Low Energy Sketching Engines on Many-Core Platform for Big Data Acceleration

Amey Kulkarni, Tahmid Abtahi, Emily Smith and Tinoosh Mohsenin
Department of Computer Science & Electrical Engineering
University of Maryland, Baltimore County
{ameyk1,abtahi1,smithe1,tinoosh}@umbc.edu

ABSTRACT

Almost 90% of the data available today was created within the last couple of years, thus Big Data set processing is of utmost importance. Many solutions have been investigated to increase processing speed and memory capacity, however I/O bottleneck is still a critical issue. To tackle this issue we adopt Sketching technique to reduce data communications. Reconstruction of the sketched matrix is performed using Orthogonal Matching Pursuit (OMP). Additionally we propose Gradient Descent OMP (GD-OMP) algorithm to reduce hardware complexity. Big data processing at real-time imposes rigid constraints on sketching kernel, hence to further reduce hardware overhead both algorithms are implemented on a domain specific many-core platform. GD-OMP algorithm is evaluated for image reconstruction on MATLAB and the proposed many-core architecture. Implementation results show that for large matrix sizes GD-OMP algorithm is 1.3× faster and consumes 1.4× less energy than OMP algorithm implementations. Compared to GPU and Quad-Core CPU implementations the proposed many-core reconstructs 5.4× and 9.8× faster respectively for large signal sizes with higher sparsity.

Categories and Subject Descriptors

B.2 [**ARITHMETIC AND LOGIC STRUCTURES**]: Design StylesParallel,Pipeline; B.2.4 [**High-Speed Arithmetic**]: Algorithms

Keywords

OMP;Compressive Sensing; Many-Core; High Performance and Reconfigurable Architecture

1. INTRODUCTION

In past decade one of challenges that the research community has faced is Big Data processing on hardware. Big Data is everywhere as sensors and servers produce large amount of data at every second. Thus to perform intelligent data analysis, large amounts of data needs to be processed fast. Three problems have been constantly discussed in Big Data hardware processing namely storage, processing and memory bandwidth bottleneck. Streaming applications generate a humongous amount of data and needs to be processed in real-time, however huge generated data sets require large memory storage and thus high processing times [7]. To solve this issues we propose to adapt Sketching techniques. Sketching represents data set concisely while keeping properties of data set intact. It can reduce the size of data set by upto 70%. However recovery of sketched data set should have less error rates with low computational complexity. Also, recovery algorithms should have lower hardware overhead in terms of energy and latency per operation.

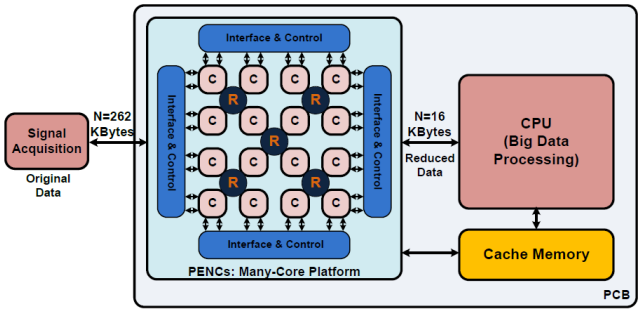

Figure 1: Low Energy Sketching Engine on Many-Core Platform for Big Data Acceleration

Big data applications need to process large amount amount of data thus it increases processing time. Therefore adapted Sketching technique to reduce processor bandwidth, needs to perform real-time tasks with low hardware overhead and should be scalable. In this paper, we take advantage of a domain specific PENC many-core architecture in order to reduce hardware overhead of sketching module. The proposed many-core platform has shown efficiency in various DSP and Machine Learning applications. Domain specific many-core architectures are considered to be better than CPUs and GPUs since it has small low power processing cores which can be operated in parallel to speed-up the application. The many-core architecture reduces processing time by implementing sketching kernels in parallel, whereas energy efficiency is increased by introducing low power processing cores with GALS routing architecture.

