2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines

SSketch: An Automated Framework for Streaming Sketch-based Analysis of Big Data on FPGA

Bita Darvish Rouhani, Ebrahim M. Songhori, Azalia Mirhoseini, and Farinaz Koushanfar
Electrical and Computer Engineering Department, Rice University
Houston Texas, USA
{bita, ebrahim, azalia, farinaz}@rice.edu

***Abstract*—This paper proposes SSketch, a novel automated computing framework for FPGA-based online analysis of big data with dense (non-sparse) correlation matrices. SSketch targets *streaming* applications where each data sample can be processed only once and storage is severely limited. The stream of input data is used by SSketch for adaptive learning and updating a corresponding ensemble of lower dimensional data structures, a.k.a., a *sketch matrix*. A new sketching methodology is introduced that tailors the problem of transforming the big data with dense correlations to an ensemble of lower dimensional subspaces such that it is suitable for hardware-based acceleration performed by reconfigurable hardware. The new method is scalable, while it significantly reduces costly memory interactions and enhances matrix computation performance by leveraging coarse-grained parallelism existing in the dataset. To facilitate automation, SSketch takes advantage of a HW/SW co-design approach: It provides an Application Programming Interface (API) that can be customized for rapid prototyping of an arbitrary matrix-based data analysis algorithm. Proof-of-concept evaluations on a variety of visual datasets with more than 11 million non-zeros demonstrates up to 200 folds speedup on our hardware-accelerated realization of SSketch compared to a software-based deployment on a general purpose processor.**

***Keywords*-Streaming model; Big data; Dense matrix; FPGA; Low-rank matrix; HW/SW co-design; Matrix sketching**

I. Introduction

Computers and sensors continually generate data at an unprecedented rate. Collections of massive data are often represented as large $m \times n$ matrices, where n is the number of samples and m is the corresponding number of features. Growth of "big data" is challenging traditional matrix analysis methods like Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). Both SVD and PCA incur a large memory footprint with an $\mathcal{O}(m^2n)$ computational complexity which limit their practicability in big data regime. This disruption of convention changes the way we analyze modern datasets and makes designing scalable factorization methods, a.k.a., *sketching algorithms*, a necessity. With a properly designed sketch matrix the intended computations can be performed on an ensemble of lower dimensional structures rather than the original matrix without a significant loss.

In the big data regime, there are at least two sets of challenges that should be addressed simultaneously to optimize the performance. The first challenge class is to minimize the resource requirements for obtaining the data sketch within an error threshold in a timely manner. This favors designing sketching methods with a scalable computational complexity which can be readily scale up for analyzing a large amount of data. The second challenge class has to do with mapping of computation to increasingly heterogeneous modern architectures/accelerators. The cost of computing on these architectures is dominated by message passing for moving the data to/from the memory and inter-cores. What exacerbates the cost is the iterative nature of dense matrix calculations that require multiple rounds of message passing. To optimize the cost, communication between the processors and memory hierarchy levels must be minimized. Therefore, the trade-off between memory communication and redundant local calculations should be carefully leveraged to improve the performance of iterative computations.

Several big data analysis applications require real-time and online processing of streaming data, where storing the original big matrix becomes superfluous and the data is read at most once [1]. In these scenarios, the sketch has to be found online. A large body of earlier work demonstrated the efficiency of using custom hardware for acceleration of traditional matrix sketching algorithms such as QR [2], LU [3], Cholesky [4], and SVD [5]. However, the existing hardware-accelerated sketching methods either have a higher-than-linear complexity [6], or are non-adaptive for online sketching [5]. They are thus unsuitable for streaming applications and big data analysis with dense correlation matrices. A recent theoretical solution for scalable sketching of big data matrices is presented in [1], which also relies on running SVD on the sketch matrix. Even this method does not scale to larger sketch sizes. To the best of our knowledge, no hardware acceleration for streaming sketches has been reported in the literature.

Despite the apparent density and dimensionality of big data, in many settings, the information may lie on a single or a union of lower dimensional subspaces [7]. We propose SSketch, a novel automated framework for efficient analysis and hardware-based acceleration of massive and densely correlated datasets in streaming applications. It leverages the *hybrid structure* of data as an ensemble of lower dimensional subspaces. SSketch algorithm is a scalable approach for dynamic sketching of massive datasets that works by factorizing the original (densely correlated) large matrix into two new matrices: (i) a dense but much smaller *dictionary matrix* which includes a set of atoms learned from the input data, and (ii) a large *block-sparse matrix* where the blocks are organized such that the subsequent matrix computations incur a minimal amount of message passings on the target platform.

As the stream of input data arrives, SSketch adaptively learns from the incoming vectors and updates the sketch of the collection. An accompanying API is also provided by our work so designers can utilize the scalable SSketch framework for rapid prototyping of an arbitrary matrix-based data analysis

978-1-4799-9969-9/15 $31.00 © 2015 IEEE
DOI 10.1109/FCCM.2015.56

algorithm. SSketch and its API target a widely used class of data analysis algorithms that model the data dependencies by iteratively updating a set of matrix parameters, including but not limited to most regression methods, belief propagation, expectation maximization, and stochastic optimizations [8].

Our framework addresses the big data learning problem by using the SSketch's block-sparse matrix and applying an efficient, greedy routine called Orthogonal Matching Pursuit (OMP) on each sample independently. Note that OMP is a key computational kernel that dominates the performance of many sparse reconstruction algorithms. Given the wide range of applications, it is thus not surprising that a large number of OMP implementations on GPUs [9], ASICs [10] and FPGAs [11] have been reported in the literature. However, the prior work on FPGA had focused on fixed-point number format. In addition, most earlier hardware-accelerated OMP designs are restricted to fixed and small matrix sizes and can handle only a few number of OMP iterations [21], [11], [12]. Such designs cannot be readily applied to massive, dynamic datasets. In contrast, SSketch's scalable methodology introduces a novel approach that enables applying OMP on matrices with varying large sizes and supports arbitrary number of iterations.

