Automated Real-Time Analysis of Streaming Big and Dense Data on Reconfigurable Platforms

BITA DARVISH ROUHANI, UC San Diego
AZALIA MIRHOSEINI and EBRAHIM M. SONGHORI, Rice University
FARINAZ KOUSHANFAR, UC San Diego

We propose SSketch, a novel automated framework for efficient analysis of dynamic big data with dense (non-sparse) correlation matrices on reconfigurable platforms. SSketch targets *streaming* applications where each data sample can be processed only once and storage is severely limited. Our framework adaptively learns from the stream of input data and updates 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. SSketch provides an automated optimization methodology for creating the most accurate data sketch for a given set of user-defined constraints, including runtime and power as well as platform constraints such as memory. To facilitate automation, SSketch takes advantage of a Hardware/Software (HW/SW) co-design approach: It provides an Application Programming Interface 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 demonstrate up to a 200-fold speedup on our hardware-accelerated realization of SSketch compared to a software-based deployment on a general-purpose processor.

CCS Concepts: • **Information systems** → **Stream management**; • **Computing methodologies** → **Online learning settings**; **Factorization methods;**

Additional Key Words and Phrases: Streaming model, big data, dense matrix, FPGA, lower dimensional embedding, HW/SW co-design, matrix sketching, matrix-based analysis

ACM Reference Format:
Bita Darvish Rouhani, Azalia Mirhoseini, Ebrahim M. Songhori, and Farinaz Koushanfar. 2016. Automated real-time analysis of streaming big and dense data on reconfigurable platforms. ACM Trans. Reconfigurable Technol. Syst. 10, 1, Article 8 (December 2016), 22 pages.
DOI: http://dx.doi.org/10.1145/2974023

1. INTRODUCTION

The ever-growing body of digital data is challenging conventional analytical techniques in machine learning, computer vision, and signal processing. Traditional analytical

This work was supported in parts by the Office of Naval Research (ONR) award (Grant No. N00014-11-1-0885) and a National Science Foundation (NSF) TrustHub award (CNS-1513063).
This work was done while Azalia Mirhosieni was at Rice University.
Authors' addresses: B. D. Rouhani and F. Koushanfar, Electrical and Computer Engineering Department, University of California San Diego, 500 Gilman Dr, La Jolla, CA 92093; emails: {bita, farinaz}@ucsd.edu; A. Mirhoseini, Google, 6100 Main St, MS 380, Houston, TX 77005; email: azalia@rice.edu; E. M. Songhori, Electrical and Computer Engineering Department, Rice University, 6100 Main St, MS 380, Houston, TX 77005; email: ebrahim@rice.edu.

DOI: http://dx.doi.org/10.1145/2974023

methods have been mainly developed based on the assumption that designers can work with data within the confines of their own computing environment. The growth of big data, however, is changing that paradigm, especially in scenarios where severe memory and computational resource constraints exist. This disruption of convention changes the way we analyze modern datasets and renders designing customizable, streaming-based data transformation methods, a.k.a., *sketching* algorithms, a necessity to holistically take into account the data structure and underlying platform constraints. A sketch matrix is a compact approximation of the original matrix embedding to lower-dimensional subspaces. With a properly designed sketching algorithm, the intended computations can be performed on an ensemble of constantly updated lower-dimensional structures rather than the original matrix without a significant loss. Note that traditional sketching methods such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) incur a large memory footprint with a quadratic computational complexity which limit their practicability in big data regime.

To optimize the performance of streaming big data learning applications, there are at least two sets of challenges that should be addressed simultaneously. 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 that can be readily applied for processing 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. This article leverages the tradeoff between memory communication and redundant local calculations to improve the performance of such costly iterative computations.

Streaming processing of data is critical for many applications in which storage is severely limited and data can be read at most once [Liberty 2013]. Some prominent examples of such applications include real-time video surveillance, medical image processing, recommendation systems, wireless communication, and Internet user's activity recognition [Council 2013]. In these scenarios, to cope with the dynamics of the streaming content just in time, the sketch has to be found inline with the data arrival. A large body of earlier work demonstrated the efficiency of using custom hardware for acceleration of traditional matrix sketching algorithms such as QR decomposition [Sergyienko and Maslennikov 2002], LU decomposition [Zhang et al. 2012], Cholesky [Greisen et al. 2013], and SVD [Ledesma-Carrillo et al. 2011]. However, the existing hardware-accelerated sketching methods either have a higher-than-linear complexity [Rajasekaran and Song 2006] or are non-adaptive for dynamic sketching [Ledesma-Carrillo et al. 2011]. 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 Liberty [2013], which also relies on running SVD on the sketch matrix. Even this method is unable to handle the changing data dynamics in real time as the SVD algorithm incurs a higher-than-linear computational complexity. Moreover, runtime and power constraints are not addressed in Liberty [2013], either. To the best of our knowledge, no hardware acceleration for streaming sketches has been reported in the literature before SSketch.

We propose SSketch, a novel automated framework for efficient analysis and hardware-based acceleration of massive and densely correlated datasets in streaming applications. It has been shown in Mirhoseini et al. [2015] that the dense and high-dimensional datasets usually have an underlying lower-dimensional *hybrid*

structure.[1] SSketch leverages this convenient property to efficiently transform the data to an ensemble of lower-dimensional subspaces. the 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* that includes a subset of samples carefully selected 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.

An important property of SSketch is its capability to customize its sketching approach based on the user-defined requirements and hardware limitations. More precisely, we provide an automated optimization approach that can be used to customize the SSketch framework to compute the best sketch matrix (with the least approximation error) under runtime, power, and memory constraints. As the stream of input data arrives, SSketch adaptively learns from the incoming vectors and updates the sketch of the collection. An accompanying Application Programming Interface (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 algorithm. SSketch and its API target a broad class of learning 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 [Montgomery et al. 2012].

Our framework addresses the big data learning problem by using 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 Graphics Processing Units (GPUs), Application-Specific Integrated Circuits (ASICs), and Field Programmable Gate Arrays (FPGAs) have been reported in the literature, for example, in Andrecut [2008], Maechler et al. [2010], and Bai et al. [2012]. However, the prior work on FPGA had focused on fixed-point number format. In addition, most earlier hardware-accelerated OMP designs are restricted to small matrix sizes [Septimus and Steinberg 2010; Bai et al. 2012] or are optimized for specific signal processing applications in which a limited number of OMP iterations (e.g., up to 32) suffices to perform the underlying data reconstruction task [Kulkarni et al. 2014]. In contrast, SSketch's scalable methodology introduces a novel generic approach that enables use of an OMP routine for processing massive, dynamic datasets without introducing restriction on the size or range of the target data.

SSketch uses 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 our sketching approach, 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 architecture. The explicit contributions of this article are as follows:

—We propose SSketch, a novel communication-minimizing framework for online (streaming) large matrix computation. SSketch adaptively learns the hybrid

[1]A datum that consists of multiple lower-dimension subspaces is referred to as a datum with hybrid structure.

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-based data transformation method for FPGA acceleration. Our sketching algorithm benefits from a fixed, low-memory footprint and an $\mathcal{O}(mn)$ computational complexity. We also provide theoretical error analysis for our proposed sketching methodology.
—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 provide an automated optimization method that can be used to customize SSketch's reconfigurable framework and compute the most accurate sketch matrix under a given set of runtime, power, and memory constraints.
—We devise SSketch with a scalable, floating-point implementation of 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 a general purpose processor and also report its overhead.

