Perform-ML: Performance Optimized Machine Learning by Platform and Content Aware Customization

Azalia Mirhoseini[1], Bita Darvish Rouhani[2], Ebrahim M. Songhori[1], Farinaz Koushanfar[2]
azalia@rice.edu, bita@uscd.edu, ebrahim@rice.edu, farinaz@ucsd.edu
[1]Rice University, [2]UC San Diego

ABSTRACT

We propose Perform-ML, the first Machine Learning (ML) framework for analysis of massive and dense data which customizes the algorithm to the underlying platform for the purpose of achieving optimized resource efficiency. Perform-ML creates a performance model quantifying the computational cost of iterative analysis algorithms on a pertinent platform in terms of FLOPs, communication, and memory, which characterize runtime, storage, and energy. The core of Perform-ML is a novel parametric data projection algorithm, called Elastic Dictionary (ExD), that enables versatile and sparse representations of the data which can help in minimizing performance cost. We show that Perform-ML can achieve the optimal performance objective, according to our cost model, by platform aware tuning of the ExD parameters. An accompanying API ensures automated applicability of Perform-ML 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 improvements in performance compared to the state of the art, within guaranteed user-defined error bounds.

1. INTRODUCTION

Efficient resource utilization plays a key role in achieving a sustainable and practical computing ecosystem. Learning and analysis of massive data is a trend that is ubiquitous among the various modern computing platforms in this ecosystem, ranging from embedded sensors and Internet-of-things devices, to smart phones, and cloud servers. The most challenging ML scenarios are the ones that involve iterative optimization on dense (non-sparse) datasets. Examples of such scenarios include decent-based algorithms ubiquitously used for solving regularized regression, deep learning, or support vector machines that are widely used in various learning applications [17, 6]. Image and video datasets which encompass the majority of the generated content in modern digital world are prominent examples of dense data.

Two broad disjoint categories of prior works have addressed efficient ML in such challenging scenarios: (i) Content aware methods in statistics and ML communities which reduce the computational load by limiting the domain to a lower-dimensional representation of data [9, 16], or, (ii) Platform-aware methods in computer engineering that map a given algorithm to a platform, but do not change anything at the data projection level [1, 21]. Note that ML acceleration methods such as Pregel [12] or GraphLab [11] that are built upon the existing sparsity of the correlation matrices become very inefficient for dense data.

In this paper, we introduce Perform-ML, the first ML performance optimization framework which is *jointly content and platform aware*. Perform-ML's efficiency is driven by our new observation that data projection has a major impact on the way mapping can be done onto modern many-core and distributed computing machines. We recognize that the degree of freedom in transforming the data to lower dimensions (for the same approximation error) can be utilized for performing platform-optimized mapping of computations. The Perform-ML dimensionality reduction is built upon a new dictionary learning, called ExD with tunable parameters that can generate different projection subspaces, or elastic dictionaries, for a given approximation error. The dictionary formation and platform aware tuning of the dictionary parameters are done in a pre-processing phase by data subsampling. This information is then used for achieving the best computing cost for processing the entire massive data. The pre-processing overhead is amortized over several iterations of the resource efficient ML algorithm on the transformed data.

The explicit contributions of this paper are: (i) Creation of the Perform-ML end-to-end framework that supports a wide range of ML applications which rely on iterative optimization over big and dense data to converge. (ii) Introduction of a quantitative performance cost for Perform-ML that is utilized to provably optimize the performance (e.g., in terms of energy, runtime, or memory) of running the ML algorithm on the target platform. (iii) Development of ExD, the first parametric data projection method which for a given error bound can generate many possible lower dimensional dictionaries. The ExD parameters are customized to the underlying platform to minimize the quantified performance cost. (iv) Inception of a new mapping method to enable efficient running of the big data ML algorithms on the pertinent computing resources which concurrently considers communication minimization and load balancing. Our mapping achieves the theoretical bounds on communication and storage that are known to be the performance hurdles. (v) Automation of Perform-ML by open-source APIs built on the MPI message passing interface to enable fast adaptation

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2898060

of the framework to new platforms and various iterative ML algorithms. Evaluations of ML applications including large-scale denoising and super-resolution demonstrate more than a 10-fold improvement in runtime, memory footprint, and energy efficiency compared with the prior art.

