Avoiding communication in primal and dual block coordinate descent methods

Aditya Devarakonda

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-196
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-196.html

December 12, 2016

Acknowledgement

This report is based on joint work with Kimon Fountoulakis, James Demmel, and Michael W. Mahoney. I thank my collaborators for the feedback and discussions that have led to this report. A special thank you to my advisors James Demmel and Michael Mahoney for their guidance and support.

Avoiding Communication in Primal and Dual Block Coordinate Descent Methods

by Aditya Devarakonda

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor James W. Demmel
Research Advisor

(Date)

* * * * * * *

Professor Michael W. Mahoney
Second Reader

(Date)

Avoiding communication in primal and dual block coordinate descent methods

Aditya Devarakonda[*]

Abstract

Primal and dual block coordinate descent methods are iterative methods for solving regularized and unregularized optimization problems. Distributed-memory parallel implementations of these methods have become popular in analyzing large machine learning datasets. However, existing implementations communicate at every iteration which, on modern data center and supercomputing architectures, often dominates the cost of floating-point computation. Recent results on communication-avoiding Krylov subspace methods suggest that large speedups are possible by re-organizing iterative algorithms to avoid communication. We show how applying similar algorithmic transformations can lead to primal and dual block coordinate descent methods that only communicate every s iterations–where s is a tuning parameter–instead of every iteration for the *regularized least-squares problem.* We derive communication-avoiding variants of the primal and dual block coordinate descent methods which reduce the number of synchronizations by a factor of s on distributed-memory parallel machines without altering the convergence rate. Our communication-avoiding algorithms attain modeled strong scaling speedups of 14× and 165× on a modern supercomputer using MPI and Apache Spark, respectively. Our algorithms attain modeled weak scaling speedups of 12× and 396× on the same machine using MPI and Apache Spark, respectively.

1 Introduction

The running time of an algorithm depends on computation, the number of arithmetic operations (F), and communication, the cost of data movement. The communication cost includes the "bandwidth cost", i.e. the number, W, of words sent either between levels of a memory hierarchy or between processors over a network, and the "latency cost", i.e. the number, L, of messages sent, where a message either consists of a group of contiguous words being sent, or is used for interprocess synchronization. On modern computer architectures, communicating data often takes much longer than performing a floating-point operation. The gap between communication and computation is continuing to increase. Therefore, it is especially important to design algorithms that minimize communication in order to attain high performance on modern computer architectures. Communication-avoiding algorithms are a new class of algorithms that exhibit large speedups on modern, distributed-memory parallel architectures through careful algorithmic transformations [4]. All of direct and iterative linear algebra have been re-organized to avoid communication and has led to significant performance improvements over existing state-of-the-art libraries [4, 3, 8, 23, 38, 45]. The results from communication-avoiding Krylov subspace methods [8, 15, 23] are particularly relevant to our work.

The origins of communication-avoiding Krylov subspace methods lie in the s-step Krylov methods work. Van Rosendale's s-step conjugate gradients method [43], Chronopoulos and Gear's s-step methods for preconditioned and unpreconditioned symmetric linear systems [11, 12],

[*]Computer Science Div. Univ. of California, Berkeley, CA 94720 (aditya@eecs.berkeley.edu).

Summary of algorithm costs					
Algorithm	Data layout	Flops cost (F)	Latency cost (L)	Bandwidth cost (W)	Memory cost (M)
BCD	1D-column	$O\left(\frac{Hb^2n}{P} + Hb^3\right)$	$O\left(H \log P\right)$	$O\left(Hb^2 \log P\right)$	$O\left(\frac{dn}{P} + b^2\right)$
CA-BCD		$O\left(\frac{Hb^2ns}{P} + Hb^3\right)$	$O\left(\frac{H}{s} \log P\right)$	$O\left(Hb^2 s \log P\right)$	$O\left(\frac{dn}{P} + b^2s^2\right)$
BDCD	1D-row	$O\left(\frac{H'b'^2d}{P} + H'b'^3\right)$	$O\left(H' \log P\right)$	$O\left(H'b'^2 \log P\right)$	$O\left(\frac{dn}{P} + b'^2\right)$
CA-BDCD		$O\left(\frac{H'b'^2ds}{P} + H'b'^3\right)$	$O\left(\frac{H'}{s} \log P\right)$	$O\left(H'b'^2 s \log P\right)$	$O\left(\frac{dn}{P} + b'^2s^2\right)$

Table 1: Flops (F), Latency (L), Bandwidth (W) and Memory per processor (M) costs comparison along the critical path of classical BCD (Thm. 1), BDCD (Thm. 2) and communication-avoiding BCD (Thm. 6) and BDCD (Thm. 7) algorithms for 1D-block column and 1D-block row data partitioning, respectively. H and H' are the number of iterations and b and b' are the block sizes for BCD, and BDCD. We assume that $X \in \mathbb{R}^{d \times n}$ is dense, P is the number of processors and s is the loop-blocking parameter.

Chronopoulos and Swanson's s-step methods for unsymmetric linear systems [10] and Kim and Chronopoulos's s-step non-symmetric Lanczos method [25] were designed to extract more parallelism than their standard counterparts. s-step Krylov methods compute s Krylov basis vectors and perform residual and solution vector updates by using Gram matrix computations and replacing modified Gram-Schmidt orthogonalization with Householder QR [44]. These optimizations enable s-step Krylov methods to use BLAS-3 matrix-matrix operations which attain higher peak hardware performance and have more parallelism than the BLAS-1 vector-vector and BLAS-2 matrix-vector operations used in standard Krylov methods. However, these methods do not avoid communication in the s Krylov basis vector computations. Demmel, Hoemmen, Mohiyuddin, and others [15, 23, 31, 32] introduced the matrix powers kernel optimization which reduces the communication cost of the s Krylov basis vector computations by a factor $O(s)$ for well-partitioned matrices. The combination of the matrix powers kernel along with extensive algorithmic modifications to the s-step methods resulted in what Carson, Demmel, Hoemmen and others call communication-avoiding Krylov subspace methods [8, 15, 23].

