Low Overhead CS-based Heterogeneous Framework for Big Data Acceleration

Amey Kulkarni, University of Maryland Baltimore County
Colin Shea, University of Maryland Baltimore County
Tahmid Abtahi, University of Maryland Baltimore County
Houman Homayoun, George Mason University
Tinoosh Mohsenin, University of Maryland Baltimore County

Big data processing on hardware gained immense interest among hardware research community to take advantage of fast processing and re-configurability. Though the computation latency can be reduced using hardware, big data processing cost is dominated by data transfers. In this paper, we propose a low overhead framework based on compressive sensing (CS) to reduce data transfers up to 67% without affecting signal quality. CS has two important kernels "sensing" and "reconstruction", in this paper we focus on CS reconstruction is using orthogonal matching pursuit (OMP) algorithm. We implement OMP CS reconstruction algorithm on domain specific PENC many-core platform, and low power Jetson TK1 platform consisting of ARM CPU, and K1 GPU. Detailed performance analysis of OMP algorithm on each platform suggests that PENC many-core platform has 15× and 18× less energy consumption and 16× and 8× faster reconstruction time as compared to low power ARM CPU, and K1 GPU, respectively. Furthermore we implement proposed CS-based framework on heterogeneous architecture, in which the PENC many-core architecture is used as an "accelerator" and processing is performed on ARM CPU platform. For demonstration, we integrate the proposed CS-based framework with hadoop MapReduce platform for face detection application. The results show that the proposed CS-based framework with PENC many-core as an accelerator achieves 26.15% data storage/transfer reduction, with an execution time and energy consumption overhead of 3.7% and 0.002%, respectively for 5000 image transfers. Compared to the CS-based framework implementation on low power Jetson TK1 ARM CPU+GPU platform, the PENC many-core implementation is 2.3× faster for the image reconstruction part, while achieving 29% higher performance and 34% better energy efficiency for complete face detection application on hadoop MapReduce platform.

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Compressive Sensing; Heterogeneous Architecture; Many-Core

1. INTRODUCTION

IoT and cloud computing applications such as health monitoring, video surveillance, and wireless sensor networks generate humongous amount of data every hour with unprecedented rate. To evaluate big data sets, hadoop platform using distributed processing on clusters of commodity computers is extensively used. Processing of big data sets demands large number of computational resources to enhance application run time, however these computing resources should have low latency and power. Therefore to improve energy efficiency, the trend has started to adopt hardware accelerators such as FPGAs, GPUs, and domain specific many-cores. Furthermore it has been also demonstrated that the domain specific many-core architectures are exceptional in terms of energy efficiency and throughput per area [Gautschi et al. 2016] [Tavana et al. 2015] [Conti and Benini 2015] [Stillmaker et al. 2012]. However, big data processing on many-core platforms faces various challenges such as storage and data transfer requirements. For example AsAP many-core platform [Stillmaker et al. 2012] [Liu and Baas 2013] has 16KB memory, whereas Power Efficient Nano Clusters (PENC) many-core platform [Kulkarni et al. 2016a], [Kulkarni et al. 2016c], [Page et al. 2016] has 382KB. Thus for efficient big data processing on hardware accelerators, reducing storage space and data transfers is of utmost importance. In this paper, we propose a low overhead and scalable CS-based framework to curtail storage requirements and data transfer.

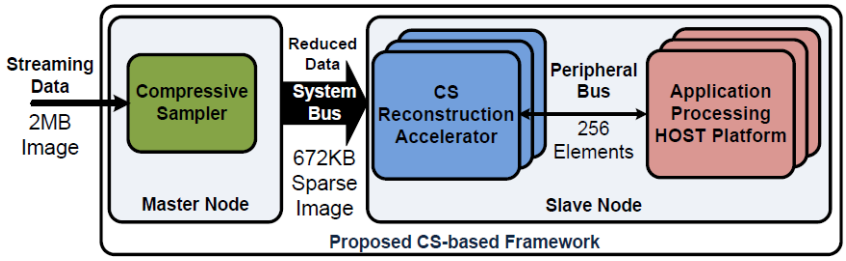

Fig. 1. Proposed low overhead CS-based heterogeneous framework for big data acceleration, compressive sampling is performed using our previous work [Kulkarni et al. 2016a], CS reconstruction is achieved on accelerator platform using OMP algorithm, and application processing is performed on host platform on reconstructed data.

In past few years various compression algorithms have been proposed to reduce data transfers, however these solutions encounter various challenges such as : 1. Inefficient for continuously changing big data sets, 2. Hardware overhead of decompression at processing platform, and 3. High decompression error rate. Compressive Sensing (CS) has demonstrated exceptional decompression (reconstruction) error rate [Tropp and Gilbert 2007], [Candès and Wakin 2010], [Needell and Vershynin 2010], however the reconstruction of CS is computationally intensive [Septimus and Steinberg 2010], [Kulkarni et al. 2014], [Kulkarni and Mohsenin 2015]. CS is a novel technique in which compression of the data set is performed by obtaining fewer linear combinations of data, thus reducing storage and data transfers. CS consists of two kernels: sensing and reconstruction; "sensing" is performed before data communications to acquire compressed measurements, whereas goal of the "reconstruction" kernel is to recover sparse data using very small number of linearly transformed measurements. Compressive sampling and OMP reconstruction architecture adopted in this paper are reconfigurable, scalable and can be readily applied to different signal processing or machine learning applications. Compressive sensing enables up to 67% of data reduction with 3.81dB SRER.

The goal of this paper is to reduce data transfers for big data acceleration to perform efficient big data processing using hardware platforms. Heterogeneous architectures have emerged as a promising solutions in energy efficient and high performance computing by allowing applications to run on a computing core that matches the resource needs more closely than a single one-size-fits-all general purpose core. Heterogeneous chip architecture integrates cores with various micro-architectures or instruction set architectures with on-chip accelerators to provide more opportunities for efficient workload mapping so that the application can find a better match among various components to achieve execution deadline and energy efficiency. Examples of heterogeneous architectures in embedded domains are Xilinx ZYNQ (CPU+FPGA), nVIDIA Tegra (CPU+GPU), Qualcomm Snapdragon (CPU+DSP+GPU) and Samsung Exynos (Big+Little CPU+GPU). Furthermore, heterogeneous big data frameworks consisting of FPGAs as accelerators such as Corse-Grained Pipelined Accelerators (CGPA) [L. et al. 2014], and Software Defined Accelerators (SODA) [Wang et al. 2015] show speedup of 4× to 46× respectively. SODA is component based programming model and consists of dataflow execution phases whereas CGPA can identify and split parallelizable sections for individual loops with complex control flow and irregular memory access. The objective of this paper is to accelerate the computationally intensive OMP CS reconstruction algorithm with low hardware overhead in terms of execution time, energy consumption, and throughput per area, while host processor performs application processing on reconstructed data.

In this paper, we propose a low overhead CS-based heterogeneous framework that can reduce data up to 67% without affecting signal quality. In the proposed framework as shown in Figure 1, we adopt deterministic sampling to sketch (sample) original signal [Kulkarni et al. 2016a], [Jafari and Mohsenin 2015] and orthogonal matching pursuit (OMP) algorithm for reconstruction of signal. We propose a flexible and fully parallel architecture for OMP which can efficiently reconstruct a wide range of signal sizes. We evaluate the proposed architecture in terms of power consumption and execution time on different platforms including, low power nvidia TK1 platform consisting of quad-core ARM CPU and K1 GPU combination, and PENC many-core platform to choose the best platform as an accelerator for CS reconstruction. Finally, for the demonstration of a real application, the proposed framework is integrated with hadoop MapReduce platform for face detection application. The performance of face detection application is shown in terms of reconstruction quality, hardware overhead cost for end-to-end framework, and overall reduction in data transfers.

The main contributions of this paper include:

— Introducing a low overhead and scalable CS-based heterogeneous framework for big data acceleration. The heterogeneous framework consists of an accelerator for low overhead CS recovery using a fully flexible and parallel reconstruction using OMP architecture and a host processor for post processing.
— OMP algorithm implementation and characterisation on various platforms including low power ARM CPU, K1 GPU combination and PENC many-core platform with respect to power, execution time and core usage parameters for wide ranges of image sizes.
— Integration of the proposed CS-based heterogeneous framework with hadoop MapReduce platform for face detection application.
— Comparison of hadoop MapReduce with CS-based framework implementation on PENC+ARM CPU platform and ARM CPU only implementation with respect to execution time and energy consumption overhead.
— Face detection quality, data transfer reduction and hardware overhead analysis of the proposed CS-based framework implementation of face detection as a case study, where PENC many-core acts as an accelerator and ARM CPU as a processing platform.

The rest of the paper is organized as follows: Section 2 presents a survey of related work on accelerating big data processing using compressive sensing. OMP algorithm used for recovery of compressive sampled data is discussed in section 3. In this paper we implement fully flexible and parallel OMP architecture on three different platforms, the processing platform architectures and implementations are discussed in section 4. Section 5 describes results and comparison analysis of OMP algorithm implementation on different platforms. Finally, section 6 demonstrates efficiency of the proposed CS-based framework with hadoop platform integration for face detection application.

2. RELATED WORK

Hardware acceleration for Big Data processing using ASIC, FPGA, GPU and domain specific many-cores interests many researchers due to its low energy consumption and fast processing capabilities. Acceleration architectures such as SparcNet [Page et al. 2017] and DianNao [Chen et al. 2016] shows speedup of about 117× and reduce up to 21× energy consumption for machine learning algorithms. SparcNet is a heterogeneous architecture consisting of ARM CPU and FPGA platforms, whereas DianNao architecture is implemented on 65nm CMOS technology. However, number of data transfers can be a potential bottleneck for large weight matrix and input data trans-

fers [Neshatpour et al. 2015], [Malik et al. 2015]. In this paper, we reduce data transfers to accelerate Big Data processing on hardware platforms using a low overhead CS-based framework.

