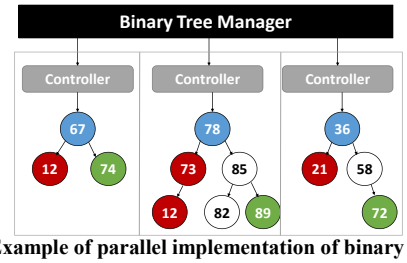


Fig. 4. Example of parallel implementation of binary tree

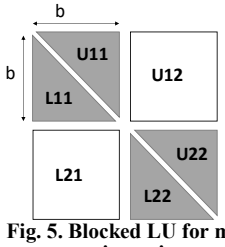


Fig. 5. Blocked LU for matrix inversion

for Tree multiplier. Therefore, the total number of operations are 2N - 1. Hence, dot product of 256 × 1 and 1 × 256 is available at every cycle. For the image size of 768 × 768, column size of ϕ^T is 256. Therefore, the dot product of 768 × 256 and 256 × 1 is computed in 768 clock cycles.

2) Implementation of sort kernel: To locate the maximum of | <ϕR> | (N×1 vector), we use the sort kernel, of computational complexity O(N). We implemented Sort algorithm by using binary tree structure [23]. If we consider S as a variable and dependent on the size of input file (image), $N/2^S$ trees are implemented in our design. In order to efficiently use parallelism, we applied concurrent sorting to $N/2^S$ as shown in Figure 4. Since our architecture needs only the highest number, we reduced the memory by cutting the left sub-tree. Each concurrent binary tree gives highest number, thereby generating $N/2^S$ highest numbers which are fed to another binary tree. Our architecture is two staged, hence 384 elements are obtained for every two cycles.

Table 2: Specification of server

Hardware Type	Parameter	Value
CPU	Model	Intel Xeon E5-2683 v4
	# Core	16
	Base Frequency	2.1 GHz
	Turbo Frequency	3 GHz
	L3 Cache	40 MB
	Memory Type Support	DDR4 1866/2133/2400
	Maximum Memory Bandwidth	76.8 GB/S
	Max Memory Channels supported	4
Disk (HDD)	Model	Seagate
	Capacity	500 GB
	Speed	7200 RPM

Table 3: Resource Utilization of Virtex-7

Resource Type	Components	Available	Used	(%)
Logic resources	Slices	75,900	52370	68%
	Logic cells	485,760	393466	81%
	CLB Flip-Flops	607,200	321817	53%
Memory resources	Distributed RAM (kb)	8,175	6050	74%
	FIFO (36 kb each)	1,030	124	12%
	Block RAM (kb)	37,080	32629	87%
Integrated IP resources	DSP Slices	2,800	1736	62%

3) Implementation of the least square kernel: Least square is the most important kernel of OMP algorithm as $(\phi^T \phi)^{-1}$ has the highest hardware implementation complexity. Since ϕ has *t* columns of size M at each iteration *t,* the new matrix (ϕ) is of size *t* × M. The result of ($\phi^T \phi$) computation is a t × t matrix. Three sub-blocks of least squares (x = $(\phi^T \phi)^{-1} \phi^T$ y) are matrix transpose, matrix multiplication, and matrix inversion. To reduce the hardware complexity and utilize minimal resources, we call matrix index in transpose order to achieve matrix transpose. For matrix multiplication, we used tree architecture discussed previously. Matrix inversion is obtained by LU decomposition leveraging the symmetric matrix. As shown in Figure 5, blocked algorithm for LU decomposition is used for efficient parallel implementation.

IV. Results

In this work, we assume that the input data is already compressed and therefore we focus on transferring compressed data from storage to memory, decompression of data, and the processing of data. In order to elucidate how high-end memory with a decompression hardware accelerator can reduce I/O bottleneck and improve performance, we present our results in three stages: stage A (without compressed data), stage B (with compressed data but without decompression accelerator), and stage C (with FPGA and GPU accelerator).

