Communication Avoiding Coordinate Descent Methods On Kernel And Regularized Linear Models

BY

ZISHAN SHAO

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE AT WAKE FOREST UNIVERSITY

in Partial Fulfillment of the Requirements for

Graduation With Honors in

Computer Science

May 2024
Winston-Salem, North Carolina

Approved By:

____
(Advisor's Name, B.S., Advisor)

____
(Committee Member's Name, B.S., Chair of Honors Committee)

____
(Department Chair's Name, B.S., Department Chair)

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Aditya Devarakonda for his invaluable support and mentorship on my journey through research. His passion for both research and teaching has been a guiding light for me in navigating the vast sea of knowledge. I am also profoundly thankful to my best friend in college, Dizhou Wu, with whom discussions on research have been both enlightening and immensely enjoyable. My deepest appreciation goes to my girlfriend, Yuka Maeyama, my parents, Kuoyi Shao and Xiuyun Fu, and my siblings, Ziqian and Zizhan, for their unwavering support and encouragement. Additionally, I would like to thank Dr. Pauca, Dr. Alqahtani, Dr. Turkett, Dr. Ballard, Cody, Adam, Sean, and everyone in the Wake Forest Department of Computer Science for their kind assistance and inspiring lectures. The support and guidance from these remarkable individuals have been instrumental in my academic journey at Wake Forest University.

This work was made possible through support under Contract No. DE-AC05-00OR22725 with the US Department of Energy and utilized resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0024170. Additionally, this research benefited from computational resources provided by the Wake Forest University (WFU) High Performance Computing Facility for testing and debugging. I am deeply grateful for the opportunity to leverage these advanced facilities and resources in the pursuit of knowledge.

TABLE OF CONTENTS

ABSTRACT

Coordinate Descent algorithms have been widely used to solve machine learning tasks such as ridge regression, SVM, regularized linear models. etc. However, the runtime performance of such algorithm are often bottlenecked by communication overhead, especially in distributed systems. This thesis aims to address these inefficiencies by focusing on the development and analysis of communication-avoiding variants of coordinate descent algorithms for Elastic-Net and kernel problems, in which we apply the s-step method to minimize data exchanges between computing nodes. By reformulating the traditional coordinate descent methods, we introduce novel communication-avoiding algorithms that reduce the communication overhead and retains the original methods' robustness and effectiveness.

1 Introduction

Coordinate Descent methods are commonly used in training machine learning models. These iterative methods prove invaluable in solving a wide range of regression and classification optimization problems, both regularized and unregularized, across various fields such as biology, computer vision, biophysics, and healthcare. However, the rapid growing size of training datasets urgently requires innovative solutions to the challenges of efficiently training machine learning models. One common approach is performing distributed computation via high-performance computing architectures to achieve efficient training of the models. In this way, the challenge of scaling these optimization methods to solve problems quickly and efficiently has become a significant focus of research.

Historically, the performance of coordinate descent methods in distributed-memory environments has been constrained by the high cost of communication due to their iterative nature, which often exceeds computation costs when dealing with large datasets. Iterative algorithms and their communication bottlenecks are well known, particularly with respect to Krylov subspace methods [1, 2, 3, 4, 5, 6]. In this work, we borrow ideas from Krylov subspace methods and apply them to the DCD and BDCD optimization methods to reduce the frequency of communication. Meanwhile, we extends the application of s-Step communication-avoiding techniques to the coordinate descent methods for solving Regularized Generalized Linear Models (R-GLMNet) problem. By adopting s-Step methods, we aim to reduce communication frequency by a user-tunable factor of s without compromising the methods' accuracy or convergence behavior. The contributions of this thesis include:

- Derivation and theoretical analysis of s-step variants of CD, DCD and BDCD methods for solving the regularized logistic regression, regularized multinomial regression, kernel SVM and kernel ridge regression problems which reduce the frequency of communication by a factor of s at the expense of additional communication bandwidth and computation.
- Empirical evaluation of the numerical stability (in MATLAB) of the s-step variants as a function of s for several small-scale benchmark classification and regression datasets.
- Performance evaluation of the strong scaling and running time breakdown of the proposed methods on a Cray EX system, which show strong scaling speedups of up to 9.8$\times$ on large-scale dense and sparse datasets.
- An empirical evaluation of the numerical stability and performance of these s-step variants, demonstrating their efficacy through MATLAB/R simulations on small-scale benchmark datasets for classification and regression.

2 Methodology

2.1 Constrained Optimization Problem

Constrained optimization problems are pivotal in the realm of machine learning, serving as the foundation for a variety of algorithms that aim to discover the optimal parameters minimizing a cost function while adhering to certain constraints. Support Vector Machines (SVM) is one such example, where the goal is to find the maximum-margin hyperplane that separates different classes. The hyperplane is represented by the weight vector $\mathbf{x} \in \mathbb{R}^n$, where n is the feature space dimension. Similarly, ridge regression aims to minimize the sum of squared residuals while penalizing the magnitude of the coefficient vector, effectively controlling the model complexity.

Both problems can be reformulated into dual problems via Lagrangian multipliers, allowing for derivative-based optimization methods such as Dual Coordinate Descent. The convex nature of SVM and ridge regression ensures that any local minimum is a global minimum, and solutions can be efficiently computed. For instance, the problem could be solved with Newton-Raphson method utilizing second-order derivative information to iteratively converge to the optimal solution.

In addition to SVM and ridge regression, Generalized Linear Models with regularization, known as GLMNet, present another constrained optimization framework. GLMNet combines $L1$ and $L2$ regularization, effectively bridging the gap between Lasso regression, which encourages sparsity in the solution, and ridge regression, which penalizes large coefficients. This hybrid approach is represented as the minimization problem:

$$\min_{\beta} \left\{ \frac{1}{m} \sum_{i=1}^{m} L(y_i, \mathbf{a}_i\beta) + \lambda \left[(1-\alpha)\|\beta\|_2^2 + \alpha\|\beta\|_1 \right] \right\},$$

where $A \in \mathbb{R}^{m\times n}$ is the data matrix, $L(y_i, \mathbf{a}_i\beta)$ is the loss function measured between the actual response y_i and the predicted response $\mathbf{a}_i\beta$, $\beta \in \mathbb{R}^n$ is the coefficient vector for n predictors, $\mathbf{a}_i$ is the feature vector of the i-th observation, m is the number of observations, λ is the regularization parameter, and α is the mixing parameter between $L1$ and $L2$ regularization.

2.2 Parallelism in C

The use of parallel computing in handling large datasets involves the distribution of data across multiple processors, which facilitates the simultaneous execution of computational tasks. This distribution significantly reduces the total processing time, especially for computationally intensive tasks inherent in large-scale data problems. In the context of linear models, such as those encountered in kernel ridge regression (K-RR), where operations like matrix multiplication AA^T are prevalent, parallelism can greatly enhance performance.

When partitioning the dataset, each processor receives a subset of the data to allow for a distributed approach of computations. This method requires the linear separability of the computation, such as the matrix multiplications. However, this approach comes with a trade-off in increasing communication cost for synchronization.

In this research, we focus on modeling the communication between processors by computing the overhead ahead of the computation in Parallel. For instance, in K-RR method, the A matrix was stored in parallel where each processor stores $\left\lfloor \frac{n}{p} \right\rfloor$ features assuming the dataset was load balanced. Each processor will compute AA^T matrix per iteration only with the stored features, and synchronize by communicating through all local computed products.

2.3 Sparse Data Storage

In scenarios where datasets are highly sparse, such as the news20.binary dataset, storing data in a dense format can lead to significant memory inefficiencies. In the context of large-scale data processing, it is crucial to utilize memory-efficient storage techniques for such sparse datasets. Sparse datasets are often represented using specialized formats that store only non-zero elements, thus dramatically reducing memory usage. Two common formats are Compressed Sparse Row (CSR) and Compressed Sparse Column (CSC).

Let us consider a sparse matrix A of dimensions $m \times n$, where m represents the number of rows and n the number of columns. In a situation where A has a sparsity of 99%, meaning 99% of its elements are zero, sparse storage formats become a practical alternative. The CSR format represents this sparse matrix as three one-dimensional arrays: one for the non-zero values, one for the extents of rows, and one for the column indices of the non-zero values. This leads to a compression of the original dataset size to $\mathcal{O}(fmn)$, where f is the density of the dataset.

Similarly, the CSC format is the column-wise counterpart to CSR. It includes three arrays: one for the non-zero values, one for the column pointers, and one for the row indices of the non-zero values. CSC format is particularly advantageous for column-wise operations and is instrumental in algorithms where column access is frequent such as coordinate descent (CD) and block coordinate descent (BCD).

3 Derivation

In this section, we introduce the Kernelized Support Vector Machine (K-SVM) and Ridge Regression (K-RR) problems. We present the dual coordinate descent (DCD) and its blocked (BDCD) variant. Finally, we perform s-step derivations of DCD for K-SVM and BDCD for K-RR.

3.1 Support Vector Machine

Support Vector Machine (SVM) [7, 8, 9] are supervised learning models used for binary classification. SVM classifies the input dataset by finding a hyperplane defined by $H := \{x \mid v^\intercal x - \rho = 0\}$, where $v \in \mathbb{R}^n$ is any vector, $\rho \in \mathbb{R}$ is the intercept, and $x \in \mathbb{R}^n$ is the normal vector to the hyperplane. Given a dataset $A \in \mathbb{R}^{m \times n}$ and a vector of binary labels $y \in \mathbb{R}^m$ s.t. $y_i \in \{-1, +1\}\ \forall\ i = 1, \dots, m$, the SVM optimization problem finds a hyperplane which maximizes the distance (margin) between the two classes. A good separation is achieved when the hyperplane, x, has the greatest distance from the nearest training data points (support vectors) from the two classes. This formulation is known as the hard-margin SVM problem, which implicitly assumes that the data points are linearly separable. For datasets containing errors or where the margin between the two classes is small, the hard-margin SVM formulation may not achieve high accuracy. The soft-margin SVM problem introduces slack variables for each data point, $\xi_i\ \forall\ i = 1, \dots, m$, so that the margin can be increased by allowing misclassification of some data points. Note that setting $C = 0$ recovers the hard-margin formulation, so we will focus on solving the soft-margin SVM problem:

$$\begin{aligned}
&\underset{x \in \mathbb{R}^n}{\arg\min} \frac{1}{2}\|x\|_2^2 + C\sum_{i=1}^{m} \xi_i \\
&\text{subject to } y_i(a_{i,:}x + \rho) \leq 1 - \xi_i \\
&\qquad\qquad\ \ \xi_i \geq 0 \\
&\qquad\qquad\ \ \forall\ i = 1, \dots, m
\end{aligned} \tag{1}$$

The constraints in (1) can be re-written in the empirical risk minimization form[1] by introducing the hinge loss for each ξ_i:

$$\begin{aligned}
&\textbf{SVM-L1:}\ \max(1 - y_i a_{i,:}x, 0) \\
&\textbf{SVM-L2:}\ \max(1 - y_i a_{i,:}x, 0)^2.
\end{aligned}$$

We refer to the hinge loss variant as the L1-SVM problem and the squared hinge-loss variant as the L2-SVM problem. For datasets that are not linearly separable in n dimensions, the SVM problem can be solved in a high-dimensional feature space by introducing a non-linear kernel function. The kernelized variants of the L1-SVM and L2-SVM problems can be obtained by deriving their Lagrangian dual problems,

$$\begin{aligned}
\textbf{KSVM-L1:}\ &\underset{\alpha \in \mathbb{R}^m}{\arg\min} \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathcal{K}(a_{i,:}, a_{j,:}) - \sum_{i=1}^{m} \alpha_i \\
&\text{subject to } 0 \leq \alpha_i \leq C, \\
\textbf{KSVM-L2:}\ &\underset{\alpha \in \mathbb{R}^m}{\arg\min} \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathcal{K}(a_{i,:}, a_{j,:}) - \sum_{i=1}^{m} \alpha_i \\
&+ \frac{1}{4C}\sum_{i=1}^{m} {\alpha_i}^2,
\end{aligned}$$

where $\alpha \in \mathbb{R}^m$ is the solution to the Lagrangian dual problem and $\mathcal{K}(a_{i,:}, a_{j,:})$ is a kernel function which defines an inner product space. Table 1 lists the kernel functions used in this work. The L1 and L2 KSVM problems can be solved using several algorithmic variants of coordinate descent [10, 11, 12]. In this work, we will focus on cyclic coordinate descent, which we will refer to as Dual Coordinate Descent (DCD).