Most researchers used hardware platforms such as FPGA or ARM micro-architectures to accelerate big data processing. A MapReduce framework on FPGA has been implemented by [Shan et al. 2010], demonstrating 31.8$\times$ speedup on FPGA as compared to CPU for RankBoost machine learning algorithm using MapReduce framework. Recent work on MapReduce framework using heterogeneous architecture has shown significant efficiency advantage over single processing architecture when running various big data benchmarks. In [Neshatpour et al. 2015] computational power of big data applications is reduced by adopting a low power xilinx zynq platform. The paper implements computationally complex kernels on FPGA to achieve speed up and energy reduction. The MapReduce application is implemented on two different micro-architectures; server and Atom platform. Overall results indicate the benefit of Atom (small cores) in terms of energy efficient hardware acceleration. [Malik et al. 2015] demonstrates that size of data and performance constraints affects choice of big vs little core-based servers for efficient big data processing. Furthermore, the paper demonstrates that required number of data transfers can be a potential bottleneck for large data transfers.

In past compressive sensing has been effectively practiced in data collection scheme for wireless sensor networks. [Liu et al. 2015] models the data collection process as a nonuniform sparse random projection (NSRP) to reduce error bound. In this paper we perform deterministic random projections to reduce hardware complexity with minimum error rate. The trend to reduce data using compressive sensing for big data applications began with [Zhang et al. 2014] and it is still in preliminary phases of research. [Zhang et al. 2014] implements compressive sensing based storage for big data analytics; authors convey that for many big data analytics workloads approximate results suffice. In [Yan et al. 2015] CS-based framework is incorporated into Hadoop and evaluated it on real web-scale production data. It shows reduction in data shuffling I/O up to 99%, and end-to-end job duration by up to 40%. In both [Zhang et al. 2014], [Yan et al. 2015] implementations were performed on software platforms thus best performance can be achieved if implemented on hardware platforms such as FPGAs or domain specific many-cores.

In our previous papers, we implemented parallel and reconfigurable OMP algorithm on FPGA virtex-7 platform to reconstruct images ranging from 128×128 to 512×512 with fixed sparsity value and 33% measurements in [Kulkarni et al. 2014]. We also discussed the effect of sparsity and number of measurements on reconstruction quality and hardware complexity of the design. Furthermore, we proposed platform independent architecture for OMP algorithm in [Kulkarni and Mohsenin 2015], [Kulkarni et al. 2014]. We also demonstrated the efficiency of CS sampling based on deterministic random matrix generator to decrease hardware overhead for machine learning application [Kulkarni et al. 2016a] [Jafari and Mohsenin 2015]. In [Kulkarni et al. 2016], [Kulkarni and Mohsenin 2017] we proposed low overhead reconstruction algorithm called GD-OMP based on OMP algorithm for big data acceleration.

3. BACKGROUND

3.1. Compressive Sensing Problem

Let us assume x to be a k-sparse signal of length N. Let ϕ be the measurement matrix projected onto the original signal, x. Measurement matrix (ϕ) must be incoherent with the basis of the sparse signal, x. If x is not sparse in its original bases, it can be transformed to another domain in which the signal is sparse. Then the measurement

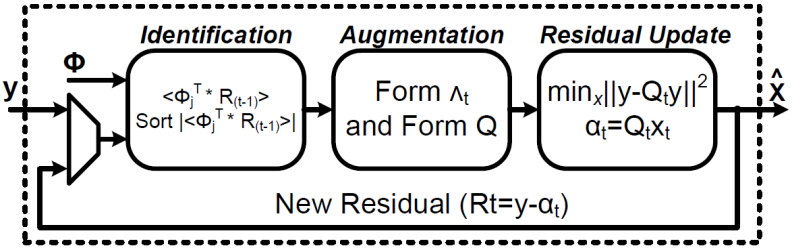

Fig. 2. **Basic block diagram of OMP CS reconstruction algorithm.**

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 Q_t=[Q_{t-1} Q_{Λ_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.

matrix has to be uncorrelated with the signal in the transformed domain [Candès and Wakin 2010]. The size of ϕ is $M \times N$, where $M<<N$ and represents the number of measurements. y is a M length vector containing the measurements obtained by the projection of ϕ onto x. Therefore, signal need to be converted to a transformed basis, Ψ, to induce sparsity and y is obtained as:

$$y = \phi\Psi x = \Phi x \tag{1}$$

In this paper, deterministic compressive sampling (DRM) technique is used to transform streaming data to lower dimensional data structure. In DRM instead of obtaining a few samples of the signal, few linear combinations are sampled. Using fewer measurements, a signal can be reconstructed almost perfectly under certain conditions [Jafari and Mohsenin 2015]. In our previous work, reconfigurable data sketching architecture using DRM technique is implemented [Kulkarni et al. 2016a]. We evaluated compressive sampling with respect to different data sizes, and impact on application error rate with respect to change in sparsity value and data reduction.

3.2. Orthogonal Matching Pursuit Algorithm

Among a variety of CS reconstruction algorithms, iterative greedy algorithms have lower complexity and low signal to reconstruction error rate [Tropp and Gilbert 2007] [Needell and Vershynin 2010]. OMP is an iterative greedy algorithm, which is

widely used due to its capability to solve large dimensional problems and competitive performance. In our previous work, we showed that, OMP algorithm is scalable to reconstruct large amount of data [Kulkarni et al. 2014] and can withstand continuously changing streaming data [Korde et al. 2013].

OMP is a greedy algorithm, which finds the sparsest solution iteratively by computing support of x and subtracting it from measurement vector y at every iteration. As shown in Figure 2 and Algorithm 1, OMP has three different phases, Identification, Augmentation and Residual Update. 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 from the Augmentation phase. The Residual update phase generates the next residual for next iteration. In this phase, first the formed Q augmented matrix is used in a Least Square regression model to find linear relationship between augmented matrix (Q) and measured vector (y). Next, the amount of contribution that column y provides is subtracted to obtain a residue. The OMP algorithm repeats sparsity (k) times to determine correct set of columns [Tropp and Gilbert 2007].

The variables used in the Algorithm 1 are defined below:

— N$\times$ N = Images size (e.g $128 \times 128...768 \times 768$)
— M = Measurements (e.g $42...252$)
— k = Sparsity (e.g 32)
— R = Residual matrix $(size : M \times 1)$
— ϕ = Measurement matrix $(size : M \times N)$
— λ = Maximum index after dot product
— t = No. of iterations (k)

The computational complexity for each step is explained below:

(1) Identification phase $< \phi R >$ requires matrix multiplication of ϕ, which is $M \times N$ matrix with R, which is a $1 \times M$ vector. Thus computational complexity of $\mathcal{O}(MN)$. Maximum of $< \phi R >$, which gives a $N \times 1$ vector. Hence, it has a computational complexity of $\mathcal{O}(N)$.
(2) In residual update phase consist of the least squares problem. At each iteration, i, Φ has i columns of size M. Hence, the new matrix, Q, is of size $i \times M$. Doing a $(Q^T Q)$ gives a $i \times i$ resulting matrix. Thus, a cost of $\mathcal{O}(iM)$. The cost of inverting this $i \times i$ matrix by LU Decomposition is $\mathcal{O}(i^3)$. The cost of $Q^T y$ is $\mathcal{O}(i^2)$.
(3) To calculate the new approximation: $\alpha_t = Q_t \; x_t$
Q is of size $i \times M$ and x is of size $1 \times i$. Thus computational complexity of $\mathcal{O}(iM)$.
(4) Compute new residual: $R_t = y - \alpha_t$, y and α_t are $M \times 1$ matrices. The subtraction at each iteration will take M computations, hence, $\mathcal{O}(M)$.
Therefore, the total cost per iteration will be $\mathcal{O}(MN)$. If the signal is k-sparse, this algorithm will be iterated k times, giving a total computation complexity of $\mathcal{O}(kMN)$.

4. PROPOSED FRAMEWORK

4.1. Processing Platform Architecture

In this paper we propose a CS-based heterogeneous framework for reducing data transfers to accelerate big data processing on hardware platforms. The heterogeneous framework consists of an accelerator for efficient CS recovery using a fully flexible and parallel reconstruction using OMP architecture and a host processor to perform post processing. An accelerator is a programmable platform adopted to perform compute-intensive CS-reconstruction while achieving low power and high-speed computations. Recent research has shown that heterogeneous architecture platforms provide significant advantages in enabling energy-efficient or area-efficient computing. For example,

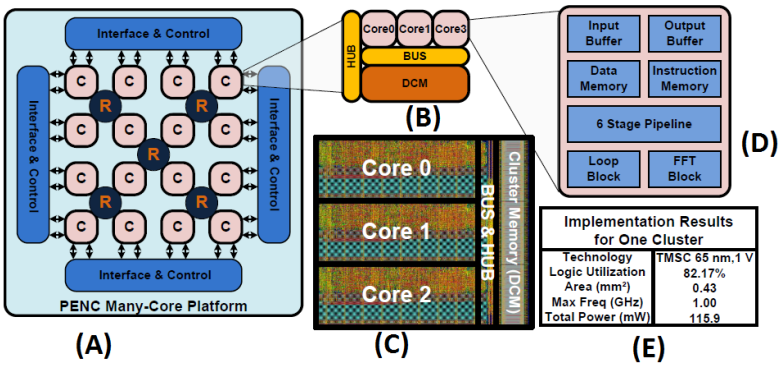

Fig. 3. (A) Power Efficient Nano Clusters (PENC), many-core architecture (B) Bus-based cluster architecture (C) Post-layout view of bus-based cluster implemented in 65nm, 1V TSMC CMOS technology (D) Block diagram of core architecture (E) Post-layout implementation results of bus-based cluster (consisting 3 cores + bus + cluster memory).

nvidia Jetson TK1 combines the benefits of the parallelism of GPU and the scalability of a multi-core CPU architecture. Although integration with GPU has provided opportunities to enhance the performance, it comes with significant power cost. To address the need for programmability, low power consumption, area efficiency and parallel computing platform, we adopt PENC many-core platform. In order to determine best platform for CS reconstruction, OMP architecture is implemented on three different platforms. nvidia Jetson TK1 platform consisting low power combination of nvidia kepler GPU and qual-core ARM CPU which provide both programmable and parallel solutions and a domain specific PENC many-core is also adopted which provides programmability, parallelism and energy efficiency. Each platform has substantially different architectural and memory sub-system, middle-ware support, and programming style. Thus to determine best accelerator, we also address performance with respect to overhead in design flow and I/O budgets.