An earlier version of SSketch was presented in Rouhani et al. [2015]. In this article, we extend our framework by (i) adding an automated and optimized constraint-driven customization module (Section 7). This module enables SSketch to create the most accurate data sketch from the streaming input data for a given set of user-defined constraints and hardware limitations. We provide an optimization strategy supported by an automated solver that can be readily used for efficient sketch computation and embedding. The input parameters (or constraints) to our solver may include runtime, power, and memory and its outputs include sketching algorithmic parameters as well as guidelines for the hardware mapping. We show the applicability and effectiveness of the proposed optimization by various new experimental evaluations (Sections 9.3 and 8) and (ii) providing theoretical guarantees for approximation error of our proposed streaming-based sketching methodology (Section 5.2). The error bound can be used for devising an sketch that meets the desired level of accuracy.

2. BACKGROUND AND PRELIMINARIES

2.1. Streaming Sketching Model

The high dimensionality of modern data collections renders usage of traditional data transformation algorithms infeasible. As such, matrix sketching methods should be designed to be scalable and pass-efficient. In pass-efficient techniques, data are 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 matrix of a collection can be obtained inline with the data arrival, which evades the requirement to store the original ever-growing dataset [Clarkson and Woodruff 2009].

2.2. 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 Algorithm 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 5.

2.3. 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 $\mathbf{A}_\lambda$ 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 $\mathbf{A}$ is of size $m \times n$ where n is the number of samples and m is the corresponding number of features and $m \ll n$ for *over-complete* matrices.

3. RELATED WORK

Developing support for streaming data is critical in many emerging applications where real-time response is required [Zinn et al. 2011]. Several recent studies have focused on system modeling and design techniques to facilitate streaming applications by exploiting task- and data-level parallelism, for example, in Plavec et al. [2013], Zinn et al. [2011], and Cong et al. [2014]. However, none of the prior work has leveraged the data geometry to further accommodate streaming applications. To the best of our knowledge, SSketch is the first automated framework that proposes a generic online data transformation that enables scalable big data analysis in streaming applications.

It is known that the most accurate low-rank approximation of a data collection is computed by SVD or PCA in settings where the column span of the data admits a lower-dimensional embedding [Golub and Reinsch 1970]. However, the large memory footprint and $\mathcal{O}(m^2n)$ computational complexity of these well-known sketching algorithms make it impractical to use them for analyzing massive and dynamic datasets. Unlike PCA, Sparse PCA (SPCA) is modified to find principal components with sparse loadings, which is desirable for interpreting data and storage reduction [Zou et al. 2006]. The computational complexity of SPCA is similar to classic SVD. Thus, even this method is not scalable for analyzing massive datasets [Zou et al. 2006] and [Papailiopoulos et al. 2013].

The efficiency of random subsampling methods to compute the lower-dimensional embedding of large datasets has been shown in Dyer et al. [2013] and Drineas and Mahoney [2005]. Random Column Subset Selection (rCSS) has been proposed as a scalable strategy for sketching large matrices [Dyer et al. 2013]. Although the authors in Dyer et al. [2013] had provided a theoretical scalable approach for large matrix sketching, but the hardware constraints are not considered in this work. The large memory footprint and non-adaptive structure of their rCSS approach make it unsuitable for streaming applications.

Mirhoseini et al. [2015] and Mirhoseini et al. [2016] have proposed scalable and sparsity-inducing methodologies to enable efficient execution of large-scale iterative learning algorithms on massive and dense datasets. Their approach, however, is static and incurs a large memory footprint, which bound their applicability to cope with dynamic ever-growing datasets. Despite the work in Mirhoseini et al. [2015] and Mirhoseini et al. [2016], 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 [Andrecut 2008; Blanchard and Tanner 2013; Fang et al. 2011], ASICs [Maechler et al. 2010], many-cores [Kulkarni et al. 2016a, 2016], and FPGAs [Septimus and Steinberg 2010; Bai et al. 2012; Ren et al. 2013; Stanislaus and Mohsenin 2013; Kulkarni et al. 2016b] are reported in the literature to speed up this complex reconstruction algorithm.

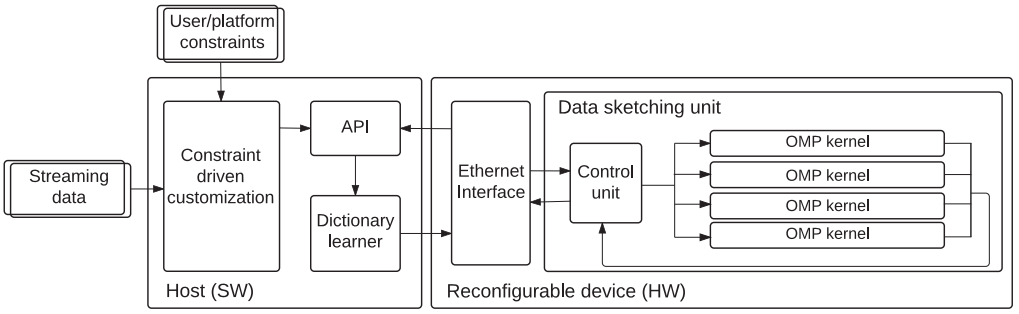


Fig. 1. High-level block diagram of SSketch. It takes a stream of data as input and adaptively learns a corresponding sketch of the collection by doing computation at the level of matrix rank. The resulting sketch is then sent back to the host for further analysis depending on the application.

FPGA implementation of OMP for problems of dimension 32×128 and 256×1024 are developed for signals with sparseness of 5 and 36, respectively [Septimus and Steinberg 2010; Bai et al. 2012]. To the best of our knowledge, none of the previous implementations of OMP are 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 dynamic large data collections.

4. SSKETCH GLOBAL FLOW

The global flow of SSketch is presented in Figure 1. SSketch takes the stream of a massive, dynamic dataset in the matrix form as its input and characterizes the underlying physical resource constraints by running an automated micro-benchmark. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as input and customizes the framework accordingly for an optimized sketch computation. The input parameters of this unit may include runtime, power and memory constraints, and its output includes sketching algorithmic parameters as well as guidelines for the hardware mapping. Our sketch formation algorithm is devised to minimize the costly message passings to/from the memory and cores, and thereby it reduces the communication delay and energy. All SSketch's computations are done in IEEE 754 single-precision floating-point format. Use of floating-point computation assures a much larger dynamic range (i.e., the largest and smallest numbers that can be represented) that is particularly important in providing a generic solution for processing massive dense data collections or datasets where the range may be unpredictable as data evolves over time.

SSketch is developed based on a novel sketching algorithm that we introduce in Section 5. As illustrated in Figure 1, SSketch consists of two main components to compute the sketch of dynamic data collections: (i) a dictionary learning unit that is devised in software and (ii) a data sketching unit that is implemented using hardware accelerators. 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 the block-sparse matrix. In the data sketching unit, the representation of each newly arriving sample is computed based on the current values of the dictionary, and the result is sent back to the host where the acquired sketch matrix is stored. As we discuss in Section 5.2, SSketch leverages a blocking approach to conform the size of the data communicated in between the hardware accelerator and the host to fit the underlying physical constraints (i.e., the I/O bandwidth in the target hardware setting), without introducing any restriction

on the size of the raw data measurement matrix. We provide an accompanying API to facilitate automation and adaptation of SSketch framework for rapid prototyping of an arbitrary matrix-based data analysis algorithm.

5. 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 [Mirhoseini et al. 2015]. The authors in Mirhoseini et al. [2015] suggest a distributed framework based on a scalable and sparsity-inducing solution to find the sketch of large and dense datasets such that:

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

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, $l \ll m \ll n$. $nnz(\mathbf{V})$ 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 that would incur a higher cost and is unsuitable for streaming applications with a single pass requirement and limited memory. Unlike the work in Mirhoseini et al. [2015], our data sketching approach is dynamic, resource aware, and well suited for streaming-based applications. SSketch tailors the solution of Equation (1) according to the underlying platform's constraints. Our approach incurs a fixed memory footprint and is well suited for scenarios where storage is severely limited.