We build on existing work by extending those results to machine learning where scalable algorithms are especially important given the enormous amount of data. Block coordinate descent methods are routinely used in machine learning to solve optimization problems [46]. Given a dataset $X \in \mathbb{R}^{d \times n}$ where the rows are features of the data and the columns are data points, block coordinate descent methods can compute the regularized or unregularized least squares solution by iteratively solving a subproblem using a block of b rows of X [35, 46]. This process is repeated until the solution converges to a desired accuracy or until the number of iterations has reached a user-defined limit. This suggests that, if the matrix is partitioned across some number of processors then the algorithm communicates at each iteration in order to solve the subproblem. This follows from the fact that the primal method uses b features (resp. data points for the dual method) and computes a Gram matrix. The running time for such methods is often dominated by communication cost which increases with the number of processors (P).

There are some frameworks and algorithms which attempt to address the communication bottleneck. For example, the CoCoA framework [24] reduces communication by performing coordinate descent on locally stored data points on each processor and intermittently communicating by summing or averaging the local solutions. CoCoA communicates fewer times than coordinate descent – although not provably so – and changes the convergence behavior. In contrast, our algorithms provably avoid communication *without* altering the convergence behavior. HOGWILD!

[34] is a lock-free approach to stochastic gradient descent (SGD) where each processor selects a data point, computes a gradient using its data point and updates the solution without synchronization. Due to the lack of synchronization (or locks) processors are allowed to overwrite the solution vector. The main results in HOGWILD! show that if the solution updates are sparse (i.e. each processor only modifies a part of the solution) then running without locks does not affect the final solution with high probability.

In contrast, our results reduce the latency cost in the primal and dual block coordinate descent methods by a factor of s on distributed-memory architectures, for dense and sparse updates without changing the convergence behavior, in exact arithmetic. Hereafter we refer to the primal method as block coordinate descent (BCD) and the dual method as block dual coordinate descent (BDCD). The principle behind our communication-avoiding approach is to unroll the BCD and BDCD iteration loops by a factor of s, compute Gram-like matrices for the s iterations and use the Gram-like matrices to update the solution vector. Table 1 summarizes our results for BCD with X stored in a 1D-block column layout and 1D-block row layout for BDCD. Our results show that the new communication-avoiding variants reduce the number of messages (i.e. synchronization events), which is the most dominant cost, by a factor of s but increase the bandwidth and computational cost by s. The algorithms we derive also avoid communication for other data layout schemes, however, we limit our discussion in this paper to the 1D-block column and 1D-block row layouts.

1.1 Contributions

We briefly summarize our contributions:

- We present communication-avoiding algorithms for block coordinate descent and block dual coordinate descent that *provably* reduce the latency cost by a factor of s.
- We analyze the computation, communication and storage costs of the classical and our new communication-avoiding algorithms under two data partitioning schemes and describe their tradeoffs.
- We compare the primal and dual methods on several machine learning datasets and explore the tradeoff between convergence of each method and properties of the dataset.
- We perform numerical experiments to illustrate that these algorithms are numerically stable for all choices of s tested and explore tradeoffs between choices of s and b.
- We show modeled performance results to illustrate that the communication-avoiding algorithms can be faster than the classical algorithms on a modern supercomputer using the MPI and Spark programming models. We observe modeled strong scaling speedups of $12\times$ and $169\times$ on MPI and Spark, respectively. We observe modeled weak scaling speedups of $14\times$ and $365\times$ on MPI and Spark, respectively.

1.2 Organization

The rest of the paper is organized as follows: Section 2 summarizes existing methods for solving the regularized least squares problem and describes the communication cost model we use to analyze our algorithms. We also present a survey of the costs of MPI communication primitives necessary to bound the communication costs of our algorithms. Section 3 presents the algorithmic transformation required to avoid communication in BCD and BDCD, respectively.

Algorithm Cost Comparison				
Algorithm	Flops cost (F)	Latency cost (L)	Bandwidth cost (W)	Memory cost (M)
BCD (section 4 Thm. 1)	$O\left(\frac{Hb^2n}{P} + Hb^3\right)$	$O(H\log P)$	$O(Hb^2\log P)$	$O\left(\frac{dn}{P} + b^2\right)$
BDCD (section 4 Thm. 2)	$O\left(\frac{H'b'^2d}{P} + H'b'^3\right)$	$O\left(H'\log P\right)$	$O\left(H'b'^2\log P\right)$	$O\left(\frac{dn}{P} + b'^2\right)$
Krylov methods [4]	$O\left(\frac{kdn}{P}\right)$	$O(k\log P)$	$O\left(k\min(d,n)\log P\right)$	$O\left(\frac{dn}{P}\right)$
TSQR [14]	$O\left(\frac{\min(d,n)^2\max(d,n)}{P}\right)$	$O(\log P)$	$O(\min(d,n)^2\log P)$	$O\left(\frac{dn}{P}\right)$

Table 2: Computation and communication costs along the critical path of BCD, BDCD, Krylov and TSQR methods. H, H', and k are the total number of iterations required for the methods, respectively, to converge to a desired accuracy. b and b' are the block sizes for BCD and BDCD, respectively. We assume that $X \in \mathbb{R}^{d\times n}$ is dense, P is the number of processors. We assume that X is partitioned in 1D-block column for BCD and 1D-block row BDCD. For Krylov methods we assume a 1D layout (block row or column depending on shape of X) where the small dimension vectors are replicated and, therefore, require a broadcast during each iteration.