4.1.1. ARM CPU/K1 GPU. Today's off-the-shelf processors provide a wide range of capabilities at different power targets. We use the popular nvidia Jetson TK1 platform which has low power commercial-of-the-shelf (COTS) in market, to evaluate power priorities related to implementing OMP architecture. The system-on-chip (SoC) combines the kepler graphics processing unit, GPU, and a 4-plus-1 ARM processor arrangement on a single chip. The 4-plus-1 processor configuration, also known as variable symmetric multiprocessing (vSMP,) consists of five cortex A15 ARM processors, four high performance processors and one low power processor. Each ARM CPU has a 32KB L1 data and instruction cache supporting 128-bit NEON™ general-purpose single instruction, and SIMD instructions. All processors in the 4-plus-1 configuration have shared access to a 2MB L2 cache. The K20a GPU is available to either processor power configuration and consists of a single streaming multiprocessor (SMX). The SMX has a CUDA™ compute capability of 3.2, which provides a majority of the architectural benefits of the K20c only scaled down to a single SMX group. The Jetson TK1 has 2GB of DDR3 memory that is shared between the ARM CPU and K1 GPU and is rated to run up to 933MHz. In this paper we use different combinations of the four high performance cortex A15 ARM processors and the GPU for comparison.

4.1.2. PENC Many-Core Architecture. Power Efficient Nano Clusters (PENC), many-core architecture is composed of 64 processing clusters (192 Cores) connected through routers in a three-level GALS hierarchical tree. The lowest level consists of four clus-

ters connected by a router with five ports: one for each cluster and one for communication to the next level. GALS hierarchical tree structure of PENC many-core allows us distributed computing and scalability, thus efficient embedding of an extra processing core or cluster to the chip. The lightweight cores also help to ensure that all used cores are fully utilized. While the lightweight cores are ideal for CS kernels, they often require large amounts of memory for their model data. This is addressed with the cluster-level shared memory that is interfaced to the bus. The shared memory can be accessed within the cluster on the bus and from other clusters through the router. Section 5.3 provides experimental results showing how each of these many-core features are well suited for OMP CS kernels. Figure 3 shows the block diagram of PENC many-core with the details of bus-based cluster and processor block diagram. It also gives brief idea of implementation results on 65nm, 1V TSMC CMOS technology. Below are key characteristics of the PENC many-core platform.

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 3 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 instructs the recipient core address of the data to be stored.

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.

Distributed Cluster Memory. Within each cluster, three 1024×16 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.

4.2. Platform Implementation and Experimental Set-up

The paper implements OMP algorithm on PENC many-core platform, low power ARM CPU and K1 GPU combination. We carefully implement OMP algorithm on different platforms while considering available cache size, processing cores and adopting to platform specific libraries.

4.2.1. ARM CPU/ K1 GPU: nvidia Jetson TK1. For each choice of platform, a series of benchmarks related to run-time and power are evaluated to determine the solution with the best energy efficiency. For the CPU solution, OpenBLAS library is selected based on its performance and usability, it builds the library specifically tuned for the targeted architecture. Single Program Multiple Thread (SPMT) approach is adopted to align with the GPU's execution model. In SPMT configuration, the OMP program utilizes one main process thread and several child processing threads. Each thread is assigned a chunk of the total number of columns to work based on equation (2).

$$Work\,Chunk = \frac{Number\,of\,Columns}{Active\,Threads} \tag{2}$$

The CPU program operates in three steps: 1. Memory allocation 2. Spawn child threads 3. Processing on allocated memory. Each child thread executes a series of matrix operations with minimal memory movements, and once the main thread is finished, it tears down the child threads leaving the completed sparse matrix present in the passed reference memory.

In case of GPU implementations, we use compute unified device architecture BLAS (cuBLAS) library in CUDA®version 6.5. The GPU implementation follows same steps as outlined for CPU implementation, however the work flow is modified for concurrency and CUDA streams. CUDA streams provided by nvidia platform allows concurrent implementation i.e. it has ability to execute a kernel while enabling asynchronous memory transfers. The amount of concurrency achieved by a device is dependent on a series of different capabilities such as: CUDA compute capability, the number of CUDA processing cores, the number of warps per multiprocessor, the number of threads per multiprocessor, and max registers per thread.

The TK1 platform has a single SMX containing 192 CUDA cores, a single DMA engine, and a true unified GPU/CPU memory architecture. Due to the low number of SMX resources the maximum kernel concurrency is dependent on the occupancy of the kernel in the SMX. We evaluated each kernel using the nvidia visual profiler to assess its performance and hence its importance in optimization. The TK1 uses a shared memory architecture with the ARM CPU and thus explicit memory movements are not required. However, in this approach we still utilize host-to-device and device-to-host copies for data movement.

Evaluation Methodology of TK1 Platform. The nvidia TK1 platform does not have an easily accessible power measurement sensors. Additionally, board layout exposes only a single load resistor for sub system power measurements. In this paper we present data based on total system level current consumption which includes the subsystems of the CPU and GPU. The current is measured using a sense resistor connected in series with the platform's 12.15V supply voltage [Stokke et al. 2015].

To efficiently measure only CPU and GPU performance and avoid measurements contributing to subsystems of TK1 platform we revised the original set up. Our revised set up use an external sensor system comprised of a TI INA219 and an Arduino Uno as shown in Figure 4. The INA219 is designed to measure the voltage across a sense resistor connected in series to a power rail. The electrical resistance of the sense resistor must be an order of magnitude smaller than the main circuity to avoid affect-

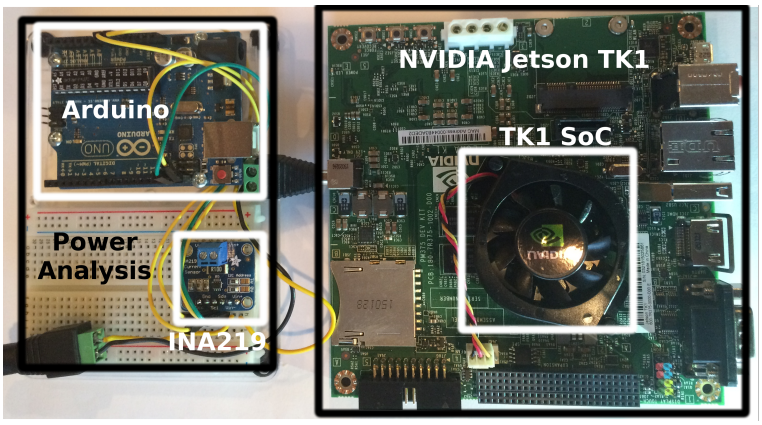

Fig. 4. TK1 current measurement setup using a TI INA219 and an Arduino Uno.

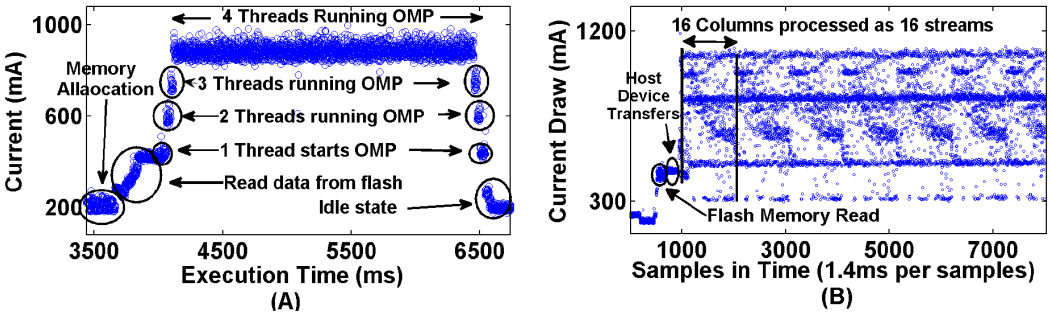

Fig. 5. **(A) Current analysis of multi-threaded OMP implementation in C++ on four ARM CPU cores at 2320.5 MHz. (B) Current analysis of OMP implementation in CUDA with one ARM CPU core and the K1 GPU at 852 MHz, with a 12.15v power supply.**

ing the total system load. The INA219 operates by sampling the voltage drop across the sense resistor and digitizes the sample to calculate the loads current drain.

For accurate power measurements inactive devices or peripherals, such as HDMI, are explicitly disabled. All external USB devices have been disconnected and only the ethernet port kept active. To eliminate OS scheduled scaling the CPU is configured to disable voltage/frequency scaling [eli 2016]. Energy estimates presented in Section 5.2 are based on the average energy consumption for a task during the OMP implementation. Figure 5A shows current analysis w.r.t execution time for multithreaded OMP implementation on CPU cores. Figure 5B shows the current analysis for the OMP algorithm on K1 GPU (nvidia Jetson TK1 platform) sampled at 667 samples per second. It shows a portion of the OMP algorithm execution focusing on the first four iterations of the processing which utilizes 16 CUDA® streams calculating sixty-four columns.

