2017 IEEE International Parallel and Distributed Processing Symposium Workshops

ExtDict: Extensible Dictionaries for Data- and Platform-Aware Large-Scale Learning

Azalia Mirhoseini*, Bita Darvish Rouhani**, Ebrahim Songhori**, and Farinaz Koushanfar**
Google Brain*, Department of Electrical and Computer Engineering at University of California San Diego**

***Abstract*—This paper proposes ExtDict, a novel data- and platform-aware framework for iterative analysis/learning of massive and dense datasets. Iterative execution is prohibitively costly for distributed architectures where the cost of moving data is continually growing compared with the cost of arithmetic computing. ExtDict creates a performance model that quantifies the computational cost of iterative analysis algorithms on a target platform in terms of FLOPs, communication, and memory, which characterize runtime, energy, and storage respectively. The core of ExtDict is a novel parametric data projection algorithm, called Extensible Dictionary, that enables versatile and sparse representations of the data to minimize this computational cost. We show that ExtDict can achieve the optimal performance objective, according to our quantified cost model, by platform-aware tuning of the Extensible Dictionary parameters. An accompanying API ensures automated applicability of ExtDict to various algorithms, datasets, and platforms. Proof-of-concept evaluations of massive and dense data on different platforms demonstrate more than an order of magnitude improvement in performance compared to the state-of-the-art, within guaranteed user-defined error bounds.**

I. Introduction

Several classes of fast-growing data, including image and video, that comprise a significant portion of newly generated mobile/web content, contain *dense* (non-sparse) dependencies, i.e., a large number of non-zeros in the data correlation matrix. These dense dependencies degrade the performance of most contemporary learning and classification algorithms, which rely on iterative updates on the correlation matrix to converge, including, but not limited to, various forms of regression, sparse approximation, power methods, and support vector machine. The challenges are exacerbated for massive and dense datasets whose handling requires distribution of content across multiple computing cores and/or accelerators. In such distributed settings, the dense correlation structure leads to a high number of costly iterative computations and inter-core communications, a.k.a., *message passing*.

Finding methods for efficient and scalable learning of massive and complex datasets is an active area of research. Broadly speaking, there are two classes of prior work relevant to addressing this standing challenge. On the one hand, the *data-aware* approaches in machine learning and computer vision attempt to ease the iterative computation by finding lower dimensional data structures, e.g., [1], [2], [3]. The limitation of the existing approaches based on the dimensionality reduction is that they are oblivious to the cost and constraints imposed by the underlying platform which may be distributed or heterogeneous.

On the other hand, *platform-aware* methodologies developed in the computer architecture and parallel computing communities provide efficient mapping of the algorithms onto distributed, heterogeneous, or reconfigurable architectures [4], [5]. As such, they must follow the data/correlation structure given by the pertinent dimensionality reduction technique, which is ubiquitous regardless of the hardware platform. Note that the methodologies for iterative analysis of data with a sparse correlation matrix/graph, such as Pregel [6], or GraphLab [7], are of a limited effectiveness for dense datasets. Performance of these techniques seriously degrades for densely correlated data as their underlying algorithms work based on a sparse correlation assumption.

This paper introduces ExtDict, a novel framework for performance-efficient and scalable (iterative) learning of massive data with dense correlations that is *simultaneously data- and platform-aware*. The key observation in ExtDict is that the projection of data into lower dimensional subspaces has a significant impact on the performance of the subsequent mapping onto the distributed or heterogeneous computing platform. The metrics used for quantifying the ExtDict performance are: (i) the number of arithmetic operations (FLOPs), (ii) the amount of messages passed among the processors (communications), and (iii) storage. Common performance indicators such as the overall timing, energy, power, and memory footprint can be directly deduced from these metrics.

The core of ExtDict is a scalable data projection (transformation) technique, called *Extensible Dictionary (ExD)*. ExD is a novel data projection method whit tunable parameters that enables producing several possible projection subspaces within a given error threshold. We leverage this new degree of freedom given by the parametric projection to best customize mapping of data to the target distributed platform. In the ExtDict *preprocessing* phase, given a target projection error[1], the ExD is scalably formed from subsamples of data. We also show how the performance cost of the iterative matrix computation on the pertinent platform can be modeled based on a small subset of the data. Given this platform-aware model, ExtDict automatically tunes the parameters of ExD to optimize the performance of distributed processing on the entire dataset. It is worth noting that the preprocessing cost of the performance minimizing projection is amortized over several runs as the target learning algorithms for dense and massive data require multiple iterations to converge. We also discuss methods for updating the projection given by ExD in case the data structure dynamically changes.

A. Contributions

The detailed contributions of this paper are as follows:

- Introducing ExtDict, the first end-to-end data- and platform-aware framework for scalable iterative learning of big and densely correlated data. Given a performance cost model and a projection error, ExtDict provably optimizes the cost of executing the learning algorithm on the underlying platform.
- Creating a performance model for quantifying the computational efficiency of ExtDict. The quantified cost metrics can be used for optimizing the performance in terms of runtime, energy, and storage.
- Devising Extensible Dictionary (ExD), the first scalable (linear-time) data transformation methodology that can produce several possible projection subspaces within the given error threshold; ExtDict tunes ExD's parameters for achieving the best performance on the underlying platform.
- Developing a new distribution strategy for ExtDict that enables efficient execution of iterative big data analysis

[1]We use transformation or projection error interchangeably.

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.171

algorithms. The proposed strategy simultaneously performs load balancing and communication minimization; it reaches the theoretical lower bounds on the memory/communication bandwidth that are known to be the performance bottlenecks.
- Providing an open-source ExtDict API in C++ that enables an end-to-end distributed implementation using the MPI standard message passing interfaces.
- Demonstrating proof-of-concept evaluations of ExtDict on several massive real-world datasets with up to 5 billion non-zeros. We evaluated ExtDict on important learning algorithms including gradient-descent based regression optimization and power method for applications including image denoising, image super-resolution, and Principle Component Analysis (PCA). The results show more than an order of magnitude improvement in runtime, energy, and memory footprint compared to existing work.

II. Preliminaries

A. Target Data Analysis Algorithms

The key to many important data analysis and learning algorithms is exploring the dependency between various data points. In particular, the pairwise correlation (*Gram*) matrix of the variables or signals is widely used for representing the dependencies. If the original data matrix is denoted by A, the Gram matrix can be written as $G = A^T A$. Using the Gram matrix, most learning algorithms iteratively update a vector of unknown parameters until they converge to a solution. Each update process requires matrix multiplications in the form of $Gx = A^T Ax$, where x is the unknown parameter vector. When it comes to big data analysis, the cost of an update process arises from iterative multiplications of very large matrices and vectors. In particular, if the dataset is too massive and has to be distributed, the cost of communication across multiple computing nodes adds to the overhead of computation.

Examples of such iterative learning algorithms include descent-based algorithms for solving regression problems such as LASSO, Elastic net, and Ridge [8], Power method for finding PCA [9], interior point methods for solving Support Vector Machines (SVM) [10], etc. Due to their universal applicability, the ExtDict framework targets iterative algorithms that operate on massive and dense Gram matrices.

For our evaluations, we consider two algorithms and their applications. The first algorithm is least squares minimization with ℓ_1 regularization, also knows as LASSO [8]. LASSO is a widely popular machine learning approach with a vast range of applications in feature selection [11], pattern recognition [8], classification [12], etc. In our experiments we apply LASSO for image denoising and image super-resolution applications. The second algorithm is Power method, which can be used for solving problems including large-scale PCA [13], spectral partitioning [14], sparse PCA [13], etc. We use Power method to find the eigenvalue and eigenvectors of the data.