5.1. SSketch Algorithm

Our platform-aware matrix sketching algorithm is summarized in Algorithm 1. The SSketch algorithmapproximates 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

Input: Measurement matrix $\mathbf{A}$, projection threshold α, sparsity level k, error threshold ϵ, and dictionary size l.
Output: Matrix $\mathbf{D}$, and coefficient matrix $\mathbf{V}$.

$\mathbf{D} \leftarrow$ empty;
$j \leftarrow 0$;
for $i = 1,...,n$ **do**
  $W(\mathbf{A}_i) \leftarrow \frac{\|\mathbf{D}(\mathbf{D}^t\mathbf{D})^{-1}\mathbf{D}^t\mathbf{A}_i-\mathbf{A}_i\|_2}{\|\mathbf{A}_i\|_2}$;
  if $W(\mathbf{A}_i) > \alpha$ **and** $j < l$ **then**
    $\mathbf{D}_j \leftarrow \mathbf{A}_i/\sqrt{\|\mathbf{A}_i\|_2}$;
    $\mathbf{V}_{ij} \leftarrow \sqrt{\|\mathbf{A}_i\|_2}$;
    $j \leftarrow j + 1$;
  else
    $\mathbf{V}_i \leftarrow OMP(\mathbf{D}, \mathbf{A}_i, k, \epsilon)$;
  end
end

For each newly arriving sample, SSketch first calculates a projection error, $W(\mathbf{A}_i)$, based on the current values of the dictionary matrix $\mathbf{D}$. This error shows how well the newly added sample can be represented in the space spanned by $\mathbf{D}$. Then, the error is

compared against a user-defined projection threshold α. If the projection error is less than the threshold, then it means the current dictionary matrix $\mathbf{D}$ is good enough to represent the new sample ($\mathbf{A}_i$). Otherwise, SSketch modifies the dictionary matrix to include the new data structure imposed by the recently added sample. SSketch makes use of the greedy OMP routine to compute/update 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). 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 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 shown in Algorithm 1, SSketch 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, $rank(\mathbf{DD}^+\mathbf{A}) = rank(\mathbf{D}) \leq l \leq m$, where $\mathbf{D}^+$ denotes the pseudo-inverse of the dictionary matrix ($\mathbf{D}^+ = (\mathbf{D}^t\mathbf{D})^{-1}\mathbf{D}^t$). 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 9, computational complexity of the projection step (line 4 of Algorithm 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 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.

ALGORITHM 2: OMP Algorithm

Input: Matrix $\mathbf{D}$, measurement $\mathbf{A}_i$, sparsity level k, threshold error ϵ.
Output: Coefficient vector $\mathbf{v}$.
1 $r^0 \leftarrow \mathbf{A}_i$;
2 $\Lambda^0 \leftarrow \emptyset$;
3 $i \leftarrow 1$;
4 **while** $i \leq k$ *and* $r > \epsilon$ **do**
5 $\quad \Lambda \leftarrow \Lambda \cup argmax_j | < r^{i-1}, \mathbf{D}_j > |$ Find best fitting column ;
6 $\quad \mathbf{v}^i \leftarrow argmin_{\mathbf{v}} \|r^{i-1} - \mathbf{D}_{\Lambda^i}\mathbf{v}\|_2^2$ LS Optimization ;
7 $\quad r^i \leftarrow r^{i-1} - \mathbf{D}_{\Lambda^i}\mathbf{v}^i$ Residual Update;
8 $\quad i \leftarrow i + 1$;
9 **end**

Algorithm 2 demonstrates the pseudocode of OMP where ϵ is a predefined error threshold and k is the target sparsity level. The Least-Squares (LS) minimization step (line 6 of Algorithm 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 the Cholesky method within the OMP algorithm would reduce its hardware implementation complexity and make it well suited for hardware accelerators [Bai et al. 2012; Stanislaus and Mohsenin 2012].

To efficiently solve the LS optimization problem in line 6 of Algorithm 2, we decide to use QR decomposition (Algorithm 3). QR decomposition returns an orthogonal matrix $\mathbf{Q}$ and an upper-triangular matrix $\mathbf{R}$. It iteratively updates the decomposition by reusing

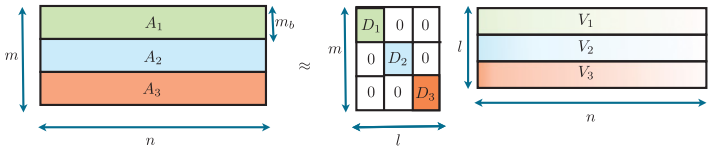


Fig. 2. Schematic depiction of blocking SSketch.

ALGORITHM 3: Incremental QR Decomposition by Modified Gram-Schmidt

Input: 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

$$R^s \leftarrow \begin{pmatrix} R^{s-1} & 0 \\ 0 & 0 \end{pmatrix}$$

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

the $\mathbf{Q}$ and $\mathbf{R}$ matrices from the last OMP iteration (we call this method OMPQR). In this approach, the residual (line 7 of Algorithm 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 as l is much smaller in compared to m and n in many real-world settings. This linear complexity enables SSketch to readily scale up for processing a large amount of data based on a streaming model.

5.2. 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 Figure 2, then the combined sketch can be as good as sketching $\mathbf{A}$ directly (*nonblocking SSketch*) in terms of error-performance tradeoff. 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 $\mathbf{D}$ 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 are 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 $\mathbf{D}$, as it is presented in Section 9, 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 $\mathbf{V}$. 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.

Theoretical bound on SSketch approximation error. In SSketch methodology, the dictionary matrix $\mathbf{D}$ is constructed such that the column space of $\mathbf{D}$ is contained in the column space of the data matrix $\mathbf{A}$. To bound the reconstruction error using SSketch, we propose Theorem 5.1.

THEOREM 5.1. *In blocking SSketch, the reconstruction error of a massive, dynamic input data* $\boldsymbol{A}$ *is* $\frac{\|\mathbf{A}-\mathbf{DV}\|_F^2}{\|\mathbf{A}\|_F^2} \leq max(\alpha, \epsilon)$.

PROOF. Let $\mathbf{A}_{uj}$ represent the uth segment of the jth column of input data matrix $\mathbf{A}$. In blocking SSketch, for each newly arriving sample, if $W(\mathbf{A}_{uj}) \geq \alpha$, then it is added to the corresponding sub-block in the dictionary matrix $\mathbf{D}$, see Figure 2. As such, the reconstruction error for the added $\mathbf{A}_{uj}$ is exactly zero via SSketch's methodology. For the remaining part of the input data, the greedy OMP routine is used to compute sparse approximation of the sample.

In our methodology, when the dictionary size l is set to be large enough (e.g., $l = m_b$ where m_b is the block size), the set of l samples that are linearly independent will span the ambient dimension of the corresponding data block $\mathbb{R}^{m_b}$, which results in exact decomposition, that is, $\|\mathbf{A}_{uj} - \mathbf{D}_u\mathbf{D}_u^+\mathbf{A}_{uj}\|_F = 0$. OMP routine does not stop unless either (i) the reconstruction error ($\frac{\|\mathbf{A}_{uj}-\mathbf{D}_u\mathbf{V}_{uj}\|_F}{\|\mathbf{A}_{uj}\|_F}$) reaches a value less than or equal to ϵ or (ii) all the column samples in $\mathbf{D}_u$ are used for reconstructing $\mathbf{A}_{uj}$, where $\mathbf{D}_u$ is the corresponding dictionary sub-block for the input segments $\mathbf{A}_u$. In case (i), the normalized reconstruction error (a.k.a., the residual) is less than ϵ, which in turn ensures that $\frac{\|\mathbf{A}_{uj}-\mathbf{D}_u\mathbf{V}_{uj}\|_F}{\|\mathbf{A}_{uj}\|_F} \leq \epsilon$ after OMP computation. In case (ii), the residual would be $\frac{\|\mathbf{A}_{uj}-\mathbf{D}_u\mathbf{D}_u^+\mathbf{A}_{uj}\|_F}{\|\mathbf{A}_{uj}\|_F}$, where $\mathbf{D}_u^+\mathbf{A}_{uj}$ is equal to $\mathbf{V}_{uj}$ since all columns of $\mathbf{D}$ are selected in the process of OMP computation for that sample. As such, the normalized reconstruction error would be less than or equal to α according to line 4 of Algorithm 1. Thereby, for each $\mathbf{A}_{uj}$ we have the following:

$$\frac{\|\mathbf{A}_{uj} - \mathbf{D}_u\mathbf{V}_{uj}\|_F^2}{\|\mathbf{A}_{uj}\|_F^2} \leq max(\alpha, \epsilon)$$

$$\|\mathbf{A}_{uj} - \mathbf{D}_u\mathbf{V}_{uj}\|_F^2 \leq max(\alpha, \epsilon)\|\mathbf{A}_{uj}\|_F^2. \tag{2}$$