A. Stage A

To demonstrate how I/O limits the performance gain by high-end memory, we evaluate the effect of CPU frequency on the performance of studied workloads. The expectation is that increasing the core frequency will put pressure on the memory subsystem and eventually leads to gain performance from high bandwidth memory. However, our results show it has a reverse effect on data intensive application. Because increasing the core frequency exacerbate disk access and therefore prevents to gain performance. Figure 6 shows that the execution time of studied applications does not drop linearly by increasing the CPU frequency, particularly when changing frequency from 1.9 GHz to 2.6 GHz. This trend indicates that some parts of workloads are I/O bound. This conclusion can further be advocated by the active state residency (C0) of processor. This can be explained as follows: If increasing the frequency of processor reduces C0, the application is I/O bound, as when a core is waiting for I/O, the core changes its state to save power. Similarly, if the

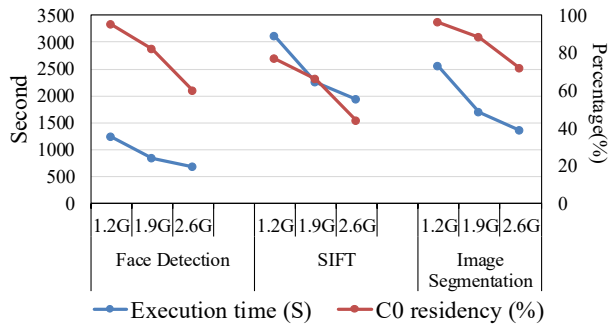


Fig. 6. Execution time & C0 residency VS core frequency

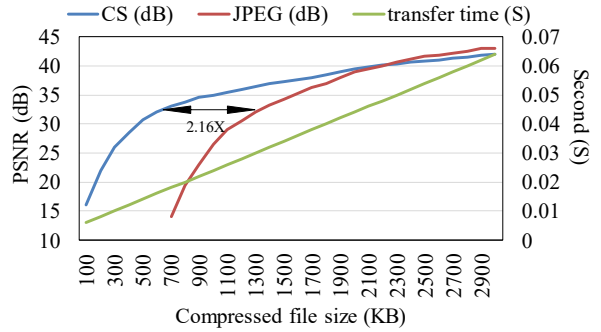


Fig. 7. Average transfer time and the quality of a compressed file

active state residency does not change, the workload is CPU bound.

Before addressing the I/O bottleneck, using high-end memory does not bring noticeable performance for data intensive applications. In the next stage, our results disclose that high bandwidth memory improves the performance only if there is high pressure on DRAM.

B. Stage B

In the second stage, we compress all the images with different compression ratios in order to determine the optimal compression rate that delivers both the quality and speedup. Figure 7 shows the time of file transfer for different input file sizes. Based on the experimental results, the vector size of image file must be less than 600000 × 1 Byte to be able lift the bottleneck from storage. Therefore, based on the trade-off between compression ratio and quality of image, we decided to compress all images into a vector with the size of 589824 × 1 (~600KB) where the PSNR of reconstructed image by CS is greater than 32 dB. Figure 7 also shows the average PSNR for different compressed file size. To show that how compress sensing works well, we compare it with JPEG (another well-known lossy image compression). The results show that CS compresses a file 2.16x (on average) more than JPEG while having the same quality (Peak signal-to-noise ratio or PSNR = 32 dB). However, this high compression ratio (which is necessary to lift the bottleneck from storage) comes with the cost of reconstruction overhead.

Table 4. Execution time and DRAM BW usage (without accelerator)

	Execution time (S)		BW usage (MBpS)	
	Before CS	After CS	Before CS	After CS
SIFT	1888	1942	1805	2752
Im. Seg.	1293	1347	6587	8395
Face Det.	659	713	3846	5556

After transferring the vector file into the memory, the processor starts to decompress the file and converts it to the original image. Then the workload uses that image to continue processing. Table 4 shows the execution time of workloads without utilizing any accelerator for decompression. The results show that using of CS increases the whole execution time while the I/O time reduces. The average time overhead of CS is 5.03%. The results also reveal that decompression puts 1.4x more processing load on the memory subsystem, on average. Figure 8 shows the impact of changing memory parameters on the average performance of studied applications before and after addressing the storage access problem.

As the CS is a lossy compression, it may result in some error on the applications outcome. Therefore, we evaluate the impact of missing data on each application and the accuracy of the results. For all three applications, we compare the output of applications with the reconstructed image and the original one. Based on our results, the average Structural Similarity (SSIM) of Image Segmentation outputs for reconstructed image and the original one is 0.936. Moreover, the average number of detected keypoints by SIFT application in a reconstructed image is 94.27% of total keypoints detected in an original image. Also, the accuracy of Face Detection application with CS is more than 98.01% (3907 faces was detected among total 3986 faces in the dataset) while it is 99.3% without CS. The results show that the accuracy of studied data-intensive multimedia applications has been only dropped 4.6% on average by CS. Figure 9 shows examples of Image segmentation and SIFT.