B. Target Datasets

While several current techniques and frameworks for running iterative update algorithms on big data rely on data/correlation sparsity to increase efficiency [15], their performance on densely correlated data heavily degrades. This is because of the massive number of arithmetics and communications incurred by the large number of non-zeros of dense data. Many dense datasets, however, benefit from a less apparent property that is known as coarse-grained data parallelism. In other words, dense datasets were shown to often belong to a single or a union of low-rank subspaces [16].

In this work, we show that several important classes of visual datasets, including Light Field, Hyperspectral, and Cancer Cell images demonstrate union of low-rank properties. In particular, we demonstrate how this property can be exploited so that versatile projections of dense data are formed.

III. Related Work

To the best of our knowledge, ExtDict is the first framework based on platform aware data projection for making subsequent ML-based processing more resource efficient. Nonetheless, prior research are related to ExtDict in terms of the problem.

Dimensionality Reduction Approaches. Traditional matrix projection methods such as SVD and PCA become infeasible in large scale as they incur $O(M^2N)$ complexities (with large constant factors) for an $M \times N$ matrix. To address this challenge, randomized algorithms known as Column Subset Selection (CSS) have been developed [17]. CSS approaches project data into a lower dimensional subspace. The projection basis, a.k.a., *dictionary*, is learned by selecting a subset of data columns. While the scalability of Randomized CSS (RCSS) techniques make them most appealing for large data [17], [18], adaptive CSS methods can create more accurate dictionaries [19], [1], [20]. Farahat [19] and Leverage Scores [1] are adaptive methods that aim to minimize the dictionary size to achieve a given approximation error. Both methods require creation and storage of the $N \times N$ correlation matrices which is impractical for dense and large data. oASIS [20] is another adaptive CSS technique that greedily chooses the most informative columns to add to the dictionary. oASIS is memory efficient and its runtime scales linearly with N [20]. Note that the above dimensionality reduction methods can replace ExD within our framework. We evaluate the performance of our framework based on ExD (our proposed platform aware dimensionality reduction) as well as RCSS and oASIS techniques.

Content Aware Methods for Accelerating ML. Previous works demonstrate the usability of data transformation methods for accelerating certain learning/linear algebra problems, including SVM [21], spectral clustering [14], dimensionality reduction [17], least squares, norm-1 minimization algorithms [22], and square-root LASSO [23]. However, unlike ExtDict, all of the above methods are tailored for the specific learning/algebraic problem and are not directly applicable to generic iterative algorithms on the correlation matrix such as LASSO, Elastic Net (least squares with ℓ_1 and ℓ_2 regularization), or Power method. Several prior works have explored the benefit of optimization based on content for performance efficiency [24], [25], [26], [27]. Our earlier work, RankMap, proposes the usability of data transformation to generic iterative update algorithms [28]. However, RankMap (unlike ExtDict) does not perform platform aware optimization. In addition, the error-based criteria for selecting the transformation basis in RankMap prevents it from creating versatile and over-complete dictionaries. Stochastic Gradient Descent (SGD) [29] is another common, greedy approach for accelerating ML that does not always guarantee convergence to the actual solution. It also does not provide memory usage reduction. We also use RankMap and SGD as our comparison basis.

Platform Aware Techniques for Hardware Mapping. Prior research extensively addressed efficient mapping of linear algebra and ML algorithms onto distributed, heterogeneous, or reconfigurable architectures [4], [30]. While such methods effectively optimize the performance with hardware aware mapping, they do not provide any customization with respect

to hidden data geometry. They instead operate on the data given by the application which is ubiquitous regardless of the platform.

IV. Global Flow of ExtDict

The ExtDict framework targets iterative algorithms that operate on the massive and dense correlation or *Gram* matrices. Many contemporary ML algorithms focus on exploring the correlations between different data samples. Examples include descent-based solutions to regression problems such as Ridge and LASSO [8], Power method for finding PCA [9], interior point methods for solving SVM [10], etc. The major cost of an iterative update arises from multiplications on the Gram matrix, i.e., $G = A^T A$, where A is the original data matrix. For large and dense data, an update becomes prohibitively costly due to the huge number of floating point operations and message passing across the processing nodes.

The overall flow of ExtDict is shown in Figure 1. ExtDict exploits the well-understood fact that many ML applications are tolerant to variations in output solution, offering the opportunity to trade the solution accuracy with resource efficiency [17], [18]. In Section V, we introduce our novel and parametric data transformation method, called ExD. ExD seeks to find a *low-dimensional dictionary matrix* D and a *sparse coefficient matrix* C such that:

$$\min \|C\|_0 \text{ } s.t. \text{ } \|A - DC\|_F \leq \epsilon \|A\|_F, \tag{1}$$

where D is $M \times L$, C is $L \times N$, and $L \ll N$. Parameter ϵ is a user-defined approximation error, $\|C\|_0$ is the number of non-zeros in C, and $\| \cdot \|_F$ is the Frobenius norm. ExD is a pre-processing step whose goal is to create a projection such that iterative updates on the transformed components i.e., $(DC)^T DC$, become much more efficient than $A^T A$. The key idea is that dictionary size L can be used to control the redundancy in D, to create different levels of sparsity in C. In other words, by elastically tuning the dictionary size, we can achieve sparser C's.

In Section VI, we propose an optimal distributed computing model to perform iterative computations on DC. We show that L governs the communication cost, thus, there is a trade-off between the number of non-zeros (or computation and the memory footprint) of the projected data and the communication overhead. We propose metrics to quantify the computing cost for our distributed model, which directly characterize memory, runtime, and energy. In Section VII, we demonstrate our approach for tuning ExD to minimize the quantified costs. In Section VIII, we provide our evaluations.

V. Extensible Dictionaries

In this section, we present ExtDict's scalable sparsity driven method for projecting the dense data matrix $A_{M\times N}$ into a product of a dictionary matrix $D_{M\times L}$ and a sparse matrix $C_{L\times N}$ such that the objective function in Equation 1 holds.

A. ExD Algorithm

In Algorithm 1, we outline ExtDict's projection method which we refer to as ExD. The first step is to create the dictionary matrix D by sub-sampling columns of A uniformly at random. Once D is created, Equation 1 becomes equivalent to a generic sparse approximation problem. Each column c_i of C is a sparse approximation of the column a_i of A with respect to D and the user-specified projection error (ϵ). The second step is to solve the sparse approximation problem. To do so, we use Orthogonal Matching Pursuit (OMP) which is a greedy sparse coding routine [31]. When the data is sufficiently sampled such that the span of columns of D is "close" to the span of columns of A, OMP finds a sparse coefficient matrix C such that the error tolerance criteria is met. Setting the error tolerance to zero ($\epsilon = 0$) guarantees achieving the same projection error as least-squares approaches. We note that sparse approximation has never been used for platform-customized performance optimization.

Algorithm 1 : **ExD Algorithm**

Input: Normalized data matrix $A \in \mathbb{R}^{M\times N}$, error tolerance ϵ, number of processors N_P, and number of columns to select L.

Output: A sparse matrix $C \in \mathbb{R}^{L\times N}$ and a dictionary $D \in \mathbb{R}^{M\times L}$ such that $\|A - DC\|_F \leq \epsilon \|A\|_F$.

0. $pid = 0$[2]creates a random subset of in dices of size L (from $1, 2, \ldots, N$), denoted by I_L and broadcasts it to other processors.
1. $pid = i$ loads $D = A(:, I_L)$.
2. $pid = i$ loads $A_i = A(:, \frac{iN}{N_P} : \frac{(i+1)N}{N_P})$.