4.2.2. PENC Many-Core Platform. In this section we discuss parallel OMP algorithm mapping on PENC many-core platform. Figure 6 shows the task graph for OMP architecture mapping on PENC platform. In identification phase, dot and sort kernel performs dot product between the measurement matrix ϕ and residual vector R, and performs on the fly sorting of evaluated dot product. We preloaded the measurement matrix ϕ into DCM whereas, index λ_t of highest element of the dot product is stored on core data memory (DMEM). The index λ_t of the measurement matrix is fetched to

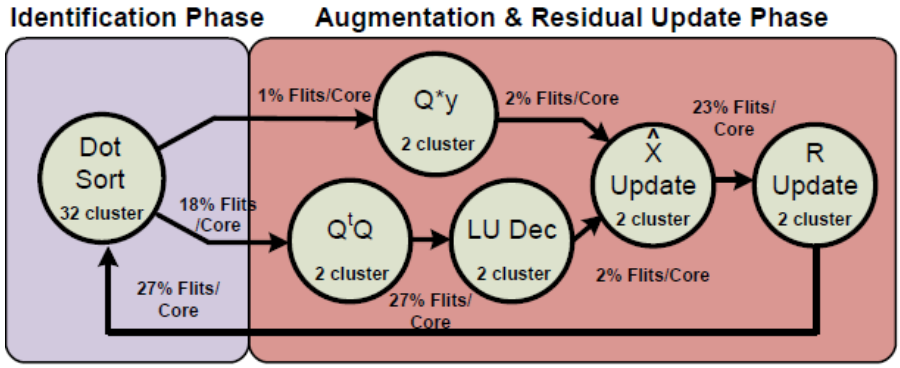

Fig. 6. Task graph for OMP algorithm mapping for 256×256 image size ($4\times$ implementation), requiring 108 cores and 32 cluster memories on PENC many-core platform.

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 DCM based on stored index λ_t in DMEM. At each iteration, the size of Q^tQ matrix increments, hence a reconfigurable matrix inversion is implemented using LU decomposition algorithm. Finally residual R is updated based on least square matrix x_t and augmented matrix Q.

OMP algorithm has interdependent kernels, thus cores can be reused to reduce resource consumption. We implemented parallel OMP algorithm in which each kernel is implemented in parallel also called as "kernel-level" parallelism. In case of kernel-level parallelism execution time per column is reduced, at the expense of core count. However in OMP algorithm each column of an image can be reconstructed independently hence keeping low core count allows us to implement more copies of OMP algorithm on PENC platform reducing overall execution time called as "core-level" parallelism. For example, the 128×128 image size, with kernel-level parallelism requires $12.41ms$ on 192 processing cores, however for core-level parallelism each column can be reconstructed in $0.96ms$ on 8 cores requiring 5.76 ms to reconstruct whole image with 24 copies on 192 cores. The advantage of core-level parallelism can be observed for higher image sizes. Analysis of reconstruction time, and energy efficiency of the algorithm is discussed in section 5.3.

PENC many-core platform has limited shared memory thus reconstructing image sizes higher than 512×512 is challenging. Therefore to allow implementation of higher matrix sizes we implement block OMP algorithm [Rouhani et al. 2015]. However block OMP needs large number of memory transfers from external cache for a new segment of ϕ matrix hence compromising overall reconstruction time. For example, we segmented 768×768 image into two segments of 384×256 which requires 32 cluster memory for storage and similarly 1024×1024 image was segmented into four segments of 256×512 each requiring 43 cluster memory.

Evaluation Methodology of PENC Many-Core Platform. 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. 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. Binary

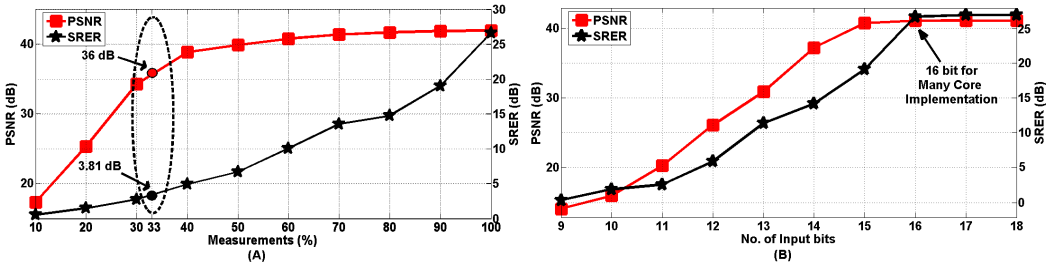

Fig. 7. (A) Quality of Reconstruction in terms of PSNR and SRER with respect to change in number of measurements. (B) Fixed point implementation effect on quality of reconstruction.

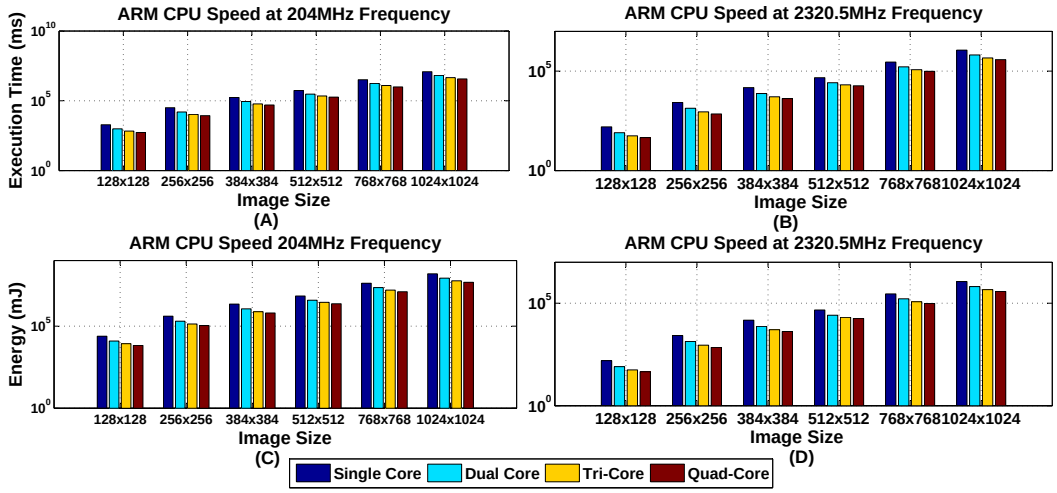

Fig. 8. Execution time and energy efficiency analysis on different ARM CPU configurations and clock rates on various image sizes (A) Execution time ARM CPU at 204MHz frequency. (B) Execution time ARM CPU at 2320.5MHz frequency. (C) Energy ARM CPU speed 204MHz frequency. (D) Energy ARM CPU speed at 2320MHz frequency.

files generated by many-core compiler are used to program each core individually. For execution time and energy consumption analysis the OMP binaries obtained using many-core compiler are mapped on the hardware model of the 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 required for ALU, branch, and communication instructions which are used for the throughput analysis of the PENC many-core architecture.

5. IMPLEMENTATION RESULT ANALYSIS

This section presents OMP CS reconstruction algorithm implementation and analysis of execution time and energy consumption on low power ARM CPU, and K1 GPU combination and PENC many-core platform. PENC many-core implementations are performed in fixed-point format, whereas CPU / GPU implementations are performed in floating point format. For each platform, signal size N ranges from 128×128 to 1024×1024, where sparsity $k = \frac{N}{8}$ and with 33% measurements, it achieves 67% data reduction. For all platforms, the measurement matrix ϕ is stored on-chip to reduce ex-

ternal memory overhead. Furthermore, Monte-Carlo simulations are performed since measurement matrix and sparse image is based on random variables. Finally in Section 5.5 we compare ARM CPU, K1 GPU and PENC many-core platform and best platform is selected for real-time application implementation.

5.1. Experimental Set-up & Performance Metric

Figure 7A shows quality of reconstruction in terms of PSNR for image reconstruction and SRER for Gaussian signal reconstruction. The SRER is measured as mean square error between original signal X and reconstructed signal $\hat{X}$. Figure 7B shows fixed point implementation analysis for reconstruction quality.

5.1.1. Experimental Set-up. Image reconstruction experiments are performed on different level of information content of $n \times n$[1] where $n = 512$, $k = 32$, $m = 128$ and $T = 1000$.

(1) Randomly generate sensing matrix using Gaussian source.
(2) Measurement vector is computed using wavelet transform and Gaussian signal vector.
(3) For each column in an image, compute measurement vector $y = \phi X$ and apply to CS reconstruction algorithm independently.
(4) Increment "T" (iteration count) and repeat experiment (step i to iii) for T times (T = 1000). Then evaluate algorithm performance using average PSNR and SRER.

5.1.2. Performance Metric

— Signal to Reconstruction Error ratio is a mean square error between original signal X and reconstructed signal $\tilde{X}$.

$$SRER = 10\ log\ (\frac{\mathbb{E}\|X\|_2}{\mathbb{E}\|X - \tilde{X}\|_2}) \tag{3}$$

— PSNR is measured via mean square error (MSE) between original image X and reconstructed image $\tilde{X}$.

$$MSE = \frac{1}{mn} \sum_{i=0}^{i=m-1} \sum_{j=0}^{j=n-1} X(i,j) - \tilde{X}(i,j)]^2 \tag{4}$$

$$PSNR = 20\ log_{10}\ (\frac{MAX_{pixel}}{\sqrt{MSE}}) \tag{5}$$

5.2. ARM CPU/K1 GPU

In this section, we discuss implementation results in terms of execution time and energy consumption analysis for low power ARM CPU/ K1 GPU combination on nvidia Jetson TK1 platform w.r.t. frequency and core usage parameters. Figure 8 shows linear increase in execution time and energy consumption with increasing computational complexity of OMP algorithm. The trend shows that, increasing the core count decreases the time necessary to solve the problem set. For all image sizes, the least amount of energy among ARM CPU configurations is observed for the quad-core configuration. Enabling second core, the average execution time is decreased by 88%, 42% drop in execution time is achieved by enabling the third core, and 22% reduction by enabling the fourth core of ARM CPU. Similarly, the energy consumption is decreased by

[1]For convenience to explain, we selected row and column size to be same. In real-time streaming data can be of different column and row sizes

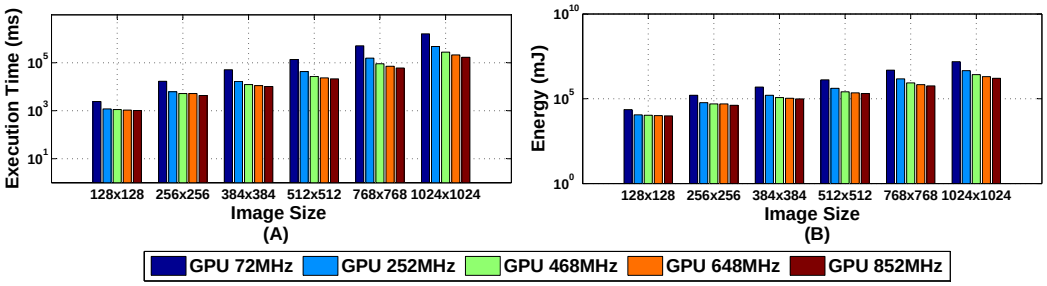

Fig. 9. Execution time and energy efficiency analysis on different K1 GPU clock rates with various images sizes at 852MHz.