[1]We omit the bias term in the derivation for presentation clarity.

Kernel function	Definition
Linear	$a_{i,:}a_{j,:}^\intercal$
Polynomial	$(c + a_{i,:}a_{j,:}^\intercal)^d$, for $c \geq 0, d \geq 1$.
Radial Basis Function (RBF)	$\exp(-\sigma\|a_{i,:} - a_{j,:}\|_2^2)$, for $\sigma > 0$.

Table 1: Kernel functions explored in this work

DCD is an iterative algorithm which reduces the KSVM problems to single-variable (or coordinate) problems which have closed-form solutions. Once a sub-problem is solved, DCD proceeds by selecting a new variable, solving the sub-problem with respect to the chosen variable, and repeating this process until convergence. Algorithm 1 shows the DCD algorithm for solving the L1 and L2 KSVM problems.

Algorithm 1 Dual Coordinate Descent (DCD) for Kernel SVM

```
 1: Input: $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m, \alpha_0 \in \mathbb{R}^m, H > 1, C \in \mathbb{R}$,
    $\mathcal{K} := A \times A \mapsto \mathbb{R}$
 2: $\begin{cases} \nu = C,\ \omega = 0, & \text{L1-KSVM} \\ \nu = \infty,\ \omega = 1/2C, & \text{L2-KSVM} \end{cases}$
 3: $\tilde{A} = \text{diag}(y) \cdot A$
 4: for $k = 1, 2, \ldots, H$ do
 5:     Choose $i_k \in [m]$ uniformly at random.
 6:     $e_{i_k} \in \mathbb{R}^m$, the $i_k$-th standard basis vector.
 7:     $u_k = \mathcal{K}\left(\tilde{A}, e_{i_k}^\intercal \tilde{A}\right)$
 8:     $\eta_k = e_{i_k}^\intercal u_k + \omega$
 9:     $g_k = u_k^\intercal \alpha_{k-1} - 1 + \omega e_{i_k}^\intercal \alpha_{k-1}$
10:     $\tilde{g}_k = \left|\min\left(\max\left(e_{i_k}^\intercal \alpha_{k-1} - g_k, 0\right), \nu\right) - e_{i_k}^\intercal \alpha_{k-1}\right|$
11:     if $\tilde{g}_k \neq 0$ then
12:         $\theta_k = \min\left(\max\left(e_{i_k}^\intercal \alpha_{k-1} - \frac{g_k}{\eta_k}, 0\right), \nu\right) - e_{i_k}^\intercal \alpha_{k-1}$
13:     else
14:         $\theta_k = 0$
15:     end if
16:     $\alpha_k = \alpha_{k-1} + \theta_k e_{i_k}$
17: end for
18: Output: $\alpha_H$
```

3.2 s-Step DCD Derivation

Note that Algorithm 1 selects a single data point from A at each iteration. This limits DCD performance to BLAS-1 and BLAS-2 operation in each iteration. This limitation also suggests that a distributed-memory parallel implementation of DCD would require communication at every iteration. We propose to improve the performance of DCD by deriving a mathematically equivalent variant we refer to as s-step DCD which avoids communication for s iterations. We begin by modifying the iteration index from k (for DCD) to $sk + j$, where $j \in \{1, 2, \ldots, s\}$ and $k \in \{0, 1, \ldots, H/s\}$. We begin the

s-step derivation by assuming that α_{sk} was just computed and show how to compute the next s solution updates. We see that from Algorithm 1 u_k, g_k, and α_k are vector quantities whereas η_k and θ_k are scalar quantities. The following two solution updates at iterations $sk+1$ and $sk+2$ require computing the gradients g_{sk+1} and g_{sk+2}, where are defined by:

$$g_{sk+1} = u_{sk+1}^{\intercal}\alpha_{sk} - 1 + \omega e_{i_{sk+1}}^{\intercal}\alpha_{sk}$$
$$g_{sk+2} = u_{sk+2}^{\intercal}\alpha_{sk+1} - 1 + \omega e_{i_{sk+2}}^{\intercal}\alpha_{sk+1}$$

However, notice we can also replace α_{sk+1} with its equivalent quantity from iteration sk.

$$\alpha_{sk+1} = \alpha_{sk} + \theta_{sk+1}e_{i_{sk+1}}$$

Given that θ_{sk+1} is a scalar quantity, g_{sk+2} can be rewritten as,

$$\begin{aligned} g_{sk+2} = u_{sk+2}^{\intercal}\alpha_{sk} - 1 + \theta_{sk+1}u_{sk+1}^{\intercal}e_{i_{sk+1}} + \\ \omega e_{i_{sk+2}}^{\intercal}\alpha_{sk} + \omega e_{i_{sk+2}}^{\intercal}\theta_{sk+1}e_{i_{sk+1}} \end{aligned}$$

This recurrence unrolling suggests that g_{sk+2} can be computed using α_{sk} provided that the quantities u_{sk+1}, u_{sk+2} and θ_{sk+1} are available. The quantities u_{sk+1} and u_{sk+2} are independent, so they can be computed simultaneously. The sequential dependence on θ_{sk+1} cannot be eliminated. We can generalize this recurrence unrolling to g_{sk+j} for an arbitrary number of iterations. Notice that g_{sk+j} is defined as

$$g_{sk+j} = u_{sk+j}^{\intercal}\alpha_{sk+j-1} - 1 + \omega e_{i_{sk+j}}^{\intercal}\alpha_{sk+j-1}.$$

Given that α_{sk+j-1} can be defined as

$$\alpha_{sk+j-1} = \alpha_{sk} + \sum_{t=1}^{j-2}\theta_{sk+t}e_{sk+t},$$

we can unroll the g_{sk+j} by substitution:

$$\begin{aligned} g_{sk+j} = u_{sk+j}^{\intercal}\alpha_{sk} + u_{sk+j}^{\intercal}\sum_{t=1}^{j-1}\theta_{sk+t}e_{i_{sk+t}} \\ - 1 + \omega e_{i_{sk+j}}^{\intercal}\alpha_{sk} + \omega e_{i_{sk+j}}^{\intercal}\sum_{t=1}^{j-1}\theta_{sk+t}e_{i_{sk+t}}. \end{aligned}$$

Since all quantities u_{sk+j} are independent, we can compute them upfront with a batched kernel computation after which all quantities θ_{sk+j} can be computed sequentially. Once all s θ_{sk+j} quantities have been computed, α_{sk+s} can be computed directly as follows

$$\alpha_{sk+s} = \alpha_{sk} + \sum_{t=1}^{s}\theta_{sk+t}e_{sk+t}.$$

Algorithm 2 shows the resulting s-step DCD algorithm for solving the L1 and L2 KSVM problems.

Algorithm 2 s-Step DCD for Kernel SVM

```
1: Input: $A \in \mathbb{R}^{m \times n}, y \in \mathbb{R}^m, \alpha_0 \in \mathbb{R}^m, C \in \mathbb{R}, s \in \mathbb{Z}^+, H \geq s$,
   $\mathcal{K} := A \times A \mapsto \mathbb{R}$
2: $\begin{cases} \nu = C,\ \omega = 0, & \text{L1-KSVM} \\ \nu = \infty,\ \omega = 1/2C, & \text{L2-KSVM} \end{cases}$
3: $\tilde{A} = \operatorname{diag}(y) \cdot A$
4: for $k = 0, \ldots, H/s$ do
5:     for $j = 1, \ldots, s$ do
6:         $i_{sk+j} \in [m]$, chosen uniformly at random.
7:         $e_{i_{sk+j}} \in \mathbb{R}^m$, the $i_{sk+j}$-th standard basis vector.
8:     end for
9:     $V_k = \left[e_{i_{sk+1}}, \ldots, e_{i_{sk+s}}\right]$
10:    $\tilde{A}_k = V_k^\intercal \tilde{A}$
11:    $U_k = \mathcal{K}(\tilde{A}, \tilde{A}_k)$
12:    $G_k = V_k^\intercal U_k + \omega I$
13:    $[\eta_{sk+1}, \ldots, \eta_{sk+s}]^\intercal = \operatorname{diag}(G_k)$
14:    for $j = 1, \ldots, s$ do
15:        $\rho_{sk+j} = e_{i_{sk+j}}^\intercal \alpha_{sk} + e_{i_{sk+j}}^\intercal \sum_{t=1}^{j-1} \theta_{sk+t} e_{i_{sk+t}}$
16:        $g_{sk+j} = (U_k e_j)^\intercal \alpha_{sk} - 1 + \omega e_{i_{sk+j}}^\intercal \alpha_{sk} + \sum_{t=1}^{j-1} \left(e_{i_{sk+t}}^T U_k e_j\right) \theta_{sk+t} + \omega e_{i_{sk+j}}^\intercal \sum_{t=1}^{j-1} \theta_{sk+t} e_{i_{sk+t}}$
17:        $\tilde{g}_{sk+j} = \left|\min\left(\max\left(\rho_{sk+j} - g_{sk+j}, 0\right), \nu\right) - \rho_{sk+j}\right|$
18:        if $\tilde{g}_{sk+j} \neq 0$ then
19:            $\theta_{sk+j} = \min\left(\max\left(\rho_{sk+j} - \frac{g_{sk+j}}{\eta_{sk+j}}, 0\right), \nu\right) - \rho_{sk+j}$
20:        else
21:            $\theta_{sk+j} = 0$
22:        end if
23:    end for
24:    $\alpha_{sk+s} = \alpha_{sk} + \sum_{t=1}^{s} \theta_{sk+t} e_{i_{sk+t}}$
25: end for
26: Output: $\alpha_H$
```

3.3 Kernel Ridge Regression

The ridge regression problem [13] for regression is defined as follows

$$\arg\min_{x \in \mathbb{R}^n} \frac{1}{2m}\|Ax - y\|_2^2 + \frac{\lambda}{2}\|x\|_2^2$$

By deriving the Lagrangian dual formulation, we obtain the Kernel Ridge Regression (K-RR) problem,

$$\arg\min_{\alpha \in \mathbb{R}^m} \frac{1}{2}\left(\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j \left(\frac{1}{\lambda}\mathcal{K}(a_{i,:}, a_{j,:}) + m\right)\right) - \sum_{i=1}^{m} \alpha_i y_i \tag{2}$$

In contrast to K-SVM, this problem can be solved in closed form. However, when $m \gg 1$, explicitly computing and storing the $m \times m$ kernel matrix can be prohibitive. In this work, we focus on iteratively solving the K-RR problem using the Block Dual Coordinate Descent (BDCD) method. BDCD solves (2) by randomly or cyclically selecting a block size, b, of samples from A and solving a subproblem with respect to just those coordinates. Algorithm 3 shows the BDCD algorithm for solving the K-RR problem.