3. $pid = i$ applies OMP to solve $a_i = Dc_i$ for the tolerance error ϵ:
   3.0. Initialize $r = a_i$,$\phi = \emptyset$
while $\|r\|_2 < \epsilon \|a_i\|_2$ **do**
   3.1. $k = argmax_j |d_j.r|$
   3.2. $\phi = (\phi, k)$.
   3.3. $y = D_\phi^+ a_i$
   3.4. $r = a_i - D_\phi y$
end while

B. Sparsity Guarantees for C

Our approach is motivated by recent results for sparse subspace clustering [12], where the goal is to discover multiple low-dimensional subspaces present in a collection of data and then cluster signals according to their subspace membership. The key intuition is that a sparse representation of a data signal (columns of A) ideally corresponds to a combination of a few other signals of A.

In particular, when columns of A lie on a union of subspaces, if enough columns are selected that represent a particular subspace (cluster), then the representations of the remaining columns in A are guaranteed to be sparse [16]. Formally, we say that a set of N columns of $A = a_1, \ldots, a_N$ each of dimension M, live on a union of N_s subspaces if $a_1, \ldots, a_N \subset \cup_{i=1}^{N_s} U_i$, where U_i is a K_i-dimensional subspace of R^N. In this case those columns of C that lie on a K_i-dimensional subspace will admit a K_i-sparse representation, i.e., at most K_i non-zeros will be used to represent a signal within this subspace. This data modeling is quite general because it can be used to describe data living on a single subspace (low rank model) and/or multiple subspaces that might be corrupted with a few outlier columns [16].

In the following, we provide an example that shows how ExD benefits from the union-of-subspace properties of the data for sparsification. Figure 2 shows a dataset in $\mathcal{R}^2$ space. Each point is represented by two non-zero elements in the direction of X and Y axis. This dataset is full-rank, i.e., an SVD factorization finds two orthogonal basis to represent the data. Figure 2 shows the orthonormal basis that is found via SVD.

[2]Processor ID's are denoted by *pid*. This notation means processor with $pid = i$ ($0 \leq i < N_P$) is in charge of the task.

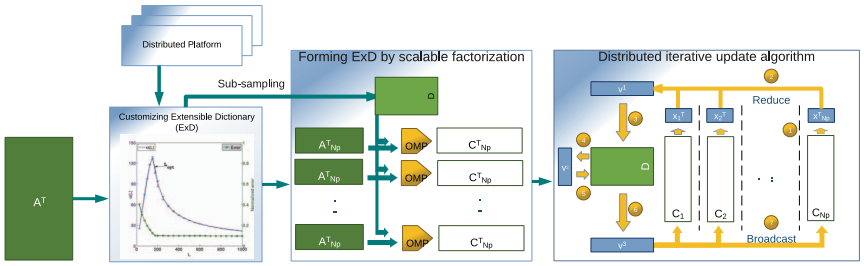


Fig. 1: Global flow of ExtDict: during pre-processing, dataset A is transformed into a dictionary D and a sparse coefficient matrix C. The transformation is platform-specific and is tailored to benefit subsequent processing of data.

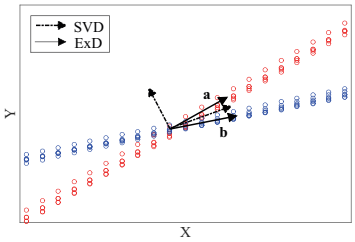


Fig. 2: A union of two 1-dimensional signal models. By selecting vectors a and b as the atoms of the dictionary, each data point can be approximated using only one atom.

However, one can observe that by choosing a more suitable basis for the data, i.e., the two vectors a and b on the figure, we can approximately represent each data point using only one non-zero element in the direction of either a or b. Thus, while the 2D dataset is not inherently low-rank in the classic definition (as its rank is 2), it lies on a union of two lower dimensional (rank-1) subspaces. Our approach uses the union of lower-rank subspaces by forming the dictionary columns, a.k.a., *dictionary atoms* from the sample data points, i.e., a and b in our example. When the data is much higher dimensional than our 2D example, there is a higher degree of freedom in the way atoms in D can be selected.

C. The Extensible Dictionary

As discussed earlier, choosing enough columns to create D is critical for ensuring that we meet the transformation error criteria and achieve sparsity. Note that when $L > M$, with a high probability, matrix D becomes full rank and thus the OMP algorithm converges (meets the error criteria). Theoretical work in the domain of subspace sampling has attempted to find bounds for L with respect to the intrinsic rank of a matrix. Recent work by Mahoney et al. [32] proves that by sampling $L \leq \Omega(\frac{k \log k}{(1-\delta)^2})$ columns at random, a maximum (least-squares) error equal to $\frac{1}{\delta}$ of the nuclear norm of the best K-dimensional approximation of the data (i.e., K-dimensional truncated SVD of A) is guaranteed.

Unlike existing decomposition approaches based on subspace sampling that create matrix C by projecting the data using $C = D^{+}A$ [3], our approach focuses on creating sparse representations in C. Our key observation is that by increasing the redundancy in D (i.e., increasing L), one can vary the sparsity level of C. We extensively use this property to tune ExD in order to optimize the performance in a distributed setting.

[3]The pseudo-inverse is calculated as: $D^{+} = (D^T D)^{-1} D^T$.

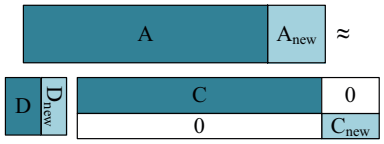


Fig. 3: Updating the transformation for evolving data.

D. Complexity Analysis

The low-overhead and scalability of ExD is crucial for its applicability to large datasets. In Algorithm 1, the main computing task is executing OMP sparse coding routine. In our implementations, we use Batch-OMP based on Cholesky factorization updates [3]. The upper bound on the complexity is $\mathcal{O}(LMN + L^2 nnz(C))$, where $nnz(C)$ is the number of non-zeros in C. As we show in our experiments for many datasets $nnz(C) \ll LN$ can be achieved. Since each column of C is computed independently, this algorithm is highly parallel. Let N_P be the number of parallel processing cores. By replicating D and a fraction of columns of A in each node (i.e., $\frac{N}{N_P}$ columns), the complexity of Step 3 would reduce to $\mathcal{O}(\frac{N}{N_P}(LM + L^2 \frac{nnz(C)}{N}))$.

E. Evolving Data

In some applications, the dataset A may dynamically evolve over time. Let us assume $A = DC$. Whenever a new set of columns are added to A, we update the coefficient matrix C by using OMP (Step 3 of ExD) to solve $A_{new} = DC_{new}$; where $A = [A, A_{new}]$ and $C = [C, C_{new}]$ represent the updated dataset and coefficient matrix. In some cases the newly added columns of A may not be expressed well by the space spanned by the current dictionary D. In other words, the OMP algorithm might not be able to find C_{new} such that the transformation error criteria is met. For example, a new set of drastically different images can expand the space of the original dataset. In such cases, D should also be modified to include those new structures. To do so, we apply ExD on A_{new} to find both D_{new} and C_{new}. Then we update the transformation of the entire dataset using zero-padding shown in Figure 3. The proposed approach enables us to update the transformation while avoiding the cost of re-applying ExD on the entire dataset.

VI. Distributed Computing Model and Performance Quantification

In this section, we first propose an efficient data partitioning and distributed computing model to perform iterative analysis on the transformed data, i.e., $(DC)^T DCx \simeq A^T Ax$. We next quantify the performance metrics based on the proposed distributed model.