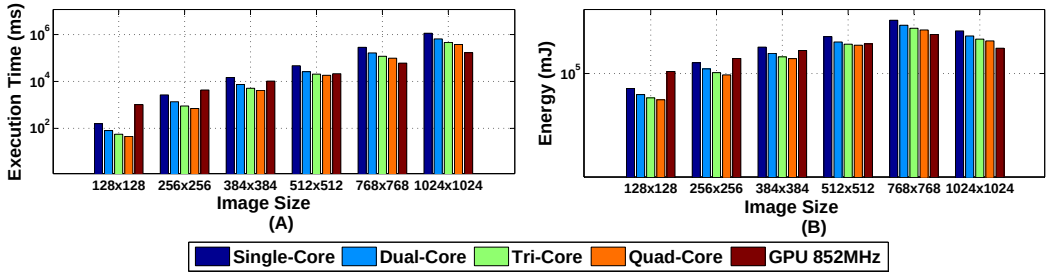

Fig. 10. Execution time and energy efficiency analysis of quad-core ARM CPU at different frequencies vs maximum GPU clock rate at 852MHz.

59% when enabling a second core, 24% reduction with three cores and 10% by enabling the fourth core of ARM CPU. Figure 9 shows increasing the clock frequency, produces a linear trend for execution time and energy consumption per frequency, however increasing clock frequency comes with the cost of increasing power consumption.

Figures 10 and 11 details the comparison in both execution time and energy consumption between the ARM CPU and K1 GPU at various configurations. Figure 10 shows analysis of quad-core ARM CPU at three different frequencies and the highest clock frequency of K1 GPU at 852MHz. The quad-core CPU at highest frequency of 2320.5MHz has the best performance up to 512×512 image size and K1 GPU performance is best for 768×768 and 1024×1024 image sizes. Figure 11 shows that the quad-core ARM CPU at its highest frequency has the best performance in terms of execution time and energy consumption for image sizes from 128×128 to 512×512, however for large image sizes K1 GPU has best performance. In conclusion, ARM CPU performs best for smaller image sizes and K1 GPU performs better for large image sizes i.e. for higher computational complexity.

5.3. PENC Many-Core

In this section, we evaluate OMP algorithm mapping on many-core platform in terms of execution time, energy consumption, and area efficiency. The execution time and energy consumption is evaluated on post layout implementation (65nm CMOS technology) using cadence tools, whereas area efficiency is measured based on throughput per core metric.

5.3.1. Execution Time Analysis. Parallel and reconfigurable OMP architecture mapping is performed to achieve higher computation-to-communication ratio. The memory transfers are hidden by computation cycles to reduce latency. Figure 12A shows ex-

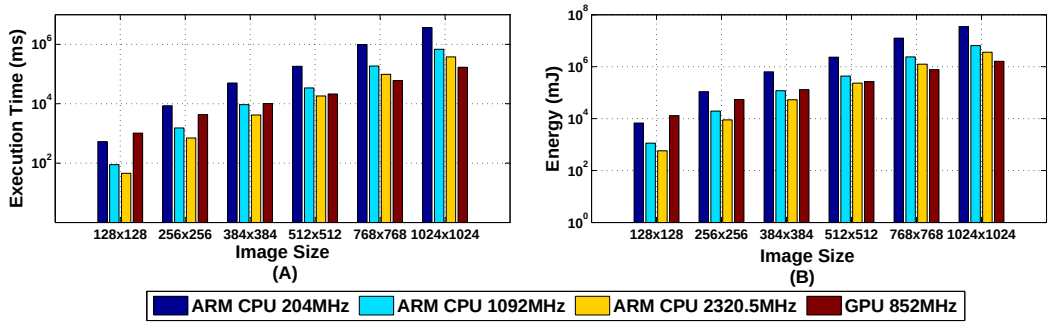

Fig. 11. Comparison of execution time and energy on different ARM CPU configurations at 2320.5MHz and K1 GPU at maximum clock rate at 852MHz.

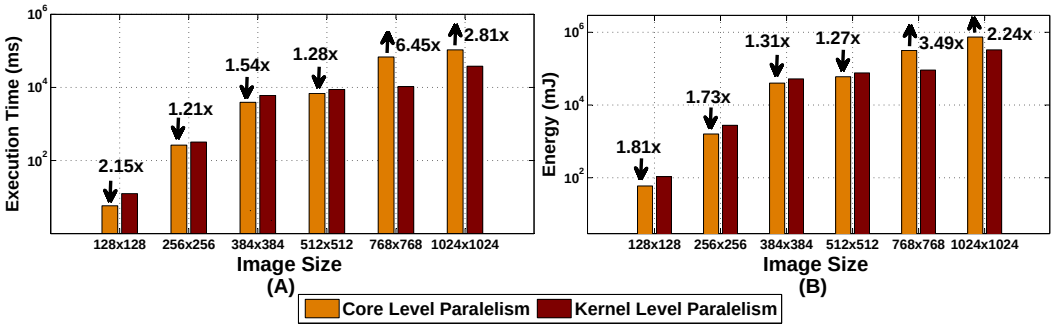

Fig. 12. Execution time and energy consumption analysis of OMP algorithm on PENC many-core platform, experiments were performed on six image sizes and different level of parallelism.

ecution time analysis for core-level and kernel-level parallelism. For 128×128 image size, core-level parallelism it requires $5.76ms$ and performs $2.1\times$ faster as compared to kernel-level parallelism. It can be observed that core-level parallelism has less latency as compared to kernel-level parallelism for small image sizes. However for large images, core-level parallelism performs better than kernel-level parallelism. For example, 1024×1024 image kernel-level parallelism is $2.8\times$ faster as compared to core-level parallelism. We also projected our findings from PENC many-core (192 core) architecture to 2000 core architecture as shown in Figure 13A. As the number of cores increased we could map more copies of independent OMP modules, thus the overall execution time decreases as a stair-case function. It can be observed that after certain number of cores, execution time saturates. The saturation point occurs when the number of cores is large enough that only one run is needed and each copy will process only one column of image data. For example 128×128 image size, 1024 cores is this saturation point.

5.3.2. Energy Efficiency Analysis. Each core on the PENC platform can operate up to 1GHz at 1v. Detailed power analysis of PENC platform is enumerated in Table I. PENC is connected through GALS hierarchical tree routing architecture hence the un-used clusters can be shut down. The power of the processor drops to leakage power when the processor clock is halted and the power consumption is reduced by almost 50% when its stalling.

Figure 12B shows the energy consumption of OMP mapping for different sizes of images on PENC many-core platform. It closely follows execution time trend, core-level

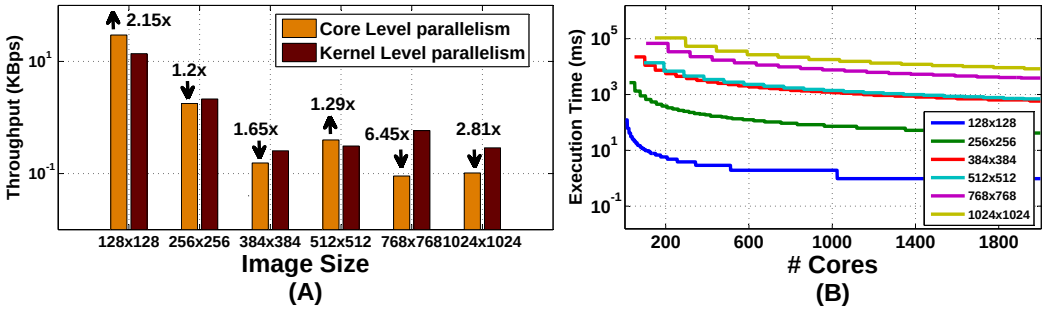

Fig. 13. (A) Execution time analysis vs number of cores with core-level parallelism. (B) Throughput per Core (TPC) analysis for core-level and kernel-level parallelism.

Table I. Post-layout power analysis of the PENC many-core architecture at 1GHz and 1V, implemented in 65nm TSMC CMOS technology.

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

parallelism consumes less energy as compared to kernel-level parallelism for small image sizes. However for large image sizes 768×768 and 1024×1024, energy consumption of kernel-level parallelism is reduced by $3.4\times$ and $2.2\times$, respectively as compared to core-level parallelism.

5.4. Area Efficiency Analysis

One of the important aspects of adopting domain specific PENC many-core is higher performance with low area. To evaluate area efficiency we use *ThroughputPerCore* (TPC) metric, where TPC is the ratio between the throughput of each design to the number of cores used for implementation as calculated in equation (6). The throughput is calculated based on number of clock cycles required to process each pixel of an image (input data).

$$Throughput\,Per\,Core = \frac{Total\,Throughput}{Number\,of\,Cores} \qquad (KBytes/Sec) \tag{6}$$

Following techniques were adapted for core area optimization:

— Dot product kernels are reused for matrix-vector multiplications in least square and residual update kernel.
— For least square kernel implementation, Q^tQ and LU decomposition kernels were fused together saving 6 cores.
— Residual update kernels i.e calculating new approximation α_t and residual R_t were fused together saving another 12 cores in the same implementation.

Figure 13B shows the throughput analysis for different image sizes, it can be observed that kernel-level parallelism has better throughput as compared to core-level parallelism irrespective of image size.