Algorithm 3 Block Dual Coordinate Descent (BDCD) for Kernel Ridge Regression

```
1: **Input:** $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m, b \in \mathbb{Z}^+ s.t. b \leq m, H > 1, \alpha_0 \in \mathbb{R}^m$
2: $\mathcal{K} := A \times A \mapsto \mathbb{R}$
3: **for** $k = 0, \ldots, H$ **do**
4:     choose $\{i_l \in [m] | l = 1, 2, \ldots, b\}$ uniformly at random without replacement
5:     $V_k = [e_{i_1}, e_{i_2}, \ldots, e_{i_b}]$
6:     $U_k = \mathcal{K}(A, V_k^\intercal A)$
7:     $\Theta_k = \frac{1}{\lambda} V_k^\intercal U_k + mI$
8:     $\Delta\alpha_k = \Theta_k^{-1}\left(V_k^\intercal y - m V_k^\intercal \alpha_{k-1} - \frac{1}{\lambda} U_k^\intercal \alpha_{k-1}\right)$
9:     $\alpha_k = \alpha_{k-1} + V_k \Delta\alpha_k$
10: **end for**
11: **Output** $\alpha_H$
```

3.4 s-Step BDCD Derivation

Similar to DCD, we can also unroll the recurrence relationship in Algorithm 3. We see that from Algorithm 3 that the matrix quantities $U_k \in \mathbb{R}^{b\times m}$ and $G_k \in \mathbb{R}^{b\times b}$ are required at every iteration to solve the subproblem and compute the vector quantity $\Delta\alpha_k \in \mathbb{R}^b$. We begin the s-step derivation by modifying the iteration index from k to $sk + j$ where $j \in \{1, 2, \ldots, s\}$ and $k \in \{0, 1, \ldots, H/s\}$, as before. We assume that α_{sk} was just computed and show how to compute the next s solution updates. We focus on the subproblem solutions defined at iterations $sk + 1$ and $sk + 2$ for the s-step derivation, which are given by,

$$\Delta\alpha_{sk+1} = \Theta_{sk+1}^{-1}\left(V_{sk+1}^{\intercal}y - mV_{sk+1}^{\intercal}\alpha_{sk} - \frac{1}{\lambda}U_{sk+1}^{\intercal}\alpha_{sk}\right)$$
$$\Delta\alpha_{sk+2} = \Theta_{sk+2}^{-1}\left(V_{sk+2}^{\intercal}y - mV_{sk+2}^{\intercal}\alpha_{sk+1} - \frac{1}{\lambda}U_{sk+2}^{\intercal}\alpha_{sk+1}\right).$$

Using the solution update, $\alpha_{sk+1} = \alpha_{sk} + V_{sk+1}\Delta\alpha_{sk+1}$, we can unroll the recurrence for $\Delta\alpha_{sk+2}$ by substitution. This yields

$$\Delta\alpha_{sk+2} = \Theta_{sk+2}^{-1}\Big(V_{sk+2}^{\intercal}y - mV_{sk+2}^{\intercal}\alpha_{sk} - mV_{sk+2}^{\intercal}V_{sk+1}\Delta\alpha_{sk+1} - \frac{1}{\lambda}U_{sk+2}^{\intercal}\alpha_{sk} - \frac{1}{\lambda}U_{sk+2}^{\intercal}V_{sk+1}\Delta\alpha_{sk+1}\Big).$$

Notice that the solutions at iteration $sk+1$ and $sk+2$ both depend on α_{sk}, but $\Delta\alpha_{sk+2}$ requires additional correction terms because α_{sk+1} is never explicitly formed. This recurrence unrolling can be generalized to an arbitrary future iteration, $sk+j$, as follows

$$\Delta\alpha_{sk+j} = \Theta_{sk+j}^{-1}\Big(V_{sk+j}^{\intercal}y - mV_{sk+j}^{\intercal}\alpha_{sk} - m\sum_{t=1}^{j-1}V_{sk+j}^{\intercal}V_{sk+t}\Delta\alpha_{sk+t} - \frac{1}{\lambda}U_{sk+j}^{\intercal}\alpha_{sk} - \frac{1}{\lambda}\sum_{t=1}^{j-1}U_{sk+j}^{\intercal}V_{sk+t}\Delta\alpha_{sk+t}\Big). \tag{3}$$

In (3) the quantities U_{sk+j} for $j \in \{1, 2, \ldots, s\}$ can be computed upfront by selecting sb coordinates and computing a kernel matrix that has a factor of s additional rows. The sequence of U_{sk+j}'s can then be extracted from the $m \times sb$ kernel matrix. The resulting s-step BDCD algorithm for K-RR is shown in Algorithm 4.

4 Analysis of Algorithm

In this section we analyze their computation and communication costs (Section 4.1) using Hockney's performance model [14]: $\gamma F + \beta W + \phi L$, where F, W and L represent the algorithm costs for computation, bandwidth, and latency; and γ, β and ϕ represent the associated hardware parameters. Given the similarities between the coordinate descent methods for K-RR and K-SVM, we focus our analysis on Algorithms 3 and 4 which are blocked generalizations of DCD. The leading-order costs of BDCD for K-RR can be specialized to DCD for K-SVM by setting the block size, $b = 1$. Independent analyses of BDCD and DCD are required in order to bound constants, which we omit in this work.

4.1 Computation and Communication Analysis

We assume that $A \in \mathbb{R}^{m\times n}$ is a sparse matrix with density f, where $0 < f \leq 1$, such that the non-zeros are uniformly distributed where each row contains fn non-zero entries. We further assume that the processors are load balanced with fmn/P

Algorithm 4 s-Step BDCD for Kernel Ridge Regression

```
1: **Input:** $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m, b \in \mathbb{Z}^+ s.t. b \leq m,$
2: $s \in \mathbb{Z}^+, H \geq s, \alpha_0 \in \mathbb{R}^m, \mathcal{K} := A \times A \mapsto \mathbb{R}$
3: **for** $k = 0, \ldots H/s,$ **do**
4:     **for** $j = 1, 2, \ldots, s$ **do**
5:         Choose $\{i_l \in [m] \mid l = 1, 2, \ldots, b\}$ uniformly
6:         at random without replacement
7:         $V_{sk+j} = [e_{i_1}, e_{i_2}, \ldots, e_{i_b}] \in \mathbb{R}^{m\times b}$
8:     **end for**
9:     $\Omega_k = [V_{sk+1}, V_{sk+2}, \ldots, V_{sk+s}] \in \mathbb{R}^{m\times sb}$
10:    $W_k = \mathcal{K}(A, \Omega_k^{\intercal} A) \in \mathbb{R}^{m\times sb}$
11:    **for** $j = 1, 2, \ldots, s$ **do**
12:        $\Psi = [e_{jb-b+1}, \ldots e_{jb}] \in \mathbb{R}^{sb\times b}$
13:        $U_{sk+j} = W_k \Psi$
14:        $\Theta_{sk+j} = \frac{1}{\lambda} V_{sk+j}^{\intercal} U_{sk+j} + mI$
```

$$\begin{aligned}\Delta\alpha_{sk+j} = \Theta_{sk+j}^{-1}\Bigg(& V_{sk+j}^{\intercal} y - m V_{sk+j}^{\intercal}\alpha_{sk} \\ & - m \sum_{t=1}^{j-1} V_{sk+j}^{\intercal} V_{sk+t}\Delta\alpha_{sk+t} \\ & - \frac{1}{\lambda} U_{sk+j}^{\intercal}\alpha_{sk} - \frac{1}{\lambda}\sum_{t=1}^{j-1} U_{sk+j}^{\intercal} V_{sk+t}\Delta\alpha_{sk+t}\Bigg)\end{aligned}$$

```
15:    **end for**
16:    $\alpha_{sk+s} = \alpha_{sk} + \sum_{t=1}^{s} V_{sk+t}\Delta\alpha_{sk+t}$
17: **end for**
18: **Output** $\alpha_H$
```

non-zeros per processor. The K-SVM and K-RR problems require non-linear kernel operations which are often more expensive than floating-point arithmetic. For example, the polynomial kernel requires a pointwise `pow` instruction for each entry of the sampled kernel matrix. Similarly, the RBF kernel requires an `exp` instruction for each entry. We model this overhead by introducing a scalar, μ, to represent the cost of applying a non-linear function relative to floating-point multiplies during the kernel computation. Note that due to the non-linear kernel function (particularly the RBF kernel), we store the $m \times b$ (sampled) kernel matrix in dense format. Finally, the BDCD and s-step BDCD algorithms require an MPI Allreduce call at each iteration. We assume that the Allreduce has a cost of $L = O(\log P)$ and $W = O(w)$ where w is the message size (in words) [15]. We begin by proving the parallel computation, communication, and storage costs of the BDCD algorithm (Theorem 1) followed by analysis of the s-step BDCD algorithm (Theorem 2).

Theorem 1. *Let H be the number of iterations of the Block Dual Coordinate Descent (BDCD) algorithm, b the block size, P the number of processors, and $A \in \mathbb{R}^{m\times n}$ the matrix that is partitioned using 1D-column layout. Under this setting, BDCD has the following asymptotic costs along the critical path:*

$$\begin{aligned}
\textbf{\textit{Computation:}}\ & \mathcal{O}\left(H\left(\frac{bfmn}{P} + \mu bm + b^3\right)\right)\ \textit{flops},\\
\textbf{\textit{Bandwidth:}}\ & \mathcal{O}\left(Hbm\right)\ \textit{words moved},\\
\textbf{\textit{Latency:}}\ & \mathcal{O}\left(H\log P\right)\ \textit{messages},\\
\textbf{\textit{Storage:}}\ & \mathcal{O}\left(bm + \frac{fmn}{P}\right)\ \textit{words of memory.}
\end{aligned}$$

Proof. Each iteration of parallel BDCD for K-RR begins by computing $U_k = \mathcal{K}(A, V_k^\intercal A)$, which costs at most $\frac{bfmn}{P}$ flops, in parallel, to form the $m \times b$ sampled columns, $AA^T V_k$, of the $m \times m$ full kernel matrix, AA^T. Each processor forms the $m \times b$ partial kernel matrix which must be sum-reduced before applying the non-linear kernel operation. This requires bm words to be moved and $\log P$ messages. After communication, the non-linear kernel operation can be applied to each entry of the kernel matrix using μbm flops. The quantity G_k can be extracted directly from U_k by selecting the b rows, corresponding to $V_k^\intercal$, sampled for this iteration. Since each processor redundantly stores y and α, the vector quantity $V_k^\intercal y - mV_k^\intercal - \frac{1}{\lambda}U_k^\intercal \alpha_{k-1}$ can be computed independently in parallel on each processor. This operation requires bm flops. Once the vector quantity is computed, the linear system can be solved in b^3 flops to compute $\Delta\alpha_k$ redundantly on each processor. Finally, α_k can be updated using b flops by updating only the coordinates of α sampled in this iteration. Summing and multiplying the above costs by H, the total number of BDCD iteration, proves the computational and communication costs. The storage costs can be obtained by noticing that each iteration of BDCD must simultaneously store A and U_k in memory, which requires $\frac{fmn}{P}$ words and bm respectively. Each iteration also requires G_k, which requires b^2 words of storage. However, storing G_k and other vector quantities are low-order terms in comparison to storing A and U_k. □

Theorem 2. *Let H be the number of iterations of the s-Step Block Dual Coordinate*

Name	Type	m	n
duke breast-cancer	binary classification	44	7129
diabetes	binary classification	768	8
abalone	regression	4177	8
bodyfat	regression	252	14

Table 2: Datasets used in the convergence experiments.