SSketch utilizes the abundant hardware resources on current FPGAs to provide a scalable, floating-point implementation of OMP for sketching purposes. One may speculate that GPUs may show a better acceleration performance than FPGAs. However, the performance of GPU accelerators is limited in our application because of two main reasons. First, for streaming applications, the memory hierarchy in GPUs increases the overhead in communication and thus reduces the throughput of the whole system. Second, in SSketch framework the number of required operations to compute the sketch of each individual sample depends on the input data structure and may vary from one sample to the other. Thus, the GPU's applicability is reduced due to its Single Instruction Multiple Data (SIMD) architecture.

The explicit contributions of this paper are as follows:

- We propose SSketch, a novel communication-minimizing framework for online (streaming) large matrix computation. SSketch adaptively learns the hybrid structure of the input data as an ensemble of lower dimensional subspaces and efficiently forms the sketch matrix of the ensemble.
- We develop a novel streaming factorization method for FPGA acceleration. Our sketching algorithm benefits from a fixed, low memory footprint and an $\mathcal{O}(mn)$ computational complexity.
- We design an API to facilitate automation and adaptation of SSketch's scalable and online matrix sketching method for rapid prototyping of an arbitrary matrix-based data analysis algorithm.
- We devise SSketch with a scalable, floating-point implementation of Orthogonal Matching Pursuit (OMP) algorithm on FPGA.
- We evaluate our framework with three different massive datasets. Our evaluations corroborate SSketch scalability and practicability. We compare the SSketch runtime to a software realization on CPU and also report its overhead.

II. Background and Preliminaries

A. Streaming Model

Modern data matrices are often extremely large which require distribution of computation beyond a single core. Some prominent examples of such massive datasets include image acquisition, medical signals, recommendation systems, wireless communication, and internet user's activities [13]. The dimensionality of the modern data collections renders usage of traditional algorithms infeasible. Therefore, matrix decomposition algorithms should be designed to be scalable and pass-efficient. In pass-efficient algorithms, data is read at most a constant number of times. Streaming-based algorithm refers to a pass-efficient computational model that requires only one pass through the dataset. By taking advantage of a streaming model, the sketch of a collection can be obtained online which makes storing the original matrix superfluous [14].

B. Orthogonal Matching Pursuit

OMP is a well-known greedy algorithm for solving sparse approximation problems. It is a key computational kernel in many compressed sensing algorithms. OMP has wide applications ranging from classification to structural health monitoring. As we describe in Alg. 2, OMP takes a dictionary and a signal as inputs and iteratively approximates the sparse representation of the signal by adding the best fitting element in every iteration. More details regarding the OMP algorithm are presented in Section V.

C. Notation

We write vectors in bold lowercase script, $\mathbf{x}$, and matrices in bold uppercase script, $\mathbf{A}$. Let $\mathbf{A}^t$ denote the transpose of $\mathbf{A}$. $\mathbf{A}_j$ represents the jth column and $\mathbf{A}_{ij}$ represents the entry at the ith row and jth column of matrix $\mathbf{A}$. $\mathbf{A}_\lambda$ is a subset of matrix $\mathbf{A}$ consisting of the columns defined in the set λ. $nnz(\mathbf{A})$ defines the number of non-zeros in the matrix $\mathbf{A}$. $\|\mathbf{x}\|_p = (\sum_{j=1}^{n} |\mathbf{x}(j)|^p)^{1/p}$ is used as the p-norm of a vector where $p \geq 1$. The Frobenius norm of matrix $\mathbf{A}$ is defined by $\|\mathbf{A}\|_F = \sqrt{(\sum_{j,j} |\mathbf{A}(i,j)|^2)}$ and $\|\mathbf{A}\|_2 = max_{\mathbf{x}\neq 0} \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2}$ is considered as spectral norm. The matrix $\mathbf{A}$ (*domain data*) is of size $m \times n$ where $m \ll n$ for *over-complete* matrices.

III. Related Work

In settings where the column span of $\mathbf{A}$ admits a lower dimensional embedding, the optimal low-rank approximation of the data is obtained by SVD or PCA algorithm [15]. However, their $\mathcal{O}(m^2n)$ complexity makes it hard to utilize these well-known algorithms for massive datasets. Unlike PCA, Sparse PCA (SPCA) is modified to find principal components with sparse loadings, which is desirable for interpreting data and storage reduction [16]. However, the computational complexity of SPCA is similar to classic SVD. Thus, it is not scalable for analyzing massive, dense datasets [16], [17].

Recently, the efficiency of random subsampling methods to compute the lower dimensional embedding of large datasets has been shown [18], [30]. Random Column Subset Selection (rCSS) approach has been used to provide scalable strategies

for factorizing large matrices [18]. Authors in [18] propose a theoretical scalable approach for large matrix factorization, but the hardware constraints are not considered in their work. The large memory footprint and non-adaptive structure of their rCSS approach make it unsuitable for streaming applications.

A recent approach in [7] suggests a distributed framework based upon a scalable and sparsity-inducing data transformation algorithm. The framework enables efficient execution of large-scale iterative learning algorithms on massive and dense datasets. SSketch framework is developed based upon a novel extension of the proposed sketching method in [7]. Unlike the work in [7], our data sketching approach is well-suited for streaming applications and is amenable to FPGA acceleration.

OMP has been shown to be very effective in inducing sparsity, although its complexity makes it costly for streaming applications. A number of implementations on GPU [9], [19], [20], ASICs [10], and FPGAs [21], [11], [22] are reported in the literature to speed up this complex reconstruction algorithm. FPGA implementation of OMP for problems of dimension 32×128 and 256×1024 are developed for signals with sparseness of 5 and 36 respectively [21], [11]. To the best of our knowledge, none of the previous implementations of OMP is devised for streaming applications with large and densely correlated data matrices. In addition, use of fixed-point format to compute and store the results limits their applicability for sketching purposes.

IV. SSketch Global Flow

The global flow of SSketch is presented in Fig. 1. SSketch takes the stream of a massive, dynamic dataset in the matrix form as its input and computes/updates a sketch matrix of the collection as its output. Our sketch formation algorithm is devised to minimize the costly message passings to/from the memory and cores, thereby it reduces the communication delay and energy. All SSketch's computations are done in IEEE 754 single precision floating-point format.