Though adopting sketching techniques demonstrates various advantages, reconstruction of sketched signal is computationally complex [6]. The paper implements OMP algo-

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Algorithm 1 OMP Reconstruction Algorithm

1:**Initialization**

- R_0 =y, Λ_0=$\emptyset$,Q_0 =$\emptyset$ and $t = 0$

2:**Identification**

- Find Index λ_t= $max_{j=1..n}$ subject to $| < \phi_j R_{t-1} > |$

3:**Augmentation**

- Update Λ_t=$\Lambda_{t-1} \bigcup \lambda_t$
- Update Qt=$[Q_{t-1}\ \phi_{\Lambda_t}]$

4:**Residual Update**

- Solve the Least Squares Problem
  x_t= $\min_x$ $||$ y - Q_y $x||^2$
- Calculate new approximation: α_t= Q_t x_t
- Calculate new residual: R_t= y-α_t

5: Increment t, and repeat from step 2 if $t < k$
After all the iterations, we can find correct sparse signals.

Algorithm 2 Stochastic Gradient Descend Function

1:**Initialization**

- Inputs: Q_t, $\theta_{Initialize}$, y, α
- Output: θ

2: Gradient Update

- $\theta_i = \theta_i - \alpha \sum_{j=1}^{j=m} [\theta * (Q^j) - y^j] * Q_i^j$;

3: Check Convergence
Cost Computation using Mean Square Error analysis
4: If Cost $> \gamma$, Increment i, and repeat from step 2

Algorithm 3 GD-OMP Reconstruction Algorithm

1:**Initialization**

- R_0 =y, Λ_0=$\emptyset$,Q_0 =$\emptyset$ and $t = 0$

2:**Identification**

- Find Index λ_t= $max_{j=1..n}$ subject to $| < \phi_j R_{t-1} > |$

3:**Augmentation**

- Update Λ_t=$\Lambda_{t-1} \bigcup \lambda_t$
- Update Q_t=$[Q_{t-1}\ \phi_{\Lambda_t}]$

4:**Residual Update**

- Function minimization using Gradient Descent
  x_t= *Stochastic_Gradient_Descent*(Q_t, $\theta_{Initialize}$, y, α
- Calculate new approximation: α_t= Q_t x_t
- Calculate new residual: R_t= y-α_t

5: Increment t, and repeat from step 2 if $t < k$ After all the iterations, we can find correct sparse signals.

rithm as reconstruction kernel, which has three interdependent kernels namely Dot product, Sort and Least Square kernel. The least square kernel has the highest hardware complexity among all OMP kernels. Therefore we propose Gradient Descent OMP (GD-OMP) algorithm to reduce hardware complexity of OMP algorithm while keeping error rate in satisfactory bounds. Figure 1 shows the proposed sketching framework on Many-core platform for Big Data processing acceleration. The proposed many-core architecture acts as co-processor working in tandem with a general purpose CPU allowing it to accelerate Big Data processing.

The main contributions to the paper include:

- Adopting Sketching framework to reduce I/O bottleneck.
- PENC: Power Efficient Nano Clusters, Many-Core platform.
- A Low hardware complexity GD-OMP Algorithm for reconstruction of sketched signal.
- Energy efficient implementations of OMP and GD-OMP algorithms.

2. BACKGROUND

2.1 Sketching Algorithms

In this section, we discuss OMP algorithm and proposed Gradient Descend (GD-OMP) technique. OMP is a greedy algorithm; it finds the sparsest solution iteratively by computing support of x and subtracting it from measurement vector y at every iteration. OMP has three different phases, Identification, Augmentation and Residual Update as explained in Algorithm 1. In Identification phase, index i of highest magnitude of $\phi * R$ is chosen as potential vector to find closest approximation to x. At each iteration, index i is added to the list of estimated support vectors in Augmentation phase. In the Residual update phase new estimation and residual are calculated for next iteration. In this phase first, formed Q augmented matrix is used in Least Square regression model to find linear relationship between augmented matrix Q and measured vector y. Finally, the amount of contribution that column y provides is subtracted to obtain a residue. The OMP algorithm takes k iterations to determine correct set of columns [10].