Algorithm 2 : **Distributing Gram Matrix Multiplication on** $DC \simeq A$

Input: Vector $x_{N\times 1}$ and transformed data matrices $D_{M\times L}$ and $C_{L\times N}$.

Output: $C^T D^T DCx$.

0.0 $pid = i$ loads $C_i = C(:, \frac{iN}{N_P} : \frac{(i+1)N}{N_P})$.
0.1 $pid = i$ loads $x_i = x(\frac{iN}{N_P} : \frac{(i+1)N}{N_P})$.
1. $pid = i$ computes $v_i^1 = C_i x_i$; v_i^1 is an $L \times 1$ vector.

Case 0: L > M
2. $pid = i$. loads D.
3. $pid = i$ computes $v_i^2 = Dv_i^1$; v_i^2 is an $M \times 1$ vector.
4. Vectors v_i^2 will be reduced in $pid = 0$.
5. $pid = 0$ computes $v^2 = \sum v_i^2$; $v^2 = DCx$ is an $M \times 1$ vector.
6. $pid = 0$ broadcasts v^2 to all other processors.
7. $pid = i$ computes $C_i^T(D^T v^2)$.

Case 1: L ≤ M
2. $pid = 0$ loads D.
3. Vectors v_i^1 will be reduced in $pid = 0$.
4. $pid = 0$ computes $v^2 = D(\sum v_i^1)$; $v^2 = DCx$ is an $M \times 1$ vector.
5. $pid = 0$ computes $v^3 = D^T v^2$; $v^3 = D^T DCx$ is an $L \times 1$ vector.
6. $pid = 0$ broadcasts v^3 to all other processors.
7. $pid = i$ computes $C_i^T v^3$.

A. Data Partitioning and Distributed Mapping

Algorithm 2 outlines ExtDict's computing model. Depending on whether $L > M$ (Case 0) or $L < M$ (Case 1), we propose two different approaches. In Case 0, we replicate matrix D in all the processors to reduce communication. However, doing so requires all the processors to do the redundant multiplication, i.e, $D^T v^2$ in Step 7. In Case 1 however, the computation corresponding to $D^T v^2$ is done only by processor 0. As discussed in Section V, most contemporary datasets are in the regime that $M,\ L \ll N$. Thus, D is a relatively small matrix that can easily fit into the memory of processor 0. Execution phase in Figure 1 shows Case 1 where $L < M$ and D is stored only in processor 0.

B. Performance Quantification

In the following, we first provide the arithmetic and communication bounds. We then use these bounds to quantify the performance in terms of runtime, energy, and memory.

Bounds on Arithmetics. The cost of arithmetics is directly dependent on the number of floating point operations for doing $(DC)^T DCx$. The computations involving C should be done in an efficient way to exploit its sparsity. This reduces the computations in Steps 1 and 7 (for both cases) to the number of non-zeros in C or $nnz(C)$. The number of floating point operations (in serial) is $2(ML + \frac{nnz(C)}{N_P})$ multiplications and $2MN_P$ additions. Here, the cost of additions is negligible because in many cases we have $N_P \ll L$.

Bounds on Communication. The communication overhead of Algorithm 2 stems from the $reduce$ and $broadcast$ activities. In Case 0, Step 4, each processor sends a message containing M words to Processor 0, and in Step 6, Processor 0 sends a message containing M words back to other processors. In a similar fashion in Case 1, at Steps 3 and 5, L words are communicated. The total number of words that are communicated simultaneously is $2 \times min(L, M)$.

We exploit the extensive work in applied numerical linear algebra to show that our computational model achieves the optimal communication. More exactly, Demmel et al.'s work on communication-optimal parallel recursive rectangular matrix multiplication directly applies to our target problem [33]. In that work, it is shown that for multiplying $Z = XY$ where dimensions of matrices X, Y, and Z belong to $\{d_1, d_2, d_3\}$ such that ($d_1 \leq d_2 \leq d_3$), if $2\frac{d_3}{d_2} > N_P$ (which is the case in our framework when $d_3 = N$), the communication lower bound (number of communicated words) is $\Omega(d_1 d_2)$. Substituting the dimensions by those of matrices D, C, and x we get $d_1 = 1$ and $d_2 = min(M, L)$ which brings the number of transferred words to $min(M, L)$. Thus, our communication achieves the optimal (minimum) bound.

1) Runtime Performance: The overall execution runtime is directly affected by the arithmetic and communication costs and is approximately proportional to:

$$ML + \frac{nnz(C)}{N_P} + \min(M, L) N_P R_{b2f}^{time}, \tag{2}$$

where R_{b2f}^{time} is the word-per-FLOPs ratio that characterizes the memory bandwidth per unit of time performance.

The first two terms reflect the computational operations and the third term reflects the adjusted communication overhead. During the message passing phases in Algorithm 2, all processors are locked and no computation is done. Although a number of other factors affect the runtime such as memory hierarchy, cache size, size of shared memory (for the cores within a computing node), and geometry of the distributed nodes, in our experiments, we quantitatively show that our approximation provides a reasonable estimation of the actual runtime.

2) Energy Performance: Similarly, the overall execution energy is approximately proportional to:

$$ML + nnz(C) + \min(M, L) N_P R_{b2f}^{energy}, \tag{3}$$

where R_{b2f}^{energy} is the word-per-FLOPs ratio that characterizes the memory bandwidth per unit of energy performance. The first two terms reflect the computational operations and the third term reflects the adjusted communication overhead.

3) Memory Performance: The sparsifying effect of ExD transformation results in significant reductions in memory footprint. Memory usage decreases due to replacing the original dense matrix A with the relatively lower dimensional dictionary matrix D and the sparse coefficient matrix C. In both proposed implementations (Algorithm 2) the memory footprint per processing node is bounded by:

$$ML + \frac{nnz(C) + N}{N_P}. \tag{4}$$

VII. Automated Customization of ExD

In this section, we present ExtDict's approach for optimizing the performance of the iterative update algorithms. More exactly, we discuss how the extensible dictionaries proposed in Section V can be tuned to minimize the performance costs quantified in Section VI.

We tune ExD by finding an optimized L as an input for Algorithm 1. The input must be such that the resulting $(L, nnz(C))$ pair minimizes the performance costs quantified in Equations 2, 3, or 4. In the following, we propose a novel scalable method to tune ExD. Our method estimates $nnz(C)$ as a function of L with preprocessing only portions of the original, massive data matrix A.

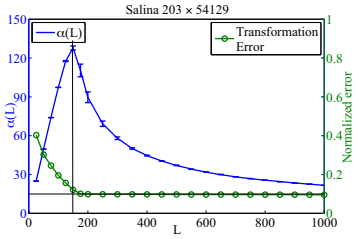


Fig. 4: Density function (or the number of non-zeros in each column of C) $\alpha(L)$ as a function of the columns sampled L. The bars show the variance for 100 trials of random sub-sampling to form D. The average transformation error $\|A - DC\|_F / \|A\|_F$ as a function of L.

Estimating $\mathbf{nnz(C)}$. Here we provide an example dataset to show the relationship between L and $nnz(C)$. Figure 4 shows the normalized transformation error ($\|A - DC\|_F / \|A\|_F$) as a function of L. The figure also plots the average per-column number of non-zeros in C as a function of L. We denote this function as:

$$\alpha(L, A, \epsilon) = \frac{nnz(C)}{N}. \quad (5)$$

The dataset is a collection of Hyperspectral signals from [34] with $M = 203$, $N = 54129$. The search space for parameter L is limited to $L \geq L_{min}$, where L_{min} is the minimum number of columns that should be sampled such that the transformation error criteria is met ($\|A - DC\|_F < \epsilon \|A\|_F$). In this example $L_{min} \approx 175$.