Summing up Equation (2) over all blocks for an input sample $\mathbf{A}_j$ results in the following:

$$\sum_{I=1}^{\frac{m}{m_b}} \|\mathbf{A}_{uj} - \mathbf{D}_i\mathbf{V}_{uj}\|_F^2 \leq \sum_{I=1}^{\frac{m}{m_b}} max(\alpha, \epsilon)\|\mathbf{A}_{uj}\|_F^2$$

$$\|\mathbf{A}_j - \mathbf{D}\mathbf{V}_j\|_F^2 \leq max(\alpha, \epsilon)\|\mathbf{A}_j\|_F^2. \tag{3}$$

Equation (3) is a result of the blocking structure of the dictionary matrix $\mathbf{D}$ (Figure 2). Finally, the overall reconstruction error can be presented by summing up Equation (3) over all n input samples,

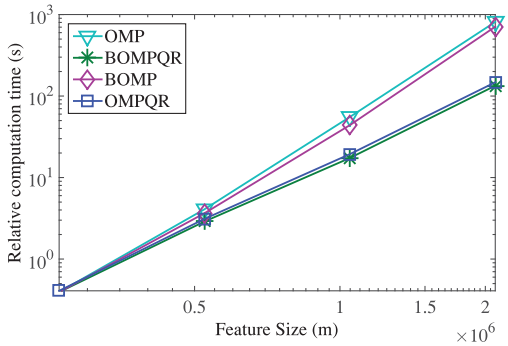


Fig. 3. Computational complexity comparison of different OMP implementations. Using QR decomposition significantly improves OMP's runtime.

$$\sum_{j=1}^{n} \|\mathbf{A}_j - \mathbf{D}\mathbf{V}_j\|_F^2 \leq \sum_{j=1}^{n} max(\alpha, \epsilon)\|\mathbf{A}_j\|_F^2$$

$$\|\mathbf{A} - \mathbf{D}\mathbf{V}\|_F^2 \leq max(\alpha, \epsilon)\|\mathbf{A}\|_F^2 \quad \square \tag{4}$$

5.3. Scalability

In Figure 3, we provide an empirical comparison between the computational complexity of different OMP implementations. 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 the OMPQR method in our target architecture since it is more memory efficient and better suited for hardware acceleration as previously shown in Stanislaus and Mohsenin [2013] and Kulkarni et al. [2014].

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 sketching 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 Algorithm 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 $\mathbf{D}\mathbf{D}^+$ with $\mathbf{Q}\mathbf{I}_t\mathbf{Q}^t$, then the projection step's computational complexity would be reduced to $(2lm + l^2m)$. Assuming $\mathbf{D} = \mathbf{Q}\mathbf{R}$, then the projection matrix can be written as follows:

$$\begin{aligned}\mathbf{D}(\mathbf{D}^t\mathbf{D})^{-1}\mathbf{D}^t &= \mathbf{Q}\mathbf{R}(\mathbf{R}^t\mathbf{Q}^t\mathbf{Q}\mathbf{R})^{-1}\mathbf{R}^t\mathbf{Q}^t \\ &= \mathbf{Q}(\mathbf{R}\mathbf{R}^{-1})(\mathbf{R}^{t^{-1}}\mathbf{R}^t)\mathbf{Q}^t = \mathbf{Q}\mathbf{I}_t\mathbf{Q}^t,\end{aligned} \tag{5}$$

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

Table I compares different sketching 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. Computational Complexity of Different Sketching Methods

Sketching Algorithm	Computational Complexity
SVD	$m^2n + m^3 \approx \mathcal{O}(m^2n)$
SPCA	$lmn + m^2n + m^3 \approx \mathcal{O}(m^2n)$
SSketch (this article)	$n(lm + l^2m) + mnl^2 \approx \mathcal{O}(mnl^2)$

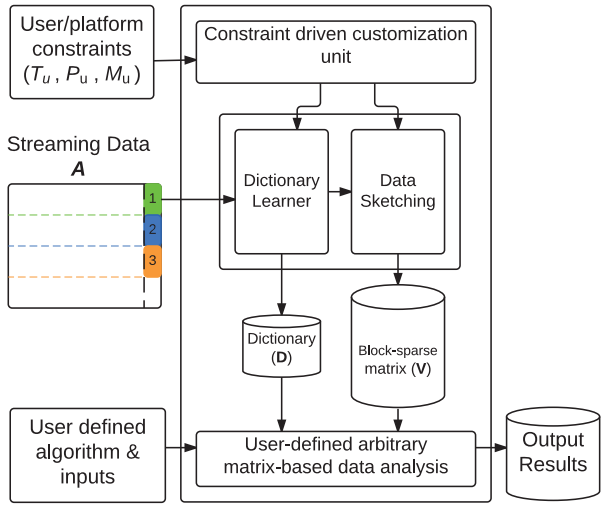


Fig. 4. High-level diagram of SSketch API. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and delivers output parameters that can be used for an optimized sketch computation.

6. SSKETCH AUTOMATED HARDWARE-ACCELERATED IMPLEMENTATION

In this section, we discuss the details of SSketch hardware-accelerated 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 $\mathbf{V}$. As the stream of data arrives, the control unit looks for the availability of OMP kernels and assigns the newly arriving 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, see Figure 4. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and customizes the framework for an optimized sketch computation. The input parameters of the constraint-driven customization unit may include runtime, power, and memory constraints (T_u, P_u, and M_u, respectively) and its outputs include sketching algorithmic parameters as well as guidelines for hardware mapping. Users can then use the transformed data to scalably perform an arbitrary matrix-based data analysis on an ensemble of lower-dimensional structures rather than the original matrix without a significant loss. Note that the 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, are programmable at runtime and can be easily changed through SSketch API.