The Identification phase mainly contributes to latency of operations whereas computational complexity of OMP algorithm is dominated by Residual Update phase. Following section 2.2 discuss about proposed reduced hardware complexity modification to OMP.

2.2 Gradient Descend OMP

Least Square kernel is the most computationally intensive kernel among all in OMP algorithm. Least Square kernel consists of Matrix Inversion module which increases the hardware area and power. In this work, we propose to use Stochastic Gradient Descend technique to reduce hardware cost instead of Matrix Inversion based least square calculations. Gradient Descend is first order iterative based function minimizing technique in which α steps are taken towards positive gradient of the function. Khandekar et.al [3] shows that gradient descent function guarantees recovery under Restricted Isometric Property (RIP) conditions. Algorithm 3 shows the integration of gradient descent function (Algorithm 2) with OMP algorithm.

2.3 Proposed Many-Core Architecture:PENC

PENC is composed of 64 processing clusters (192 Cores)

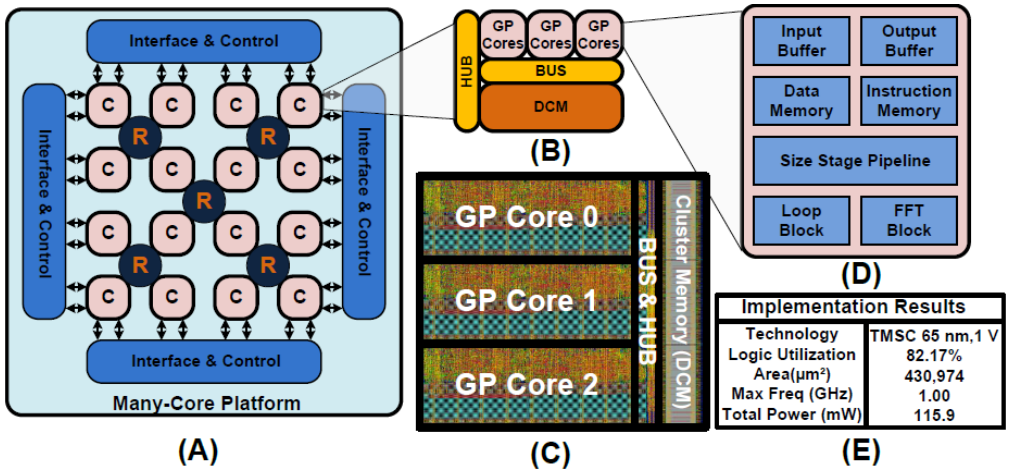

Figure 2: (A) PENC:Power Efficient Nano Clusters, Many-Core Architecture (B) Bus-based Cluster Architecture (C) Layout of Bus-based Cluster implemented in 65nm TSMC CMOS technology (D) Block Diagram of Core Architecture (E) Post Layout Implementation Results of Bus-based Cluster

connected through routers in a three-level hierarchical tree. The lowest level consists of four clusters connected by a router with five ports: one for each cluster and one for communication to the next level. Three processing nodes and a shared memory compose a cluster and communicate with each other and a low-latency bus. Figure 2 shows PENC architecture and its kernel with post layout analysis of bus-based cluster architecture on 65nm CMOS technology.

2.3.1 *Bus-Based Cluster*

Each 16-bit core consists of a six-stage processor pipeline, 128-deep instruction and data memories, and 16 registers. For all ALU instructions, the sources and destinations can be either registers or local data memory references. In either case, the read data is available before the execute stage, eliminating the need for separate `LD` and `ST` instructions for applications whose state fits in the local data memory. Register accesses are resolved in the Instruction-Decode stage, and accesses to a core's local data memory are resolved in the Memory-Decode stage.

Cores use the `IN` and `OUT` instructions to communicate with each other. When a core executes an `OUT` instruction, the data and relevant addressing information is packetized and sent to its output FIFO. When data is present in a core's output FIFO, it requests to use the bus. The bus then arbitrates between requests, only granting those whose transactions can be completed. Each core has an input FIFO, and if the input FIFO corresponding to the `OUT` is not full, the `OUT` can be completed.