It can be seen that function $\alpha(L, A, \epsilon)$ is decreasing for $L > L_{min}$. This is due to the fact that a larger L would result in a greater ensemble of signals in D. This enables representing columns of A as a linear combination of fewer signals from D. In an extreme case when $L = N$ (or $D = A$), then the i^{th} column of A (a_i) can be transformed as $a_i = De_i$, where e_i is a unit vector whose i^{th} entry is 1 and all of its other entries are 0. In this case $\alpha(N, A, \epsilon) = 1$, however, the cost in Equation 2 becomes too large due to the large L. When $L < L_{min}$, the OMP method cannot reconstruct some of the a_i columns even if all the columns in D are selected in the reference set ϕ (see Algorithm 1).

The bars on the graph shows the variation in $\alpha(L, A, \epsilon)$ for 100 different initial ensemble collection for D. It can be seen that the variations are very negligible for a fixed L (dispersion index is less than 4% for this example).

Estimating $\boldsymbol{\alpha(L, A, \epsilon)}$ from Smaller Subsets of A. In order to optimize the performance cost (e.g., Equation 2), function $\alpha(L, A, \epsilon)$ needs to be characterized. While a Brute Force approach (i.e., running Algorithm 1 for various L values) would provide us with the best L, it is not an efficient nor a feasible solution for massive N. To provide an effective way to find $\alpha(L, A, \epsilon)$, we rely on the following two important observations.

First, let A be a data with a union of subspace signal model. Let A_s be a random subset of A such that $|A_s|^4 = n$, then for $n \to N$: $E[\alpha(L, A_s, \epsilon)] = E[\alpha(L, A, \epsilon)]$. Second, recalling our formal definition of union-of-subspace data (which is a generalization of low-rank datasets) in Section IV, we assume columns of A are a collection of signals from N_s subspaces where each subspace U_i is K_i-dimensional (for $1 \leq i \leq N_s$). In this case if n_i columns of A lie on subspace U_i, then the coefficient matrix corresponding to those n_i columns have at most $K_i n_i$ non-zeros [16], [12]. Based on our definition of the density measure, we have $\alpha(L, A, \epsilon) \leq \sum_{1 \leq i \leq N_s} K_i \frac{n_i}{N}$. Let us create A_s by selecting n columns at random from A. The expected number of columns in A_s that belong to subspace U_i is $\frac{n_i}{N} n$. If we apply Algorithm 1 to A_s, we get $A_s = D_s C_s$. Thus, the following expected upper bound is achieved for $\alpha(L, A_s, \epsilon) \leq \sum K_i \frac{n_i}{N}$ which is the same as the bound on $\alpha(L, A, \epsilon)$.

The above observations indicate that finding $\alpha(L, A_s, \epsilon)$ from a subset of A yields a close estimation of $\alpha(L, A, \epsilon)$. One can run ExD for random subsets of A denoted by A_i, such that $|A_1| < |A_2| < \cdots < |A|$ until the discrepancy in $\alpha(L, A_i, \epsilon)$ reduces to a prespecified threshold. In Section VIII-B2 we experimentally verify the above observations.

VIII. Evaluation

Implementation and API. We implement our proposed tunable ExD transformation (Section V) and the distributed iterative model on the transformed data (Sections VI and VII) in C++ using the standard message passing system (MPI). We use Eigen library for linear algebra computation. Our API takes the following user inputs: dataset A, transformation error ϵ, and the learning algorithm as an iterative update function on Gram matrix. We experimentally measure the platform-specific relative cost of arithmetic vs. communication (R^{time}_{b2f}).

All our evaluations are done on IBM iDataPlex (node type: Intel-Xeon-X5660@2.80GHz) cluster. Within the IBM iDataPlex platform, we studied several configuration of nodes and cores within each node to emulate various platforms. Note that all previous work in data transformation literature are oblivious to platform and always return the same outputs for the same approximation error.

Datasets. Our evaluations are done on the datasets shown in Table I.

TABLE I: Datasets used for various applications.

Denoising Gradient-descent	Super-Resolution Gradient-descent	PCA Power method		
Light Field [35]	Light Field [35]	Salina [34]	Cancer Cell [5]	Light Field [35]
1600 × 146000 1.86GB	576 × 146000 672.7MB	203 × 54129 87.9MB	1024 × 111296 911.7MB	18496 × 272320 40.3GB

A. Applications and Baselines

Applications and Algorithms. We evaluate 3 applications: image denoising, image super-resolution, and PCA. In the following, we describe each application and our baselines for comparison.

In the first two applications, we use gradient-descent to solve a LASSO objective. The LASSO objective function is written as follows: $\|Ax - y\|_2 + \lambda |x|_1$, where λ is a learning parameter. As discussed in Section II-A, LASSO is a widely popular approach with numerous applications including feature selection [11], pattern recognition [8], and classification [12].

For image denoising, y is a noisy image, A is a dataset of denoised Light Field pixels. The reconstructed signal, i.e., Ax would become the denoised image. For image super-resolution, y is a lower-resolution image that is reconstructed via LASSO objective using a scaled down version of A, where A is a dataset of high-resolution pixels. To be more precise, we consider a scenario where A_{lf} is a dataset created from 8×8 patches of a 5×5 light-field camera setting. Thus, A_{lf} has 1600

[4] $|\cdot|$ is the cardinality or number of columns.

[5] This dataset consists of cancer tumor morphologies collected in MD-Anderson cancer center.

rows. y is an image derived from a 3×3 subset of lightfield cameras and has 576 rows. We create a subset of A_{lf}, denoted by A, corresponding to the 3×3 camera subset. Thus, A also has 576 rows. We then solve LASSO using A and y to find x. Once x is found, $A_{lf}x$ would yield a higher-resolution version of y with 1600 rows. To solve the LASSO problem, we use iterative gradient descent approach. Each iteration performs an update of type $Gx^t - A^T y$. We use the Adagrad method for updating the gradient [36].

Our last application is the Power method, a widely used algorithm for finding the principle components of large-scale datasets (PCA analysis). Finding PCA of a large-scale data is a highly expensive computational task. Once the principal components (singular and eigen-values) are found, they can be readily used to solve a wide range of learning problems including image classification [37], [14], feature extraction [13], and face recognition [38].

Power method iteratively performs matrix-vector multiplication on the Gram matrix, i.e., $x^{t+1} = \frac{Gx^t}{\|Gx^t\|_2}$, until convergence (to the largest eigenvalue) is achieved. Vector x^t and the norm $\|Gx^t\|_2$ denote the estimated eigenvector and eigenvalue at the t^{th} iteration respectively. After convergence, the content associated with the found eigenvalue is subtracted from the data. Power method then re-iterates itself to find the next largest eigenvalue.

In our experiments, we provide extensive runtime, memory, and error analysis. We do not provide energy analysis. However, the energy usage is dominantly governed by the number of floating point operations and communication (Equation 3). Thus, the runtime and memory analysis directly translate to energy as well.

Baselines. We consider two types of comparisons. The first type compares our preprocessing, i.e., ExD transformation, with other state-of-the-art existing scalable transformations including Random Column Subset Selection (RCSS) [32], Adaptive column sampling for kernel matrix approximation method (oASIS) [20], and the sparsifying columns subset selection-based methos (RankMap) [28], [39]. Each of these transformations can substitute ExD within our proposed framework. In Section VIII-B3, we compare the memory and runtime performance of the transformed data, using each of the above methods against ExD and demonstrate the advantages of using our platform-aware approach.