2. RELATED WORK

To the best of our knowledge, Perform-ML 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 Perform-ML 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 [2]. 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 [2, 10], adaptive CSS methods can create more accurate dictionaries [4, 9, 14, 3]. Farahat [4] and Leverage Scores [9] 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 [14] 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 [14]. 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 [7], spectral clustering [8], dimensionality reduction [2], least squares, norm-1 minimization algorithms [15], and square-root LASSO [20]. However, unlike Perform-ML, 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. Our earlier work, RankMap, proposes the usability of data transformation to generic iterative update algorithms [13]. However, RankMap (unlike Perform-ML) 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) [22] 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 [1, 21]. 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.

3. GLOBAL FLOW OF Perform-ML

The Perform-ML 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 [17], Power method for finding PCA [6], interior point methods for solving SVM [5], 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 Perform-ML is shown in Figure 1. Perform-ML 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 [2, 10]. In Section 4, we introduce our novel and parametric data tranformation method, called ExD. ExD seeks to find a *low-dimensional dictionary matrix* D and a *sparse coefficient matrix* C such that:

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

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 5, 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 6, we demonstrate our approach for tuning ExD to minimize the quantified costs. In Section 7, we provide our evaluations.

4. Perform-ML TRANSFORMATION

Algorithm 1 outlines ExD, Perform-ML's projection method. 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 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 [16]. 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

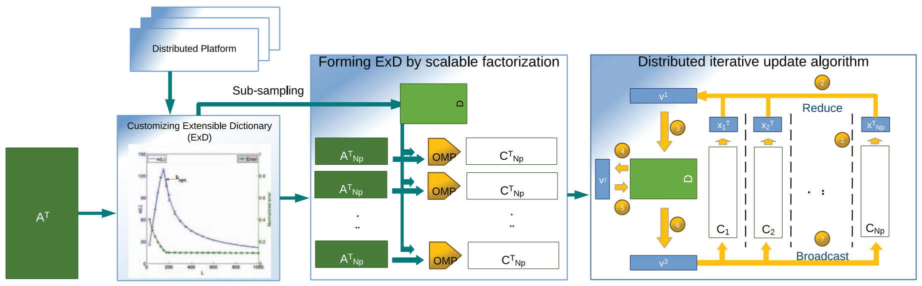
Distributed Platform
Sub-sampling
Customizing Extensible Dictionary (ExD)
A^T
Forming ExD by scalable factorization
D
A^T_Np
OMP
C^T_Np
Distributed iterative update algorithm
v^1
Reduce
x_1^T
x_2^T
x^T_Np
C_1
C_2
C_Np
Broadcast
v^3

Figure 1: Global flow of Perform-ML: 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.

Algorithm 1 : ExD Transformation

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. $pid = 0$[1]creates a random subset of in dices of size $L$ (from $1, 2, \dots, N$), 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 $\epsilon$:
    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
```

tolerance to zero ($\epsilon = 0$) guarantees achieving the same projection error as least-squares approaches. Note that sparse approximation has never been used for platform aware performance optimization.

Selecting enough columns to create D is critical to ensure meeting the approximation error criteria and sparsity. When $L > M$, with a high probability, matrix D becomes full rank and thus the OMP algorithm 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 [10] 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.

Our key observation is that by increasing the dictionary size L, one can vary the sparsity level of C. We extensively use this property to tune ExD in order to minimize the cost of iterative updates in a platform aware manner.

Complexity Analysis. The main complexity of Algorithm 1 arises from executing OMP. We implement the efficient Batch-OMP based on Cholesky factorization updates (Rubinstein et al. 2008). The upper bound on the complexity is $\mathcal{O}(LMN + L^2 nnz(C))$, where $nnz(C)$ measures the number of non-zeros in C. As we show in our experiments for many datasets $nnz(C) \ll LN$. Each column of C is computed independently. Let N_P be the number of parallel processing cores. In this case the complexity of OMP would reduce to $\mathcal{O}(\frac{N}{N_P}(LM + L^2\frac{nnz(C)}{N}))$.

[1]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.

Algorithm 2 : Distributing Gram Matrix Update.

Input: Vector $x_{N\times 1}$, $D_{M\times L}$, 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 $C_i^T v^3$.
```

5. DISTRIBUTED COMPUTING MODEL

Algorithm 2 outlines our proposed data partitioning and distributed computing model to perform an update, i.e., $(DC)^T DCx \simeq A^T Ax$, where x is the $N \times 1$ solution vector. 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 4, we target datasets that are in the regime where 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.