The node wraps the processing core pipeline with layers of buffering and is the level that interacts with the bus. The architecture in Figure 2 shows input and output FIFOs to store data to be sent to and received from the bus. The destination core is used by the bus to forward the packet to the appropriate location, and the source core is used by the requesting node to satisfy its corresponding `IN` instruction. The destination address and data fields tell the recipient core where to store the data.

The processing core contains additional buffering on the input in the form of a 32-element content-addressable memory (CAM). It is used to store packets from the bus and allow a finite state machine to find a word where the source core field corresponds to that in the IN instruction itself. For example, if the core is executing `IN 3`, the FSM searches through the CAM to find the first word whose source core is equal to three. This word is then presented to the processing core and processing continues.

2.3.2 *Distributed Cluster Memory*

Within each cluster, three 1024x16 SRAM cells compose a Distributed Cluster Memory (DCM). The processor nodes within the cluster can all access the cluster memory via the bus. To access the memory, cores use two memory instructions: `LD` and `ST`. The maximum depth of the cluster memory is 2^{16} words since registers and data memory are both 16-bits wide and can therefore supply a 16-bit memory address.

Using data memory as operands for instructions is still beneficial to using `LD` and `ST` from an efficiency standpoint because of the one-cycle read/write capability. Referencing data from the cluster memory has latency and requires a separate instruction, which reduces the overall instructions per cycle that the pipeline can complete. However, the `LD` and `ST` instructions enable the use of a much larger addressable space, which allows the PENC to support this application.

2.3.3 *Programming and Evaluation Methodology*

Our many-core development environment includes an architecture simulator written in Java. The simulator serves as a reference implementation of the architecture; its purpose is to make testing, refining, and enhancing the architecture easier. Each task of OMP algorithm is first implemented in assembly language on every processing core using many-core simulator. ASCII files generated by many-core simulator are used to program each core individually. The simulator reads in the assembled code as well as an initial state for the register file and data memory in each core. It then models the functionality of the processor and calculates the final state of register files and data memories. For execution time and energy consumption analysis, both algorithms are implemented on the hardware model of the PENC many-core platform and simulated using Cadence NC-Verilog. The activity factor is then derived and is used by the Cadence Encounter tool for accurate power computation.The many-core simulator reports statistics such as the number of cycles

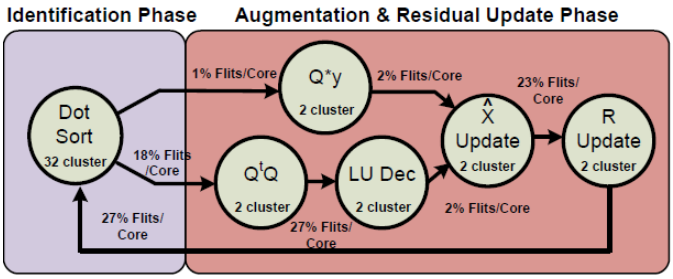

Figure 3: Task Graph for OMP algorithm mapping for 256×256 image size ($4\times$ implementation), requiring 108 cores and 32 cluster memories

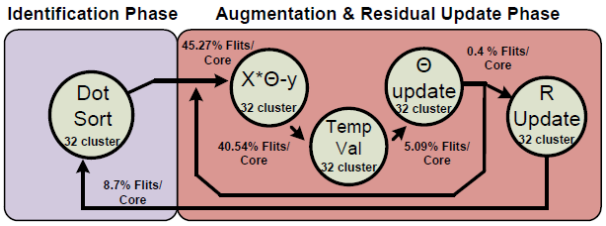

Figure 4: Task Graph for GD-OMP algorithm mapping for 256×256 image size ($4\times$ implementation), requiring 96 cores and 32 cluster memories, achieving $3.7\times$ improvement in Energy-Area-Delay Product as compared to similar OMP implementation

required for ALU, branch, and communication instructions which are used for the throughput analysis of the PENC many-core architecture.