The second type, compares our approach with other practices for solving learning objectives for large datasets. More precisely, for the denoising and image super-resolution applications, we compare ExtDict's implementation of gradient-descent based approach, with Stochastic Gradient Descent (SGD). SGD is a popular but approximate method that circumvents operations on large kernel matrices by using only a subset (or a batch) of data in each iteration. The subset, denoted by A_b, is randomly constructed from a rows of A. The update is done using $A_b^T A_b x$ instead of $A^T Ax$. Different subsets of rows are selected at each iteration. We implemented a distributed SGD method using Adagrad to update the gradients. The drawback of SGD is in its sub-optimality, non-guaranteed, and slow convergence, since only portions of data is used to update the solution [29]. ExtDict, however, runs the provably converging gradient-decent algorithm on the entire (but transformed) dataset.

For PCA application, we compare Power method on original data A as opposed to using the transformed data DC.

B. Evaluating ExtDict's Preprocessing

Our framework is developed based on the proposed tunable ExD transformation for preprocessing data. In the following, we investigate several aspects of ExD including its tunability, sparsifying effect, and overhead. We also compare its performance with other transformation methods.

1) Tunability of ExD: We verify the ability of ExD to create versatile sparse transformations. Figure 5 shows the effect of varying the input parameters of ExD, namely dictionary size L, and transformation error ϵ, to achieve different sparsity levels in the coefficient matrix C. The y axis demonstrates the average number of non-zeros per column of C denoted by $\alpha(L)$ [6]. On each figure, the name and size of datasets are shown. As it can be seen, $\alpha(L)$ can be significantly lower than the number of non-zeros in the original data. For example, for Light Field data, when $L = 1000$ the average number of non-zeros per column is reduced from 18496 in A to approximately 800, 600, and 200 for ($\epsilon = 0.01$, 0.05, and 0.1 respectively) in C.

The following two novel and critical properties of ExD are evident: (i) by increasing the redundancy in the dictionary (large L), we can achieve sparse coefficient matrices. (ii) by increasing the transformation error-tolerance, we can achieve sparser solutions. ExtDict takes advantage of these tunable characteristics to tune the transformation w.r.t. the platform requirement. Recall that the tradeoff is that increasing L would yield a higher communication (Equation 2), and increasing error might yield to unwanted error in learning applications that use the transformed data.

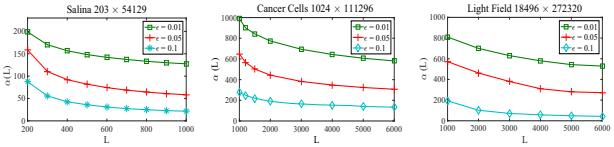
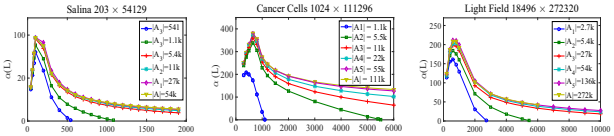


Fig. 5: Tunablity of ExD transformation. The average number of non-zeros per column of C (i.e., $\alpha(L)$) versus L for different transformation errors ϵ are shown. Both increasing the redundancy in dictionary and increasing error tolerance yield sparser transformations.

2) Low-overhead ExD Tuning and Executing: We experimentally verify our claim in Section VII, that we can use subsets of data to tune parameter L. Figure 6, demonstrates $\alpha(L)$ as a result of applying ExD on the subsets of the data, i.e., $A_1 \subset A_2 \subset A_3 \subset A_4 \subset A_5 \subset A$. The transformation error ϵ is set to 0.1 (10%). It is evident that a reasonable estimation of $\alpha(L)$ can be achieved by running ExD on subsets of data, thus, reducing the overhead of tuning ExD during the preprocessing phase. For example for $L = 1000$ using only 10% of data is sufficient to estimate $\alpha(L)$ within less that 14% error for all datasets. Once, $\alpha(L)$ is estimated for various Ls, target performance model (Equations 2, 3, or 4) can be used to find the optimal L that reduces the costs.

Table II shows the total preprocessing time overhead, which accounts for both tuning and running ExD for the optimal L. The computations are done on 64 cores (8 nodes each with 8 cores). As we will see later, the overhead of ExD, which is a one-time preprocessing step, is amortized when iterative algorithms are run on the transformed data. Note that even though Light Field has a larger dimension than Cancer Cells

[6] We abbreviate $\alpha(L, A, \epsilon)$ with $\alpha(L)$.



(a) (b) (c)

Fig. 6: Effective ExD tuning based on subsets of A. As the sizes of the subsets increase, $\alpha(L)$ converges to that of the full data A.

TABLE II: Preprocessing overhead in (ms) which includes ExD tuning and execution time. The computations are done on 64 cores (8 nodes each with 8 cores).

	Salina 203×54129	Cancer Cells 1024×111296	Light Field 18496×272320
Tuning	1687	120042	175780
Transformation	931	564092	164522
Overall	2618	684134	340302

data, the latter incurs a higher preprocessing overhead. This is due to the data-dependent nature of ExD. As can be seen in Figure 5, Cancer Cells have a denser geometry that requires more iterations of the OMP algorithm (Step 3 in Algorithm 1) to achieve a given ϵ.

3) Comparison with Other Transformations: We show the effectiveness of our customized preprocessing approach by substituting ExD with other existing scalable data transformation methods within our framework. More precisely, we report ExtDict's runtime and memory performance when each of the projection methods are used to preprocess the data.

Runtime Analysis. Figure 7 demonstrates the runtime improvement achieved by using ExtDict for Gram matrix updates, over other approaches including the original A^TA, and the state-of-the-art scalable transformations RCSS, oASIS, and RankMap as discussed in Section VIII-A. All the transformations (including ExD) are done for the same transformation error of $\epsilon = 0.1$. We compare the runtime of an iterative update on the Gram matrix, i.e., A^TAx, while using the transformed data ($(DC)^TDCx$) instead. $A_{M\times N}$ is the dataset and $x_{N\times 1}$ is a random vector. We measure the runtime for four platforms: 1×1, 1×4, 2×8, and 8×8 configurations, where the first number indicates the nodes and the second indicates the cores per node. We tune ExD to optimize for runtime (Equation 2) on each platform.

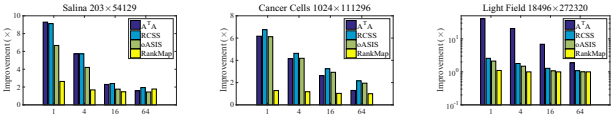
In all cases, ExD yields better or equal runtime (for an iterative update on the Gram matrix) compared with other schemes. We observe up to 40.78× (runtime) improvement over A^TA, 9.12× improvement over RCSS, 6.67× over oASIS, and 2.63× improvement over RankMap. For Light Field we achieve comparable runtime with RankMap (ExtDict achieves 10% runtime improvement for $N_P = 1$ and equal runtime for other N_Ps). This is because, as shown in Figure 8, for this data the optimally tuned dictionary size is very close to the smallest possible value that meets the transformation error. Thus, similar to RankMap, in this case ExtDict chooses the smallest transformation basis.

Note that we do not provide comparison of the runtime performance of other transformations for the three learning applications as we already demonstrated ExD's superiority over them in Figure 7; Since all the transformation methods are applied for the same error tolerance ϵ, the number of iterations required for convergence of learning algorithms (e.g., LASSO or Power method) are similar. Therefore, the relative performance of different transformations remain the same as what is shown in Figure 7.