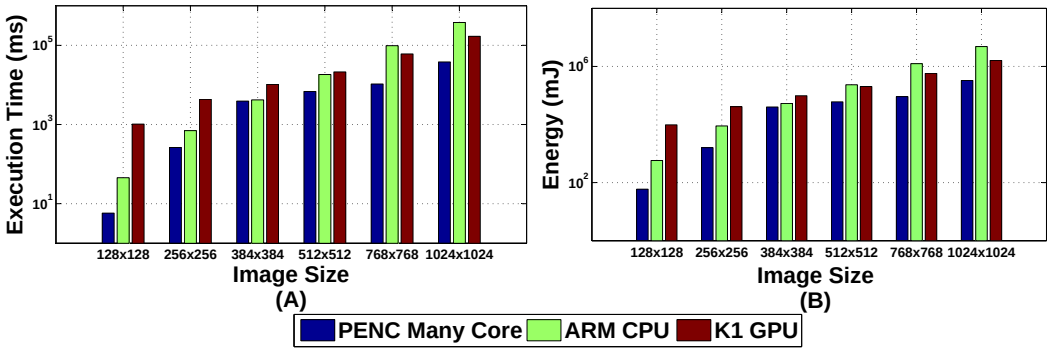

Fig. 14. Comparison of OMP algorithm on quad-core ARM CPU, K1 GPU at maximum clock rate of 2320.5MHz and 852MHz respectively, with PENC many-core implementations at 1GHz (A) execution time (B) energy consumption analysis.

Table II. Comparison of OMP algorithm implementation on different platforms.

Platforms	Signal Size	Sparsity	Tech-nology (nm)	Max Frequency (MHz)	Chip Area (mm^2)	Vdd (V)	Power (W)	Execution Time (ms)
ARM CPU **(This Work)**	128×128	16	28	2320.5	16	0.9	12.75	45.3
	256×256	32					12.75	701.7
	384×384	48					12.75	4,182
	512×512	64					12.75	18,307
	768×768	96					12.75	97,869
	1024×1024	128					12.75	378,120
K1 GPU **(This Work)**	128×128	16	28	852	37	0.9	9.52	1,023
	256×256	32					9.52	4,323
	384×384	48					9.52	10,295
	512×512	64					9.52	21,328
	768×768	96					9.52	60,543
	1024×1024	128					9.52	169,225
PENC Many-Core **(This Work)**	128×128	16	65	996	29.2	1	6.39	5.7
	256×256	32					8.67	318.83
	384×384	48					6.39	6,041
	512×512	64					8.67	6,859
	768×768	96					8.67	10,532
	1024×1024	128					8.67	38,019

5.5. Comparison Results

In last decade, different modifications to OMP algorithm are proposed to improve PSNR, SRER, Structural Similarity Index measure (SSIM) for specific application. For example, Look Ahead OMP improves the SRER performance for Gaussian signals [Swamy et al. 2014], Thresholding OMP (tOMP) improves reconstruction time, and Gradient Descent OMP (GDOMP) reduces hardware complexity for image processing application while achieving satisfactory range of PSNR [Kulkarni and Mohsenin 2017]. To allow different OMP architectures based on application requirements, processing platform needs to be programmable. In our previous work [Kulkarni et al. 2016] we implemented tOMP and GDOMP algorithm in PENC many-core platform to reduce area and execution time overhead, respectively, of CS-based framework for image processing application.

In [Kulkarni and Mohsenin 2017] we implemented OMP and its variants on 65nm CMOS technology (ASIC platform), it can reconstruct the signal 4.5$\times$ faster and consume 8$\times$ less energy as compared to PENC many-core solution, however they are not

programmable. In [Kulkarni et al. 2014] we implemented OMP algorithm on Xilinx FPGA platform, it can reconstruct 1.8× faster. In OMP algorithm, identification and residual update phase require at least "n×m" multiplications and "n" subtraction for efficient parallel implementation. These operations are performed using DSP slices, however low power FPGA devices have rigid constraints on number of DSP slices. Limiting DSP slices will restrict parallel implementation thus reducing reconstruction performance for large image sizes. In conclusion, though ASICs and FPGAs are energy efficient solutions, they are application-specific hardware. Therefore we do not compare them with PENC many-core and GPU acceleration platforms which allow programmability, reconfigurability, and energy efficient parallel implementation.

To choose best platform for CS reconstruction acceleration in CS-based framework, we evaluate each platform with respect to execution time, energy consumption and chip area as shown in Table II. Compared to ARM CPU and K1 GPU implementation PENC many-core platform performs 8× and 177× faster and saves 15× and 200× energy consumptions, respectively. Additionally considering chip area for TK1 platform in 28nm and PENC platform in 65nm, PENC many-core platform is the most efficient choice for OMP CS kernels.

6. APPLICATION CASE STUDY AND ANALYSIS

To demonstrate the efficiency of the proposed CS-based framework, we integrate it with hadoop MapReduce for face detection application as shown in Figure 15. Table II shows that PENC many-core platform has lowest hardware overhead as compared to ARM CPU and K1 GPU platforms. Therefore we select PENC many-core platform for efficient CS-OMP reconstruction in the proposed CS-based framework. We demonstrate efficiency of CS-based framework targeting two different AR-face detection database [Martinez and Benavente 1998] and UMass-FDDB [Jain and Learned-miller 2010] big data benchmarks for face detection. AR-face detection data set contains 4000 of images and UMass-FDDB data set contains 2845 images. In this section we discuss hadoop MapReduce integration with the proposed CS-based framework and its implementation results for face detection application.

The MapReduce framework is a parallel programming model, designed to process big data applications. In hadoop framework data is uploaded into the file systems before processing commences. This process of adding data requires bandwidth and storage to be available on the distributed file system and there needs to be enough available storage to allow for redundancy. Depending on the size of the data there can be a long delay before an application can be executed, as the system must first distribute the data to each of the MapReduce nodes. A particular example of this would be in the use of MapReduce for image and video processing.

We propose to use compressive sensing to reduce data storage and transfer requirements. However data reduction brings two important challenges: 1. cost of computation, 2. decompression error rate. In this paper we measure computation cost for data reduction in terms of execution time and energy consumption overhead, whereas decompression error rate is measured in terms of mis-classification rate. For the demonstration we implemented face detection application on MapReduce with CS-based framework as discussed in Section 6.1.

6.1. Face Detection Application

In last two years much research has been performed in big data and face detection, thus to demonstrate efficiency of the proposed CS-based framework we implement face detection application. We implement Haar feature based cascade classifier in OpenCV [Viola and Jones 2001],[Girshick 2015]. Cascade classifier consists of several simple classifier stages applied to region of interest until the candidate image is passed

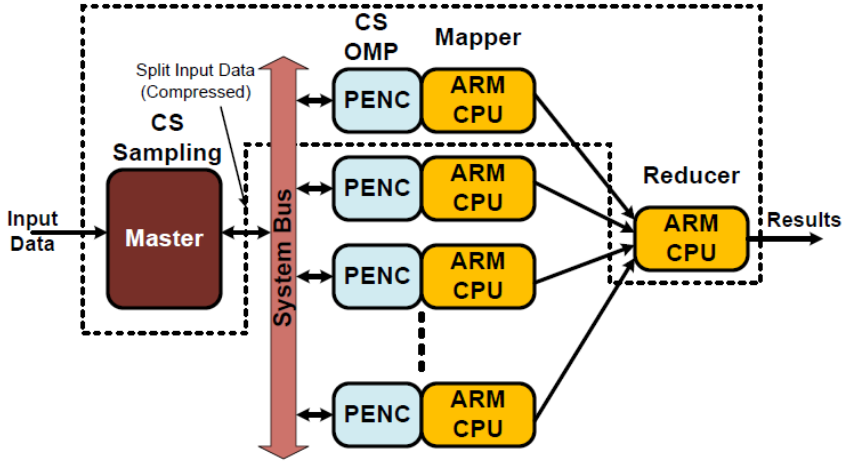

Fig. 15. Integration of proposed CS-based heterogeneous framework with hadoop MapReduce, compressive sampling is performed using our previous work [Kulkarni et al. 2016a] and compressive reconstruction is achieved using OMP algorithm on PENC many-core platform, Map and Reduce is performed on ARM CPU We perform all tests on a single mapper as shown by dotted lines.

A) Original B) Reconstructed Image C) Facial Recognition of CS Image

Fig. 16. Visual representation of image before and after each stage of processing. (A) Original Image (B) Reconstructed image from the stored compressed image (C) Successful facial identification of the reconstructed image

or rejected [Cas 2016]. Cascade classifier typically consists of two stages training and detection. In this experiment we use 20-stage cascaded classifier trained with 5000 images of size 512×512 consisting of 60% positive sample [Lienhart et al. 2003],[Sinha et al. 2006]. For detection stage, different number of images ranging from 25 to 1000 with up to 6 faces in each image. The CS-based algorithm can be efficiently adopted for different machine learning algorithms. CS-based framework has been evaluated for different machine learning algorithms including support vector machine, nave bayes, logistic regression, and k-nearest neighbors [Kulkarni et al. 2016a]. For seizure detection application, computations are reduced by 16× while energy consumption of processing is reduced up to 68%.

In following Section 6.2 we perform face detection analysis with CS-framework.