3. APPLICATION MAPPING

3.1 OMP Algorithm Mapping

Figure 3 shows the task graph for OMP mapping on PENC platform. In identification phase, Dot and Sort kernel performs dot product between the measurement matrix (Φ) and Residual vector (R). We preloaded the measurement matrix (ϕ) into Cluster Memory. The index of highest element of the dot product is stored on data memory of one of the cores. The index λ_t of the measurement matrix is fetched to obtain Q matrix in augmentation phase. The residue update phase consists of Least Square kernel, it performs matrix-matrix multiplication, matrix-vector multiplication and matrix inversion. The Q matrix is accessed from cluster memory. At each iteration size of (Q^t) matrix increments, hence reconfigurable matrix inversion is implemented using LU decomposition algorithm. OMP algorithm has interdependent kernels, thus cores can be reused to reduce resource consumption.

We implement OMP algorithm mapping for three different image sizes i.e. 128×128, 256×256,and 512×512 with $2\times$, $4\times$, and $8\times$ parallelism among the kernels. For the serial implementation ($1\times$) of OMP algorithm for 128×128 image size with 16 sparsity (k), 9 cores are utilized with measurement matrix stored in 2 cluster memories. It has been observed that DS kernel residing in Identification phase is the bottleneck of the implementation, it needs 46% of overall clock cycles. The throughput of the $1\times$ implementation is 6760 clock cycles per column equaling 53 clock cycles per Byte. However for $2\times$ implementation 18 cores were utilized with measurement matrix stored in 4 cluster memories. For $2\times$ implementation DS kernel has been implemented on 16 cores, the cores are reused for matrix-matrix, matrix-vector and vector-vector operations. The throughput of the $2\times$ implementation is 3272 clock cycles per column equaling 25 clock cycles per Byte. Furthermore we experimented with $4\times$ and $8\times$ implementations to reduce execution time and increase throughput, the results shows $4\times$ implementation takes 14 clock cycles per Byte whereas $8\times$ implementation requires 9 clock cycles per Byte.

For image size of 256×256 with sparsity (k) of 32, the measurement matrix (Φ) is stored on 8 cluster memories. In serial implementation of 256×256, identification and residual update phase required 42% and 46% of overall clock cycles. The throughput for serial implementation is 36,455 clock cycles per column equaling 142 clock cycles per Byte. Whereas $2\times$ implementation of 256×256 image size requires 54 cores, achieving 15,156 clock cycles per column equaling 59 clock cycles per Byte throughput. Furthermore, $4\times$ and $8\times$ implementations show increase in throughput achieving 38 and 23 clock cycles per Byte respectively.

For image size of 512×512 with 64 sparsity, we could implement only till $2\times$ implementations as it requires 192 cores exhausting processing tiles. To store measurement matrix of 512×512, 28 cluster memories were required. In serial implementation of 512×512, residual update phase requires highest number of clock cycles (54%) among all phases. The throughput for serial implementation is 47098 clock cycles per column equaling 91 clock cycles per Byte whereas for $2\times$ implementation requires 37221 clock cycles per column equaling 72 clock cycles per Byte. The proposed implementations are scalable and can be reconfigured till 1024×1024 image sizes without using external cache to store measurement matrix. Executing time, area and energy efficiency of the algorithm is discussed in Section 4.

3.2 GD-OMP Algorithm Mapping

GD-OMP algorithm reduces hardware complexity of the OMP algorithm by replacing least square kernel residing in residual update phase with stochastic gradient descent function. Thus GD-OMP algorithm does not include LU decomposition, it only consist of simple matrix multiplication kernels. The stochastic gradient descent function is repeated for 100 iterations for best reconstruction quality. Identification and augmentation phase of the both OMP and GD-OMP algorithm is implemented in same way. In GD-OMP the residual phase consists of matrix-vector multiplications only, thus cores used for Identification phase are reused.