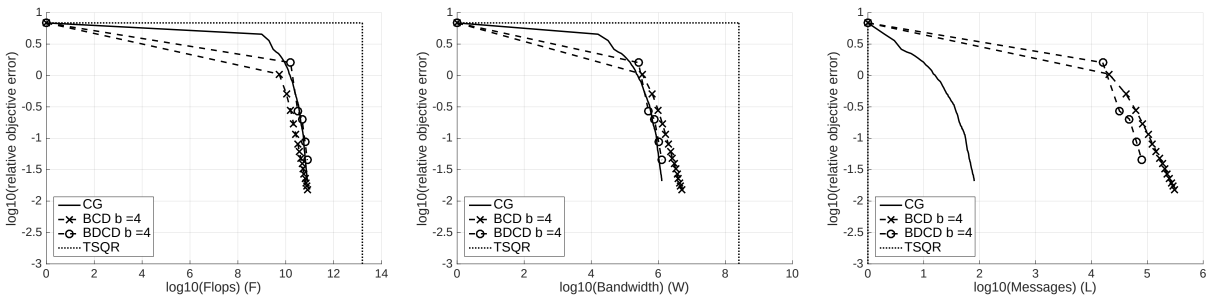
Section 4 analyzes the worst case computation, communication and storage costs of the classical and communication-avoiding algorithms under the 1D-block column and 1D-block row data layouts. Section 5 provides numerical experiments that compare the convergence behavior of the communication-avoiding variants to their standard counterparts and modeled strong scaling and weak scaling speedups on a modern supercomputer. Finally, we conclude in Section 6 and provide remarks about directions for future work based on the results in this paper.

2 Background

We begin by surveying existing methods for solving the regularized least squares problems. We compare the algorithm costs of these methods and describe their tradeoffs in order to motivate the need for coordinate methods. Then we describe the communication cost model and summarize the cost of various communication patterns. The communication costs depend heavily on the underlying message routing algorithms, therefore, we describe the cost of communicating using the Message Passing Interface (MPI) [16]. The underlying message routing algorithms in MPI have been well-studied and, in most cases, attain known communication lower bounds.

2.1 Survey of Regularized Least Squares Methods

Unregularized and regularized least squares problems have been well-studied in literature from direct matrix factorization approaches [14] to Krylov [5, 8, 36] and (primal and dual) block coordinate descent methods [6, 24, 37, 40, 46]. Although these methods solve the same problem, their computation and communication complexities differ. The complexity analysis in this section assumes that the data matrix $X \in \mathbb{R}^{d\times n}$ is dense for simplicity. TSQR [14, 23] can be used to implicitly solve the normal equations and costs $O\left(\min(d,n)^2\max(d,n)\right)$ flops. This is the same Big-O cost as forming the normal equations and solving them explicitly, but is always numerically stable. In Krylov methods, the most expensive computations are matrix-vector multiplies at each iteration. If Krylov methods require k iterations to converge, then they cost $O(kdn)$ flops. BCD and BDCD methods iteratively select b features, i.e. rows of length n, (BCD) or b' data points, i.e. columns of length d (BDCD), and solve least squares on the subproblem. We assume that the subproblem is solved implicitly by first constructing the Gram matrix and computing its Cholesky factorization. If BCD and BDCD methods require H and H' iterations, respectively, then they cost $O\left(Hb^2n + Hb^3\right)$ and $O\left(H'b'^2d + H'b'^3\right)$ flops.



(a) Flops (b) Bandwidth (c) Messages

Figure 1: Comparison of the objective function convergence behavior of BCD, BDCD, CG and TSQR versus their theoretical algorithm costs on a $d = 62061$ by $n = 15935$ matrix. We set an accuracy limit of 10^{-2} and $b = 4$ for BCD and BDCD. The x and y axis are on $\log_{10}$-scale with $\left(0, \frac{f(X,w_{opt},y)-f(X,0,y)}{f(X,w_{opt},y)}\right)$ as a starting point. The relative error for TSQR is fixed until all flops are computed, after which the error drops to machine precision. The figures only show TSQR accuracy up to 10^{-3} and, since TSQR needs a single reduction, the curve overlaps with the y-axis in Figure 1c.

Table 2 summarizes the parallel algorithm costs of the various least-squares methods. From these costs we observe that the tradeoff space between these algorithms is large and varies based on the data and choices of algorithm parameters.

We explore the tradeoffs between the methods in Table 2 by comparing their convergence rates and their computation and communication costs. Figures 1a, 1b and 1c illustrate the tradeoff space between convergence behavior and flops, number of words communicated (bandwidth) and number of messages sent (latency). We allow each algorithm to converge to 10^{-2} accuracy and plot their convergence behavior by measuring the relative objective error