6.2. Face Detection Application Analysis

A MapReduce face detection implementation was created using both PENC many-core platform and the low power nvidia Jetson TK1 platform, where PENC is used as an accelerator and TK1 platform as the main processing engine. We adopt hadoop

Table III. **Execution time analysis of MapReduce integrated with CS-based framework for face detection application. In ARM CPU only implementation the CS reconstruction and processing performed on ARM CPU whereas in PENC+ARM CPU, CS reconstruction is performed on PENC and processing on ARM CPU.**

Size of Data	Data Transfer Reduction Through CS (%)	Application Execution Time			Execution time Overhead(%)
		ARM Only (secs)	PENC + ARM (secs)	Improvement (%)	
25 Images (11MB)	26.15	17	14	22.26	8.76
50 Images (22MB)	26.15	34	27	26.22	6.75
250 Images (108MB)	26.15	183	144	26.97	4.49
500 Images (217MB)	26.15	359	281	27.90	4.36
1000 Images (434MB)	26.15	741	583	27.01	4.08
2500 Images(1.08GB)	26.15	1,884	1,473	27.90	3.99
5000 Images(2.17GB)	26.15	3,906	3,015	29.55	3.78

Table IV. **Energy consumption analysis of MapReduce integrated with CS-based framework for face detection application. In ARM CPU only implementation the CS reconstruction and processing performed on ARM CPU whereas in PENC+ARM CPU, CS reconstruction is performed on PENC and processing on ARM CPU.**

Size of Data	Application Energy Consumption			Energy Consumption Overhead(%)
	ARM Only (Joules)	PENC + ARM (Joules)	Improvement (%)	
25 Images (11MB)	223	166	34.31	0.007
50 Images (22MB)	436	322	35.52	0.005
250 Images (108MB)	2,309	1,736	32.94	0.003
500 Images (217MB)	4,536	3,391	33.74	0.003
1000 Images (434MB)	9,349	7,060	32.41	0.003
2500 Images (1.08GB)	22,769	17,044	33.58	0.003
5000 Images (2.17GB)	46,946	35,210	33.33	0.002

2.6.3 [apc 2016] platform, and the native hadoop libraries were built from source for the platform. Figure 15 shows integration of hadoop MapReduce framework with the proposed CS-based framework, to evaluate face detection application with all tests on a single mapper. The experiment is performed in four different stages: 1. Images to be analyzed are converted to grey scale and sampled using compressive sensing with 33% measurements i.e only 33% of the image is transferred. The resulting transformed images are stored as binary files for distribution. 2. *SequenceFile* is used to create a persistent data structure for binary key-value pairs. The key is generated from the name of the file and a value is the binary data from the compressed file. It ensures that the binary data of each image is not segmented before facial recognition occurs. 3. At the consumer (mapper) end, reconstruction of the sampled image is performed using PENC many-core platform. While the reconstructed image is placed into a queue for the consumer thread, the producer thread reads in the next key-value pair. 4. Finally, the consumer thread passes reconstructed data to OpenCV's Haar feature based cascade classifiers for face detection application.

In order to analyse decompression error rate, reconstruction performance analysis is performed in terms of mis-classifcation analysis. Figure 16 shows the example of original image transition to reconstructed image and face detection on hadoop MapReduce platform with CS-based framework. Table III shows execution analysis of the proposed CS-based framework integrated with hadoop MapReduce for face detection application. The proposed CS-based framework reduces data transfers by 26% with 67% reduction in data. To demonstrate efficiency of the PENC CS reconstruction acceleration, we implemented MapReduce platform in two different cases, 1. ARM CPU is used for CS reconstruction and processing i.e for master, CS reconstruction and mapper, reducer. 2. Combination of PENC and ARM CPU, in which PENC is used for CS reconstruction and ARM CPU is used for master, mapper and reducer. Compare to ARM

CPU implementation, PENC + ARM implementation reduces application processing time by 22-29% and saves 32-34% energy consumption as shown in Table IV. Additionally we also perform hardware overhead analysis of CS reconstruction on hadoop MapReduce platform. CS framework has very low execution time overhead of 3.7% and negligible energy consumption overhead of 0.002% when tested for 5000 images. Table III and IV shows that, the increase in number of data sizes CS-based framework will have insignificant execution and energy consumption overhead.

7. CONCLUSION

In this paper we propose novel a CS-based framework for efficient big data processing on hardware. We focus on computationally complex CS-reconstruction kernel which we implemented using OMP algorithm. We performed characterisation and implementation of OMP algorithm on various platforms including domain specific PENC many-core platform, and low power Jetson TK1 platform consisting of ARM CPU, and K1 GPU. The PENC many-core platform is evaluated for execution time, energy efficiency and throughput per core on 65nm ,1v CMOS technology. Low power nvidia Jetson TK1 platform consisting of ARM CPU and K1 GPU is evaluated based on different frequencies, and different number of cores for execution time and energy consumption. All three platforms are compared based on execution time, energy consumption, and chip area for 6 different image sizes. OMP algorithm analysis to reconstruct image data from 33% of measurements shows that PENC many-core architecture consume 15$\times$ and 18$\times$ less energy and 16$\times$ and 8$\times$ faster reconstruction time as compared to low power ARM CPU, and K1 GPU respectively while achieving satisfactory range of signal quality. Based on hardware overhead analysis we chose PENC platform as "accelerator" and ARM CPU platform for big data processing. To demonstrate efficiency of the proposed framework, we integrated CS-based framework with hadoop MapReduce platform for face detection application. The master who schedules the tasks and the mapper which executes face detection are implemented on ARM CPU platform whereas, CS reconstruction is performed on PENC many-core platform. The CS-framework achieves 26.17% reduction in data transfers with very low execution overhead of 3.7% and negligible energy overhead of 0.002% when tested for 5000 number of images on PENC many-core platform. Additionally we compare PENC+ARM CPU implementation with ARM CPU only implementation, the results show that PENC+ARM CPU implementation reduces 29% execution time and saves 34% energy consumption of face detection application.

7.1. Discussions

The 3Vs in big data sets, Volume, Velocity, and Variety, provide challenges in many different aspects of real-time systems. CS-based framework is adopted to reduce data transfers while achieving low hardware overhead with lower reconstruction error rates. In this paper, the experiments are conducted on nvidia TK1 platform with data sizes from 11MBytes to 2.17GBytes, we observe that as the data size increases execution time and energy consumption overhead is reducing. The execution and energy overhead, largely depends on accelerating platform efficiency, based on available cache size, number of parallel threads, allowed maximum frequency, and its overall architecture. Therefore, careful characterization of acceleration platform needs to be performed. Based on the experiments conducted in section 5 and 6, we believe that affordable volume and velocity for the heterogeneous CS-based framework depends on acceleration platform, and processing platform.

Current embedded big data processing platform should adapt to continuously changing data, our scalable CS-based framework is practiced with three different applications including seizure detection [Kulkarni et al. 2016b], face detection, and object

identification [Kulkarni et al. 2017] to provide data reduction with low hardware overhead. In [Kulkarni et al. 2016a], CS-based framework is presented to accelerate multi-channel EEG based Seizure detection application and its performance efficiency is experimented for four different machine learning algorithms. The CS-based framework could reduce up to 72% data transfers while achieving 2% and 2.9% degradation in sensitivity and specificity. For each sample window, computations are reduced by 16$\times$ while energy consumption of processing is reduced up to 68%. In [Kulkarni et al. 2017] we show that the CS-based framework not only reduces data but also enables encryption for embedded big data applications. To demonstrate the efficiency, we performed experiments on two different datasets for object identification application. CS-based framework requires 2$\times$ less transfer time and achieves 2.25$\times$ higher throughput per watt compared to MapReduce platform. Furthermore, [Zhang et al. 2014] shows that CS-based frameworks can be adopted for real web-scale production data. Thus CS-based frameworks have potential in most of the big data applications.

REFERENCES

2016. Apache kernel description. *http://www.apache.org/* (Feb 2016).

2016. Haar feature-based cascade classifier for object detection. *http://docs.opencv.org/* (Feb 2016).

2016. Jetson TK1. *http://www.elinux.org/Jetson_TK1* (Feb 2016).

M Andrecut. 2008. Fast GPU implementation of sparse signal recovery from random projections. (2008). http://www.arxiv.org/PS_cache/arxiv/pdf/0809/0809.1833v1.pdf

R. Baraniuk and P. Steeghs. 2007. Compressive radar imaging. In *Radar Conference, 2007 IEEE*. 128 –133. DOI:http://dx.doi.org/10.1109/RADAR.2007.374203

P. Blache, H. Rabah, and A. Amira. 2012. High level prototyping and FPGA implementation of the orthogonal matching pursuit algorithm. In *Information Science, Signal Processing and their Applications (ISSPA), 2012 11th International Conference on*. 1336–1340. DOI:http://dx.doi.org/10.1109/ISSPA.2012.6310501

E. Candès and M. Wakin. 2010. An introduction to compressive sampling. *Signal Processing Magazine, IEEE* 25, 2 (Mar 2010), 21–30.

Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam. 2016. DianNao Family: Energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (Oct. 2016), 105–112. DOI:http://dx.doi.org/10.1145/2996864

Y. Chen and X. Zhang. 2010. High-speed architecture for image reconstruction based on compressive sensing. *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on* (2010), 1574–1577.

J. Constantin, A. Dogan, O. Andersson, P. Meinerzhagen, J.N. Rodrigues, D. Atienza, and A. Burg. 2012. TamaRISC-CS: An ultra-low-power application-specific processor for compressed sensing. In *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*. 159 –164. DOI:http://dx.doi.org/10.1109/VLSI-SoC.2012.6379023

F. Conti and L. Benini. 2015. A Ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15)*. EDA Consortium, San Jose, CA, USA, 683–688. http://dl.acm.org/citation.cfm?id=2755753.2755910

Y. Fang, L. Chen, J. Wu, and B. Huang. 2011. GPU implementation of orthogonal matching pursuit for compressive sensing. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*. 1044 –1047. DOI:http://dx.doi.org/10.1109/ICPADS.2011.158

M. Gautschi, M. Schaffner, F. K. Grkaynak, and L. Benini. 2016. 4.6 A 65nm CMOS 6.4-to-29.2pJ/FLOP@0.8V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. 82–83. DOI:http://dx.doi.org/10.1109/ISSCC.2016.7417917

R. Girshick. 2015. Fast R-CNN. In *The IEEE International Conference on Computer Vision (ICCV)*.

G. Huang and L. Wang. 2012. High-speed signal reconstruction with orthogonal matching pursuit via matrix inversion bypass. In *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*. 191–196. DOI:http://dx.doi.org/10.1109/SiPS.2012.26

G. Huang and L. Wang. 2014. High-speed signal reconstruction for compressive sensing applications. *Journal of Signal Processing Systems* 81, 3 (2014), 333–344. DOI:http://dx.doi.org/10.1007/s11265-014-0954-4

A. Jafari and T. Mohsenin. 2015. A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix. In *IEEE Biomedical Circuits and Systems (Biocas) Conference*.