SSketch framework is developed based upon a novel sketching algorithm that we introduce in Section V. As illustrated in Fig. 1, SSketch consists of two main components to compute the sketch of dynamic data collections: (i) a dictionary learning unit, and (ii) a data sketching unit. As the stream of data comes in, the first component adaptively learns a dictionary as a subsample of input data such that the hybrid structure of data is well captured within the learned dictionary. Next, the data sketching unit solves a sparse approximation problem using the OMP algorithm to compute a block-sparse matrix. In the data sketching unit, the representation of each new arrival sample is computed based on the current values of the dictionary, and the result is sent back to the host. SSketch's accompanying API facilitates automation and adaptation of our proposed framework for rapid prototyping of an arbitrary matrix-based data analysis algorithm.

V. SSketch Methodology

Many modern massive datasets are either low-rank or lie on a union of lower dimensional subspaces. This convenient property can be leveraged to efficiently map the data to an ensemble of lower dimensional data structures [7]. The authors in [7] suggest a distributed framework based upon a scalable and sparsity-inducing solution to find the sketch of large and dense datasets such that:

$$\underset{\mathbf{D}\varepsilon R^{m\times l},\mathbf{V}\varepsilon R^{l\times n}}{\text{minimize}} \quad \|\mathbf{A}-\mathbf{D}\mathbf{V}\|_F \quad \text{subject to} \quad \|\mathbf{V}\|_0 \leq kn, \qquad (1)$$

where $\mathbf{A}_{m\times n}$ is the input data, $\mathbf{D}_{m\times l}$ is the dictionary matrix, $\mathbf{V}_{l\times n}$ is the block-sparse matrix, and $l \ll m \ll n$. $\|\mathbf{V}\|_0$ measures the total number of non-zeros in $\mathbf{V}$, and k is the target sparsity level for each input sample. Their approach, however, is "static" and does not adaptively update the dictionary at runtime. The only way to update is to redo the dictionary computation which would incur a higher cost and is unsuitable for streaming applications with a single pass requirement and limited memory. We develop SSketch based upon a novel extension of [7] for streaming applications. SSketch tailors the solution of (1) according to the underlying platform's constraints. Our approach, incurs a lower memory footprint and is well-suited for scenarios where storage is severely limited.

A. SSketch Algorithm

Our platform-aware matrix sketching algorithm is summarized in Alg. 1. SSketch algorithm, approximates matrix $\mathbf{A}$ as a product of two other matrices ($\mathbf{A}_{m\times n} \approx \mathbf{D}_{m\times l}\mathbf{V}_{l\times n}$) based on a streaming model.

Algorithm 1 SSketch algorithm

Inputs: Measurement matrix A, projection threshold α, sparsity level k, error threshold ϵ, and dictionary size l.
Output: Matrix D, and coefficient matrix V.

```
D ← empty
j ← 0
for i = 1,...,n do
    W(A_i) = ||D(D^tD)^{-1}D^tA_i − A_i||_2 / ||A_i||_2
    if W(A_i) > α and j < l then
        D_j = A_i / sqrt(||A_i||_2)
        V_ij = sqrt(||A_i||_2)
        j ← j + 1
    else
        V_i ← OMP(D, A_i, k, ε)
    end if
end for
```

For each newly arriving sample, our algorithm first calculates a projection error, W($\mathbf{A}_i$), based on the current values of the dictionary matrix $\mathbf{D}$. Next, it compares the calculated error with a user-defined projection threshold α and updates the sketch accordingly. SSketch locally updates the dictionary matrix $\mathbf{D}$ based on each arriving data sample and makes use of the greedy OMP routine to compute the block-sparse matrix $\mathbf{V}$. OMP can be used, either by fixing the number of non-zeros in each column of $\mathbf{V}$ (sparsity level k) or by fixing the total amount of approximation error (error threshold ϵ) for each sample. Factorizing the input matrix $\mathbf{A}$ as a product of two matrices with much fewer non-zeros than the original data, induces an approximation error that can be controlled by tuning the error threshold (ϵ), dictionary size (l), and projection

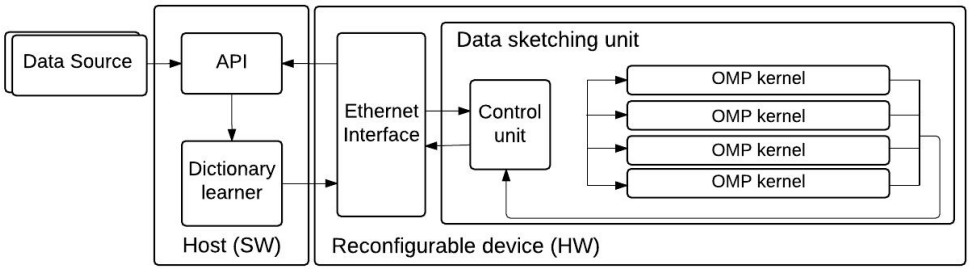


Fig. 1: High level block diagram of SSketch. It takes stream of data as input and adaptively learns a corresponding low-rank sketch of the collection by doing computation at the level of matrix rank. The resulting sketch is then sent back to the host for further analysis depending on the application.

threshold (α) in SSketch framework. In our experiments, we consider Frobenius norm error ($Xerr = \frac{\|\mathbf{A}-\mathbf{DV}\|_F}{\|\mathbf{A}\|_F}$), as sketch accuracy metric.

As can be seen, SSketch algorithm requires only one pass through each arriving sample. This method only requires storing a single column of the input matrix $\mathbf{A}$ and the matrix $\mathbf{D}$ at a time. Note that the dictionary matrix $\mathbf{D}_{m\times l}$ is constructed by columns of data matrix $\mathbf{A}_{m\times n}$. The column space of $\mathbf{D}$ is contained in the column space of $\mathbf{A}$. Thus, $rank(\mathbf{DD}^{+}\mathbf{A}) = rank(\mathbf{D}) \leq l \leq m$. It simply implies that for over-complete datasets OMP computation is required for $n-l$ columns and the overhead time of copying $\mathbf{D}$ is ignorable due to its small size compared to $\mathbf{A}$.