$$\frac{f(X, w_{opt}, y) - f(X, w_{alg}, y)}{f(X, w_{opt}, y)}$$

where $f(X, w, y) = \frac{1}{2n}\|X^T w - y\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$, w_{opt} is computed *a priori* from conjugate gradients with a tolerance of 10^{-15}, and w_{alg} is the solution obtained from CG, BCD or BDCD. Figure 1a shows that, although TSQR requires a single pass over the data, it performs approximately 100 times more flops than the iterative methods (all plots have x and y axes on $\log_{10}$-scale). The iterative methods perform equal numbers of flops with at most a constant factor difference. Since the test matrix has $d > n$, BDCD performs computations on d-dimensional vectors and costs $O\left(\frac{d}{n}\right)$ more flops per iteration than BCD. The bandwidth costs in Figure 1b show a similar tradeoff to the flops comparison. TSQR communicates more data since it performs a reduction on an $n \times n$ local triangular matrix, whereas CG communicates a single n-dimensional vector per iteration and the block coordinate methods communicate a $b \times b$ Gram matrix per iteration. BCD and BDCD communicate less data per iteration but require more iterations. As a result, TSQR communicates the most data, CG and BDCD communicate less data and, since $H > H'$, BCD communicates slightly more data. Ignoring TSQR, the bandwidth costs of CG, BCD and BDCD differ by constant factors. Finally, Figure 1c shows the number of messages (modulo $\log P$ factors) required for each algorithm. Since TSQR is a single-pass algorithm, it only requires one reduction (10^0) to perform the factorization. In comparison, CG requires k reductions, BCD requires H and BDCD requires H'. BCD and BDCD were comparable to or a constant factor worse than CG in terms of flops and bandwidth, however, they require *orders of magnitude* more messages than CG. Since the latency cost is the dominant factor, reducing it by even a small factor of s could lead to BCD and BDCD algorithms that are s times faster. Our work addresses this by reducing the number of messages sent by a factor of s.

2.2 Modeling Communication

Algorithms have traditionally been analyzed by counting arithmetic (the number of floating-point operations). However, data movement (communication) is another important cost that often dominates arithmetic cost [19, 22]. By combining the arithmetic and communication costs we obtain the following running time model

$$\mathrm{T}_{\text{algorithm}} = \underbrace{\gamma F}_{\text{Computation Cost}} + \underbrace{\alpha L + \beta W}_{\text{Communication Cost}} \tag{1}$$

where $\gamma, \alpha,$ and β are machine-specific parameters that correspond to the time per floating-point operation, overhead time per message, and time per words moved, respectively. $F, L,$ and W are algorithm-specific parameters that represent the total number of floating-point operations computed, the number of messages sent and the number of words moved, respectively. Communication models have been well-studied in literature from the complex LogP [13] and LogGP [2] models to the simpler α-β model (see equation 1). The LogP and LogGP models are refinements of the α-β model, therefore, we use the latter for simplicity. The α-β model applies to both sequential and parallel computations but we focus on the latter in this paper.

3 Communication-Avoiding Primal and Dual Block Coordinate Descent

In this section, we re-derive the block coordinate descent (BCD) (in section 3.1) and block dual coordinate descent (BDCD) (in section 3.2) algorithms starting from the respective minimization problems. The derivation of BCD and BDCD lead to recurrences which can be unrolled to derive communication-avoiding versions of BCD and BDCD, which we will refer to as CA-BCD and CA-BDCD respectively.

3.1 Derivation of Block Coordinate Descent

The coordinate descent algorithm minimizes the problem:

$$\arg\min_{w \in \mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{2n}\left\|X^T w - y\right\|_2^2 \tag{2}$$

where $X \in \mathbb{R}^{d\times n}$ is the data matrix whose rows are features and columns are data points, $y \in \mathbb{R}^n$ are the labels, $w \in \mathbb{R}^d$ are the weights, and $\lambda > 0$ is a regularization parameter. The minimization problem in (2) can be solved by block coordinate descent with the b-dimensional update

$$w_h = w_{h-1} + \mathbb{I}_h \Delta w_h \tag{3}$$

where $w_h \in \mathbb{R}^d$ and $\mathbb{I}_h = \left[e_{i_1}, e_{i_2}, \ldots, e_{i_b}\right] \in \mathbb{R}^{d\times b}$ and $i_k \in [d]$ for $k = 1, 2, \ldots, b$. By substitution in (2) we obtain the minimization problem

$$\arg\min_{\Delta w_h \in \mathbb{R}^b} \frac{\lambda}{2}\|w_{h-1} + \mathbb{I}_h \Delta w_h\|_2^2 + \frac{1}{2n}\|X^T w_{h-1} + X^T \mathbb{I}_h \Delta w_h - y\|_2^2$$

Algorithm 1 Block Coordinate Descent (BCD) Algorithm

```
1: Input: $X \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$, $H > 1$, $w_0 \in \mathbb{R}^d$, $b \in \mathbb{Z}_+$ s.t. $b \leq d$
2: for $h = 1, 2, \cdots, H$ do
3:     choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
4:     $\mathbb{I}_h = [e_{i_1}, e_{i_2}, \cdots, e_{i_b}]$
5:     $\Gamma_h = \frac{1}{n}\mathbb{I}_h^T XX^T\mathbb{I}_h + \lambda\mathbb{I}_h^T\mathbb{I}_h$
6:     $\Delta w_h = \Gamma_h^{-1}\left(-\lambda\mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X\alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T Xy\right)$
7:     $w_h = w_{h-1} + \mathbb{I}_h\Delta w_h$
8:     $\alpha_h = \alpha_{h-1} + X^T\mathbb{I}_h\Delta w_h$
9: Output $w_H$
```

with the closed-form solution

$$\Delta w_h = \left(\frac{1}{n}\mathbb{I}_h^T XX^T\mathbb{I}_h + \lambda\mathbb{I}_h^T\mathbb{I}_h\right)^{-1}\left(-\lambda\mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T XX^T w_{h-1} + \frac{1}{n}\mathbb{I}_h^T Xy\right). \tag{4}$$

Note, however, that the closed-from solution requires a matrix-vector product using the entire data matrix when computing $\frac{1}{n}\mathbb{I}_h^T XX^T w_{h-1}$. This expensive operation can be avoided by introducing the auxiliary variable:

$$\alpha_h = X^T w_h$$

which, by substituting (3), can be re-arranged into a vector update of the form

$$\begin{aligned}\alpha_h &= X^T w_{h-1} + X^T\mathbb{I}_h\Delta w_h \\ &= \alpha_{h-1} + X^T\mathbb{I}_h\Delta w_h\end{aligned} \tag{5}$$

and the closed-form solution can be written in terms of α_{h-1}

$$\Delta w_h = \left(\frac{1}{n}\mathbb{I}_h^T XX^T\mathbb{I}_h + \lambda\mathbb{I}_h^T\mathbb{I}_h\right)^{-1}\left(-\lambda\mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X\alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T Xy\right) \tag{6}$$