C. Stage C

We repeat the experiment with different memory configurations to present the effect of high-end memory (highest frequency and number of channels which provides

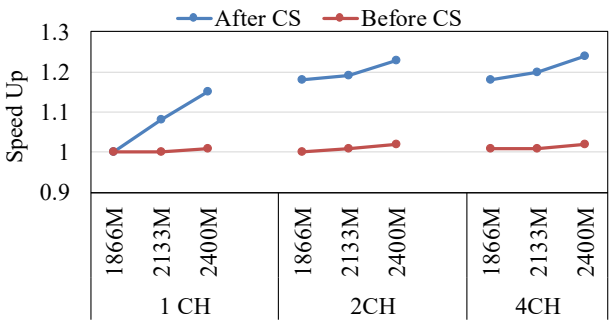
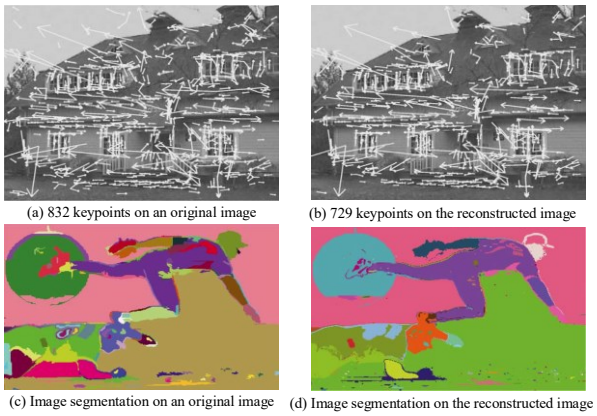


Fig. 8. Average speed up by memory parameters



(a) 832 keypoints on an original image (b) 729 keypoints on the reconstructed image

(c) Image segmentation on an original image (d) Image segmentation on the reconstructed image

Fig. 9. Example of SIFT and image segmentation

76.8 GBpS memory bandwidth) on the performance. We present time distribution of transferring, decompression, and processing data in the following tables. Table 5 presents the time overhead of input transferring from storage to server before and after compressive sensing. Table 6 shows only the time of decompression. It shows that increasing the number of memory channels and memory frequency decreases the decompression time and consequently reduces execution time. Based on the results, using high bandwidth memory is beneficial on the speedup gain of the whole application when compressive sensing is exploited. In fact, compressive sensing increases the memory bandwidth utilization of high-end memory, a problem which was highlighted in recent work [4]. Table 7 presents the processing time of each workload without considering the file transferring and decompression time after compressive sensing.

D. Discussion

We observed that the decompression time will be higher than the transferring of an original file when the system does not use an accelerator. Deploying hardware or software accelerator takes advantage of the slack time provided by transferring a compressed file from storage to the memory. In this work, the application by itself has not been accelerated and only the decompression phase has been accelerated. The results show that using accelerator improves the performance for all studied workloads. On average, FPGA implementation has the best speed up gain in an end-to-end execution (29%, 46%, and 16% for Face Detection, SIFT, and Image Segmentation respectively presented in Table 8).

Further, to evaluate the energy efficiency, we use Energy Delay Product (EDP) metric. Our obtained results presented in Figure 10 indicate that the FPGA accelerator is the most energy-efficient approach to alleviate storage access. As the power consumption is of importance in data centers, we present the breakdown of power as well in Figure 11. Finally, Figure 12 shows that C0 residency of the processor after hardware acceleration. 19.2% improvement in C0 state residency indicates that the bottleneck has been lifted from the I/O. Therefore, compressive sensing as a lossy data

Table 5. Overhead of transferring input file

Before CS	194.7 (s)
After CS	23.6 (s)

Table 6. Decompression time with acceleration

Platform	Decompression time (s)	Speed up
Bare CPU	225.3	Base
CPU + High-End Mem	181.0	1.24x
Virtex-7	0.2	1126.5x
GPU - Tesla	104.8	2.1x
GPU - GeForce	30.1	7.4x

Table 7. Processing time of different workloads

Image Segmentation	1060.2 (s)
SIFT	1000.6 (s)
Face Detection	448.1 (s)

Table 8. End-to-end execution time and speed up using FPGA accelerator

Application	Before CS	After CS + Accl	Speed up
Image Segmentation	1293	1083	16.2 %
SIFT	1888	1024	45.7 %
Face Detection	659	471	28.5 %