OMP with QR Decomposition. As we describe in Section VII, computational complexity of the projection step (line 4 of Alg. 1) is small compared to the $\mathcal{O}(mnl^2)$ complexity of the OMP algorithm. Thus, the computational bottleneck of SSketch algorithm is OMP. To boost the computational performance of SSketch for analyzing a large amount of data on FPGA, it is necessary to modify the OMP algorithm such that it maximally benefits from the available resources and incurs a scalable computational complexity.

Alg. 2 demonstrates the pseudocode of OMP where ϵ is the error threshold, and k is the target sparsity level. The Least Squares (LS) minimization step (line 5 of Alg. 2) involves a variety of operations with complex data flows that introduce an extra hardware complexity. However, proper use of factorization techniques like QR decomposition or Cholesky method within the OMP algorithm would reduce its hardware implementation complexity and make it well-suited for hardware accelerators [11], [23].

Thus, to efficiently solve the LS optimization problem in line 5 of Alg. 2, we decide to use QR decomposition (Alg. 3). QR decomposition returns an orthogonal matrix $\mathbf{Q}$ and an upper-triangular matrix $\mathbf{R}$. It iteratively updates the decomposition by reusing the $\mathbf{Q}$ and $\mathbf{R}$ matrices from the last OMP iteration. In this approach, the residual (line 6 of Alg. 2) can be updated by $\mathbf{r}^i \leftarrow \mathbf{r}^{i-1}\mathbf{Q}^i(\mathbf{Q}^i)^t\mathbf{r}^{i-1}$. The final solution is calculated by performing back substitution to solve the inversion of the matrix $\mathbf{R}$ in $\mathbf{v}^k = \mathbf{R}^{-1}\mathbf{Q}^t\mathbf{A}_i$.

Assuming that matrix $\mathbf{A}$ is of size $m \times n$ and $\mathbf{D}$ is of size $m \times l$, then the complexity of the OMPQR is $\mathcal{O}(mnl^2)$. This complexity is linear in terms of m and n since in many settings l is much smaller in compared to m and n. This linear complexity enables SSketch to readily scale up for processing a large amount of data based on a streaming model.

Algorithm 2 OMP algorithm

Inputs: Matrix D, measurement $\mathbf{A}_i$, sparsity level k, threshold error ϵ.
Output: Support set Λ and k-dimensional coefficient vector v.

1: $r \leftarrow \mathbf{A}_i$
2: $\Lambda^0 \leftarrow \emptyset$
3: **for *i = 1,...,k* do**
4: $\quad \Lambda \leftarrow \Lambda \cup argmax_j | < r^{i-1}, \mathbf{D}_j > |$ **Find best fitting column**
5: $\quad v^i \leftarrow argmin_v \|r^{i-1} - \mathbf{D}_{\Lambda^i} v\|_2{}^2$ **LS Optimization**
6: $\quad r^i \leftarrow r^{i-1} - \mathbf{D}_{\Lambda^i} v^i$ **Residual Update**
end for

Algorithm 3 Incremental QR decomposition by modified Gram-Schmidt

Inputs: New column $\mathbf{D}_{\Lambda^s}$, last iteration $\mathbf{Q}^{s-1}$, $\mathbf{R}^{s-1}$.
Output: $\mathbf{Q}^s$ and $\mathbf{R}^s$.

1: $$R^s \leftarrow \begin{pmatrix} R^{s-1} & 0 \\ 0 & 0 \end{pmatrix}$$
2: $\xi^s \leftarrow \mathbf{D}_{\Lambda^s}$
3: **for *j = 1,...,s-1* do**
4: $\quad \mathbf{R}_{js}{}^s \leftarrow (\mathbf{Q}^{s-1})_j{}^H \xi^s$
5: $\quad \xi^s \leftarrow \xi^s - \mathbf{R}_{js}{}^s \mathbf{Q}_j{}^{s-1}$
end for
6: $\mathbf{R}^s_{ss} \leftarrow \sqrt{\|\xi^s\|_2{}^2}$
7: $\mathbf{Q}^s \leftarrow [\mathbf{Q}^{s-1}, \frac{\xi^s}{\mathbf{R}_{ss}{}^s}]$

B. Blocking SSketch.

Let $\mathbf{A}$ = [$\mathbf{A}_1$; $\mathbf{A}_2$; $\mathbf{A}_3$] be a matrix consisting of rows $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ that are stacked on the top of one another. Our key observation is that if we obtain the sketch of each block independently and combine the resulting sketches (*blocking SSketch*) as illustrated in Fig. 2, then the combined sketch can be as good as sketching $\mathbf{A}$ directly (*nonblocking SSketch*) in terms of error-performance trade-off. This property can be generalized to any number of partitions of $\mathbf{A}$. We leverage this convenient property to increase the performance of our proposed framework for sketching massive datasets based on a streaming model. In blocking SSketch, the data matrix $\mathbf{A}$

is divided into more manageable sized blocks such that there exist enough block RAMs on FPGA to store the corresponding **D** and a single column of that block. The blocking SSketch achieves a significant bandwidth saving, faster load/store, less communication traffic between kernels, and a fixed memory requirement on FPGA. The methodology also provides the capability of factorizing massive, dense datasets in an online streaming model.

Independent analysis of each block is especially attractive if the data is distributed across multiple machines. In such settings, each platform can independently compute a local sketch. These sketches can then be combined to obtain the sketch of the original collection. Given a fixed memory budget for the matrix **D**, as it is presented in Section VII, blocking SSketch results in a more accurate approximation compared with nonblocking SSketch. The blocking SSketch computations are done on smaller segments of data which confers a higher system performance. The achieved higher accuracy is at the cost of a larger number of non-zeros in **V**. Note that as our evaluation results imply, designers can reduce number of non-zeros in the computed block-sparse matrix by increasing the error threshold ϵ in SSketch algorithm.

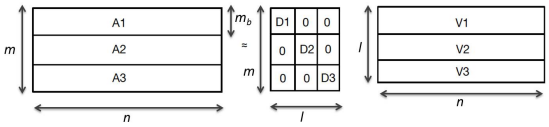


Fig. 2: Schematic depiction of blocking SSketch.