Descent (s-Step BDCD) algorithm, b the block size, P the number of processors, and $A \in \mathbb{R}^{m\times n}$ the matrix that is partitioned using 1D-column layout. Under this setting, s-Step BDCD has the following asymptotic costs along the critical path:

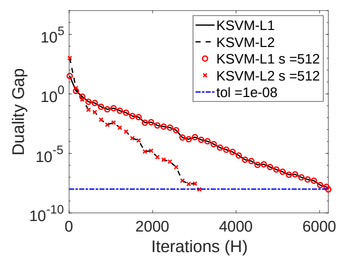
$$\textbf{\textit{Computation:}}\ \mathcal{O}\left(\frac{H}{s}\left(\frac{sbmfn}{P} + \mu sbm + sb^3 + \binom{s}{2}b^2\right)\right) \textit{ flops},$$

$$\textbf{\textit{Bandwidth:}}\ \mathcal{O}\left(\frac{H}{s}sbm\right) \textit{ words moved},$$

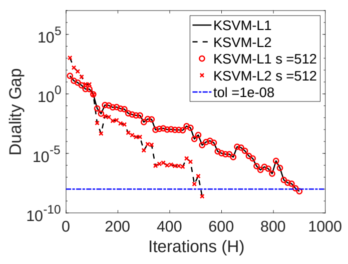
$$\textbf{\textit{Latency:}}\ \mathcal{O}\left(\frac{H}{s}\log P\right) \textit{ messages},$$

$$\textbf{\textit{Storage:}}\ \mathcal{O}\left(\frac{fmn}{P} + sbm\right) \textit{ words of memory}.$$

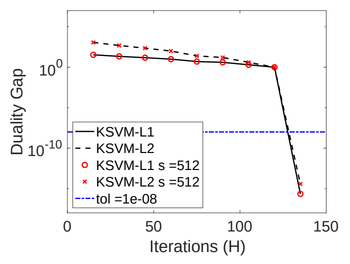
Proof. The s-Step BDCD algorithm for K-RR computes a factor of s larger kernel matrix, $Q_k = \mathcal{K}(A, \Omega_k A)$, which costs at most $\frac{sbfmn}{P}$ flops, in parallel, to partially form. The partial Q_k's on each processor must be sum-reduced prior to applying the non-linear kernel operation, which requires sbm words to be moved and $\log P$ messages. Once Q_k is sum-reduced the non-linear kernel computation can be performed redundantly on each processor. Since the non-linear operation is required for each entry of $Q_k \in \mathbb{R}^{m\times sb}$, forming the sampled kernel matrix costs μsbm flops. As with parallel BDCD, the $b \times b$ matrices G_{sk+j} for $j \in \{1, 2, \ldots, s\}$ can be extracted from Q_k and is a low-order cost. However, the s-Step BDCD algorithm requires additional matrix-vector computations to form the right-hand side of the linear system. Forming the right-hand side requires a total of s $U_{sk+j}^\intercal \alpha_{sk}$ matrix-vector computations which costs sbm flops and a sequence of matrix-vector computations, $U_{sk+j}^\intercal V_{sk+t}\Delta\alpha_{sk+t}$ for $t \in \{1, \ldots, j-1\}$, to correct the right-hand side. There are a total of $\binom{s}{2}$ such corrections with each requiring b^2 flops. Once the corrections have been performed, s linear systems can be solved using sb^3 flops. The leading-order cost of the inner loop (indexed by j) is $\binom{s}{2}b^2 + sbm + sb^3$ flops. Once the sequence of s $\Delta\alpha_{sk+j}$ vectors have been computed, α_{sk+s} can be computed by summing with appropriate indices of α_{sk}, which requires sb flops. The s-Step BDCD computes s solutions every outer iteration, so a total of H/s outer iterations are required to perform the equivalent of H BDCD iterations. Multiplying these costs by H/s proves the computation and communication costs. Finally, this algorithm requires storage of A, Q_k, G_{sk+j}, and several vector quantities. However, the leading order costs are the storage of A and Q_k which costs $\frac{fmn}{P} + sbm$ words. □



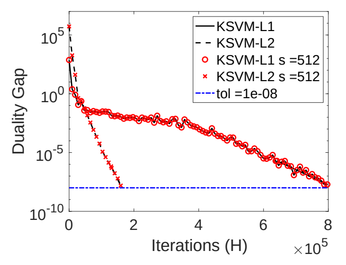
(a) duke, linear



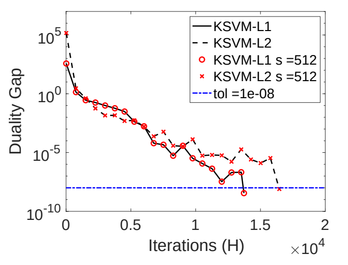
(b) duke, poly



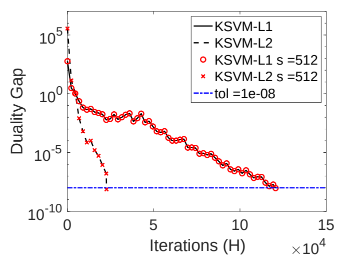
(c) duke, gauss



(d) diabetes, linear



(e) diabetes, poly



(f) diabetes, gauss

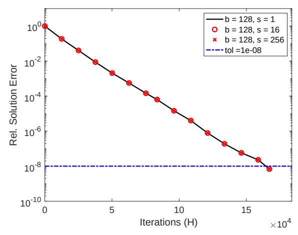
Figure 1: Comparison of DCD and s-step DCD convergence behavior for K-SVM-L1 and K-SVM-L2 problems.

5 Experiment

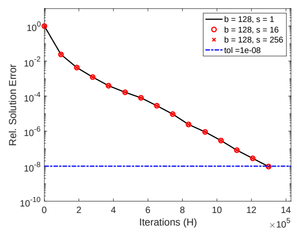
This section presents the numerical and performance experiments of the s-step DCD and BDCD algorithms for K-SVM and K-RR, respectively. Prior work on s-step Krylov methods [1, 5] showed that the additional computation led to numerical instability due to the Krylov basis becoming rank-deficient. The s-step DCD and BDCD algorithms require computation of factor of s larger kernel matrix and requires gradient correction due to the deferred update on α. We begin by studying the convergence behavior of the s-step methods relative to the classical DCD and BDCD methods on several binary classification and regression datasets. Then, we show the performance trade off and scaling behavior of the high-performance, distributed-memory implementations (written in C and MPI) of the s-Step DCD and BDCD methods relative to the standard DCD and BDCD methods on a Cray EX cluster.

5.1 Convergence Experiments

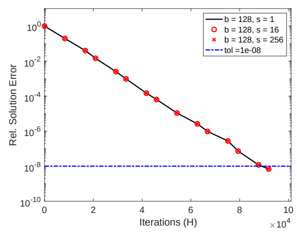
We measure the convergence behavior of the s-step methods against the classical methods and perform ablation studies by varying the values of s, kernel choice (shown in Table 1), and the block size (for the K-RR problem). The datasets used in the experiments were obtained from the LIBSVM repository [16] whose properties are shown in Table 2. All algorithms were implemented and tested in MATLAB R2022b Update 7 on a MacBook Air 2020 equipped with an Apple M1 chip. We measure the convergence of the K-SVM problem by plotting the duality gap for each setting of the s-step DCD method in comparison to the classical DCD method. Since K-SVM is a convex problem, we should expect the duality gap to approach machine precision. However, in these experiments we set the duality gap tolerance to 10^{-8}. We plot the duality gap over



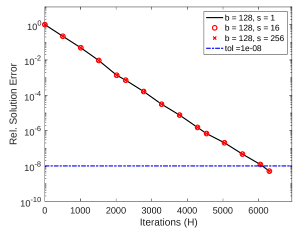
(a) abalone, linear



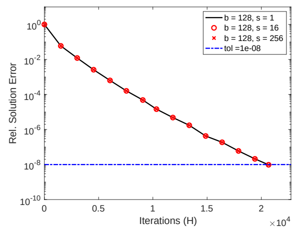
(b) abalone, poly



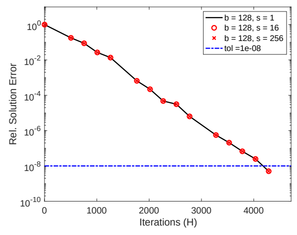
(c) abalone, gauss



(d) bodyfat, linear



(e) bodyfat, poly



(f) bodyfat, gauss

Figure 2: Comparison of BDCD and s-step BDCD convergence behavior for K-RR problem.

H iterations until the gap converges to the specified tolerance. Duality gap is defined as follows, $D(x) - P(x)$, where the $D(x)$ refers to the objective value of the K-SVM problem (Lagrangian dual problem) and $P(x)$ which refers to the objective value of the primal K-SVM problem as computed by LIBSVM [10, 16]. The primal and dual SVM problems are defined in (1) and (2), respectively. Note that we report convergence for both the K-SVM-L1 and (smoothed) K-SVM-L2 problems. Figure 1 shows the convergence behavior (in terms of duality gap) of the DCD and s-step DCD methods on the datasets in Table 2 and the kernels described in Table 1. We polynomial kernel utilizes a degree $d = 3$ and $c = 0$. The RBF kernel utilizes $\sigma = 1$. We can observe for both datasets and all kernels, that the s-step DCD methods for K-SVM-L1 and K-SVM-L2 (red markers) exhibit the same convergence behavior as the DCD methods and attain the same solution, α_H, up to machine precision. Note that we expect the convergence behavior of K-SVM-L1 and K-SVM-L2 to differ since they solve different problems and attain different solutions. We also perform experiments for the K-RR problem shown in (2). Since K-RR has a closed-form solution, we use relative solution error to illustrate convergence behavior of BDCD and s-step BDCD. The relative solution error is defined as follows, $\|\alpha_k - \alpha^*\|_2/\|\alpha^*\|_2$, where α_k is the partial solution at iteration k for BDCD (or iteration $sk+j$ for s-step BDCD) and α^* is the optimal solution obtained via matrix factorization. In order to compute α^* we first compute the full, $m \times m$ kernel matrix and solve the linear system for α^*. Figure 2 compares the convergence behavior of BDCD and CA-BDCD for $s < 1$ with all three kernels. Figure 2 shows the convergence behavior for the regression datasets in Table 2 for the three kernels in Table 1 for a large setting of b and s. Similar to K-SVM, we observe that s-step BDCD attains the same solution at BDCD for every iteration and converges to a relative error tolerance of 10^{-8} (single-precision accuracy). The abalone dataset is the largest MATLAB dataset tested, so we set the block size to $b = 128$. We report two settings for s to study the convergence behavior at small and large values, where $s = 16$ and $s = 256$, respectively.

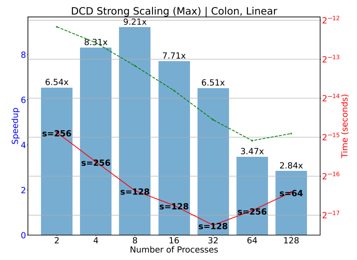
Dataset	m	n	nnz	Sparsity
colon-cancer	62	2,000	124000	0
duke breast-cancer	44	7,129	313676	0
synthetic	2,000	800,000	16,000,000	99%
news20.binary	19,996	1,355,191	909,7916	99.97%

Table 3: Datasets used in the performance experiments.

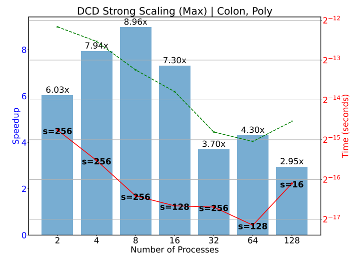
We use a smaller block size, $b = 64$, for the smaller bodyfat dataset and use the same settings for s. As Figure 2 illustrates, s-step BDCD is numerically stable even when $b \gg 1$. Furthermore, we can observe that the convergence of $s = 256$ and $s = 16$ (red markers) match for both datasets tested. Since K-RR, in particular, can utilize $b \gg 1$, we can expect the s-step BDCD method to achieve smaller performance gains over classical BDCD due to the bandwidth-latency trade off exhibited by the s-step methods. So we should expect the s-step methods to be numerically stable for smaller values of b. Figures 1 and 2 both illustrate that the s-step methods are numerically stable for the tested ranges and are practical for many machine learning applications. As a result, the maximum value of s depends on the computation, bandwidth, and latency trade off for a given dataset and hardware parameters of a candidate parallel cluster.