By introducing the auxiliary quantity α_h and the vector update (5) we can avoid operations using the data matrix. In order to make the communication-avoiding BCD derivation easier, let us define

$$\Gamma_h = \frac{1}{n}\mathbb{I}_h^T XX^T\mathbb{I}_h + \lambda\mathbb{I}_h^T\mathbb{I}_h$$

Then (6) can be re-written as

$$\Delta w_h = \Gamma_h^{-1}\left(-\lambda\mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X\alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T Xy\right). \tag{7}$$

This re-arrangement leads to Algorithm 1 which we refer to as BCD in residual form. The recurrence in lines 6, 7, and 8 of Algorithm 1 allow us to unroll the BCD iteration loop and avoid communication. We begin by changing the loop index from h to $sk + j$ where k is the outer loop parameter, s is the loop-blocking parameter and j is the inner loop parameter. Assume that we are at the beginning of iteration $sk + 1$ and w_{sk} and α_{sk} were just computed. Then Δw_{sk+1} can be computed by

$$\Delta w_{sk+1} = \Gamma_{sk+1}^{-1}\left(-\lambda\mathbb{I}_{sk+1}^T w_{sk} - \frac{1}{n}\mathbb{I}_{sk+1}^T X\alpha_{sk} + \frac{1}{n}\mathbb{I}_{sk+1}^T Xy\right)$$

Algorithm 2 Communication-Avoiding Block Coordinate Descent (CA-BCD) Algorithm

```
1: Input: $X \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$, $H > 1$, $w_0 \in \mathbb{R}^d$, $b \in \mathbb{Z}_+$ s.t. $b \leq d$
2: for $k = 0, 1, \cdots, \frac{H}{s}$ do
3:     for $j = 1, 2, \cdots, s$ do
4:         choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
5:         $\mathbb{I}_{sk+j} = [e_{i_1}, e_{i_2}, \cdots, e_{i_b}]$
6:     let $Y = \left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]^T X$.
7:     compute the Gram matrix, $G = \frac{1}{n}YY^T + \lambda I$.
8:     for $j = 1, 2, \cdots, s$ do
9:         $\Gamma_{sk+j}$ are the $b \times b$ diagonal blocks of $G$.
10:        $\Delta w_{sk+j} = \Gamma_{sk+j}^{-1}\Big( -\lambda \mathbb{I}_{sk+j}^T w_{sk} - \lambda \sum_{t=1}^{j-1} \left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t} \Delta w_{sk+t}\right)$
               $-\frac{1}{n}\mathbb{I}_{sk+j}^T X\alpha_{sk} - \frac{1}{n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T XX^T \mathbb{I}_{sk+t}\Delta w_{sk+t}\right) + \frac{1}{n}\mathbb{I}_{sk+j}^T Xy\Big)$
11:        $w_{sk+j} = w_{sk+j-1} + \mathbb{I}_{sk+j}\Delta w_{sk+j}$
12:        $\alpha_{sk+j} = \alpha_{sk+j-1} + X^T \mathbb{I}_{sk+j}\Delta w_{sk+j}$
13: Output $w_H$
```

By unrolling the recurrence for w_{sk+1} and α_{sk+1} we can compute Δw_{sk+2} in terms of w_{sk} and α_{sk}

$$\Delta w_{sk+2} = \Gamma_{sk+2}^{-1}\Bigg(-\lambda \mathbb{I}_{sk+2}^T w_{sk} - \lambda \mathbb{I}_{sk+2}^T \mathbb{I}_{sk+1}\Delta w_{sk+1} - \frac{1}{n}\mathbb{I}_{sk+2}^T X\alpha_{sk} - \frac{1}{n}\mathbb{I}_{sk+2}^T XX^T\mathbb{I}_{sk+1}\Delta w_{sk+1} + \frac{1}{n}\mathbb{I}_{sk+1}^T Xy\Bigg).$$

By induction we can show that Δw_{sk+j} can be computed using w_{sk} and α_{sk}

$$\Delta w_{sk+j} = \Gamma_{sk+j}^{-1}\Bigg(-\lambda \mathbb{I}_{sk+j}^T w_{sk} - \lambda \sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t}\Delta w_{sk+t}\right) - \frac{1}{n}\mathbb{I}_{sk+j}^T X\alpha_{sk} - \frac{1}{n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T XX^T\mathbb{I}_{sk+t}\Delta w_{sk+t}\right) + \frac{1}{n}\mathbb{I}_{sk+j}^T Xy\Bigg). \quad (8)$$

for $j = 1, 2, \ldots, s$. Due to the recurrence unrolling we can defer the updates to w_{sk} and α_{sk} for s steps. Notice that the first summation in (8) computes the intersection between the coordinates chosen at iteration $sk+j$ and $sk+t$ for $t = 1, \ldots, j-1$ via the product $\mathbb{I}_{sk+j}^T\mathbb{I}_{sk+t}$. Communication can be avoided in this term by initializing all processors to the same seed for the random number generator. The second summation in (8) computes the Gram-like matrices $\mathbb{I}_{sk+j}^T XX^T\mathbb{I}_{sk+t}$ for $t = 1, \ldots, j-1$. Communication can be avoided in this computation by computing the $sb \times sb$ Gram matrix $G = \left(\frac{1}{n}\left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]^T XX^T\left[+\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right] + \lambda I\right)$ once before the inner loop and redundantly storing it on all processors. Finally, at the end of the s inner loop iterations we can perform the vector updates

$$w_{sk+s} = w_{sk} + \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta w_{sk+t}\right) \quad (9)$$

$$\alpha_{sk+s} = \alpha_{sk} + X^T\sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta w_{sk+t}\right) \quad (10)$$