TABLE III: Comparison of different transformations in terms of memory. The transformation error is set to 0.1 (10%). ExD is the only transformation that can be customized to platform. All values are in MB.

	Original data	RCSS	oASIS	RankMap	ExtDict $N_p = 1$	ExtDict $N_p = 4$	ExtDict $N_p = 16$	ExtDict $N_p = 64$
Salina	87.9	86.9	65.1	38.2	10.11	10.11	10.11	19.1
Cancer Cells	911.7	898.5	808.7	254.6	172.6	172.6	206.7	254.6
Light Field	40294.6	2326.5	1977.5	567.5	517.9	567.5	567.5	567.5

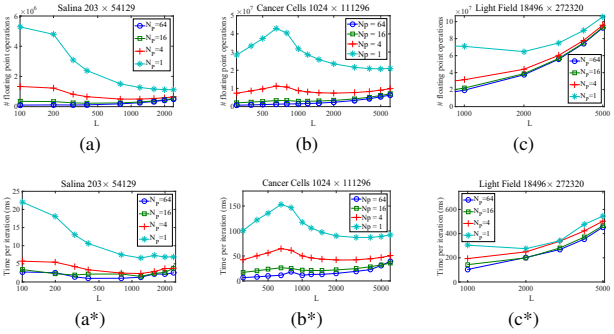
Memory Analysis. Table III compares the memory performance of different approaches. The values show the memory consumption for storing matrices C and D, achieved from different transformations. The memory usage of the original data (matrix A) is also provided. Other than ExtDict, other approaches always result in the same memory footprint regardless of the platform. It can be seen that ExtDict results in up to 77.8× (memory usage) improvement over A^TA, 8.6× improvement over RCSS, 6.4× improvement over oASIS, and 3.8× improvement over RankMap. ExtDict achieves its memory reduction by choosing over-complete dictionaries that can produce sparse coefficient matrices. Other approaches do not offer such flexibility.



(a) (b) (c)

Fig. 7: Runtime improvement achieved by ExtDict on different platforms. The runtime corresponds to execution of an iterative update on the transformed Gram matrix.

C. Evaluating ExtDict's Proposed Performance Model



(a) (b) (c)

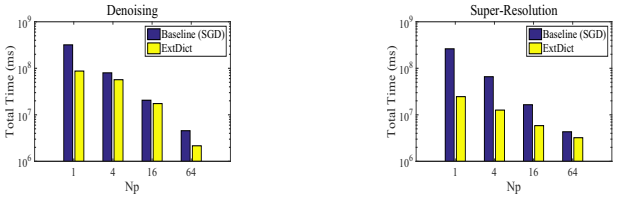
(a*) (b*) (c*)

Fig. 8: Verification of performance model: top row is the estimated and bottom row is the actual performance of running $(DC)^TDCx$ on various platforms. Our predicted runtime (in terms of number of floating point operations) closely follows the trend (in terms of milliseconds) in the actual evaluations.

We verify the accuracy of our performance model for predicting the runtime trend (as quantified in Equation 2) by comparing it against the measured runtime on the platform. Figure 8 shows the results for running one iteration of Gram matrix update of the form $(DC)^TDCx$. The measured runtimes are averaged over 100 iterations. The tests are done

on various platforms. Recall that the number of distributed processing cores are denoted by N_P's (1×1, 1×4, 2×8, 8×8 configurations). As can be observed, the measured performance (bottom) is very similar to our estimated performance (top), corroborating that one can use our performance model to tune ExD without having to measure the actual runtimes.

D. Evaluating Learning Applications

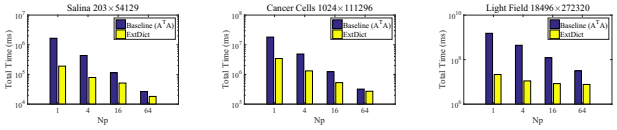


(a) (b)

Fig. 9: Runtime comparison of image denoising and super-resolution applications vs. SGD on various platforms.

1) Runtime Analysis: Figures 9a and 9b compare the overall runtime performance of ExtDict for image denoising and image super-resolution applications. Our gradient-descent approach not only benefits from a guaranteed convergence (as opposed to SGD), it also yields faster convergence for both applications. Another advantage of our approach over SGD is in reducing the memory usage for storing the data. Unlike our approach, SGD does not affect the memory usage. In both applications the ϵ in ExD is set to 0.1 (10%). It can be seen that ExtDict achieves up to $3.7\times$ (runtime) improvement for denoising application and up to $10.9\times$ improvement for super-resolution application over SGD.

For the same dataset (fixed N), as it is suggested by Equation 2, by increasing the number of processors the cost of communication becomes dominant over the cost of FLOPs. SGD's communication is limited to the batch-size, which in our experiments is set to 64. Thus, SGD'd communication in each iteration is lower than ExtDict's communication, i.e., $min(M, L)$. However, SGD requires many more iterations to converge compared with the gradient-descent approach. Reversely, for a fixed platform setup, as the size of data increases (increasing N), the cost of FLOPs become dominant over the cost of communication.



(a) (b) (c)

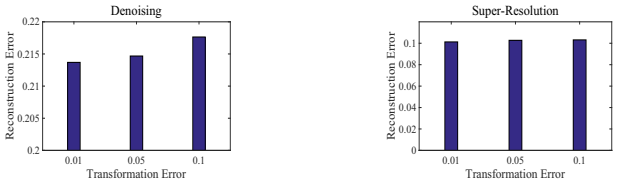
Fig. 10: Runtime comparison of PCA application (while running Power method) using $(DC)^T DCx$ vs. using original data $A^T Ax$ on various platforms.

Figure 10 compares the runtime performance of Power method for finding the first 100 eigenvalues of different datasets. The baseline is the case where the Gram matrix is computed using $A^T A$. The ϵ in ExD is set to 0.1 (10%). The results show significant runtime improvement when ExD is applied. It can be seen that ExtDict achieves up to $8.68\times$, $5.29\times$, and $71.2\times$ (runtime) improvement for Salina, Cancer Cells, and Light Field datasets over baseline respectively.

2) Error Analysis: We evaluate the trade-off between the preprocessing error tolerance versus the final learning error. Figures 11a and 11b provide the reconstruction error of ExtDict for image denoising and super-resolution applications. The reported error is calculated as $\frac{\|y-\widehat{y}\|_2{}^2}{\|y\|_2}$, where $\widehat{y}$ is the approximated solution achieved by solving LASSO and y is the denoised or high-resolution benchmark.

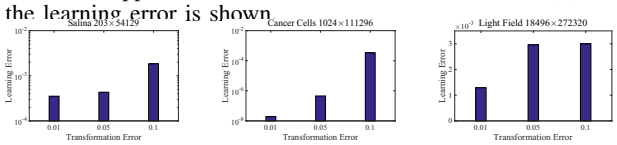
A more intuitive way to evaluate the quality of the denoised or higher-resolution reconstructed images is by using the Peak Signal to Noise Ratio (PSNR) metric. PSNR is the ratio between the maximum power of a signal and the power of the corrupting noise. PSNR is defined as $10\log_{10}(\frac{MAX}{\sqrt[2]{MSE}})$ (dB), where MAX is the maximum pixel value of the original image patch and MSE is the mean square reconstruction error. Typical recommended PSNR values in vision applications are 25 dB and higher [40]. For denoising application, our output PSNR is 29.39 dB when input SNR of the noisy image is 15.15 dB. For super-resolution application, our output PSNR is 24.69 dB.