5.2 Performance Experiments

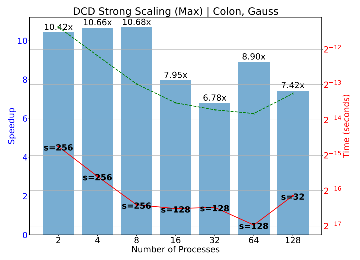
In this section, we study the performance of the s-step methods for K-SVM and K-RR problems. We implement all algorithms in C using Intel MKL for sparse and dense BLAS routines and MPI for distributed-memory parallel processing. We show performance results for each of the kernels shown in Table 1. The linear and polynomial kernels utilize the Intel MKL SparseBLAS library for sparse GEMM computations. The polynomial kernel with degree d requires additional elementwise operations on the output of the sparse GEMM. We compute the RBF kernel by using the definition of the dot-product to expand $\|a_{i,:} - a_{j,:}\|_2^2$ into a sparse GEMM computation. Finally, we use the $-O2$ compiler optimization flag additional performance optimization. The datasets for the performance experiments are shown in Table 5. We use datasets obtained from the LIBSVM repository and synthetic sparse dataset in order to study performance characteristics on benchmark machine learning datasets (which may not be load-balanced) and a perfectly load-balanced, sparse synthetic matrix. All datasets are processed in compressed sparse row (CSR) format. We use a Cray EX cluster to perform experiments. We run the DCD and BDCD methods for a fixed number of iterations and vary s for the s-step DCD and BDCD methods. We partition the dataset and store it in 1D-column layout (i.e. feature partitioning) so that each MPI process stored roughly n/P columns. Each CPU node contains two sockets equipped with AMD EPYC 7763 processors with each containing 64 physical cores. We did not see any benefits from utilizing simultaneous multi-threading, so we limit the number of MPI processes per node to 128 processes. We use MPI process binding and disable dynamic frequency scaling to ensure comparable performance as the number of MPI processes and number of nodes is varied.



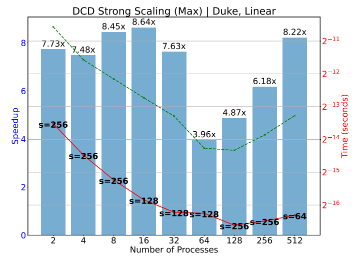
(a) colon cancer, linear



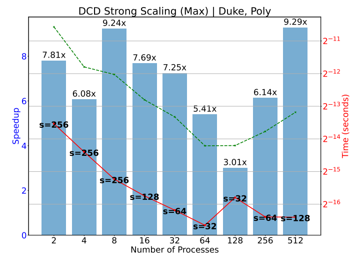
(b) colon cancer, polynomial



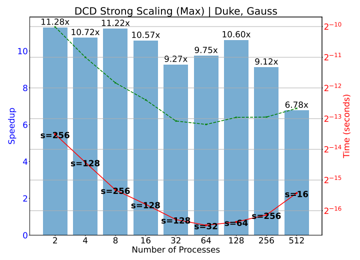
(c) colon cancer, RBF



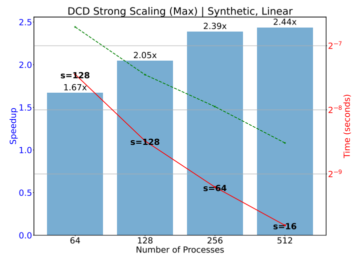
(d) duke, linear



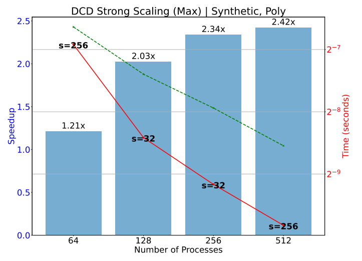
(e) duke, polynomial



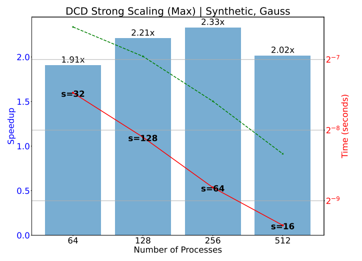
(f) duke, RBF



(g) synthtic, linear

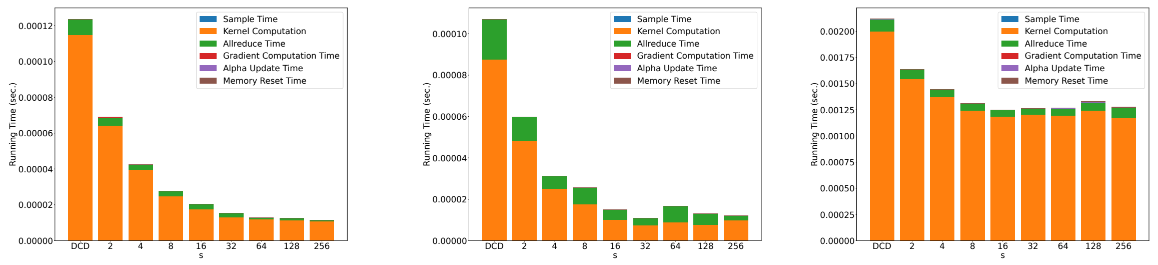


(h) synthetic, polynomial



(i) synthetic, RBF

Figure 3: Strong Scaling of DCD and s-step DCD for K-SVM.



(a) colon-cancer, P = 64, RBF (b) duke, P = 64, RBF (c) synthetic, P = 512, RBF

Figure 4: Running Time Breakdown of DCD and s-step DCD for values of P with fastest running times.

5.2.1 Strong Scaling

We explore the strong scaling behavior of the DCD and s-step DCD methods for K-SVM in Figure 3. We also present performance results for BDCD and s-step BDCD methods for K-RR, specifically, as the block size is varied. DCD is limited to $b = 1$ since SVM does not have closed-form solution, thus, we expect better scaling and performance. Given the computation, bandwidth, and latency trade off, we expect s-step BDCD to achieve reduced performance benefits as b is increased. We report running time and speedups with respect to the slowest processor. We performed offline tuning of s to obtain the setting which achieves the best running time. Note that we limit the values of s to powers of two, thus, additional performance may be attainable with fine-grained tuning of s. The small, colon cancer dataset exhibits scalability to $O(10)$ processors in Figure 3. The DCD method is latency bound, so we observe large speedups from the s-step DCD method. On the colon cancer dataset, the s-step DCD method attains speedups up to $3.5\times, 4.3\times$, and $8.9\times$ on the linear, polynomial, and RBF kernels, respectively. On the duke dataset, the s-step DCD method attains speedups up to $4.8\times, 5.4\times$, and $9.8\times$ on the linear, polynomial, and RBF kernels, respectively. Finally, the larger, synthetic dataset attains speedups up to $2.4\times, 2.4\times$, and $2\times$ on the linear, polynomial, and RBF kernels, respectively. Finally, the s-step BDCD method attains more modest speedups as the block size is increased for all kernels and all datasets, as illustrated in Table 4. We see decreasing benefits as the block size is increased for the colon-cancer and duke datasets. For the colon-cancer dataset, we observed speedups of up to $4.78\times$ at $b = 1$ and $1.7\times$ at $b = 4$. For the colon-cancer dataset, we observed speedups of up to $5.48\times$ at $b = 1$ and $1.68\times$ at $b = 4$.

5.2.2 Runtime Breakdown

Figure 4 presents the running time breakdown of the DCD and s-step DCD methods for the colon-cancer, duke, and synthetic datasets for various settings of s. We show results for the values of P that achieve the fastest strong scaling running time and show only the results for the RBF kernel. A notable observation in Figure 4 is the decrease in kernel computation time as we increase s. Since DCD is limited to a block size of 1, the kernel computation is limited to computing a single row of the kernel matrix at each iteration. In contrast, the s-step DCD method computes s rows of

Dataset	Kernel	Speedup		
		$b = 1$	$b = 2$	$b = 4$
colon-cancer	Linear	4.18 ×	3.63 ×	1.86 ×
	Polynomial	4.08 ×	2.41 ×	1.71 ×
	Gauss	4.78 ×	3.63 ×	2.48 ×
duke	Linear	5.48 ×	3.50 ×	2.61 ×
	Polynomial	4.08 ×	2.41 ×	1.71 ×
	Gauss	3.59 ×	2.54 ×	1.68 ×
news20.binary	Linear	2.03 ×	1.32 ×	1.11 ×
	Polynomial	1.74 ×	1.16 ×	1.09 ×
	Gauss	1.63 ×	1.17 ×	1.11 ×

Table 4: Speedups attained by s-step BDCD over BDCD for solving the K-RR problem for different block sizes, b.

the kernel matrix every (outer) iteration. As a result, the SparseBLAS routines have better single-node memory-bandwidth utilization (in addition to decreasing latency cost). Figure 4 also shows a decrease in MPI allreduce time, which is expected when latency cost dominates. However, for the synthetic dataset when $s > 16$, we observe that the allreduce time increases. Note that this increase in communication time is also expected as the bandwidth term begins to dominate. Thus, the value of s must be tuned carefully to achieve the best performance. The s-step methods require the same total bandwidth as the classical methods, but the per message bandwidth increases by a factor of s. For the colon-cancer and duke datasets, we see significant improvements in kernel computation and allreduce times with $s = 256$ and $s = 32$ being the optimal setting for colon-cancer and duke, respectively. For the synthetic dataset, we see a smaller factor of improvement in kernel computation and allreduce times. This suggests that the synthetic dataset can be strong scaled further before the DCD runtime becomes dominated by DRAM/network communication.

6 Coordinate Descent Derivation

The application of the Coordinate Descent algorithm for solving Generalized Linear Model (GLM) problems is comprehensively specified in [17]. However, most existing derivations primarily focus on least-squares regression problems with an elastic net penalty term, with less attention given to the nuances of the Coordinate Descent algorithm in solving regularized least squares problems. In the subsequent section, I will elaborate on the derivation of the Coordinate Descent algorithm specifically for regularized logistic regression. This will include both naive and covariance update strategies. Additionally, I will provide a detailed derivation for one variant of the s-Step Coordinate Descent updates.

6.1 Regularized Logistic Regression

In this section, we introduce the Regularized Logistic Regression (RLog) and Regularized Multinomial Regression (RMR) problems. We present the coordinate descent

(CD) algorithm used to solve these problem and, finally, we perform s-step derivation of CD for each problem.