GD-OMP algorithm is also implemented for three different image sizes i.e. 128×128, 256×256,and 512×512 with $2\times$, $4\times$, and $8\times$ parallelism among the kernels. The GD-OMP can be mapped into one kernel, since its algorithmic complexity is less than the OMP algorithm. However since it needs to store the measurement matrix on shared memory, hence the number of cluster is at least ceiling (Size of measurement matrix / Size of share memory). For 128×128 image size, the serial implementation requires 6 cores and 2 cluster memories. The results show that Residual update phase dominates the execution time and takes 58% of total clock cycles. The throughput of GD-OMP serial implementation is 19,783 clock cycles per column equaling 154 clock cycles per Byte. The $2\times$ implementation achieves throughput of 7875 clock cycles per column, increasing by $2.4\times$ as

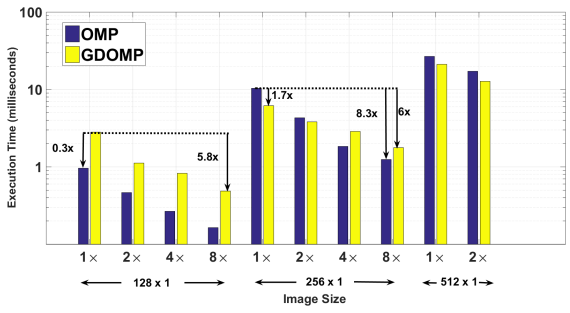

Figure 5: Single Column Execution Time Analysis of OMP and GD-OMP algorithm on Proposed PENC Platform, experiments were performed on three image sizes and different level of parallelism

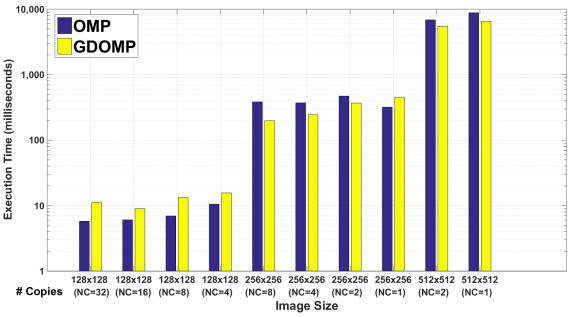

Figure 6: Execution Time Analysis with multiple copies of OMP and GD-OMP algorithm on Proposed PENC Platform for different image sizes and level of parallelism, where NC= Number of Copies made

compared to 2× implementation of OMP for 128×128 image size. However the core count also reduced by 1.5× requiring only 12 cores. For 4× ad 8× implementations computation-to-communication ratio (CCR) decreases, hence reducing performance of the GD-OMP algorithm. Throughput in terms of clock cycles per column drops by 2.9× and 3× respectively.

For 256 × 256 image size with sparsity of 32, serial implementation of GD-OMP requires 24 cores with 8 cluster memories. The throughput of GD-OMP algorithm is 21,729 clock cycles per column equaling 84 clock cycles per Byte. The GD-OMP implementation shows 1.7× improvement in clocks per column and achieves almost 2× improvement in Area-Delay-Product (ADP) as compared to serial implementation of OMP algorithm for 256 × 256. Parallel implementation (2×) also shows 1.2× improvement in clock cycles per column and 1.4×. However for 4× and 8× implementation CCR reduces thus curtailing throughput performance of GD-OMP algorithm by 1.5× as compared to OMP algorithm for both 4× and 8× implementations.

4. IMPLEMENTATION RESULTS

Big Data processing establish rigid constraints on processing time, in this paper Sketching technique is introduced before Big data processing to reduce I/O bandwidth i.e. data

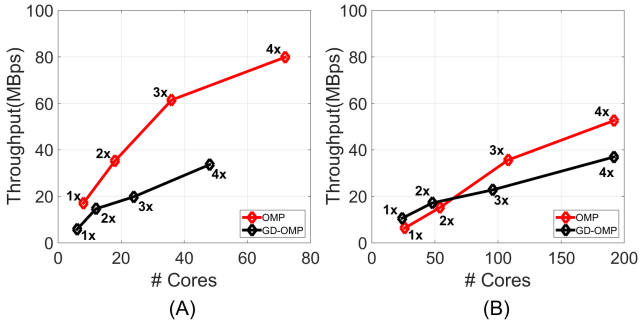

Figure 7: Throughput vs Number of Cores Analysis for OMP and GD-OMP algorithm (A) 128×128 **(B)** 256×256 **image size**