VI. SSketch Automated Hardware Implementation

In this section, we discuss the details of SSketch hardware implementation. After applying preprocessing steps on the stream of the input data for dictionary learning, SSketch sends the data to FPGA through a 1Gbps Ethernet port. SSketch is devised with multiple OMP kernels and a control unit to efficiently compute the block-sparse matrix **V**. As the stream of data arrives, the control unit looks for availability of OMP kernels and assigns the newly arrived sample to an idle kernel for further processing. The control unit also has the responsibility of reading out the outputs and sending back the results to the host. SSketch API provides designers with a user-friendly interface for rapid prototyping of arbitrary matrix-based data analysis algorithms and realizing streaming applications on FPGAs (Fig. 3). Algorithmic parameters of SSketch including the projection threshold α, error threshold ϵ, dictionary size l, target block-sparsity level k, and block-size m_b, can be easily changed through the SSketch API. The SSketch API is open source and is freely available on our website [24].

In OMP hardware implementation, we utilize several techniques to reduce the iteration interval of two successive operations and exploit the parallelism within the algorithm. We observe that the OMP algorithm includes multiple dot product computations which result in frequent appearance of for-loops requiring an operation similar to $a \mathrel{+}= b[i] \times c[i]$. We use a *tree-based reduction module* by implementing a tree-based adder to accelerate the dot product and norm computation

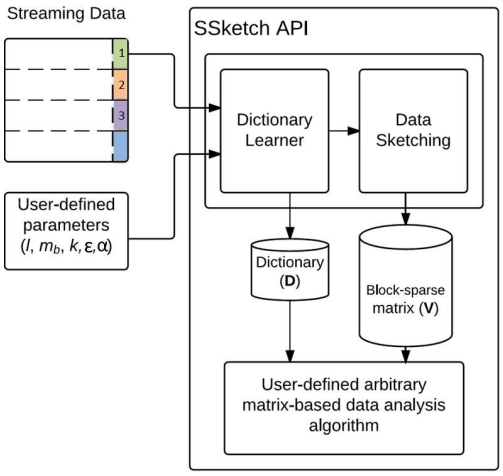


Fig. 3: High level diagram of SSketch API. It takes the stream of input data in the matrix form as its input. SSketch divides the input data matrix into more manageable blocks of size m_b and adaptively learns the corresponding sketch of the data. SSketch gets its algorithmic parameters such as the projection threshold α, error threshold ϵ, dictionary size l, block-sparsity level k, and block-size m_b from user and computes the sketch accordingly. This learned data sketch can then be used for building a domain-specific architecture that scalably performs data analysis on an ensemble of lower dimensional structures rather than the original matrix without a significant loss.

steps that appear frequently in the OMP routine. By means of the reduction module, SSketch is able to reduce the iteration interval and handle more operations simultaneously. As such, SSketch requires multiple concurrent loads and stores from a particular RAM. To cope with the concurrency, instead of having a large block RAM for matrices **D** and **Q**, we use multiple smaller sized block memories and fill these block RAMs by cyclic interleaving. Thus, we can perform a faster computation by accessing multiple successive elements of the matrices and removing the dependency in the for-loops.

Using the block RAM is desirable in FPGA implementations because of its fast access time. The number of block RAMs on one FPGA is limited, so it is important to reduce the amount of utilized block memories. We reduce block RAM utilization in our realization by a factor of 2 compared to the naive implementation. This reduction is a consequence of our observation that none of the columns of matrix **D** would be selected twice during one call of the OMP algorithm. Thus, for computing line 4 of Alg. 2 we only use the indices of **D** that are not selected during the previous OMP iterations. We instead use the memory space that was originally assigned to the selected columns of **D** to store the matrix **Q**. By doing so we reduce the block RAM utilization, which allows SSketch to employ more OMP kernels in parallel.

VII. Experimental Evaluations

For evaluation purposes, we apply our methodology on three sets of data: (i) Light field, (ii) Hyperspectral images, and (iii) Synthetic data. To ensure that dictionary learning is

independent of the data order, for each fixed set of algorithmic parameters we shuffle the data before each calling of SSketch algorithm. The mean error value for 10 calls of SSketch algorithm and its variance are reported in Fig. 4, and 5. In all cases, we observe that the variance of the error value is two to three orders of magnitude less than the mean value, implying that SSketch algorithm has a low dependency on the data arrival order. This convenient property of SSketch algorithm is promising for adaptive dictionary learning and sketching purposes in streaming applications.

A. Light Field Experiments

A light field image is a set of multi-dimensional array of images that are simultaneously captured from slightly different viewpoints. Promising capabilities of light field imaging include the ability to define the field's depth, focus or refocus on a part of image, and reconstruct a 3D model of the scene [25]. For evaluating SSketch algorithm accuracy, we run our experiments on a light field data consisting of 2500 samples each of which constructed of 25 8×8 patches. The light field data results in a data matrix with 4 million non-zero elements. We choose this moderate input matrix size to accommodate the SVD algorithm for comparison purposes and enable the exact error measurement especially for correlation matrix (a.k.a., *Gram matrix*) approximation. The Gram matrix of a data collection consists of the Hamiltonian inner products of data vectors. The core of several important data analysis algorithms is the iterative computation on the data Gram matrix. Examples of Gram matrix usage include but are not limited to kernel-based learning and classification methods, as well as several regression and regularized least squares routines [31].

In SSketch, the number of selected columns "l" for the dictionary has a direct effect on the convergence error rate as well as speed. Factorization with larger l achieves a smaller convergence error but decreases the performance due to the increase in computation. Fig. 4a and 4d report the results of applying both nonblocking and blocking SSketch on the data. We define the approximation error for the input data and its corresponding Gram matrix as $Xerr = \frac{\|\mathbf{A}-\tilde{\mathbf{A}}\|_F}{\|\mathbf{A}\|_F}$, and $Gerr = \frac{\|\mathbf{A}^t\mathbf{A}-\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}\|_F}{\|\mathbf{A}^t\mathbf{A}\|_F}$, where $\tilde{\mathbf{A}} = \mathbf{DV}$.