We have a response variable $y \in \{0,1\}^m$ and a sparse data vector $\mathbf{a_j} = A \cdot e_j^T \in \mathbb{R}^n$, where n represents the number of features and $\mathbf{a_i}$ represents one individual coordinate selected from the m observations. The logistic regression problem has the following structure, aiming to solve a binary classification problem with classes $K = 1, 2$. The logistic regression model, often represented using the sigmoid function, is given as:

$$\sigma(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{z}}} \tag{4}$$

where $\mathbf{z} = \mathbf{a_j}\beta$ is the linear combination of the features $\mathbf{a_j}$ and the regression coefficients β.

$$Sig(z) = \frac{1}{1 + e^{-z}}$$

The binary logistic regression problem is defined as:

$$\log\left(\frac{\sigma(z_i)}{1 - \sigma(z_i)}\right) = \sum_{j=0}^{n} a_{ij}\beta_j, \quad i = 1, \ldots, m. \tag{5}$$

where $\sigma(z_i)$ is the probability for the observation i to be in class $K = 1$ is given as

$$\sigma(z_i) = \frac{1}{1 + e^{-\mathbf{a_i}\beta}}, \tag{6}$$

Alternatively, the $\sigma(z_i)$ is complementary to the chance of i to be in class $K = 2$ for binary problems. Therefore, the probability of observation i to be in class $K = 2$ was given as

$$1 - \sigma(z_i) = \frac{1}{1 + e^{\mathbf{a_i}\beta}} \tag{7}$$

The objective of the binary logistic regression is to find a set of coefficients β that assign most accurate estimates of probabilities for each observation to correct class. The optimization problem to find the optimal β is as follows,

$$\ell(\beta) = \frac{1}{n}\sum_{i=1}^{N}\left[y_i \cdot \mathbf{a_i}\beta - \log(1 + e^{\mathbf{a_i}\beta})\right]. \tag{8}$$

This convex problem could be solved by the Newton method through maximizing the log-likelihood of the above formula through iterative re-weighted least squares [17]. Hence if the current estimates of the parameters are β, we form a quadratic approximation to the log-likelihood (Taylor expansion about current estimates), which is as mentioned in [17]

$$\ell(\beta) = -\frac{1}{2n}\sum_{i=1}^{N} w_i(z_i - \mathbf{a_i}\beta)^2 \tag{9}$$

where,

$$w_i = \sigma(z_i)(1-\sigma(z_i)) \qquad \text{(weights)} \tag{10}$$

$$z_i = \mathbf{a_i}\beta + \frac{y_i - \sigma(z_i)}{\sigma(z_i)(1-\sigma(z_i))} \qquad \text{(working response)} \tag{11}$$

$\sigma(z_i)$ is evaluated at the current parameters with the sigmoid function, which represents the probability of the current observation at $K = 1$ class given its features. Therefore, by incorporating the least-square constraint $\lambda P_\alpha(\beta)$, we have obtained the penalized weighted least-squares problem ℓ_Q to be minimized below, which is solvable with the coordinate descent.

$$\ell_Q(\beta, \alpha, \lambda) = \min_{\beta\in\mathbb{R}^n} \left\{-\ell(\beta) + \lambda P_\alpha(\beta)\right\}. \tag{12}$$

where

$$P_\alpha(\beta) = (1-\alpha)\frac{1}{2}\|\beta\|_{l_2}^2 + \alpha\|\beta\|_{l_1} \tag{13}$$

It is obvious that the optimal solution of the concave optimization problem above could be obtained at the single minimum point at zero derivative as other least square problems. Consequently, by iterative optimizing the partial derivative in respect to each β_j value selected, the convergence was obtained.

$$\left.\frac{\partial \ell_Q(\beta,\alpha,\lambda)}{\partial \beta_j}\right|_{\beta_j=\beta_j} = -\frac{1}{m}\sum_{i=1}^{m} w_i a_{ij}(z_i - \mathbf{a_i}\beta^{(j)}) + \lambda(1-\alpha)\beta_j + \lambda\alpha. \tag{14}$$

Where $\beta^{(j)}$ indicates the β without the j-th feature. All the derivative was done in according to the current β values. In this way, we have obtained the gradient of selected coefficient value of the j-th feature, we denote the updated coefficient as $\hat{\beta}_j$

$$\hat{\beta}_j \leftarrow \frac{S\left(\frac{1}{m}\sum_{i=1}^{m} w_i a_{ij}(z_i - \mathbf{a_i}\beta^{(j)}), \lambda\alpha\right)}{\frac{1}{m}\sum_{i=1}^{m} w_i a_{ij}^2 + \lambda(1-\alpha)}. \tag{15}$$

where $S(\omega, \gamma)$ is the soft-thresholding operator with value

$$\operatorname{sign}(\omega)(|\omega| - \gamma)_+ = \begin{cases} \omega - \gamma & \text{if } \omega > 0 \text{ and } \gamma < |\omega| \\ \omega + \gamma & \text{if } \omega < 0 \text{ and } \gamma < |\omega| \\ 0 & \text{if } \gamma \geq |\omega|. \end{cases} \tag{16}$$

6.1.1 Naive Updates

The $\beta^{(j)}$ requires the computation of the dot product with data matrix without the j-th feature considered, which can be inefficient when performing matrix operation for problem solving. Fortunately, the gradient computation could be reformulated to a matrix-friendly format for updating, which is called naive updates [17]. In this scenario, we have

$$\begin{aligned} z_i - \mathbf{a_i}\beta^{(j)} &= z_i - \mathbf{a_i}\beta + a_{ij}\beta_j && (17) \\ &= r_i + a_{ij}\beta_j && (18) \end{aligned}$$

In this way, the weighted sum could be rewritten as

$$\begin{aligned} g &= \frac{1}{m}\sum_{i=1}^{m} w_i a_{ij}(z_i - \mathbf{a_i}\beta^{(j)}) && (19) \\ &= \frac{1}{m}\sum_{i=1}^{m} w_i a_{ij}(z_i - \mathbf{a_i}\beta + a_{ij}\beta_j) && (20) \end{aligned}$$

or in matrix format equivalently,

$$g = \frac{1}{m}\left[(A \cdot e_j^T) \cdot W(Z - A\beta)\right] + \frac{1}{m}\left[(A \cdot e_j^T)^2 \cdot W\right]\beta_j \tag{21}$$

Where:

- $(A \cdot e_j^T)^2$ is the element-wise square of the j-th column of A.
- Z denoting the working response z_i for every observation

6.1.2 Covariance Updates

The covariance updates is generally more efficient than the naive updates. According to the experiment result of the referred paper [17], covariance updates are generally computationally efficient for large number of samples in comparison to the feature numbers. It is achieved by further unfolding the residual r_i from the naive updates. The general formula of covariance update was as following.

$$\sum_{i=1}^{m} a_{ij} r_i = \langle \mathbf{a_j}, y\rangle - \sum_{k=1}^{n}\langle \mathbf{a_j}, \mathbf{a_k}\rangle\beta_k, \tag{22}$$

where $\langle \mathbf{a_j}, y\rangle = \sum_{i=1}^{m} a_{ij} y_i$. The first term is computed by the inner product of the label with each feature of the dataset, which can be pre-computed and stored at each process before the loop.

Thus, we have the covariance updates as following:

$$g = \frac{1}{m}\sum_{i=1}^{m} a_{ij}y_i - \frac{1}{m}\sum_{i=1}^{m} a_{ij}\sigma(z_i) + \frac{\beta_j}{m}\sum_{i=1}^{m} w_i a_{ij}^2 \tag{23}$$

or in matrix format equivalently,

$$g = \frac{1}{m}{A_j}^T y - \frac{1}{m}A_j^T sig(A\cdot\beta) + \frac{\beta_j}{m}W^T A_j^2 \tag{24}$$

Algorithm 5 Pathwise Coordinate Descent for Regularized Regression

```
1: **Input:** $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m, \alpha \in \mathbb{R}, \lambda \in \mathbb{R}, H > 1$
2: Let $\lambda$ be a sequence of values given by the user
3: $\tilde{A} = y^T \cdot A$
4: $\rho = \lambda \times \alpha$
5: while $\lambda = \lambda_{\max}$ to $\lambda_{\min}$ do
6:     Initialize $\beta_0 \leftarrow 0$
7:     for $k = 0, 1, \ldots, H$ do
8:         Choose $j_k \in [n]$ uniformly at random.
9:         $e_{j_k} \in \mathbb{R}^n$, the $j_k$-th standard basis vector.
10:        $\mathcal{G}_k = A \cdot \beta_k$
11:        $W_k = sig(\mathcal{G}_k) \otimes (\vec{1} \ominus sig(\mathcal{G}_k))$
12:        $g = \frac{1}{m}\tilde{A}\cdot e_{j_k}^T - \frac{1}{m}A\cdot e_{j_k}^T sig(\mathcal{G}_k) + \frac{\beta_{k_{j_k}}}{m}W_k^T(A\cdot e_{j_k}^T)^2$
13:        $\omega_k = \frac{1}{m}W_k^T(A\cdot e_{j_k}^T)^2$
14:        if $g_k > \rho$ then
15:            $\Delta\beta_{j_k} = \frac{g_k-\rho}{\omega_k+\lambda(1-\alpha)} - \beta_{k_{j_k}}$
16:        else if $g_k > \rho$ then
17:            $\Delta\beta_{j_k} = \frac{g_k+\rho}{\omega_k+\lambda(1-\alpha)} - \beta_{k_{j_k}}$
18:        else
19:            $\Delta\beta_{j_k} = -\beta_{k_{j_k}}$
20:        end if
21:        $\beta_{k+1} = \beta_k + \Delta\beta_k e_{j_k}$
22:    end for
23:    Store the solution for $\beta$ at the current $\lambda$
24: end while
25: For $\beta$ value obtained across $\lambda_{min} \ldots \lambda_{max}$, select the $\beta$ value that gives result with
    smallest mean squared error and assign as $\beta_{opt}$
26: **Output:** $\beta_{opt}$
```

6.2 s-Step Regularized Logistic Regression

The s-step method could be adapted in this algorithm to reduce the communication time in parallel setting. In this section, I will derive the s-step method for both the naive updates and the covariance updates.

The s-step method communicate every s iteration, in which we need to computes the weight vector W and working response U locally in the inner loop. For both the naive and covariance updates, the weight vector is computed for every iteration. Let $k = 1, ..., \frac{H}{s}$ be the current iteration of outer loop, and $t = 1, ..., s$ denotes the current iteration inside the inner loop for s.

$$W_{sk} = sig(A \cdot \beta_{sk}) \otimes (\vec{1} \ominus sig(A \cdot \beta_{sk})) \tag{25}$$

where the β_{sk} vector is updated for every iteration of t. The new β_{sk+1} is obtained with $\beta_{sk} + e_{j_{sk}} \Delta\beta_{sk}$ where $e_{j_{sk}}$ is the j-th basis vector and $\Delta\beta_j$ is the gradient of j-th β entry for updates, which is

$$\Delta\beta_{sk+1} = \beta_{sk} - S(g_{sk}, \lambda\alpha) \tag{26}$$

Therefore, at iteration $sk + t$, the current weight vector W can be unfolded as:

$$W_{sk+t} = sig(A \cdot (\beta_{sk} + \sum_{l=1}^{t} e_{j_{sk+l}} \Delta\beta_{sk+l})) \otimes (\vec{1} \ominus sig(A \cdot (\beta_{sk} + \sum_{l=1}^{t} e_{j_{sk+l}} \Delta\beta_{sk+l}))) \tag{27}$$

In the equations above, the operators are defined as follows:

- The operator $\otimes$ denotes element-wise multiplication.
- The operator $\ominus$ denotes element-wise subtraction.
- The operator $\oslash$ denotes element-wise division.

The matrix A is partitioned across processes, and, in the setting of s-step, we can then compute the $A \cdot \beta_{sk+s}$, which effectively reduce the number of communications.