Table 1: Power Analysis of the proposed PENC many-core architecture at 1GHz and 1V

Kernel	100% Active (mW)	Stall (mW)	Leakage (mW)
Processor	37.5	19.49	0.3
Cluster-Bus	3.19	-	0.05
Hierarchical Router	97.3	-	0.8
Cluster	115.8	-	1.69

transfers between external memory and Big data processing platform as shown in Figure 1. OMP and GD-OMP sketching algorithms are mapped on PENCs, which works in tandem with general purpose CPU platform [5],[4]. Therefore sketching algorithms should have lower hardware overhead i.e. area and energy efficient with lower execution time. In this section, we first perform hardware overhead analysis and later compare our work with previously published work on ASIC, FPGA, and GPU platforms.

4.1 Execution Time Analysis

Both OMP and GD-OMP algorithms are implemented on PENCs in serial (1×), and parallel (2×, 4× and 8×). The serial implementation of the algorithm requires least number of cores but execution time is higher, whereas parallel implementations require higher number of cores with reduction in execution time. Figure 5 shows execution time analysis for single column of image size for both algorithms. We implemented multiple copies of implementation to reduce overall image reconstruction (execution) time. In case of serial implementation we could implement more copies as compared to parallel implementations, thus reducing execution time. Figure 6 execution time required to reconstruct complete image using 192 cores. For 128×128 image size least execution time is $5.7ms$ using 32 copies of OMP algorithm, whereas for 256×256 image size least execution time is $197.7ms$ using 8 copies of GD-OMP algorithm. Moreover, GD-OMP also achieves least execution time $5.4secs$ for 512×512 image size.

4.2 Energy Efficiency Analysis

Each core on the PENC platform can operate up to 1 GHZ at 1.3 V. Detailed power analysis of PENC platform is enumerated in Table 1. PENC is connected through GALS

Table 2: Throughput and Energy-Delay-Area Product (EDAP) Analysis for Different Implementations

Implementations	OMP			GD-OMP		
	#Cores	Throughput (MBps)	EDAP	Cores	Throughput (MBps)	EDAP
128×128 (1×)	9	17	0.7	6	5.8	2.2
128×128 (2×)	18	35.2	.9	12	14.6	1.2
128×128 (4×)	36	61.5	.8	24	19.8	2.8
128×128 (8×)	72	99.9	1	48	33.7	3.7
256×256 (1×)	26	6.3	595.3	24	10.6	157.6
256×256 (2×)	54	15.2	411.7	48	17.2	227.4
256×256 (4×)	108	35.6	299.6	96	22.8	529.4
256×256 (8×)	192	52.6	393	192	37	795.6
512×512 (1×)	96	9.8	46103	90	12.4	25821
512×512 (2×)	192	15.2	75085	180	20.5	36973

Table 3: Comparison with Previous Work on OMP algorithm

Platforms	Signal Size	Sparsity	Tech (nm)	Area (mm^2)	Vdd (v)	Power (W)	Execution Time (ms)
FPGA (Virtex-5) [9]	128×128	5	65	-	2.3	-	10 (μs)
ASIC [8]	256×256	8	65	0.69	1.3	-	13.69 (μs)
GPU (Nvidia GeForce) [1]	1024×1	12	90	484	1.8	67.5	37.5
CPU (Core i7) [2]	1024×1	12	45	263	1.15	16.25	68
PENC **(This Work)**	128×128	16	65	5.5	1.3	6.1	6.9
	256×256	32				5.3	370.8
	512×512	64				8.7	8814

hierarchical tree routing architecture hence the un-used clusters can be shut off. The power of the processor drops to leakage power when the clocks is halted. Also, the processor power drops by almost 50% when its stalling.