In OMP hardware implementation, we utilize several techniques to reduce the iteration interval of two successive operations and exploit the parallelism within the

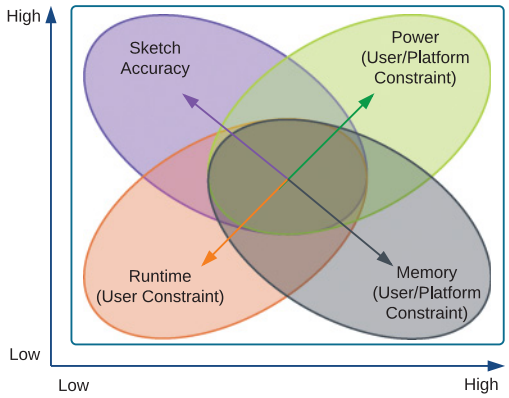


Fig. 5. Overview of automated SSketch customization. SSketch leverages the degree of freedom in producing several data projection subspaces to optimize for the pertinent resource provisioning including runtime, power, and memory.

algorithm. We observe that the OMP algorithm includes multiple dot product computations that result in frequent appearance of for-loops requiring an operation similar to $a += b[i] \times c[i]$. We use a *tree-based reduction module* by implementing a tree-based adder to accelerate the dot product and norm computation 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 memory. To cope with the concurrency, instead of having a large block Random Access Memory (RAM) for matrices $\mathbf{D}$ and $\mathbf{Q}$, 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 optimize the amount of utilized block memories. We reduce block RAM utilization in SSketch's 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 $\mathbf{D}$ would be selected twice during one call of the OMP algorithm. This is particularly because the updated residual at the end of each iteration is made orthogonal to the selected dictionary samples. Thereby, for computing line 5 of Algorithm 2, we only use the indices of $\mathbf{D}$ that are not selected during the previous OMP iterations. We instead use the memory space that was originally assigned to the selected columns of $\mathbf{D}$ to store the newly added columns of matrix $\mathbf{Q}$. By doing so, we reduce the block RAM utilization, which allows SSketch to employ more OMP kernels in parallel.

7. AUTOMATED SSKETCH CUSTOMIZATION

As illustrated in Figure 5, there are four main directions along which the underlying data transformation could be optimized: accuracy, runtime, memory bandwidth, and power. SSketch is devised with an automated constraint-driven customization unit to deliver best data projection under a given set of physical resources and constraints.

As we demonstrate in Section 9, there is a tradeoff between the sketch matrix accuracy and SSketch's physical performance that can be carefully leveraged to improve the efficiency of sketch matrix computation. By increasing the dictionary size l, SSketch can better capture the hybrid structure of the input data, which results in more non-zero elements in the block-sparse matrix $\mathbf{V}$ and typically higher sketch accuracy.

Note that in the SSketch framework each dictionary sub-block is constructed from the columns of its corresponding data sub-matrix, see Figure 2. The column space of the dictionary is contained in that of the data matrix, which implies that the rank of each dictionary sub-block is less than m_b, where m_b is the block size. Therefore, in the blocking SSketch approach, each dictionary sub-block can at most consist of m_b independent samples. This automatically ensures that the memory constraint is satisfied. However, in practice, one might need to customize the architecture such that not only memory constraint is addressed but also runtime and power constraints are taken into consideration. SSketch takes the runtime, power, and memory constraints into account and subsequently tunes its algorithmic parameters to deliver the sketch matrix with the least approximation error in each scenario.

7.1. Constraint-Driven Optimization

Memory constraint on computing platforms is one of the main limitations in the big data regime. Blocking SSketch computes the sketch of dynamic data collections by breaking up the data into more manageable blocks according to the memory budget. In the SSketch framework, the memory requirement can be approximated by $((lm_b + m_b) \times (n_k + 1) + n_k l^2) \times 4$ bytes, where n_k is the number of OMP kernels, m_b is the block size, and l is the number of samples in the dictionary matrix. The $(lm_b + m_b)$ term corresponds to the resource requirement to store a sub-block of dictionary matrix $\mathbf{D}$ and a single column of the input matrix $\mathbf{A}$ at a time, while the l^2 factor denotes the memory storage assigned to the matrix $\mathbf{R}$ in each OMP kernel.

Total runtime in SSketch framework ($T_{SSketch}$) can be expressed as:

$$\begin{aligned} T_{SSketch} &\approx \underset{learning}{T_{dictionary}} + \underset{Overhead}{T_{Communication}} + \underset{Computation}{T_{FPGA}} \\ &\approx \beta_0 mn + \beta_1 mn(l + l^2) + \beta_2 \frac{mn(kl + k^2)}{n_k}, \end{aligned} \tag{6}$$

where β_is are constant coefficients that characterize the runtime requirement per unit of floating point operation. The latter term in Equation (6) represents the runtime cost of computing the block-sparse matrix $\mathbf{V}$, which is the dominant factor in SSketch's runtime, as we experimentally illustrate in Section 8. To deliver the most accurate sketch matrix, SSketch solves the optimization objective described in Equation (7). It maximally exploits the existing sparsity in a dataset to effectively improve the performance of iterative matrix computations while adhering to a set of user/platform physical constraints. The SSketch's constraint-driven optimization can be expressed as follows:

$$\underset{l,\, m_b,\, n_k}{minimize}(approximation\ error), \tag{7}$$

$$\begin{aligned} subject\ to: \quad & l \leq m_b, \\ & m_b \leq m, \\ & n_k \in \mathbb{N}, \\ & \beta_2 mnl^2/n_k \leq T_u, \\ & P_{SSketch} \leq P_u, \\ & (l+1)(n_k+1)m_b + n_k l^2 \leq M_u, \end{aligned}$$

where T_u, P_u, and M_u are a set of user-defined parameters that imply the underlying physical constraints in terms of runtime, power, and memory, respectively. The term mnl^2/n_k reflects the computational complexity of SSketch (Table I), where n_k samples can be processed in parallel using n_k OMP kernels available in the data sketching

Table II. Virtex-6 Resource Utilization

	Slice Registers	Slice LUTs	RAM B36E1	DSP 48E1s
OMP Kernel	12285×4	21333×4	85×4	89×4
Ethernet Interface	1,172	2,022	45	0
Controller Unit	5	96	28	0
Total Utilization (Percentage)	50,317 (16%)	87,450 (58%)	413 (99%)	356 (46%)

unit. As we show in Section 8, the total power consumption of SSketch ($P_{SSketch}$) has a linear correlation with the number of OMP kernels that are concurrently in use for data sketching.

SSketch approximates the solution of Equation (7) using the Karush-Kuhn-Tucker conditions. To efficiently capture the hybrid structure of streaming data collections, SSketch automatically customizes its framework according to the application and tunes the algorithmic parameters including block size m_b, dictionary size l, as well as the number of OMP kernels n_k. To facilitate automation, we provide a solver for our optimization approach. The solver gets the constraints from the user as inputs and uses our Mathematica-based computational software program to solve the optimization. Note that this constraint-driven optimization is a one-time process that incurs a constant and negligible overhead regardless of data size.

8. HARDWARE SETTINGS AND RESULTS

We use Xilinx Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit as our hardware accelerator 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 and software-based realization of SSketch (used for comparison purposes). We employ Xilinx standard IP cores for single precision floating-point operations. Xilinx ISE 14.6 is used to synthesize, place, route, and program the FPGA.

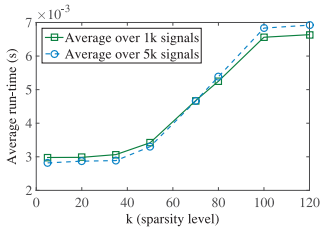
Table II breaks down the Virtex-6 resource utilization for our realization of SSketch framework. Our implementation includes four OMP kernels, a controller unit, and an Ethernet interface. For factorizing matrix $\mathbf{A}_{m\times n}$, there is no specific limitation on the size n due to the streaming nature of SSketch. However, the FPGA block RAM 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 is set to be less than 256. As such, each block of the dictionary consists of a maximum 256 samples ($l \leq 256$). We set the sparsity level k equal to the dictionary size l to give users the flexibility to process different data samples without inducing a limitation on the target sparsity. Note that SSketch's parameters are changeable in SSketch API. So if a designer decides to choose a different set of algorithmic parameters for any reason, she can easily modify the parameters at runtime corresponding to the underlying application.