Bounds on Arithmetics. The cost of arithmetics depends on the number of floating point operations for executing $(DC)^T DCx$. The number of floating point operations (in serial) is $2(ML + \frac{nnz(C)}{N_P})$ multiplications and $2MN_P$ addi-

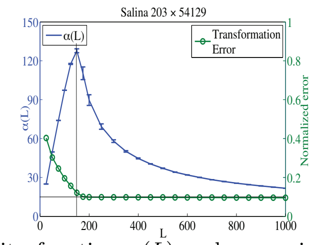

Figure 2: Density function $\alpha(L)$ and approximation error as a function of dictionary size L.

tions, where $nnz(C)$ shows the number of non-zeros in C. 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 with M words to Processor 0, and in Step 6, Processor 0 sends a message with M words back to other processors. In a similar fashion in Case 1, in 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, recent work on communication-optimal parallel recursive rectangular matrix multiplication directly applies to our target problem [1]. 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 $(d_1 \le d_2 \le d_3)$, if $2\frac{d_3}{d_2} > N_P$ (which is the case in our framework when $d_3 = N$), the communication lower bound 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.

Runtime and Energy Performance. The overall execution runtime is approximately proportional to: $ML + \frac{nnz(C)}{N_P} + \min(M, L) N_P R_{b2f}^{time}$, where R_{b2f}^{time} is the word-per-FLOPs ratio that characterizes memory bandwidth per unit of time. The first two terms show the computational operations and the third term reflects the adjusted communication overhead. Although a number of other factors such as memory hierarchy can affect the runtime/energy, we experimentally show that our model provides a reasonable estimation of the actual runtime. Similarly, energy performance can be characterized by replacing R_{b2f}^{time} with R_{b2f}^{energy}.

Memory Performance. The sparsifying effect of ExD results in memory savings. In both proposed implementations (Algorithm 2) the memory footprint per processing node is bounded by $ML + \frac{nnz(C)+N}{N_P}$.

6. AUTOMATED ExD'S CUSTOMIZATION

Perform-ML optimizes the performance of iterative Gram matrix based algorithms by minimizing the quantified performance cost. To do so, it adaptively finds a platform-specific L such that the resulting $(L, nnz(C))$ pair minimizes the performance costs in terms of runtime, memory, or energy. We propose a novel scalable method to tune ExD. Our method estimates $nnz(C)$ as a function of L with preprocessing only a subset of data matrix A.

Figure 2 shows the normalized error $(\|A - DC\|_F / \|A\|_F)$ as a function of L for a sample dataset[2]. The figure also plots the average per-column number of non-zeros in C as a function of L. We denote this function as $\alpha(L, A, \epsilon) = nnz(C)/N$. The search space for parameter L is limited to $L \ge L_{min}$, where L_{min} is the minimum number of columns such that $(\|A - DC\|_F \le \epsilon \|A\|_F)$. In this example $L_{min} \approx 175$. $\alpha(L, A, \epsilon)$ is decreasing for $L > L_{min}$. Since a larger L would result in a greater ensemble of signals in D, a higher sparsity in C can be achieved. The bars on the graph show the result's variation for 100 different initial ensemble collection for D. Note that the variations are very negligible for a fixed L (i.e., less than 4% for this example).

Estimating $\boldsymbol{\alpha(L, A, \epsilon)}$ from Subsets of A. To optimize the performance cost, $\alpha(L, A, \epsilon)$ needs to be efficiently characterized. We make the following two important observations. First, let A be a data with a union of subspace signal model, and A_s be a random subset of A's columns such that $|A_s| = n$ ($|.|$ denotes cardinality). Then, for $n \to N$: $E[\alpha(L, A_s, \epsilon)] = E[\alpha(L, A, \epsilon)]$. Second, given that dataset A admits a union of subspace model, columns of A can be represented as a collection of signals from N_s subspaces where each subspace U_i is K_i-dimensional (for $1 \le i \le N_s$). 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. Based on our definition of the density measure, we have $\alpha(L, A, \epsilon) \le \sum_{1 \le i \le N_s} K_i \frac{n_i}{N}$. 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, the following expected upper bound is achieved for $\alpha(L, A_s, \epsilon) \le \sum K_i \frac{n_i}{N}$, which is the same bound as if the entire dataset was used. Thus, 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 pre-specified threshold.