Algorithm 3 Block Dual Coordinate Descent (BDCD) Algorithm

```
1: **Input**: $X = [x_1, x_2, \dots x_n] \in \mathbb{R}^{d \times n}, y \in \mathbb{R}^n$, $H' > 1$, $\alpha_0 \in \mathbb{R}^n$, $b' \in \mathbb{Z}_+$ s.t. $b' \leq n$
2: **Initialize**: $w_0 \leftarrow \frac{-1}{\lambda n} X \alpha_0$
3: **for** $h = 1, 2, \cdots, H'$ **do**
4:     choose $\{i_m \in [n] | m = 1, 2, \dots, b'\}$ uniformly at random without replacement
5:     $\mathbb{I}_h = \left[e_{i_1}, e_{i_2}, \cdots, e_{i_{b'}}\right]$
6:     $\Theta_h = \frac{1}{\lambda n^2} \mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n} \mathbb{I}_h^T \mathbb{I}_h$
7:     $\Delta\alpha_h = \frac{1}{n} \Theta_h^{-1} \left(-\mathbb{I}_h^T X^T w_{h-1} + \mathbb{I}_h^T \alpha_{h-1} + \mathbb{I}_h^T y\right)$
8:     $\alpha_h = \alpha_{h-1} + \mathbb{I}_h \Delta\alpha_h$
9:     $w_h = w_{h-1} - \frac{1}{\lambda n} X \mathbb{I}_h \Delta\alpha_h$
10: **Output** $\alpha'_H$ and $w'_H$
```

The resulting communication-avoiding BCD (CA-BCD) algorithm is shown in Algorithm 2.

3.2 Derivation of Block Dual Coordinate Descent

The solution to the primal problem (2) can also be obtained by solving the dual minimization problem:

$$\arg\min_{\alpha \in \mathbb{R}^n} \frac{\lambda}{2} \left\| \frac{1}{\lambda n} X \alpha \right\|_2^2 + \frac{1}{2n} \|\alpha + y\|_2^2 \tag{11}$$

such that

$$w = -\frac{1}{\lambda n} X \alpha. \tag{12}$$

The minimization problem (11) can be solved using block coordinate descent which iteratively solves a sub-problem in $\mathbb{R}^{b'}$, where $1 \leq b' \leq n$ is a tunable block-size parameter. Let us first define the dual vector update for $\alpha_h \in \mathbb{R}^n$

$$\alpha_h = \alpha_{h-1} + \mathbb{I}_h \Delta\alpha_h \tag{13}$$

Where h is the iteration index, $\mathbb{I}_h = \left[e_{i_1}, e_{i_2}, \dots e_{i_{b'}}\right] \in \mathbb{R}^{n \times b'}$, $i_k \in [n]$ for $k = 1, 2, \dots b'$ and $\Delta\alpha_h \in \mathbb{R}^{b'}$. By substitution in (11), $\Delta\alpha_h$ is the solution to a minimization problem in $\mathbb{R}^{b'}$ as desired:

$$\arg\min_{\Delta\alpha_h \in \mathbb{R}^{b'}} \frac{1}{2\lambda n^2} \|X\alpha_{h-1} + X\mathbb{I}_h \Delta\alpha_h\|_2^2 + \frac{1}{2n} \|\alpha_{h-1} + \mathbb{I}_h \Delta\alpha_h + y\|_2^2 \tag{14}$$

Finally, due to (12) we obtain the primal vector update for $w_h \in \mathbb{R}^d$

$$w_h = w_{h-1} - \frac{1}{\lambda n} X \mathbb{I}_h \Delta\alpha_h. \tag{15}$$

From (13), (14), and (15) we obtain a block coordinate descent algorithm which solves the dual minimization problem. Henceforth, we refer to this algorithm as block dual coordinate descent (BDCD). Note that by setting $b' = 1$ we obtain the SDCA algorithm [37] with the least-squares loss function.

Algorithm 4 Communication-Avoiding Block Dual Coordinate Descent (CA-BDCD) Algorithm

```
1: Input: $X = [x_1, x_2, \dots x_n] \in \mathbb{R}^{d \times n}, y \in \mathbb{R}^n$, $H' > 1$, $\alpha_0 \in \mathbb{R}^n$, $b' \in \mathbb{Z}_+$ s.t. $b' \leq n$
2: Initialize: $w_0 \leftarrow \frac{-1}{\lambda n} X \alpha_0$
3: for $k = 0, 1, \cdots, \frac{H'}{s}$ do
4:     for $j = 1, 2, \cdots, s$ do
5:         choose $\{i_m \in [n] | m = 1, 2, \dots, b'\}$ uniformly at random without replacement
6:         $\mathbb{I}_{sk+j} = \left[e_{i_1}, e_{i_2}, \cdots, e_{i_{b'}}\right]$
7:     let $Y = X\left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \dots, \mathbb{I}_{sk+s}\right]$.
8:     compute the Gram matrix, $G' = \frac{1}{\lambda n^2} Y^T Y + \frac{1}{n} I$.
9:     for $j = 1, 2, \cdots, s$ do
10:        $\Theta_{sk+j}$ are the $b' \times b'$ diagonal blocks of $G'$.
11:        $\Delta\alpha_{sk+j} = -\frac{1}{n}\Theta_{sk+j}^{-1}\Big( - \mathbb{I}_{sk+j}^T X^T w_{sk} + \frac{1}{\lambda n} \sum_{t=1}^{j-1} \left(\mathbb{I}_{sk+j}^T X^T X \mathbb{I}_{sk+t} \Delta\alpha_{sk+t}\right)$
                   $+\mathbb{I}_{sk+j}^T \alpha_{sk} + \sum_{t=1}^{j-1} \left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t} \Delta\alpha_{sk+t}\right) + \mathbb{I}_{sk+j}^T y\Big)$
12:        $\alpha_{sk+j} = \alpha_{sk+j-1} + \mathbb{I}_{sk+j} \Delta\alpha_{sk+j}$
13:        $w_{sk+j} = w_{sk+j-1} - \frac{1}{\lambda n} X \mathbb{I}_{sk+j} \Delta\alpha_{sk+j}$
14: Output $\alpha_{H'}$ and $w_{H'}$
```