The GD-OMP implementation requires less number of cores as compared to OMP implementations, however execution time performance is best when implemented for large matrix sizes. Thus GD-OMP has lower energy efficiency for 128×128 image size and lower energy consumption of $0.49mJ$ is achieved when OMP is mapped with 9 cores and 2 cluster memories. For 256×256 image size, GD-OMP consumes low energy $6.9mJ$, 2× less as compared to OMP algorithm. GD-OMP also consumes least energy $88mJ$ among all 512×512 implementations.

4.3 Area Efficiency Analysis

In terms of our 64 cluster many-core architecture less modular area which is represented by number of cores implies we can map several modules to operate in parallel. To evaluate area efficiency we use *ThroughputPerCore* (TPC) metric. TPC is the ratio between the throughput of each design to the number of cores used for implementation. Following techniques were adapted for core area optimization:

- In OMP implementation Q^tQ and LU decomposition kernels were fused together saving 6 cores for 256×256 4× implementation. In a similar way, residual update kernels were fused together saving another 12 cores in the same implementation.

- In GD-OMP implementation DS, GD and Residual update kernels were fused together to save 48 cores in 256×256 4× implementation.

Figure 7 shows the throughput trend over cores for OMP and GD-OMP with 128×128 and 256×256 image size with different parallel mapping variations. The overall throughput increases with increased parallelism. Higher the TPC greater is the area efficiency. For 128×128 image size TPC is higher for OMP, however it needs 1.6× additional cores as compared to GD-OMP. On the other hand, for 256×256 and 512×512 implementations GD-OMP exhibits higher TDP as well as requires less number of cores as compared to OMP.

5. CONCLUSION

In this paper we adopt sketching framework to reduce I/O bottleneck for Big data acceleration. We propose PENCs many-core architecture consisting of 192 processing core connected through hierarchical tree routers to reduce hardware overhead. PENCs work as a co-processor in tandem with general purpose CPU to accelerate Big Data processing it. The GD-OMP algorithm is proposed to reduce area overhead of the sketching kernel. We implemented OMP and GD-OMP algorithm for three different image sizes with four level of parallelism. The results shows that GD-OMP is efficient for large matrix sizes, for 512×512 image size it requires 1.4× less execution time, and 2× less Energy-Delay-Area product. However OMP works better for 128×128 matrix size, it requires 2.9× less execution time, and 3.5× less Energy-Delay-Area product. Compared to GPU and Quad-Core CPU implementations the proposed many-core reconstructs 5.4× and 9.8× faster respectively for large signal sizes with higher sparsity.

6. REFERENCES

[1] M. Andrecut. Fast gpu implementation of sparse signal recovery from random projections, 2008.

[2] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin. High-speed compressed sensing reconstruction on fpga using omp and amp. In *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, pages 53–56, 2012.

[3] R. Garg and R. Khandekar. Gradient descent with sparsification: An iterative algorithm for sparse recovery with restricted isometry property. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 337–344, New York, NY, USA, 2009. ACM.

[4] M. Khavari Tavana, A. Kulkarni, A. Rahimi, T. Mohsenin, and H. Homayoun. Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14, pages 275–278, New York, NY, USA, 2014. ACM.

[5] A. Kulkarni, A. Jafari, C. Sagedy, and T. Mohsenin. Sketching-based high-performance biomedical big data processing accelerator. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, 2016.

[6] A. M. Kulkarni, H. Homayoun, and T. Mohsenin. A parallel and reconfigurable architecture for efficient omp compressive sensing reconstruction. In *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, GLSVLSI '14, pages 299–304, New York, NY, USA, 2014. ACM.

[7] K. Neshatpour, M. Malik, M. A. Ghodrat, A. Sasan, and H. Homayoun. Energy-efficient acceleration of big data analytics applications using fpgas. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 115–123, Oct 2015.

[8] J. Stanislaus and T. Mohsenin. High performance compressive sensing reconstruction hardware with qrd process. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 29–32, 2012.

[9] J. Stanislaus and T. Mohsenin. Low-complexity fpga implementation of compressive sensing reconstruction. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 671–675, Jan 2013.

[10] J. Tropp and A. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. on Information Theory*, 53(12):4655–4666, 2007.