To corroborate the scalability and practicability of SSketch, we use synthetic data with dense (non-sparse) correlations of different sizes, as well as the hyperspectral [Salina 2014] and light field image datasets [LightField 2014] that we discuss more in detail in Section 9. Table III compares the required runtime per sample for processing different-sized synthetic data using blocking SSketch versus an optimized software-only realization on a single-core 3.40GHz Central Processing Unit (CPU) using the OMP implementation in Rubinstein [2009]. In this experiment, sparsity level k is set to be 50% of the feature space size m, and the overall runtime is averaged over 1,000 samples. As illustrated, the required runtime for updating SSketch's block-sparse matrix $\mathbf{V}$ corresponding to each arriving sample is a linear function of block-size m_b (Section 5.3), which is set to be 256 for SSketch evaluation in Table III.

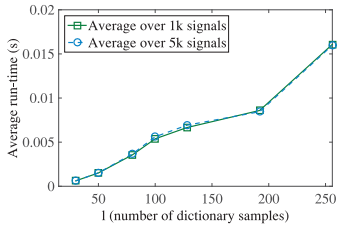
Figure 6 demonstrates the average runtime of SSketch as a function of sparsity level (k), dictionary size (l), and number of OMP kernels (n_k). According to the OMP

Table III. Average Runtime Improvement per Sample Achieved by SSketch Compared to an Optimized Software-Based OMP Implementation

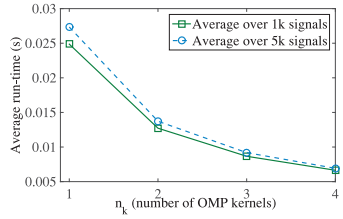
Feature Size (m)	Average Runtime per Sample (Software-only Implementation)	Average Runtime per Sample (SSketch HW/SW Co-design)	Improvement
256	12.5ms	3.41ms	3.68×
512	63.82ms	6.78ms	9.41×
1,024	511.50ms	13.69ms	37.36×
2,048	5.91×10^3ms	27.61ms	214.05×



(a) Runtime per signal in blocking SSketch vs. the sparsity level, k, with $l = 128$, m_b = 256, $\epsilon = 0.001$, and n_k = 4.



(b) Runtime per signal in blocking SSketch vs. the dictionary size, l, with $k = l$, m_b = 256, $\epsilon = 0.1$, and n_k = 4.



(c) Runtime per signal in blocking SSketch vs. number of OMP kernels, n_k, with $k = l = 128$, m_b = 256, and $\epsilon = 0.1$.

Fig. 6. SSketch's FPGA-accelerated implementation performance. l, k, ϵ, and m_b are SSketch's algorithmic parameters that indicate the number of samples in the dictionary matrix, target sparsity level, error threshold, and block size, respectively. n_k denotes the number of OMP kernels that work in parallel in the data sketching unit. In both (b) and (c), we set $k = l$ to let k be as large as the dictionary size and use ϵ as the stopping criteria in the OMP algorithm.

Table IV. SSketch Total Processing Time Is Linear in Terms of the Number of Processed Samples. In This Experiment, $m_b = 256$ and $\alpha = 0.1$

Number of Samples (n)	$T_{SSketch}$ ($l = 128$)	$T_{SSketch}$ ($l = 64$)
1K	3.63s	2.31s
5K	21.03s	12.01s
10K	43.44s	24.32s
20K	90.76s	48.52s
50K	219.65s	123.07s

algorithm, for each newly arriving sample, we expect $(m(kl + k^2))$ operations for updating the block-sparse matrix $\mathbf{V}$. As illustrated in Figures 6(a) and (b), our hardware-accelerated implementation of SSketch follows the same trend as predicted by the OMP algorithm complexity. Note that none of the columns of matrix $\mathbf{D}$ would be selected twice during one call of the OMP algorithm. In our hardware-accelerated realization, we compute line 5 of Algorithm 2 by searching among those columns of the dictionary that have not been selected during the previous OMP iterations. Our hardware implementation approach results in a milder slope for larger ks in Figure 6(a). For example, this property can be seen for $100 \leq k \leq 128$ versus $80 \leq k \leq 100$ in Figure 6(a). In Figure 6(c) the average runtime of SSketch is illustrated as a function of n_k, where n_k is the number of OMP kernels. The kernels are used in parallel to compute the block-sparse matrix $\mathbf{V}$. As Figure 6(c) demonstrates, the average runtime of SSketch is proportional to $1/n_k$, which experimentally confirms that the runtime of SSketch ($T_{SSketch}$) is dominated by the runtime cost of computing the block-sparse matrix $\mathbf{V}$.

The processing time of SSketch is dominated by the OMP computation in the FPGA (Section 7.1). Table IV reflects the total runtime of SSketch ($T_{SSketch}$) as defined in

Table V. Power Consumption on Virtex 6 for Different Numbers of OMP Kernels

Number of Kernels (n_k)	Power Consumption
1	0.358W
2	0.447W
3	0.546W
4	0.634W

Equation (6) for processing different numbers of samples. As shown, the total latency is a linear function of the number of processed samples, which experimentally confirms the scalability of our HW/SW co-design approach to compute the sketch matrix of a collection as data evolve over time. The average overhead delay for communicating between the processor (host) and accelerator contributes less than 4% to the total runtime.

Table V reflects the total power consumption of our hardware-accelerated implementation of SSketch on Xilinx Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit. The values are simulated by Xpower analyzer tools [XPower 2012] in Xilinx ISE and account for both leakage and dynamic power. The FPGA is programmed with a speed grade of -1 and the board is operating at 2.5V. As demonstrated in Table V, the total power consumption of SSketch is a linear function of the number of OMP kernels that are employed in data sketching unit.

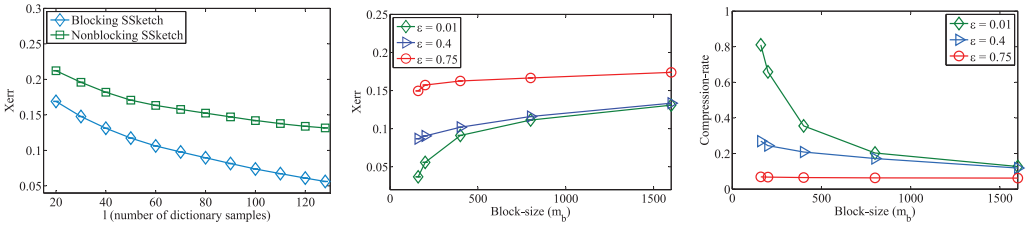
9. PRACTICAL DESIGN EXPERIMENTS

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 arriving order, for each fixed set of algorithmic parameters we shuffle the data before applying SSketch algorithm. The mean error value for 10 calls of SSketch and its variance are reported in Figures 7 and 8. 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 the SSketch algorithm has a low dependency on the data arrival order. This is particularly of interest in streaming application where the data matrix evolves over the course of time.

9.1. 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 three-dimensional model of the scene [Marwah et al. 2013]. For evaluating SSketch algorithm accuracy, we run our experiments on a light field data consisting of 2,500 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 the 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 [Tibshirani 1996].

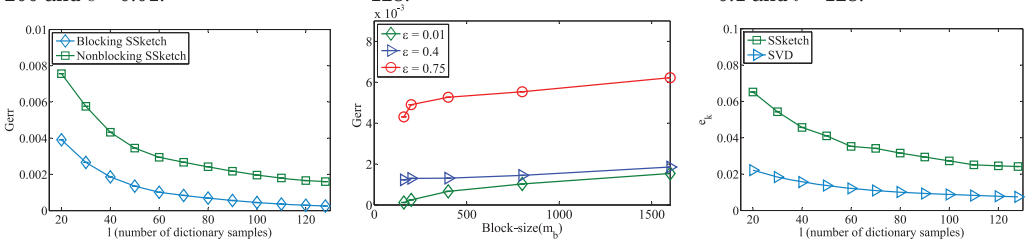
In the SSketch framework, the dictionary size l has a direct effect on the achieved approximation error as well as system speed. Transformation with larger l results in a smaller approximation error at the cost of decreasing system performance in terms of runtime due to the increase in computation. Figures 7(a) and 7(d) report the results of applying both nonblocking and blocking SSketch on the data. We define



(a) $Xerr$ vs. l with α = 0.1, m_b = 200 and ϵ = 0.01.