7. EVALUATION

We implement Perform-ML using the standard message passing system (MPI) in C++. Eigen library is used for linear algebra computations. Our API's inputs include: dataset A, approximation error ϵ, and the iterative update function on Gram matrix. We experimentally find the platform-specific relative cost of computation versus communication (e.g., R_{b2f}^{time}). Our evaluations are done on IBM iDataPlex (node type: Intel-Xeon-X5660@2.80GHz) cluster. Within this cluster, we studied several configuration of nodes and cores (within each node) to emulate various platforms. Table 1 shows the three datasets used in our evaluations.

Salina [18]	Cancer Cell [3]	Light Field [19]
203×54129 87.9MB	1024×111296 911.7MB	18496×272320 40.3GB

Table 1: Datasets used for various applications.

Applications and Baselines. We evaluate two applications: image denoising and image super-resolution. For these applications, we use gradient-descent method to solve the widely popular LASSO objective [17]. Note that our approach is generic and applicable to any regularized or penalized L_2 norm optimization.

We consider two types of comparisons. The first type compares ExD with other existing scalable transformations

[2]The dataset is a collection of Hyperspectral signals (Salina 2015) with $M = 203$, $N = 54129$.

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

including RCSS, oASIS, and RankMap. Each of these transformations can substitute ExD within our proposed end-to-end framework. The second type, compares our approach with other ML acceleration practices. More precisely, we compare Perform-ML's implementation of gradient-descent based approach, with SGS. 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. In each iteration, a subset, denoted by A_b, is randomly selected from a rows of A. The update is done using $A_b^T A_b x$ instead of $A^T A x$. 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 [22]. Perform-ML, however, runs the provably converging gradient-descent on the entire (but transformed) data.

7.1 Evaluating Perform-ML's Pre-processing

We verify the ability of ExD to create versatile sparse transformations. Figure 3 shows the effect of varying ExD parameters, namely dictionary size L, and error ϵ, to achieve different sparsity levels in C. The y axis demonstrates the average number of non-zeros per column of C denoted by $\alpha(L)$ [4]. 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 ($\epsilon = 0.1$, 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 error-tolerance, we can achieve sparser solutions. Perform-ML takes advantage of these tunable characteristics to tune the transformation w.r.t. the platform. Recall that the trade-off is that increasing L would yield a higher communication (Section 5), and increasing ϵ might yield to degradation of learning accuracy.

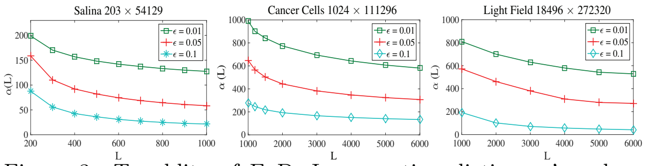

Figure 3: Tunablity of ExD. Incrementing dictionary's redundancy and projection error yields to sparser results.

Figure 4, shows that a highly accurate estimation of $\alpha(L)$ can be achieved by running ExD on subsets of data, thus, reducing the overhead of tuning ExD during the pre-processing. 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 characterized for various Ls, it can be used to reduce our quantified performance model.

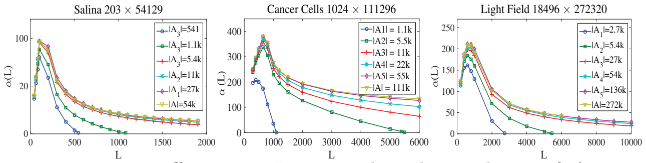

Figure 4: Effective ExD tuning based on subsets of A.

Table 2 shows the total pre-processing time overhead, which includes tuning and running ExD for optimal L. The computations are done on 64 cores (8 nodes each with 8 cores). The overhead of ExD, which is a one-time pre-processing step is amortized when iterative algorithms are run on the transformed data.