A notable observation is that although higher transformation errors can result in meaningful runtime and memory improvements (Sections VIII-B3), they may not drastically affect the reconstruction error.



(a) (b)

Fig. 11: Error comparison of image denoising and super-resolution applications. Effect of transformation error (ϵ) on the learning error is shown.



(a) (b) (c)

Fig. 12: Error comparison of PCA application. Effect of transformation error (ϵ) on the learning error is shown. The learning error is computed by comparing the reconstructed eigen-values computed by using ExD against the actual eigen-values computed by using A.

Figure 12 demonstrates the learning error in finding singular-values of different datasets. The learning error is the normalized cumulative error of the first 100 eigen-values found by running Power method using ExtDict. The baseline singular values are derived from running Power method on A. It can be seen that ExtDict results in negligible learning error while drastically improving the runtime.

E. Discussions

Our evaluations demonstrate the significance of ExD's preprocessing step to reduce the cost of immensely expensive learning algorithms. As our results show, even moderate size datasets can be (computationally) extremely expensive. For example, on a 64-core platform, Power method on a Light Field dataset with size 18496×272320 takes more than 8 hours and 42 minutes to converge (Figure 10). ExtDict reduces this time to less than 2 hours and 8 minutes on the same platform (Figure 10). Our one-time preprocessing overhead for this dataset on the same platform is less than 6 minutes

(Table II). This drastic improvement also translates to dollar-cost reduction on clouds.

Note that several learning algorithms (e.g., LASSO or other regression algorithms) require model selection. This means the learning objective should be tuned for varying parameters (e.g., regularization variables such as λ in LASSO objective) until the best ones are identified. Model selection requires running the learning algorithms several times, further amortizing the one-time preprocessing overhead.

IX. Conclusion

In this paper we present ExtDict, the first data- and platform-aware solution for highly efficient execution of iterative learning algorithms on massive datasets. We introduce the novel idea of Extensible Dictionary (ExD) which leverages coarse-grained parallelism in the data to create tunable sparse transformations. The transformation is low-overhead and highly scalable. Our framework reduces the performance cost by adaptively tuning the transformation with respect to the properties of the underlying computing platform. We provide a distributed API that enables applying ExtDict to a wide rang of learning problems in large scale. Our extensive evaluations demonstrate the importance of platform-aware tuning. ExtDict can achieve more than an order of magnitude improvement in execution runtime, energy, and memory footprint compared to existing work.

Acknowledgment

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

References

[1] A. Gilbert, M. Strauss, J. Tropp, and R. Vershynin, "One sketch for all: Fast algorithms for compressed sensing," 2007.

[2] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *JCGS*, pp. 265–286, 2006.

[3] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit," *CS Technion'08*.

[4] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication-optimal parallel recursive rectangular matrix multiplication," in *IPDPS'13*.

[5] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, "Minimizing communication in sparse matrix solvers," in *SC'09*.

[6] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," *SIGMOD*, 2010.

[7] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "GraphLab: A new parallel framework for machine learning," *UAI*, 2010.

[8] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Royal Statist. Soc B*, vol. 58, no. 1, pp. 267–288, 1996.

[9] M. Figueiredo, R. Nowak, and S. Wright, "Gradient projections for sparse reconstruction: Application to compressed sensing and other inverse problems," *IEEE J. Select. Top. Signal Processing*, vol. 1, no. 4, pp. 586–597, 2007.

[10] M. C. Ferris and T. S. Munson, "Interior-point methods for massive support vector machines," *SIAM J. on Optimization*, pp. 783–804, 2002.

[11] I. Ramirez, P. Sprechmann, and G. Sapiro, "Classification and clustering via dictionary learning with structured incoherence and shared features," *CVPR'10*.

[12] E. Elhamifar and R. Vidal, "Sparse subspace clustering: algorithm, theory, and applications," *TPAMI'13*.

[13] M. Journee, Y. Nesterov, P. Richtárik, and R. Sepulchre, "Generalized power method for sparse pca," *JMLR'10*.

[14] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nystrom method," *TPAMI'04*.

[15] J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," *OSDI*, 2012.

[16] E. Dyer, A. Sankaranarayanan, and R. Baraniuk, "Greedy feature selection for subspace clustering," *JMLR'13*.

[17] P. Drineas and M. Mahoney, "On the nyström method for approximating a gram matrix for improved kernel-based learning," *JMLR*, pp. 2153–2175, 2005.

[18] A. Gittens and M. Mahoney, "Revisiting the nystrom method for improved large-scale machine learning," *ICML'13*.

[19] A. Farahat, A. Elgohary, A. Ghodsi, and M. Kamel, "Greedy column subset selection for large-scale data sets," *Knowledge and Information Systems*, pp. 1–34, 2014.

[20] R. Patel, T. Goldstein, E. Dyer, A. Mirhoseini, and R. Baraniuk, "oasis: Adaptive column sampling for kernel matrix approximation," *arXiv preprint:1505.05208*, 2015.

[21] S. Fine and K. Scheinberg, "Efficient svm training using low-rank kernel representations," *JMLR'02*.

[22] V. Pham, L. Ghaoui, and F. A., "Lsrn: A parallel iterative solver for strongly over or under-determined systems," *SIAM Journal of Scientific Computing*, 2014.

[23] A. F. Vu Pham, Laurent El Ghaoui, "Robust sketching for multiple square-root lasso problems," *Optimization and Control*, 2014.

[24] A. Mirhoseini, B. D. Rouhani, E. M. Songhori, and F. Koushanfar, "Perform-ml: Performance optimized machine learning by platform and content aware customization," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 20.

[25] B. D. Rouhani, A. Mirhoseini, E. M. Songhori, and F. Koushanfar, "Automated real-time analysis of streaming big and dense data on reconfigurable platforms," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 1, p. 8, 2016.

[26] B. D. Rouhani, E. M. Songhori, A. Mirhoseini, and F. Koushanfar, "Ssketch: An automated framework for streaming sketch-based analysis of big data on fpga," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 187–194.

[27] E. M. Songhori, A. Mirhoseini, X. Lu, and F. Koushanfar, "Ahead: automated framework for hardware accelerated iterative data analysis," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 942–947.

[28] A. Mirhoseini, E. Dyer, E. Songhori, R. Baraniuk, and F. Koushanfar, "Rankmap: A platform-aware framework for distributed learning from dense datasets," *arXiv preprint:1503.08169*, 2015.

[29] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," ser. ICML '04.

[30] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," *USENIX CHTCC'10*.

[31] Y. Pati, R. Rezaifar, and P. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," *Proc. Asilomar Conf. Signals, Systems, and Computers*, Nov. 1993.

[32] A. Gittens and M. Mahoney, "Revisiting the nystrom method for improved large-scale machine learning," *JMLR'13*.

[33] J. Demmel, D. Eliahu, A. Fox, A. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication optimal parallel multiplication of sparse random matrices," in *SPAA '13*.

[34] "Aviris salinas valley and rosis pavia university hyperspectral datasets," http://www.ehu.es/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

[35] The stanford light field archive. [Online]. Available: http://lightfield.stanford.edu/

[36] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," ser. JMLR '11.

[37] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[38] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," *tpami*, 1997.

[39] A. Mirhoseini, E. Songhori, B. Rouhani, and F. Koushanfar, "Flexible transformations for learning big data," *SIGMETRICS*, 2015.

[40] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *TSP*, pp. 4311–4322, 2006.