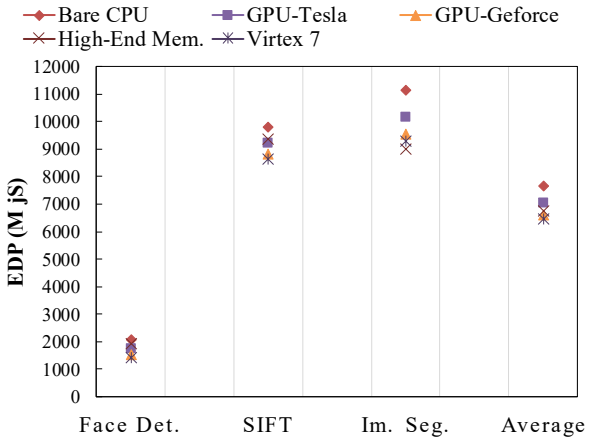


Fig. 10. EDP of applications on different platforms

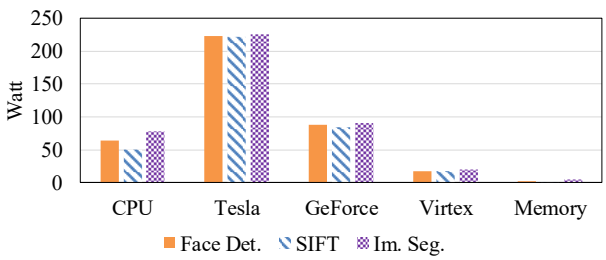


Fig. 11. Power breakdown

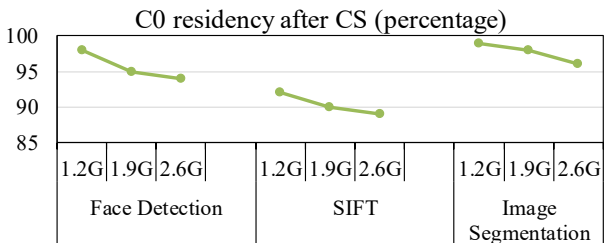


Fig. 12. CPU frequency VS C0 when using FPGA Accelerator

compression technique can be used in data-intensive multimedia applications if a suitable platform, which supports the acceleration of decompression, is exploited.

V. CONCLUSION

Despite many efforts in designing improved memory subsystems and high-speed storage, the I/O still remains a bottleneck, in particular for data intensive applications. In this work we show that using a high-end memory is not an effective solution for I/O intensive applications unless the bottleneck shifts from storage to memory. We propose data transmission with compressed sensing to reduce I/O access time. To mitigate the large processing times of decompression involved in CS, software and hardware accelerators of OMP reconstruction are implemented and deployed in this work. Compared to a system without compressive sensing, the C0 state of the CPU is increased by 19% on average, an indication of lifting bottleneck from the storage I/O. Our result shows that compressive sensing of storage in conjunction with OMP hardware (FPGA), and software (GPU) accelerated reconstruction achieves up to 46% and 11% performance gain respectively with only 4.6% accuracy degradation. Moreover, FPGA accelerator is the most energy efficient solution for CS reconstruction.

REFERENCES

[1] Lee, Dong Uk, et al. "25.2 A 1.2 V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV." Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International. IEEE, 2014.

[2] Singh, Dilpreet, and Chandan K. Reddy. "A survey on platforms for big data analytics." Journal of Big Data 2.1 (2015): 8.

[3] Chen, CL Philip, and Chun-Yang Zhang. "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data." Information Sciences 275 (2014): 314-347.

[4] Awan, Ahsan Javed, et al. "Performance characterization of in-memory data analytics on a modern cloud server." Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on. IEEE, 2015.

[5] Fabian, Nathan, et al. "The paraview coprocessing library: A scalable, general purpose in situ visualization library." Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on. IEEE, 2011.

[6] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." HotCloud 10.10-10 (2010): 95.

[7] Nicolae, Bogdan. "High Throughput Data-Compression for Cloud Storage." Globe 10 (2010): 1-12.

[8] Zou, Hongbo, et al. "Improving I/O performance with adaptive data compression for big data applications." Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International. IEEE, 2014.

[9] Caulfield, Adrian M., et al. "A cloud-scale acceleration architecture." Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016.

[10] Foucart, Simon, and Holger Rauhut. A mathematical introduction to compressive sensing. Vol. 1. No. 3. Basel: Birkhäuser, 2013.