Given a fixed memory budget for the matrix $\mathbf{D}$, Fig. 4a and 4d illustrate that the blocking SSketch results in a more accurate sketch compared with the nonblocking one. Number of rows in each block of input data (*block-size*) has a direct effect on SSketch's performance. Fig. 4b, 4e, and 4c demonstrate the effect of block-size on the data and Gram matrix approximation error as well as matrix *compression-rate*, where the compression-rate is defined as $\frac{nnz(\mathbf{D})+nnz(\mathbf{V})}{nnz(\mathbf{A})}$. In this setting, the higher accuracy of blocking SSketch is at the cost of a larger number of non-zeros in the block-sparse matrix $\mathbf{V}$. As illustrated in Fig. 4b, 4e, and 4c designers can easily reduce the number of non-zeros in the matrix $\mathbf{V}$ by increasing the SSketch error threshold ϵ. In SSketch algorithm, there is a trade-off between the sketch accuracy and the number of non-zeros in the block-sparse matrix $\mathbf{V}$. The optimal performance of SSketch methodology is determined based on the error threshold and the hardware constraints.

Considering the spectral norm error ($e_k = \frac{\|\mathbf{A}-\tilde{\mathbf{A}}\|_2}{\|\mathbf{A}\|_2}$) instead of Frobenius norm error, the theoretical minimal error can be expressed as $\sigma_{k+1} = min(\|\mathbf{A} - \mathbf{A}_k\|_2 : \mathbf{A}_k \text{ has rank k})$, where σ_k is the exact k'th singular value of $\mathbf{A}$ and $\mathbf{A}_k$ is obtained by SVD [26]. Fig. 4f compares e_k and the theoretical minimal error for the light field dataset.

B. Hyperspectral Experiments

A hyperspectral image is a sequence of images generated by hundreds of detectors that capture the information from across the electromagnetic spectrum. With this collected information, one would obtain a spectrum signature for each pixel of the image that can be used for identifying or detecting the material [27]. Hyperspectral imaging is a new type of high-dimensional image data and a promising tool for applications in earth-based and planetary exploration, geo-sensing, and beyond. This fast and non-destructive technique provides a large amount of spectral and spatial information on a wide number of samples. However, the large size of hyperspectral datasets limits the applicability of this technique, especially for scenarios where online evaluation of a large number of samples is required.

We adapt SSketch framework to capture the sketch of each pixel for the purpose of enhancing the computational performance and reducing the total storage requirement for hyperspectral images. In this experiment, the SSketch algorithm is applied on two different hyperspectral datasets. The first dataset [28] is 148×691614 and the second one [29] is of size 220×21025. Our experiments show that SSketch algorithm results in the same trend as in 4a and 4d for both hyperspectral datasets. As Fig. 5 illustrates, the Gram matrix factorization error reaches to less than 0.2×10^{-6} for $l \geq 10$ in both datasets.

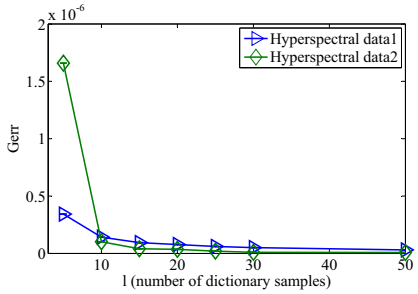
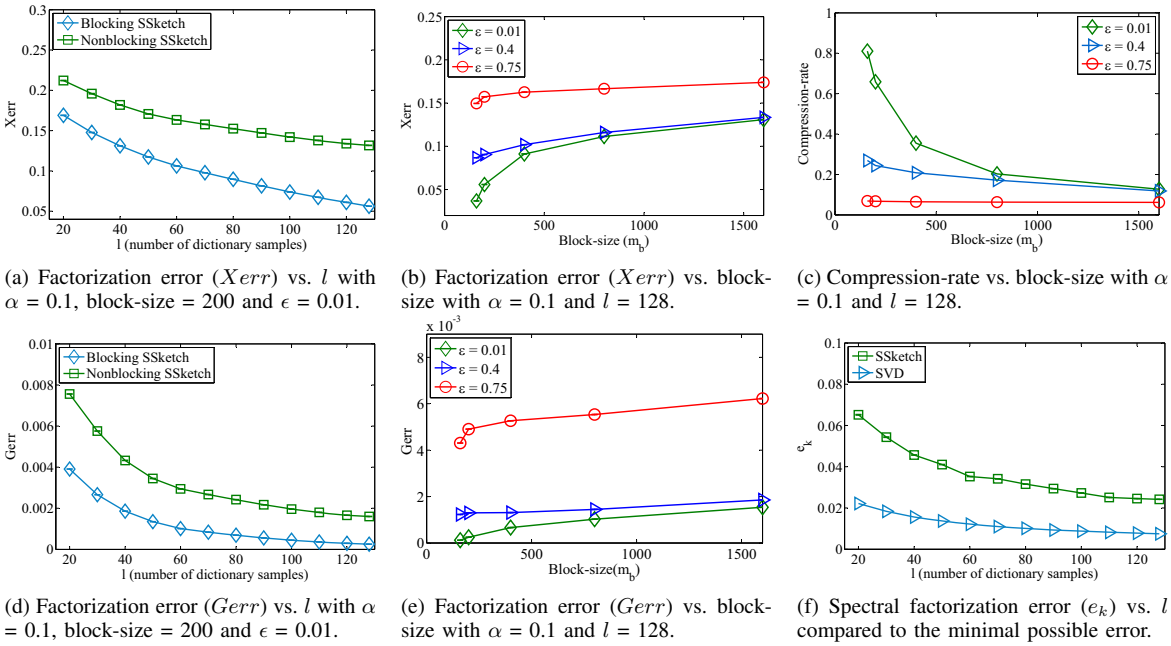


Fig. 5: Factorization error ($Gerr$) vs. l for two different hyperspectral datasets. The SSketch algorithmic parameters are set to α = 0.1 and ϵ = 0.01, where α is the projection threshold and ϵ is the error threshold.