The optimization problem (14) which computes the solution along the chosen coordinates and has the closed-form

$$\Delta\alpha_h = -\left(\frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T \mathbb{I}_h\right)^{-1}\left(\frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T \alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T y\right). \qquad (16)$$

Let us define $\Theta_h \in \mathbb{R}^{b' \times b'}$ such that

$$\Theta_h = \left(\frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T \mathbb{I}_h\right)$$

From this we have that at iteration h, we compute the solution along the b' coordinates of the linear system

$$\Delta\alpha_h = -\frac{1}{n}\Theta_h^{-1}\left(-\mathbb{I}_h^T X^T w_{h-1} + \mathbb{I}_h^T \alpha_{h-1} + \mathbb{I}_h^T y\right) \qquad (17)$$

and obtain the BDCD algorithm shown in Algorithm 3. The recurrence in lines 7, 8, and 9 of Algorithm 3 allow us to unroll the BDCD iteration loop and avoid communication. We begin by changing the loop index from h to $sk + j$ where k is the outer loop parameter, s is the loop-blocking parameter and j is the inner loop parameter. Assume that we are at the beginning of iteration $sk + 1$ and w_{sk} and α_{sk} were just computed. Then $\Delta\alpha_{sk+1}$ can be computed by

$$\Delta\alpha_{sk+1} = -\frac{1}{n}\Theta_{sk+1}^{-1}\left(-\mathbb{I}_{sk+1}^T X^T w_{sk} + \mathbb{I}_{sk+1}^T \alpha_{sk} + \mathbb{I}_{sk+1}^T y\right).$$

Furthermore, by unrolling the recurrence for w_{sk+1} and α_{sk+1} we can analogously to (8) show by induction that

$$\Delta\alpha_{sk+j} = -\frac{1}{n}\Theta_{sk+j}^{-1}\Bigg(- \mathbb{I}_{sk+j}^T X^T w_{sk} + \frac{1}{\lambda n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T X^T X \mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) + \mathbb{I}_{sk+j}^T\alpha_{sk} + \sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T\mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) + \mathbb{I}_{sk+j}^T y\Bigg) \quad (18)$$

for $j = 1, 2, \ldots, s$. Note that due to unrolling the recurrence we can compute $\Delta\alpha_{sk+j}$ from w_{sk} and α_{sk} which are the dual and primal solution vectors from the previous outer iteration. Since the solution vector updates require communication, the loop unrolling allows us to defer those updates for s iterations at the expense of additional computation. The solution vectors can be updated at the end of the inner iterations by

$$w_{sk+s} = w_{sk} - \frac{1}{\lambda n} X \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) \quad (19)$$

$$\alpha_{sk+s} = \alpha_{sk} + \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) \quad (20)$$

The resulting communication-avoiding BDCD (CA-BDCD) algorithm is shown in Algorithm 4.

4 Analysis of Algorithms

From the derivations in Section 3, we can observe that the primal and dual block coordinate descent algorithms perform computations on XX^T and X^TX, respectively. This implies that, along with the convergence rates, the shape of X is a key factor in choosing between the two methods. Furthermore, the data partitioning scheme used to distribute X between processors may cause one method to have a lower communication cost than the other. In this section we analyze the cost of Block Coordinate Descent and Block Dual Coordinate Descent under two data partitioning schemes: 1D-block row (feature partitioning) and 1D-block column (data point partitioning). In both cases, we derive the associated computation and communication costs in order to compare the classical algorithms to our communication-avoiding variants. We describe the tradeoffs between the choice of data partitioning scheme and its effect on the communication cost of the BCD and BDCD algorithms. We assume that the matrix $X \in \mathbb{R}^{d\times n}$ is dense to simplify the analysis. We further assume that vectors in $\mathbb{R}^n$ are partitioned and vectors in $\mathbb{R}^d$ are replicated for 1D-block column. The reverse holds if X is stored in a 1D-block row layout. We begin in Section 4.1 with the analysis of the classical BCD and BDCD algorithms and then analyze our new, communication-avoiding variants in Section 4.2.

4.1 Classical Algorithms

We begin with the analysis of the BCD algorithm with X stored in a 1D-block column layout and show how to extend this proof to BDCD with X in a 1D-block row layout.