(b) $Xerr$ vs. m_b with α = 0.1 and l = 128.

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



(d) $Gerr$ vs. l with α = 0.1, m_b = 200 and ϵ = 0.01.

(e) $Gerr$ vs. m_b with α = 0.1 and l = 128.

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

Fig. 7. Experimental evaluations of SSketch. α, ϵ, l, and m_b are SSketch's algorithmic parameters that indicate the projection threshold, error threshold, number of samples in the dictionary matrix, and block size, respectively. *Xerr* represents the data matrix sketching error and *Gerr* represents the approximation error for the corresponding Gram matrix. The spectral transformation error is denoted by e_k.

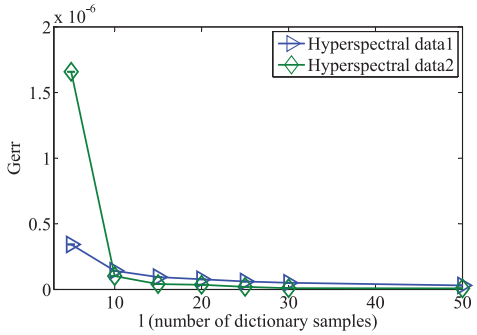


Fig. 8. Approximation error (*Gerr*) vs. l for two different hyperspectral datasets. The SSketch algorithmic parameters are set to $\alpha = 0.1$ and $\epsilon = 0.01$, where α is the projection threshold and ϵ is the error threshold.

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 dictionary matrix $\mathbf{D}$, blocking SSketch could result in a more accurate sketch compared with the nonblocking approach, as illustrated in Figures 7(a) and (d). In this setting, the higher accuracy of blocking SSketch is at the cost of a larger number of non-zeros per column of the block-sparse matrix $\mathbf{V}$. Number of rows in each block of input data (block size) has a direct effect on SSketch's performance. Figures 7(b), (e), and (c) 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 $\frac{nnz(\mathbf{D})+nnz(\mathbf{V})}{nnz(\mathbf{A})}$. As illustrated, one can always reduce the number of non-zeros per column of matrix $\mathbf{V}$ by increasing the SSketch error threshold

ϵ. In particular, there is a tradeoff between the data sketch accuracy and the number of non-zeros in the block-sparse matrix $\mathbf{V}$. This tradeoff ultimately controls the data compression rate and the memory bandwidth required to perform the subsequent machine-learning algorithm using the computed data sketch matrix. Thereby, SSketch tunes its algorithmic parameters to optimize for the underlying constraints imposed by the application data and/or hardware platform.

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 kth singular value of $\mathbf{A}$ and $\mathbf{A}_k$ is obtained by SVD [Martinsson et al. 2010]. Figure 7(f) compares e_k and the theoretical minimal error for the light field dataset.

9.2. 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 [Plaza et al. 2011]. 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 the 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 [Stanford 2014] is $148 \times 691{,}614$ and the second one [Salina 2014] is of size $204 \times 54{,}129$. Our experiments show that SSketch algorithm results in the same trend as in Figures 7(a) and (d) for both hyperspectral datasets. As Figure 8 illustrates, the Gram matrix approximation error reaches to less than 0.2×10^{-6} for $l \geq 10$ in both datasets.

9.3. Hardware Customized Experiments

Here, we provide three experiments to demonstrate the advantage of SSketch's automated tuning approach for user/platform specific customization and performance optimization. In all these experiments, we compare the approximation results with the actual optimal values. As can be seen, our approximations for SSketch algorithmic parameters are very close to the real optimized values, which implies the accuracy of SSketch automated constraint-driven customization. Our experiment platform is a Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit with 1872kB available block RAM memory [Datasheet 2014].

Experiment (i): Consider a scenario where a user requires computing the sketch matrix of a dynamic data collection with a size of $m = 256$ and $n = 5{,}000$ within a runtime budget of 30s. In this case, the memory and power are only limited by the target platform; in other words, there are no user constraints in terms of memory or power. Since no power constraint exists, n_k (the number of OMP kernels) is set at its maximum value, which is determined by the memory budget. As it shown in Table II, four OMP kernels fit on the Virtex-6-XC6VLX240T FPGA with 1,872kB of available block RAM memory. Figure 9 demonstrates the average runtime for a sample sketch update as a function of dictionary size in our real-world evaluation on FPGA. As Figure 9(a) illustrates, $l = 107$ is the optimal dictionary size one can use to compute the data sketch within the aforementioned set of user/platform constraints. On the

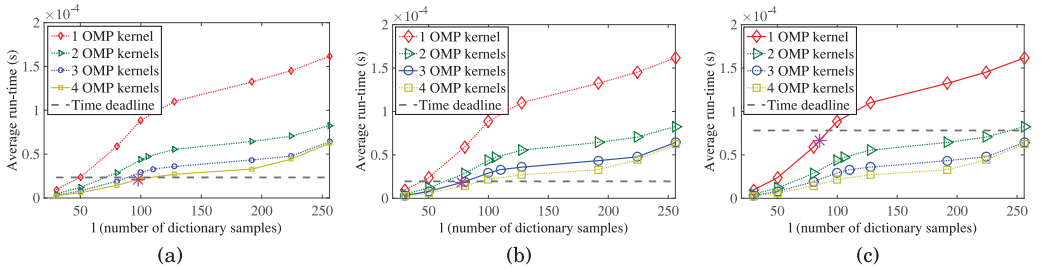


Fig. 9. SSketch's automated constraint-driven customization. Each plot demonstrates the average runtime required to process one sample as a function of the dictionary size. The dashed horizontal line reflects the user runtime deadline, and the star point illustrates the SSketch's automated customization output. The user power constraint selects the number of OMP kernels that can work in parallel. The crossing point of the dashed line and the corresponding runtime curve (solid curve) is the actual optimized point in each setting.

other hand, solving our constraint-driven optimization in Equation (7) results in $l \approx 98$ (star-marked point in Figure 9(a)). Thus, our parametric optimization outputs efficiently model the real-world hardware-accelerated implementation.

Experiment (ii): Consider a scenario where a user requires computing the sketch matrix of a dynamic data collection with a size of $m = 256$ and $n = 5{,}000$ within a runtime budget of 25s and a power budget of 0.6W. In this case, the memory is limited by the target platform. Due to the power constraint, one can make use of three OMP kernels to compute the sketch (Table V). As Figure 9(b) illustrates, $l = 80$ is the optimal dictionary size one can use to compute the data sketch within the aforementioned set of user/platform constraints. On the other hand, solving our constraint-driven optimization in Equation (7) results in $l \approx 78$ (star-marked point in Figure 9(b)).

Experiment (iii): Consider a scenario where a user requires computing the sketch matrix of a dynamic data collection with a size of $m = 512$ and $n = 2500$ within a runtime budget of 91s and a power budget of 0.4W. In this case, the memory is limited to the 1,872kB block RAM memory available within our experiment platform. Due to the power constraint, only one OMP kernel can be used to compute the sketch matrix (Table V). As Figure 9(c) illustrates, $l = 95$ is the optimal dictionary size that can be used to compute the data sketch within the aforementioned set of user/platform constraints. On the other hand, solving our constraint-driven optimization in Equation (7) results in $l \approx 85$ (star-marked point in Figure 9(c)).

The above experiments further demonstrate the applicability and efficiency of our constraint-driven optimization approach proposed in Section 7.