C. Scalability

We provide a comparison between the complexity of different implementations of the OMP routine in Fig. 6. Batch OMP (BOMP) is a variation of OMP that is especially optimized for sparse-coding of large sets of samples over the same dictionary. BOMP requires more memory space compared with the conventional OMP, since it needs to store $\mathbf{D}^T\mathbf{D}$ along with the matrix $\mathbf{D}$. BOMPQR and the OMPQR both have near linear computational complexity. We use OMPQR method in our target architecture as it is more memory-efficient.



(a) Factorization error ($Xerr$) vs. l with α = 0.1, block-size = 200 and ϵ = 0.01.

(b) Factorization error ($Xerr$) vs. block-size with α = 0.1 and l = 128.

(c) Compression-rate vs. block-size with α = 0.1 and l = 128.

(d) Factorization error ($Gerr$) vs. l with α = 0.1, block-size = 200 and ϵ = 0.01.

(e) Factorization error ($Gerr$) vs. block-size with α = 0.1 and l = 128.

(f) Spectral factorization error (e_k) vs. l compared to the minimal possible error.

Fig. 4: Experimental evaluations of SSketch. α and ϵ represent the user-defined projection threshold, and the error threshold respectively. l is the number of samples in the dictionary matrix, and m_b is the block-size.

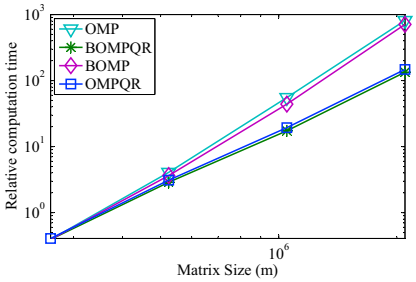


Fig. 6: Computational complexity comparison of different OMP implementations. Using QR decomposition significantly improves OMP's runtime.

The complexity of our OMP algorithm is linear both in terms of m and n, so dividing $\mathbf{A}_{m\times n}$ into several blocks along the dimension of m and processing each block independently does not add to the total computational complexity of the algorithm. However, it shrinks the data size to fit into the FPGA block RAMs and improves the performance.

Let $T_{OMP}(m,l,k)$ stand for the number of operations required to obtain the sketch of a vector of length m with target sparsity level k. Then the runtime of the system is a linear function of T_{OMP} which makes the proposed architecture scalable for factorizing large matrices. The complexity of projection step in SSketch algorithm (line 4 of Alg. 1) is $(l^3 + 2lm + 2l^2m)$. However, if we decompose $\mathbf{D}_{m\times l}$ to $\mathbf{Q}_{m\times m} \times \mathbf{R}_{m\times l}$ and replace $\mathbf{DD}^+$ with $\mathbf{QI}_t\mathbf{Q}^t$, then the projection step's computational complexity would be reduced to $(2lm+l^2m)$. Assuming $\mathbf{D} = \mathbf{QR}$ then the projection matrix can be written as:

$$\mathbf{D}(\mathbf{D}^t\mathbf{D})^{-1}\mathbf{D}^t = \mathbf{QR}(\mathbf{R}^t\mathbf{Q}^t\mathbf{QR})^{-1}\mathbf{R}^t\mathbf{Q}^t$$
$$= \mathbf{Q}(\mathbf{RR}^{-1})(\mathbf{R}^{t^{-1}}\mathbf{R}^t)\mathbf{Q}^t = \mathbf{QI}_t\mathbf{Q}^t, \quad (2)$$

which we use to decrease the projection step's complexity.

Table I compares different factorization methods with respect to their complexity. The SSketch's complexity indicates a linear relationship with n and m. In SSketch, computations can be parallelized as the sparse representation can be independently computed for each column of the sub-blocks of $\mathbf{A}$.

TABLE I: Complexity of different factorization methods.

Algorithm	Complexity
SVD	$m^2n + m^3$
SPCA	$lmn + m^2n + m^3$
SSketch (this paper)	$n(lm + l^2m) + mnl^2 \approx 2mnl^2$

D. Hardware Settings and Results

We use Xilinx Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit as our hardware platform. An Intel core i7-2600K processor with SSE4 architecture running on the Windows OS is utilized as our general purpose processing unit hosting the FPGA. In this work, we employ Xilinx standard IP cores for single precision floating-point operations. We used Xilinx ISE 14.6 to synthesize, place, route, and program the FPGA.

Table II shows Virtex-6 resource utilization for our heterogeneous architecture. SSketch includes four OMP cores plus an Ethernet interface. For factorizing matrix $\mathbf{A}_{m\times n}$, there is no specific limitation on the size n due to the streaming nature of SSketch. However, the FPGA block RAM's size is limited. To fit into the RAM, we decide to divide input matrix $\mathbf{A}$ to blocks of size $m_b \times n$ where m_b and k are set to be less than 256. Note that these parameters are changeable in SSketch API. So, if a designer decides to choose a higher m_b or k for any reasons, she can easily modify the parameters according to the application.

To corroborate the scalability and practicability of our framework, we use synthetic data with dense (non-sparse)

TABLE II: Virtex-6 resource utilization.

	Used	Available	Utilization
Slice Registers	50888	301440	16%
Slice LUTs	81585	150720	54%
RAM B36E1	382	416	91%
DSP 48E1s	356	768	46%

correlations of different sizes as well as a hyperspectral image dataset [29]. The runtime of SSketch for the different-sized synthetic datasets is reported in Table III, where the total delay of SSketch ($T_{SSketch}$) can be expressed as:

$$T_{SSketch} \approx \underset{learning}{T_{dictionary}} + \underset{Overhead}{T_{Communication}} + \underset{Computation}{T_{FPGA}} . \quad (3)$$

As it is shown in Table III, the total delay of SSketch is a linear function of the number of processed samples, which experimentally confirms the scalability of our proposed architecture. According to Table III, the whole system including the dictionary learning process takes 21.029s to process 5K samples where each of them has 256 elements. 4872 of these samples pass through OMP kernels and each of them requires 89 iterations on average to complete the process. In this experiment, an average throughput of 169Mbps is achieved by SSketch.

For both hyperspectral dataset [29] of size 204×54129 and synthetic dense data, our HW/SW co-design approach (with target sparsity level (k) of 128) achieves up to 200 folds speedup compared to the software-only realization on a 3.40 GHz CPU. The average overhead delay for communicating between the processor (host) and accelerator contributes less than 4% to the total delay.