Similarly, the working response U vector could also be unfolded at $sk + t$ iteration. Given the U's equation of updates,

$$U_{sk} = A_{:sk}^{T} \cdot \beta_{sk} + (b - sig(A \cdot (\beta_{sk}))) \oslash W_{sk} \tag{28}$$

At iteration $sk + t$, it was expressed as following, which can then be expanded by the iterative updates

$$\begin{aligned} U_{sk+t} &= \left((A \cdot e_{sk+t}^{T})^{T} \cdot \beta_{sk+t}\right) + (b - \mathrm{sig}(A \cdot \beta_{sk+t})) \oslash W_{sk+t} && (29) \\ &= \left((A \cdot e_{sk+t}^{T})^{T} \cdot (\beta_{sk} + \sum_{l=1}^{t} e_{j_{sk+l}} \Delta\beta_{sk+l})\right) + && (30) \\ &\quad \left(b - \mathrm{sig}(A \cdot (\beta_{sk} + \sum_{l=1}^{t} e_{j_{sk+l}} \Delta\beta_{sk+l}))\right) \oslash W_{sk+t} && (31) \end{aligned}$$

6.2.1 Naive Updates

Similarly, the naive updates of g value can also be unfolded in terms of s iterations. Consider the initial setting of g_{sk},

$$g_{sk} = \frac{1}{m}\left[(A \cdot e_j^T) \otimes W_{sk}(U_{sk} - A \cdot \beta_{sk})\right] + \frac{1}{m}\left[(A \cdot e_j^T)^2 \otimes W_{sk}\right]\beta_{sk} \tag{32}$$

at $sk + t$ iteration, the $g_{j_{sk+t}}$ has the formula as below,

$$g_{sk+t} = \frac{1}{m}\left[(A \cdot e_{sk+t}^T) \otimes W_{sk+t} \cdot (U_{sk+t} - A \cdot \beta_{sk+t})\right] + \frac{1}{m}\left[(A \cdot e_{sk+t}^T)^2 \otimes W_{sk+t}\right] \cdot \beta_{sk+t} \tag{33}$$

It is worth mentioning that the W_{sk+t} and U_{sk+t} is unfolded as previous derived equations

$$g_{sk+t} = \frac{1}{m}\left[(A \cdot e_{sk+t}^T) \otimes W_{sk+t} \cdot (U_{sk+t} - A \cdot \beta_{sk+t})\right] + \frac{1}{m}\left[(A \cdot e_{sk+t}^T)^2 \otimes W_{sk+t}\right] \cdot \beta_{sk+t} \tag{34}$$

6.2.2 Covariance Updates

The covariance updates has the matrix formula of

$$g_{sk} = \frac{1}{m}\left((A \cdot e_{sk}^T)^T y - \frac{1}{m}A_{:sk}^T \cdot \mathrm{sig}(A \cdot \beta_{sk}) + \frac{\beta_j}{m} W_{sk}^T A_{:sk}^2\right) \tag{35}$$

at $sk + t$ iteration, the $g_{j_{sk+t}}$ has the formula as below,

$$g_{sk+t} = \frac{1}{m}\left((A \cdot e_{sk+t}^T)^T y - \frac{1}{m}(A \cdot e_{sk+t}^T)^T \cdot \mathrm{sig}(A \cdot \beta_{sk+t}) + \frac{\beta_{sk+t}}{m} W_{sk+t}^T \cdot (A \cdot e_{sk+t}^T)^2\right) \tag{36}$$

the W_{sk+t} and β_{sk+t} are expandable according to equation (22) and (23) in the previous section,

$$g_{sk+t} = \frac{1}{m}\left((A \cdot e_{sk+t}^T)^T y\right) - \frac{1}{m}\,\mathrm{sig}\left(A \cdot \left(\beta_{sk} + \sum_{l=1}^{t} e_{j_{sk+l}} \Delta\beta_{sk+l}\right)\right) + \frac{\beta_{sk+t}}{m}\left(W_{sk+t}^T \cdot (A \cdot e_{sk+t}^T)^2\right) \tag{37}$$

It is worth mentioning that the computation of W vector is not linearly separable, but we could achieve more efficient computing by incrementing the inner product of $A \cdot \beta$ iteratively, which can be computed in parallel.

Algorithm 6 s-Step Coordinate Descent for Regularized Regression

```
1: Input: $A \in \mathbb{R}^{m\times n}, y \in \mathbb{R}^m, \alpha \in \mathbb{R}, \lambda \in \mathbb{R}, s \in \mathbb{R}, H > 1$
2: $\tilde{A} = y^T \cdot A$
3: $\rho = \lambda \times \alpha$
4: while $\lambda = \lambda_{\max}$ to $\lambda_{\min}$ do
5:     Initialize $\beta_0 \leftarrow 0$
6:     for $k = 0, \dots, H/s$ do
7:         $\mathcal{G}_k = A \cdot \beta_k$
8:         for $t = 1, \dots, s$ do
9:             $j_{sk+t} \in [n]$, chosen uniformly at random.
10:            $e_{j_{sk+t}} \in \mathbb{R}^n$, the $j_{sk+t}$-th standard basis vector.
11:        end for
12:        $J_k = \left[e_{j_{sk+1}}, \dots, e_{j_{sk+s}}\right]$
13:        $\tilde{A}_k = \tilde{A} \cdot {J_k}^T$
14:        $U_k = A \cdot {J_k}^T$
15:        for $t = 0, \dots, s$ do
16:            $\mathcal{G}_{sk+t} = \mathcal{G}_{sk} + \sum_{h=1}^{t} U_{sk+h} \Delta\beta_{sk+h}$
17:            $W_{sk+t} = sig(\mathcal{G}_{sk+t}) \otimes (\vec{1} \ominus sig(\mathcal{G}_{sk+t})$
18:            $g_{sk+t} = \frac{1}{m}\tilde{A}_{sk+t} - \frac{1}{m}U^T_{sk+t} sig(\mathcal{G}_{sk+t}) + \frac{\beta_{j_{sk+t}}}{m} W^T_{sk+t} U^2_{sk+t}$
19:            $\omega_{sk+j} = \frac{1}{m} W^T_{sk+j} U^2_{sk+t}$
20:            if $g_{sk+t} > \rho$ then
21:                $\Delta\beta_{j_{sk+t}} = \frac{g_{sk+t}-\rho}{\omega_{sk+t}+\lambda(1-\alpha)} - \beta_{j_{sk+t}}$
22:            else if $g_{sk+t} > \rho$ then
23:                $\Delta\beta_{j_{sk+t}} = \frac{g_{sk+t}+\rho}{\omega_{sk+t}+\lambda(1-\alpha)} - \beta_{j_{sk+t}}$
24:            else
25:                $\Delta\beta_{j_{sk+t}} = -\beta_{j_{sk+t}}$
26:            end if
27:            $\beta_{sk+t} = \beta_{sk} + \sum_{h=1}^{t} \Delta\beta_{j_{sk+t}} e_{j_{sk+t}}$
28:        end for
29:    end for
30:    Store the solution for $\beta$ at the current $\lambda$
31: end while
32: For $\beta$ value obtained across $\lambda_{min} \dots \lambda_{max}$, select the $\beta$ value that gives result with
    smallest mean squared error and assign as $\beta_{opt}$
33: Output: $\beta_{opt}$
```

6.3 Regularized Multinomial Regression

The regularized multinomial regression extends the problem of regularized logistic regression from binary to multi-logit model. Given a dataset of $K > 2$ classes, we have the model as below and the solution of the multi-nomial problem is a set of β_ℓ for each class of values.

$$\Pr(G = \ell|\mathbf{a_i}) = \frac{e^{\mathbf{a_i}\beta_\ell}}{\sum_{k=1}^{K} e^{\mathbf{a_i}\beta_k}}, \quad \text{where } \ell = 1, \dots, K-1. \tag{38}$$

It is worth mentioning that the multi-class case could be computed with one-versus-all strategy. For instance, suppose that the $K = 1$ is the based case to be optimized, we will compare the $K = 1$ case against all other cases so we can solve the problem as a binary logistic regression problem. However, it is worth mentioning that the balances between class cases is crucial for convergence of the algorithm. If there is limited sample of certain class, β may fail to converge.

Algorithm 7 Coordinate Descent for Regularized Multinomial Regression

```
 1: **Input:** $A \in \mathbb{R}^{m \times n}, y \in \mathbb{R}^m, \alpha \in \mathbb{R}, \lambda \in \mathbb{R}, H > 1$
 2: **while** l = 1, 2, ... K **do**
 3:     Create a new binary label vector $b_l \in \mathbb{R}^m$, such that:
 4:     **for** $i = 0$ **to** $m - 1$ **do**
 5:         **if** $y_{l_i} = l$ **then**
 6:             $y_{l_i} = 0$
 7:         **else**
 8:             $y_{l_i} = 1$
 9:         **end if**
10:     **end for**
11:     Update according to coordinate descent algorithm for binary logistic regression:
        Algorithm 5
12: **end while**
13: For $\beta$ value obtained across $\lambda_{min} \dots \lambda_{max}$, select the $\beta$ value that gives result with
    smallest mean squared error and assign as $\beta_{opt}$
14: **Output:** $\beta_{opt}$
```

6.4 s-Step Regularized Multinomial Regression

Similarly, we can add an outer while loop and doing s-Step coordinate descent algorithm on binary logistic regression problem.

Name	Type	m	n
australian	binary	690	14
ionosphere-scale	binary	351	34
splice-scale	binary	1000	60
synthetic-multiclass-dataset	multi-class	2000	16

Table 5: Datasets used in the convergence experiments.

Algorithm 8 Coordinate Descent for Regularized Multinomial Regression

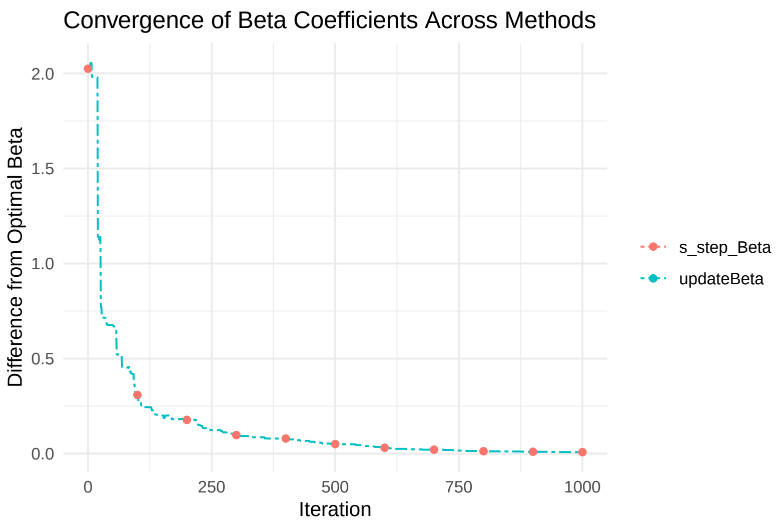
```
1: Input: $A \in \mathbb{R}^{m \times n}, y \in \mathbb{R}^m, \alpha \in \mathbb{R}, \lambda \in \mathbb{R}, s \in \mathbb{R}, H > 1$
2: while l = 1, 2, ... K do
3:     Create a new binary label vector $y_l \in \mathbb{R}^m$, such that:
4:     for $i = 0$ to $m - 1$ do
5:         if $y_{l_i} = l$ then
6:             $y_{l_i} = 0$
7:         else
8:             $y_{l_i} = 1$
9:         end if
10:    end for
11:    Update according to coordinate descent algorithm for binary logistic regression:
       Algorithm 6
12: end while
13: For $\beta$ value obtained across $\lambda_{min} \dots \lambda_{max}$, select the $\beta$ value that gives result with
    smallest mean squared error and assign as $\beta_{opt}$
14: Output: $\beta_{opt}$
```

7 Numerical Experiments

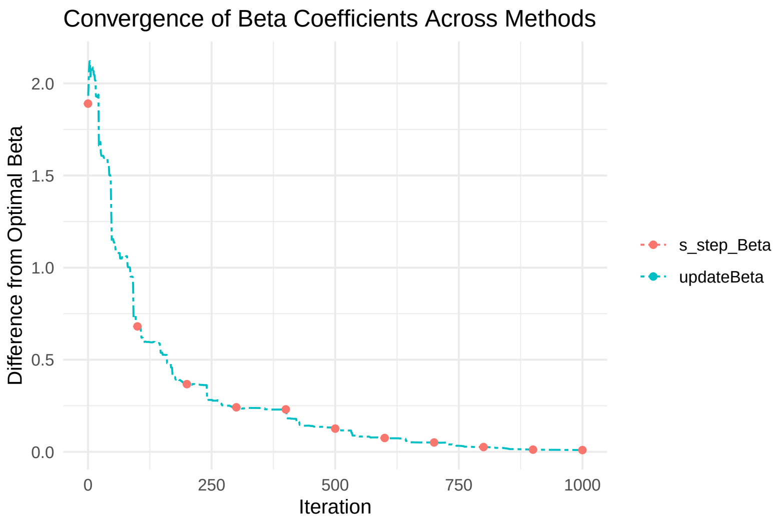
7.1 Coordinate Descent Convergence Experiments

This section presents the numerical and performance experiments of the CD and s-step CD algorithm for logistic regression. The convergence of an algorithm is a critical factor that determines its reliability and effectiveness in solving optimization problems. Convergence, by definition, guarantees that the algorithm will reach an optimal solution within a finite number of steps, provided certain conditions are met. Previous studies on s-step Krylov methods [1, 5] indicated that additional computations might lead to numerical instability due to the Krylov basis becoming rank-deficient. The s-step CD and BCD algorithms involve computations with a factor of s larger matrix and require gradient correction owing to the deferred update on the parameter β. This thesis examines the convergence properties of our proposed s-Step methods, ensuring they behave as expected under various conditions comparing with unoptimized algorithms.