V. Jain and E. Learned-miller. 2010. *FDDB: A benchmark for face detection in unconstrained settings*. Technical Report.

A. Korde, D. Bradley, and T. Mohsenin. 2013. Detection performance of radar compressive sensing in noisy environments. *International SPIE Conference on Defense, Security, and Sensing* (May 2013).

A. Kulkarni, T. Abtahi, E. Smith, and T. Mohsenin. 2016. Low energy sketching engines on many-core platform for big data acceleration. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI (GLSVLSI '16)*. ACM, New York, NY, USA, 57–62. DOI:http://dx.doi.org/10.1145/2902961.2902984

A. Kulkarni, H. Homayoun, and T. Mohsenin. 2014. 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)*. ACM, New York, NY, USA, 299–304. DOI:http://dx.doi.org/10.1145/2591513.2591598

A. Kulkarni, A. Jafari, C. Sagedy, and T. Mohsenin. 2016a. Sketching-based high-performance biomedical big data processing accelerator. In *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. 1138–1141.

A. Kulkarni, A. Jafari, C. Shea, and T. Mohsenin. 2016b. CS-based Secured Big Data Processing on FPGA. In *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. 201–201. DOI:http://dx.doi.org/10.1109/FCCM.2016.59

A. Kulkarni and T. Mohsenin. 2015. Accelerating compressive sensing reconstruction OMP algorithm with CPU, GPU, FPGA and domain specific many-core. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*. 970–973. DOI:http://dx.doi.org/10.1109/ISCAS.2015.7168797

A. Kulkarni and T. Mohsenin. 2017. Low overhead architectures for OMP compressive sensing reconstruction algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers* 99 (2017), 1–13. DOI:http://dx.doi.org/10.1109/TCSI.2017.2648854

A. Kulkarni, Y. Pino, M. French, and T. Mohsenin. 2016c. Real-time anomaly detection framework for many-core router through machine-learning techniques. *Journal on Emerging Technologies in Computing (JETC)* 13, 1, Article 10 (June 2016), 22 pages. DOI:http://dx.doi.org/10.1145/2827699

A. Kulkarni, C. Shea, H. Homayoun, and T. Mohsenin. 2017. LESS: Big data sketching and encryption on low power platform. In *2017 Design, Automation Test in Europe Conference Exhibition (DATE)*.

A. Kulkarni, J. Stanislaus, and T. Mohsenin. 2014. Parallel heterogeneous architectures for efficient OMP compressive sensing reconstruction. (2014). DOI:http://dx.doi.org/10.1117/12.2050530

Feng L., S. Ghosh, N. P. Johnson, and D. I. August. 2014. CGPA: Coarse-Grained Pipelined Accelerators. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. DOI:http://dx.doi.org/10.1145/2593069.2593105

R. Lienhart, A. Kuranov, and V. Pisarevsky. 2003. *Pattern Recognition: 25th DAGM Symposium, Magdeburg, Germany, September 10-12, 2003. Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Empirical analysis of detection cascades of boosted classifiers for rapid object detection, 297–304. DOI:http://dx.doi.org/10.1007/978-3-540-45243-0_39

B. Liu and B.M. Baas. 2013. Parallel AES encryption engines for many-core processor arrays. *Computers, IEEE Transactions on* 62, 3 (March 2013), 536–547. DOI:http://dx.doi.org/10.1109/TC.2011.251

X. Liu, Y. Zhu, L. Kong, C. Liu, Y. Gu, A. Vasilakos, and M. Wu. 2015. CDC: Compressive Data Collection for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems* 26, 8 (Aug 2015), 2188–2197. DOI:http://dx.doi.org/10.1109/TPDS.2014.2345257

P. Maechler, C. Studer, D.E. Bellasi, A. Maleki, A. Burg, N. Felber, H. Kaeslin, and R.G. Baraniuk. 2012. VLSI design of approximate message passing for signal restoration and compressive sensing. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on* 2, 3 (2012), 579–590. DOI:http://dx.doi.org/10.1109/JETCAS.2012.2214636

M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun. 2015. System and architecture level characterization of big data applications on big and little core server architectures. In *Big Data (Big Data), 2015 IEEE International Conference on*. 85–94. DOI:http://dx.doi.org/10.1109/BigData.2015.7363745

A. Martinez and R. Benavente. 1998. The AR Face Database. In *CVC Technical Report 24)*.

O. Maslennikow, P. Ratuszniak, and A. Sergyienko. 2007. Implementation of cholesky LLT-decomposition algorithm in FPGA-based rational fraction parallel processor. *Mixed Design of Integrated Circuits and Systems, 2007. MIXDES '07. 14th International Conference on* (2007), 287–292.

P. Meher, B.K. Mohanty, and T. Srikanthan. 2014. Area-delay efficient architecture for MP algorithm using reconfigurable inner-product circuits. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. 2628–2631.

D. Needell and R. Vershynin. 2010. Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. *Selected Topics in Signal Processing, IEEE Journal of* 4, 2 (April 2010), 310–316. DOI:http://dx.doi.org/10.1109/JSTSP.2010.2042412

K. Neshatpour, M. Malik, A. Ghodrat, Mohammad, A. Sasan, and H. Homayoun. 2015. Energy-efficient acceleration of big data analytics applications using FPGAs. In *Big Data (Big Data), 2015 IEEE International Conference on*. 115–123. DOI:http://dx.doi.org/10.1109/BigData.2015.7363748

A. Page, N. Attaran, C. Shea, H. Homayoun, and T. Mohsenin. 2016. Low-power manycore accelerator for personalized biomedical applications. In *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI (GLSVLSI '16)*. ACM, New York, NY, USA, 63–68. DOI:http://dx.doi.org/10.1145/2902961.2902986

A. Page, A. Jafari, C. Shea, and T. Mohsenin. 2017. SPARCNet: A hardware accelerator for efficient deployment of sparse convolutional networks. *Journal on Emerging Technologies in Computing (JETC)*, Article 10 (Jan. 2017), 22 pages.

H. Rabah, A. Amira, B.K. Mohanty, S. Almaadeed, and P.K. Meher. 2014. FPGA Implementation of Orthogonal Matching Pursuit for Compressive Sensing Reconstruction. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* PP, 99 (2014), 1–1. DOI:http://dx.doi.org/10.1109/TVLSI.2014.2358716

B. Rouhani, E. Songhori, A. Mirhoseini, and F. Koushanfar. 2015. SSketch: An automated framework for streaming sketch-based analysis of big data on FPGA. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. 187–194. DOI:http://dx.doi.org/10.1109/FCCM.2015.56

A. Septimus and R Steinberg. 2010. Compressive sampling hardware reconstruction. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. 3316–3319.

P. Sermwuthisarn, S. Auethavekiat, and V. Patanavijit. 2009. A fast image recovery using compressive sensing technique with block based Orthogonal Matching Pursuit. In *Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on*. 212 –215. DOI:http://dx.doi.org/10.1109/ISPACS.2009.5383863

Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang. 2010. FPMR: MapReduce framework on FPGA. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '10)*. ACM, New York, NY, USA, 93–102. DOI:http://dx.doi.org/10.1145/1723112.1723129

P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. 2006. Face Recognition by Humans: Nineteen Results All Computer Vision Researchers Should Know About. *Proc. IEEE* 94, 11 (Nov 2006), 1948–1962. DOI:http://dx.doi.org/10.1109/JPROC.2006.884093

A. Stillmaker, L. Stillmaker, and B. Baas. 2012. Fine-grained energy-efficient sorting on a many-core processor array. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*. 652–659. DOI:http://dx.doi.org/10.1109/ICPADS.2012.93

K. Stokke, H. Stensland, C. Griwodz, and P. Halvorsen. 2015. Energy Efficient Video Encoding Using the Tegra K1 Mobile Processor. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, New York, NY, USA, 81–84. DOI:http://dx.doi.org/10.1145/2713168.2713186

P.B. Swamy, S.K. Ambat, S. Chatterjee, and K.V.S. Hari. 2014. Reduced look ahead orthogonal matching pursuit. In *Communications (NCC), 2014 Twentieth National Conference on*. 1–6. DOI:http://dx.doi.org/10.1109/NCC.2014.6811329

M. Tavana, D. Pathak, M. Hajkazemi, M. Malik, I. Savidis, and H. Homayoun. 2015. Realizing complexity-effective on-chip power delivery for many-core platforms by exploiting optimized mapping. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. 581–588. DOI:http://dx.doi.org/10.1109/ICCD.2015.7357168

J. Tropp and A. Gilbert. 2007. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on* 53, 12 (dec. 2007), 4655 –4666. DOI:http://dx.doi.org/10.1109/TIT.2007.909108

D. Truong, W. Cheng, T. Mohsenin, Zhiyi Y., A. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Zhibin X., E. Work, J. Webb, P. Mejia, and B. Baas. 2009. A 167-processor computational platform in 65 nm CMOS. *Solid-State Circuits, IEEE Journal of* 44, 4 (Apr. 2009), 1130–1144. DOI:http://dx.doi.org/10.1109/JSSC.2009.2013772

P. Viola and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, Vol. 1. I–511–I–518 vol.1.

C. Wang, X. Li, and X. Zhou. 2015. SODA: Software defined FPGA based accelerators for big data. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15)*. EDA Consortium, San Jose, CA, USA, 884–887. http://dl.acm.org/citation.cfm?id=2757012.2757017

Y. Yan, J. Zhang, B. Huang, X. Sun, J. Mu, Z. Zhang, and T. Moscibroda. 2015. Distributed outlier detection using compressive sensing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. ACM, New York, NY, USA, 3–16. DOI:http://dx.doi.org/10.1145/2723372.2747641

J. Zhang, Y. Yan, L. J. Chen, M. Wang, T. Moscibroda, and Z. Zhang. 2014. Impression Store: Compressive sensing-based storage for big data analytics. In *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'14)*. USENIX Association, Berkeley, CA, USA, 1–1. http://dl.acm.org/citation.cfm?id=2696535.2696536