TABLE III: SSketch total processing time is linear in terms of the number of processed samples.

Size of n	$T_{SSketch}$ $(l = 128)$	$T_{SSketch}$ $(l = 64)$
$n = 1K$	$3.635sec$	$2.31sec$
$n = 5K$	$21.029sec$	$12.01sec$
$n = 10K$	$43.446sec$	$24.32sec$
$n = 20K$	$90.761sec$	$48.52sec$

VIII. Conclusion

This paper presents SSketch, an automated computing framework for FPGA-based online analysis of big and densely correlated data matrices. SSketch utilizes a novel streaming and communication-minimizing methodology to efficiently capture the sketch of the collection. It adaptively learns and leverages the hybrid structure of the streaming input data to effectively improve the performance. To boost the computational efficiency, SSketch is devised with a novel scalable implementation of OMP on FPGA. Our framework provides designers with a user-friendly API for rapid prototyping and evaluation of an arbitrary matrix-based big data analysis algorithm. We evaluate the method on three different large contemporary datasets. In particular, we compare SSketch runtime to the software realization on a CPU and also report the delay overhead for communicating between the processor (host) and the accelerator. Our evaluations corroborate SSketch scalability and practicability.

Acknowledgments

This work was supported in parts by the Office of Naval Research grant (ONR- N00014-11-1-0885).

References

[1] E. Liberty. "Simple and deterministic matrix sketching." Proceedings of the 19th ACM SIGKDD, 2013.
[2] A. Sergyienko and O. Maslennikov. "Implementation of Givens QR-decomposition in FPGA." Parallel Processing and Applied Mathematics, 2002.
[3] W. Zhang, V. Betz, and J. Rose. Portable and scalable fpgabased acceleration of a direct linear system solver. ICECE Technology, 2008.
[4] P. Greisen, M. Runo, P. Guillet, S. Heinzle, A. Smolic, H. Kaeslin, and M. Gross. Evaluation and fpga implementation of sparse linear solvers for video processing applications. IEEE TCSVT, 2013.
[5] L. M. Ledesma-Carrillo, E. Cabal-Yepez, R. de J Romero-Troncoso, et al. "Reconfigurable FPGA-Based unit for Singular Value Decomposition of large mxn matrices." ReConFig, 2011.
[6] S. Rajasekaran and M. Song. "A novel scheme for the parallel computation of SVDs." HPCC, Springer Berlin Heidelberg, 2006.
[7] A. Mirhoseini, E. L. Dyer, E. M. Songhori, R. Baraniuk, and F. Koushanfar. "RankMap: A Platform-Aware Framework for Distributed Learning from Dense Datasets." arXiv:1503.08169, 2015.
[8] D. C. Montgomery, E. A. Peck, and G. G. Vining. Introduction to linear regression analysis. John Wiley and Sons, 2012.
[9] M. Andrecut. "Fast GPU implementation of sparse signal recovery from random projections." arXiv:0809.1833, 2008.
[10] P. Maechler, P. Greisen, N. Felber, and A. Burg, "Matching pursuit: Evaluation and implementation for LTE channel estimation," ISCAS, 2010.
[11] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin. "High-speed compressed sensing reconstruction on FPGA using OMP and AMP." ICECS, 2012.
[12] A. M. Kulkarni, H. Homayoun, and T. Mohsenin. "A parallel and reconfigurable architecture for efficient OMP compressive sensing reconstruction." Proceedings of the 24th edition of the great lakes symposium on VLSI. ACM, 2014.
[13] "Frontiers in Massive Data Analysis", The National Academies Press, 2013.
[14] K. L. Clarkson and D. P. Woodruff. "Numerical linear algebra in the streaming model." Proceedings of the 41th ACM symposium on Theory of computing, 2009.
[15] G. H Golub and C. Reinsch. Singular value decomposition and least squares solutions. Numerische Mathematik, 1970.
[16] H. Zou, T. Hastie, and R. Tibshirani. "Sparse principal component analysis." Journal of computational and graphical statistics 15.2, 2006.
[17] D. S. Papailiopoulos, A. G. Dimakis, and S. Korokythakis. "Sparse PCA through Low-rank Approximations." arXiv:1303.0551, 2013.
[18] E. L. Dyer, A. C. Sankaranarayanan, and R. G. Baraniuk. "Greedy feature selection for subspace clustering." JMLR 14.1, 2013.
[19] J. D. Blanchard and J. Tanner. "GPU accelerated greedy algorithms for compressed sensing." Mathematical Programming Computation, 2013.
[20] Y. Fang, L. Chen, J. Wu, and B. Huang. "GPU implementation of orthogonal matching pursuit for compressive sensing." ICPADS, 2011.
[21] A. Septimus and R. Steinberg, "Compressive sampling hardware reconstruction,". ISCAS, 2010.
[22] J. L. Stanislaus, and T. Mohsenin. "Low-complexity FPGA implementation of compressive sensing reconstruction." ICNC, 2013.
[23] J. L. Stanislaus, and T. Mohsenin. "High performance compressive sensing reconstruction hardware with QRD process." ISCAS, 2012.
[24] http://www.aceslab.org/.
[25] K. Marwah, G. Wetzstein, Y. Bando, and R. Raskar. "Compressive light field photography using overcomplete dictionaries and optimized projections." ACM TOG, 2013.
[26] G. Martinsson, A. Gillman, E. Liberty, et al. "Randomized methods for computing the Singular Value Decomposition of very large matrices." Workshop on Algorithms for Modern Massive Datasets, 2010.
[27] A. Plaza, J. Plaza, A. Paz, and S. Sanchez. "Parallel hyperspectral image and signal processing." Signal Processing Magazine, IEEE 2011.
[28] http://scien.stanford.edu/index.php/landscapes
[29] http://www.ehu.es/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes
[30] Drineas, Petros, and M. Mahoney. "On the Nyström method for approximating a Gram matrix for improved kernel-based learning." JMLR 6, 2005.
[31] R. Tibshirani, Regression shrinkage and selection via the lasso, JSTOR, 1996.