[11] Yang, Chi, et al. "A spatiotemporal compression based approach for efficient big data processing on cloud." Journal of Computer and System Sciences 80.8 (2014): 1563-1583.

[12] Jain, Deepak, Gordon McFadden, and Brian Will. "Hardware Based Compression in Big Data." Data Compression Conference (DCC), 2016. IEEE, 2016.

[13] Zou, Hongbo, et al. "FlexAnalytics: a flexible data analytics framework for big data applications with I/O performance improvement." Big Data Research 1 (2014): 4-13.

[14] Sayood, Khalid. Introduction to data compression. Newnes, 2012.

[15] Kontoyiannis, Ioannis. "Pattern matching and lossy data compression on random fields." IEEE Transactions on Information Theory 49.4 (2003): 1047-1051.

[16] Yuan, Xin, and Raziel Haimi-Cohen. "Image Compression Based on Compressive Sensing: End-to-End Comparison with JPEG." arXiv preprint arXiv:1706.01000 (2017).

[17] Bai, Lin, et al. "High-speed compressed sensing reconstruction on FPGA using OMP and AMP." Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on. IEEE, 2012.

[18] Kulkarni, Amey M., Houman Homayoun, and Tinoosh Mohsenin. "A parallel and reconfigurable architecture for efficient OMP compressive sensing reconstruction." Proceedings of the 24th edition of the great lakes symposium on VLSI. ACM, 2014.

[19] Kulkarni, Amey, and Tinoosh Mohsenin. "Accelerating compressive sensing reconstruction OMP algorithm with CPU, GPU, FPGA and domain specific many-core." Circuits and Systems (ISCAS), 2015.

[20] Available at: https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[21] Lei et al. "Bigdatabench: A big data benchmark suite from internet services," in IEEE 20th HPCA, pp. 488-499, 2014.

[22] Xiang, Siyuan, and Lin Cai. "Scalable video coding with compressive sensing for wireless videocast." Communications (ICC), 2011 IEEE International Conference on. IEEE, 2011.

[23] Mihhailov, Dmitri, et al. "Parallel FPGA-based implementation of recursive sorting algorithms." Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on. IEEE, 2010.

[24] Zhang, Jiaxing, et al. "Impression Store: Compressive Sensing-based Storage for Big Data Analytics." HotCloud. 2014.

[25] Rozenberg, Eyal, and Peter Boncz. "Faster across the PCIe bus: a GPU library for lightweight decompression: including support for patched compression schemes." Proceedings of the 13th International Workshop on Data Management on New Hardware. ACM, 2017.

[26] Makrani, Hosein Mohammadi, and Houman Homayoun. "MeNa: A Memory Navigator for Modern Hardware in a Scale-out Environment." In 2017 IEEE International

Symposium on Workload Characterization (IISWC), pp. 2-11. IEEE, 2017.

[27] Makrani, Hosein Mohammadi, and Houman Homayoun. "Memory requirements of hadoop, spark, and MPI based big data applications on commodity server class architectures." In Workload Characterization (IISWC), 2017 IEEE International Symposium on, pp. 112-113. IEEE, 2017.

[28] Makrani, Hosein Mohammadi, et al. "Main-Memory Requirements of Big Data Applications on Commodity Server Platform." IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID), 2018.

[29] Makrani, Hosein Mohammadi, et al. "A comprehensive Memory Analysis of Data Intensive Workloads on Server Class Architecture." ACM International Symposium on Memory Systems (MEMSYS), 2018.

[30] Makrani, Hosein Mohammadi, et al. "Understanding the role of memory subsystem on performance and energy-efficiency of Hadoop applications." Eight International Green and Sustainable Computing Conference (IGSC), pp. 1-6, 2018.

[31] Makrani, Hosein Mohammadi, "Storage and Memory Characterization of Data Intensive Workloads for Bare Metal Cloud." arXiv preprint arXiv:1805.08332, 2018.

[32] Neshatpour, Katayoun, et al. "Architectural considerations for FPGA acceleration of Machine Learning Applications in MapReduce." International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation(SAMOS), 2018.

[33] Neshatpour, Katayoun, et al. "Design Space Exploration for Hardware Acceleration of Machine Learning Applications in MapReduce." The 26th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2018.

[34] Sayadi, Hossein, et al. "Customized Machine Learning-Based Hardware-Assisted Malware Detection in Embedded Devices." IEEE TrustCom, 2018.