10. CONCLUSION

This article 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 a massive, dynamic data 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 scalable implementation of OMP on FPGA. We propose a constraint-driven optimization method that automatically tunes the sketch matrix computation and embedding properties with respect to user-defined and hardware specifications. To enable handling datasets with arbitrary large number of features, a blocking model is devised that efficiently partitions the feature space for sketch matrix computing. We provide theoretical bounds for the sketch approximation error in the blocking mode. 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 the SSketch runtime to the software realization on a general purpose processor. We also report the delay overhead for communicating between the processor (host) and the accelerator (FPGA). Our evaluations corroborate SSketch scalability and practicability.

REFERENCES

Mircea Andrecut. 2008. Fast GPU implementation of sparse signal recovery from random projections. *arXiv preprint arXiv:0809.1833*.

Lin Bai, Patrick Maechler, Michael Muehlberghuber, and Hubert Kaeslin. 2012. High-speed compressed sensing reconstruction on FPGA using OMP and AMP. In *Proceedings of the 2012 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 53–56.

Jeffrey D. Blanchard and Jared Tanner. 2013. GPU accelerated greedy algorithms for compressed sensing. *Math. Program. Comput.* 5, 3 (2013), 267–304.

Kenneth L. Clarkson and David P. Woodruff. 2009. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 205–214.

Jason Cong, Muhuan Huang, and Peng Zhang. 2014. Combining computation and communication optimizations in system synthesis for streaming applications. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. ACM, 213–222.

N. Council. 2013. Frontiers in massive data analysis. (2013).

Xilinx Datasheet. 2014. Xilinx Virtex 6 Datasheet. Retrieved 2014 from http://www.xilinx.com/publications/prod_mktg/Virtex6_Product_Table.pdf.

Petros Drineas and Michael W. Mahoney. 2005. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.* 6 (2005), 2153–2175.

Eva L. Dyer, Aswin C. Sankaranarayanan, and Richard G. Baraniuk. 2013. Greedy feature selection for subspace clustering. *J. Mach. Learn. Res.* 14, 1 (2013), 2487–2517.

Yong Fang, Liang Chen, Jiaji Wu, and Bormin Huang. 2011. GPU implementation of orthogonal matching pursuit for compressive sensing. In *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 1044–1047.

Gene H. Golub and Christian Reinsch. 1970. Singular value decomposition and least squares solutions. *Numer. Math.* 14, 5 (1970), 403–420.

Pierre Greisen, Marian Runo, Patrice Guillet, Simon Heinzle, Aljoscha Smolic, Hubert Kaeslin, and Markus Gross. 2013. Evaluation and FPGA implementation of sparse linear solvers for video processing applications. *IEEE Trans. Circ. Syst. Vid. Technol.* 23, 8 (2013), 1402–1407.

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, 57–62. DOI:http://dx.doi.org/10.1145/2902961.2902984

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

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

Amey M. Kulkarni, Houman Homayoun, and Tinoosh 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*. ACM, 299–304.

Luis M. Ledesma-Carrillo, Eduardo Cabal-Yepez, Rene de J. Romero-Troncoso, Arturo Garcia-Perez, Roque Osornio-Rios, Tobia D. Carozzi, and others. 2011. Reconfigurable FPGA-Based unit for singular value decomposition of large mxn matrices. In *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 345–350.

Edo Liberty. 2013. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 581–588.

Stanford Dataset Archive LightField. 2014. Retrieved from http://lightfield.stanford.edu/.

Patrick Maechler, Pierre Greisen, Norbert Felber, and Andreas Burg. 2010. Matching pursuit: Evaluation and implementatio for LTE channel estimation. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 589–592.

Gunnar Martinsson, Adrianna Gillman, Edo Liberty, Nathan Halko, Vladimir Rokhlin, Sijia Hao, Yoel Shkolnisky, Patrick Young, Joel Tropp, Mark Tygert, and others. 2010. Randomized methods for computing the singular value decomposition (SVD) of very large matrices. In *Proceedings of the Workshop on Algorithms for Modern Massive Data Sets, Palo Alto*.

Kshitij Marwah, Gordon Wetzstein, Yosuke Bando, and Ramesh Raskar. 2013. Compressive light field photography using overcomplete dictionaries and optimized projections. *ACM Trans. Graph.* 32, 4 (2013), 46.

Azalia Mirhoseini, Eva Dyer, Ebrahim Songhori, Richard Baraniuk, Farinaz Koushanfar, and others. 2015. RankMap: A platform-aware framework for distributed learning from dense datasets. *arXiv preprint arXiv:1503.08169* (2015).

Azalia Mirhoseini, Bita Darvish Rouhani, Ebrahim M. Songhori, and Farinaz Koushanfar. 2016. Perform-ML: Performance optimized machine learning by platform and content aware customization. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 20.

Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. 2012. *Introduction to Linear Regression Analysis*, Vol. 821. John Wiley & Sons.

Dimitris S. Papailiopoulos, Alexandros G. Dimakis, and Stavros Korokythakis. 2013. Sparse pca through low-rank approximations. *arXiv preprint arXiv:1303.0551* (2013).

Franjo Plavec, Zvonko Vranesic, and Stephen Brown. 2013. Exploiting task-and data-level parallelism in streaming applications implemented in FPGAs. *ACM Trans. Reconf. Technol. Syst.* 6, 4 (2013), 16.

Antonio Plaza, Javier Plaza, Alexander Paz, and Sergio Sanchez. 2011. Parallel hyperspectral image and signal processing [applications corner]. *Sign. Process. Mag.* 28, 3 (2011), 119–126.

Sanguthevar Rajasekaran and Mingjun Song. 2006. A novel scheme for the parallel computation of SVDs. In *High Performance Computing and Communications*. Springer, 129–137.

Fengbo Ren, Richard Dorrace, Wenyao Xu, and Dejan Markovic. 2013. A single-precision compressive sensing signal reconstruction engine on FPGAs. In *Proceedings of the 2013 23rd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

Bita Darvish Rouhani, Ebrahim Songhori, Azalia Mirhoseini, and Farinaz Koushanfar. 2015. SSketch: An automated framework for streaming sketch-based analysis of big data on FPGA. In *Proceedings of the 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines Conference (FCCM)* (2015).

R. Rubinstein. 2009. Omp-Box v10. (2009).

Hyperspectral Remote Sensing Dataset Salina. 2014. Retrieved 2014 from http://www.ehu.es/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

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

Anatoli Sergyienko and Oleg Maslennikov. 2002. Implementation of givens QR-decomposition in FPGA. In *Parallel Processing and Applied Mathematics*. Springer, 458–465.

Hyperspectral Dataset Stanford. 2014. Retrieved 2014 from http://scien.stanford.edu/index.php/landscapes.

Jerome L. V. M. Stanislaus and Tinoosh Mohsenin. 2012. High performance compressive sensing reconstruction hardware with QRD process. In *Proceedings of the 2012 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 29–32.

Jerome L. V. M. Stanislaus and Tinoosh Mohsenin. 2013. Low-complexity FPGA implementation of compressive sensing reconstruction. In *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 671–675.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *J. Roy. Stat. Soc. Ser. B* (1996), 267–288.

Wei Zhang, Vaughn Betz, and Jonathan Rose. 2012. Portable and scalable FPGA-based acceleration of a direct linear system solver. *ACM Trans. Reconfig. Technol. Syst.* 5, 1 (2012), 6.

Daniel Zinn, Quinn Hart, Timothy McPhillips, Bertram Ludascher, Yogesh Simmhan, Michail Giakkoupis, and Viktor K. Prasanna. 2011. Towards reliable, performant workflows for streaming-applications on cloud platforms. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 235–244.

Hui Zou, Trevor Hastie, and Robert Tibshirani. 2006. Sparse principal component analysis. *J. Comput. Graph. Stat.* 15, 2 (2006), 265–286.

Received July 2015; revised April 2016; accepted July 2016