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

Table 2: Pre-processing overhead in ms.

7.2 Comparison with Other Transformations

Runtime Analysis. Figure 5 demonstrates the runtime improvement achieved by using ExD for Gram matrix updates over other approaches including the baseline $A^T A$, and the state of the art scalable transformations RCSS, oASIS, and RankMap (all used within the Perform-ML framework). All the transformations are done for $\epsilon = 0.1$. We compare the runtime of an iterative update on the Gram matrix, i.e., $A^T Ax$, while using the transformed data ($(DC)^T DCx$) instead. $A_{M \times N}$ is the dataset and $x_{N \times 1}$ is a random vector. We measure the runtime on 4 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 on each platform. In comparison, in all cases ExD yields better or equal runtime. We observe up to 40.78× (runtime) improvement over $A^T A$, 9.12× improvement over RCSS, 6.67× over oASIS, and 2.63× improvement over RankMap. For Light Field we achieve comparable runtime with RankMap (Perform-ML achieves 10% runtime improvement for $N_P = 1$ and equal runtime for other N_Ps). This is because for Light Field, ExD similar to RankMap, chooses the smallest basis (Figure 6). The relative (overall) runtime of each of the methods for completing an ML solution would be proportional to that of one iteration.

Memory Analysis. Table 3 compares the required memory for storing matrices C and D for different transformations. The memory usage of the original matrix A is also provided. Other than ExD, other methods always result in the same memory footprint regardless of the platform. Perform-ML results in up to 77.8× (memory usage) improvement over $A^T A$, 8.6× improvement over RCSS, 6.4× improvement over oASIS, and 3.8× improvement over RankMap. The memory efficiency in ExD is achieved by flexibly creating overcomplete, dictionaries that result in sparse C's.

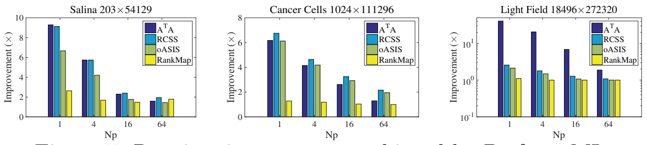

Figure 5: Runtime improvement achieved by Perform-ML.

	Original data	RCSS	oASIS	RankMap	Perform-ML $N_p = 1$	Perform-ML $N_p = 4$	Perform-ML $N_p = 16$	Perform-ML $N_p = 64$
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

Table 3: Memory comparison (MB).

7.3 Evaluating Performance Model

We evaluate our quantified performance model by comparing the predicted runtime trend with the measured one. Figure 6 shows the results for running one iteration of Gram matrix, i.e., $(DC)^T DCx$. The measured runtimes are averaged over 100 iterations. The tests are done on various

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

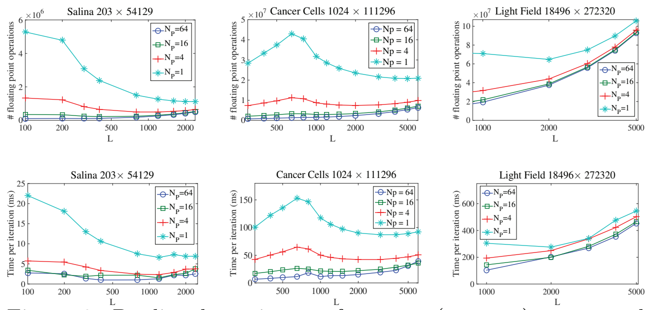
Figure 6: Predicted runtime performance (top row) vs. actual measured runtime (bottom row).

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 shown, the measured performance (bottom) is very similar to our estimated performance (top), corroborating that the quantified model can be used to tune ExD.

7.4 Evaluating Learning Applications

Figure 7 compares Perform-ML's total runtime for denoising and super-resolution applications. Our gradient-descent approach not only benefits from a guaranteed convergence (unlike SGD), it also yields faster convergence. Another advantage is ExD's ability to reduce the data storage overhead. In both applications the ϵ in ExD is set to 0.1. It can be seen that Perform-ML achieves up to 3.7× (runtime) improvement for denoising application and up to 10.9× improvement for super-resolution application over SGD.