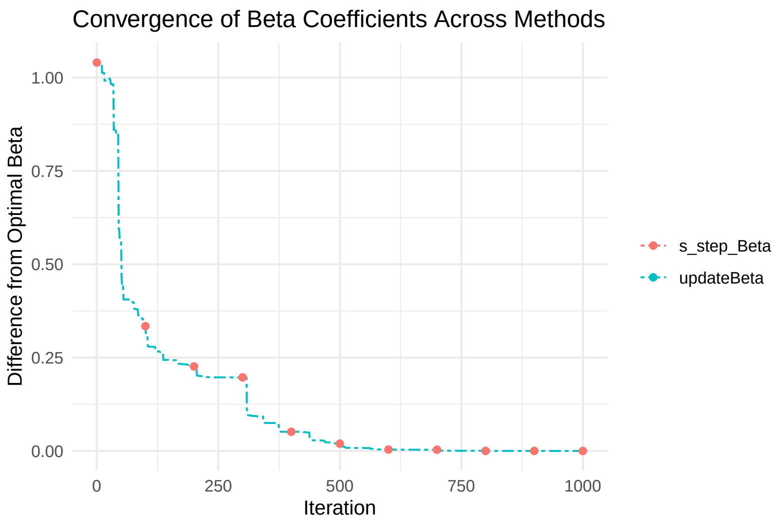
We initiate our examination by evaluating the convergence behavior of the s-step methods in comparison to the classical CD and BCD methods across various binary classification datasets.



(a) australian



(b) ionosphere-scale



(c) splice-scale

Figure 5: Comparison of CD and *s*-step CD convergence behavior for Regularized Logistic Regression problems.

We measure the convergence behavior of the s-step methods against the classical methods by varying the values of s and the dataset sizes. The datasets used in the experiments were sourced from the LIBSVM repository [16], whose properties are outlined in Table 2. All algorithms were implemented and evaluated in R version 4.2.1 (2022-06-23) on a MacBook Air 2020 equipped with an Apple M1 chip. We quantify the convergence of the logistic regression problem by plotting the mean squared difference between the optimal β value and the β value at each iteration for each setting of the s-step CD method compared to the classical CD method. Logistic regression, being a convex problem, should see this difference approach zero. The mean squared difference is defined as follows: $\|\beta_k - \beta^*\|^2$, where β_k is the beta value at iteration k for CD (or iteration $sk + j$ for s-step CD), and β^* is the optimal beta value obtained via logistic regression's optimization process.

Figure 5 shows the convergence behavior (in terms of mean squared difference) of the CD and s-step CD methods on the datasets in Table 2. The convergence behavior observed suggests that both the s-step CD and classical CD methods eventually achieve the same solution, β_opt, up to the precision limits of the computational hardware. The experiments indicate that the s-step methods are numerically stable for the tested ranges and are feasible for many machine learning applications involving logistic regression.

8 Conclusion

We demonstrated that the s-step DCD and s-step BDCD methods for KSVM and KRR, respectively, attain large speedups when latency is the dominant cost. We show that this conclusion holds for dense and sparse datasets as well as datasets with non-uniform nonzero distributions that lead to load imbalance. We also show that the performance benefits of the s-step methods are moderate as allreduce bandwidth becomes the dominant cost. This observation underscores the importance of dataset characteristics and machine balance in determining the performance of the proposed methods in high-performance computing environments. Our research extends prior work on s-step methods to kernelized machine learning models for classification and regression. We show that in contrast to prior s-step coordinate descent and stochastic gradient descent methods, the kernel methods do not increase the total communication bandwidth (in theory) and attain speedups for a greater range of values of s.

However, the s-step methods to coordinate descent algorithms indicates an increasing communication cost in computation. The joint non-linearity of the sigmoid function and the soft-thresholding makes the computation linearly non-separable. In this case, the communication over the locally stored dataset will lead to increase in synchronization cost. The alternative expansion of the π term with taylor expansion of indicates numerical instable solution of the β vector, in which the coefficient explodes.

In the future, we plan to further optimize the s-step methods' kernel computation and gradient correction overheads by approximating the sampled kernel matrix (for example using the Nyström method). This performance optimization would enable the s-step method to scale to larger block sizes at the expense of weaker convergence. We also aim to study the performance characteristics of the proposed methods in distributed environments (e.g. federated or cloud environments) where network latency costs are more prohibitive and where s-step methods may yield impactful performance improvements.

References

[1] Mark Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California, Berkeley, 2010.

[2] James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. Avoiding communication in sparse matrix computations. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, Miami, FL, USA, 2008. IEEE.

[3] Anthony T. Chronopoulos and C. William Gear. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel computing*, 11(1):37–53, 1989. Citation Key: chronopoulos89.

[4] SK Kim and AT Chronopoulos. An efficient nonsymmetric lanczos method on parallel vector computers. *Journal of Computational and Applied Mathematics*, 42(3):357–374, 1992.

[5] Erin Carson. *Communication-avoiding Krylov subspace methods in theory and practice*. PhD thesis, EECS Department, University of California, Berkeley, August 2015.

[6] E. Carson, N. Knight, and J. Demmel. Avoiding communication in nonsymmetric lanczos-based krylov subspace methods. *SIAM Journal on Scientific Computing*, 35(5):S42–S61, 2013.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, page 144–152, Pittsburgh Pennsylvania USA, July 1992. ACM.

[8] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, Burlington, MA, USA, 1992. Morgan-Kaufmann.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[10] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 408–415, New York, NY, USA, 2008. Association for Computing Machinery.

[11] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft, April 1998.

[12] Yang You, Xiangru Lian, Ji Liu, Hsiang-Fu Yu, Inderjit S Dhillon, James Demmel, and Cho-Jui Hsieh. Asynchronous parallel greedy coordinate descent. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, Red Hook, NY, USA, 2016. Curran Associates, Inc.

[13] George Kimeldorf and Grace Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

[14] Roger W. Hockney. The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel Computing*, 20(3):389–398, 1994.

[15] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19(1):49–66, February 2005.

[16] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[17] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[18] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[19] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.

[20] Aditya Devarakonda, Kimon Fountoulakis, James Demmel, and Michael Mahoney. Avoiding communication in primal and dual block coordinate descent methods. *SIAM Journal on Scientific Computing*, 41, 12 2016.

[21] A. T. Chronopoulos. s-step iterative methods for (non)symmetric (in)definite linear systems. *SIAM Journal on Numerical Analysis*, 28(6):1776–1789, 1991.

[22] Aditya Devarakonda and James Demmel. Avoiding communication in logistic regression. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 91–100, Piscataway, NJ, USA, 2020. IEEE.

[23] A. Devarakonda. *Avoiding Communication in First Order Methods for Optimization.* PhD thesis, UC Berkeley, 2018. ProQuest ID: Devarakonda_berkeley_0028E_18070. Merritt ID: ark:/13030/m56b2136.

[24] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14:567–599, 2013.

[25] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild! a lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 693–701, Red Hook, NY, USA, 2011. Curran Associates Inc.

[26] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3068–3076, Cambridge, MA, USA, 2014. MIT Press.

[27] Virginia Smith, Simone Forte, Michael I. Jordan, and Martin Jaggi. L1-regularized distributed optimization: A communication-efficient primal-dual framework. *CoRR*, abs/1512.04011:1–23, 2015.

[28] Dizhou Wu and Freddie R. Jr. Salsbury. Unraveling the role of hydrogen bonds in thrombin via two machine learning methods. *Journal of Chemical Information and Modeling*, 63(12):3705–3718, 2023. PMID: 37285464.

[29] Aditya Devarakonda. *Avoiding Communication in First Order Methods for Optimization.* Phd thesis, UC Berkeley, 2018.

[30] Aditya Devarakonda, Kimon Fountoulakis, James Demmel, and Michael W. Mahoney. Avoiding synchronization in first-order methods for sparse convex optimization. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 409–418, Piscataway, NJ, USA, 2018. IEEE.

[31] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. Association for Computing Machinery.

[32] Saeed Soori, Aditya Devarakonda, Zachary Blanco, James Demmel, Mert Gurbuzbalaban, and Maryam Mehri Dehnavi. Reducing communication in proximal newton methods for sparse least squares problems. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP '18, New York, NY, USA, 2018. Association for Computing Machinery.

[33] Samuel Williams, Mike Lijewski, Ann Almgren, Brian Van Straalen, Erin Carson, Nicholas Knight, and James Demmel. s-step krylov subspace methods as bottom solvers for geometric multigrid. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1149–1158, Piscataway, NJ, USA, 2014. IEEE.

[34] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[35] Zeyuan Allen Zhu, Weizhu Chen, Gang Wang, Chenguang Zhu, and Zheng Chen. P-packSVM: Parallel primal gradient descent kernel svm. In *Proceedings of the 9th IEEE international conference on data mining*, ICDM '09, page 677–686, Piscataway, NJ, USA, 2009. IEEE Computer Society.

[36] Anthony Chronopoulos. *A class of parallel iterative methods implemented on multiprocessors.* PhD thesis, University of Illinois at Urbana-Champaign, 1987.

[37] Sebastian U. Stich. Local SGD converges fast and communicates little. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, pages 1–17, New Orleans, LA, USA, 2019. OpenReview.net.

[38] Gustavo Chávez, Yang Liu, Pieter Ghysels, Xiaoye Sherry Li, and Elizaveta Rebrova. Scalable and memory-efficient kernel ridge regression. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 956–965, New Orleans, LA, USA, 2020. IEEE.

[39] Scott Sallinen, Nadathur Satish, Mikhail Smelyanskiy, Samantika S. Sury, and Christopher Ré. High performance parallel stochastic gradient descent in shared memory. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 873–882, Chicago, IL, USA, 2016. IEEE.

[40] Yang You, James Demmel, Kenneth Czechowski, Le Song, and Richard Vuduc. Ca-svm: Communication-avoiding support vector machines on distributed systems. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 847–859, Hyderabad, India, 2015. IEEE.

[41] Yang You, James Demmel, Cho-Jui Hsieh, and Richard Vuduc. Accurate, fast and scalable kernel ridge regression on parallel and distributed systems. In *Proceedings of the 2018 International Conference on Supercomputing*, ICS '18, page 307–317, New York, NY, USA, 2018. Association for Computing Machinery.