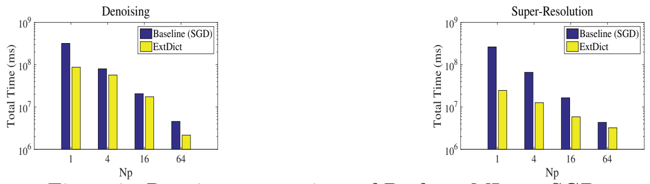

Figure 7: Runtime comparison of Perform-ML vs. SGD.

To evaluate the quality of reconstruction we use the Peak Signal to Noise Ratio (PSNR) metric. PSNR is the ratio between the maximum signal power and noise ($10\log_{10}(\frac{MAX}{\sqrt[2]{MSE}})$ (dB)). Recommended PSNR values in vision applications are $25dB$ and higher (Aharon et al. 2006). For denoising application, our output PSNR is $29.39dB$ when input SNR of the noisy image is $15.15dB$. For super-resolution application, our output PSNR is $24.69dB$.

Our evaluations demonstrate the significant impact of content and platform aware customization to gain efficiency. The cost of Perform-ML's pre-processing is rapidly paid off over several runs of iterative updates. Moreover, most ML problems require model selection which translates to running the algorithms for several modeling parameters. This, further amortizes the one-time pre-processing overhead. We also observe that while higher transformation errors can result in meaningful runtime and memory improvements, they may not drastically affect the reconstruction error.

8. ACKNOWLEDGEMENT

This work was supported by The Office of Naval Research (ONR) Grant No. R17460.

9. CONCLUSION

We propose Perform-ML, an end-to-end solution for highly efficient execution of iterative ML algorithms on massive data. We introduce a new data projection method (called ExD) which leverages coarse-grained parallelism in the data to create tunable sparse transformations. The transformation is low-overhead and highly scalable. Perform-ML reduces the performance cost by adaptively customizing the transformation for the underlying platform. We provide a distributed API that enables applying Perform-ML to a wide range of learning problems in large scale. Our extensive evaluations show that Perform-ML can achieve significant improvements in execution runtime, energy, and memory footprint compared to prior art.

10. REFERENCES

[1] Demmel, J., Eliahu, D., Fox, A., Lipshitz, B., and Spillinger, O. Communication optimal parallel recursive rectangular matrix multiplication. *IPDPS'13*.

[2] Drineas, P., and Mahoney, M. On the nyström method for approximating a gram matrix for improved kernel-based learning. *JMLR'05*.

[3] Dyer, E., Goldstein, T., Patel, R., Kording, K., and Baraniuk, R. Self-expressive decompositions for matrix approximation and clustering. *arXiv:1505.00824* (2015).

[4] Farahat, A., Elgohary, A., Ghodsi, A., and Kamel, M. Greedy column subset selection for large-scale data sets. *Knowledge and Information Systems* (2014).

[5] Ferris, M. C., and Munson, T. S. Interior-point methods for massive support vector machines. *SIAM J. on Optimization* (2002).

[6] Figueiredo, M., Nowak, R., and Wright, S. Gradient projections for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE J. Select. Top. Signal Processing 1*, 4 (2007).

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

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

[9] Gilbert, A., Strauss, M., Tropp, J., and Vershynin, R. One sketch for all: Fast algorithms for compressed sensing. *STOC '07*.

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

[11] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. GraphLab: A new parallel framework for machine learning. *UAI'10*.

[12] Malewicz, G., Austern, M., Bik, A., Dehnert, J., Horn, I., Leiser, N., and Czajkowski, G. Pregel: a system for large-scale graph processing. *SIGMOD'10*.

[13] Mirhoseini, A., Baraniuk, R., and Koushanfar, F. Rankmap: A platform-aware framework for distributed learning from dense datasets. *preprint:1503.08169* (2015).

[14] Patel, R., Goldstein, T., and Baraniuk, R. Adaptive column sampling and nystrom approximation via oasis. *SDM'16*.

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

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

[17] Tibshirani, R. Regression shrinkage and selection via the lasso. *J. Royal Statist. Soc B 58*, 1 (1996).

[18] www.ehu.es/ccwintco/. Aviris hyperspectral data.

[19] www.lightfield.stanford.edu. The Light Field Archive.

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

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

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