Theorem 1. *H iterations of the Block Coordinate Descent (BCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block column partitions with a block size b, on P processors along the critical path costs*

$$F = O\left(\frac{Hb^2n}{P} + Hb^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + b^2\right) \textit{words of memory},$$

$$W = O\left(Hb^2\log P\right) \textit{words moved, and} \qquad L = O\left(H\log P\right) \textit{messages.}$$

Proof. The BCD algorithm computes a $b \times b$ Gram matrix $\Gamma_h = \frac{1}{n}\mathbb{I}_h^T X X^T \mathbb{I}_h + \lambda \mathbb{I}_h^T \mathbb{I}_h$, solves the $b \times b$ linear system $\Delta w_h = \Gamma_h^{-1}\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X \alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right)$ and updates the vectors w_h and α_h. Computing the Gram matrix requires that each processor locally compute a $b \times b$ block of inner-products and then perform an all-reduce (a reduction and broadcast) to sum the partial blocks. This operation requires $O\left(\frac{b^2 n}{P}\right)$ flops, communicates $O\left(b^2 \log P\right)$ words, and requires $O\left(\log P\right)$ messages. In order to solve the sub-problem redundantly on all processors, a local copy of the residual $\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X \alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right)$ is required. Computing the residual requires $O\left(\frac{bn}{P}\right)$ flops, and communicates $O\left(b \log P\right)$ words, in $O\left(\log P\right)$ messages. Once the residual is computed the sub-problem can be solved redundantly on each processor in $O\left(b^3\right)$ flops. Finally, the vector updates to w_h and α_h can be computed without any communication in $O\left(b + \frac{bn}{P}\right)$ flops on each processor. The critical path costs of H iterations of this algorithm are $O\left(\frac{Hb^2 n}{P} + Hb^3\right)$ flops, $O\left(Hb^2 \log P\right)$ words, and $O\left(H \log P\right)$ messages. Each processor requires enough memory to store w_h, Γ_h, Δw, $\mathbb{I}_h$ and $\frac{1}{P}$-th of X, α_h, and y. Therefore the memory cost of each processor is $d + b^2 + 2b + \frac{dn+2n}{P} = O\left(\frac{dn}{P} + b^2\right)$ words per processor. □

Theorem 2. *H' iterations of the Block Dual Coordinate Descent (BDCD) algorithm with the matrix $X \in \mathbb{R}^{d \times n}$ stored in 1D-block row partitions with a block size b', on P processors along the critical path costs*

$$F = O\left(\frac{H' {b'}^2 d}{P} + H' {b'}^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + {b'}^2\right) \textit{words of memory},$$

$$W = O\left(H' {b'}^2 \log P\right) \textit{words moved, and} \qquad L = O\left(H' \log P\right) \textit{messages}.$$

Proof. The BDCD algorithm computes a $b' \times b'$ Gram matrix $\Theta_h = \frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T \mathbb{I}_h$. Since Θ_h requires inner-products between columns of X, a 1D-block row partitioning scheme ensures that all processors contribute to each entry of Θ_h. A similar cost analysis to the one used in Theorem 1 proves this theorem. □

If X is stored in a 1D-block row layout, then each processor stores a disjoint subset of the features of X. The BCD algorithm selects b features at each iteration, however, due to the 1D-block row partitioning this could possibly lead to load imbalance. In the worst case, $P - b$ processors could be idle and the b features chosen could be assigned to the remaining b processors. Therefore, we begin by proving a bound on the worst case maximum number of features stored on a single processor. This bound only holds with high probability since the features are chosen uniformly at random.

Lemma 3. *Given a matrix $X \in \mathbb{R}^{d \times n}$ and P processors such that each processor stores $\Theta\left(\left\lfloor \frac{d}{P} \right\rfloor\right)$ features, if s features are chosen uniformly at random such that $b < P$, then the worst case maximum number of features assigned to a single processor w.h.p. is $O\left(\frac{\ln b}{\ln \ln b}\right)$.*

Proof. This is the well-known randomized load balancing variant of the balls and bins problem [21] [30, Lem.2.14]. The worst case load balance occurs when $P - b$ processors are idle and only b processors share the load. Since the number of processors sharing the load and the number of rows chosen are the same (i.e. there are b balls and b bins) we can directly apply the max load bound derived by Gonnet [21] and Mitzenmacher [30, Lem.2.14]. Therefore, the worst case maximum load on a single processor is $O\left(\frac{\ln b}{\ln \ln b}\right)$. If we assume $b < \frac{P}{\log P}$ a tighter bound of $O\left(\frac{\log P}{\log \frac{P}{b}}\right)$ applies. This is proved by Mitzenmacher in his thesis [30]. □

This bound can then be used to bound the communication cost required to perform load-balancing. A similar result holds for the BDCD algorithm with X stored in a 1D-block column layout.

Theorem 4. *H iterations of the Block Coordinate Descent (BCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block row partitions with a block size b, on P processors along the critical path costs w.h.p.*

$$F = O\left(\frac{Hb^2n}{P} + Hb^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + b^2\right) \textit{words of memory},$$

$$\textit{and } W = O\left(Hb^2\log P + \frac{Hn\ln b}{\ln\ln b}\log P\right) \textit{words moved}, \qquad L = O\left(H\log P\right) \textit{ for small messages}$$

or for large messages

$$W = O\left(Hb^2\log P + \frac{Hn\ln b}{\ln\ln b}\right) \textit{words moved}, \qquad L = O\left(HP\right) \textit{messages.}$$

Proof. The 1D-block row partitioning scheme implies that the $b\times b$ Gram matrix $\Gamma_h = \frac{1}{n}\mathbb{I}_h^T XX^T\mathbb{I}_h + \lambda\mathbb{I}_h^T\mathbb{I}_h$ computation may be load imbalanced. Since we randomly select b rows, some processors may hold multiple rows while others hold none. In order to balance the computational load we perform an all-to-all to convert the $b\times n$ sampled matrix into the 1D-block column layout. The amount of data moved is bounded by the max-loaded processor, which from Lemma 3, stores $O\left(\frac{\ln b}{\ln\ln b}\right)$ rows in the worst-case. This requires $W = O\left(\frac{n\ln b}{\ln\ln b}\log P\right)$ and $L = O\left(\log P\right)$ for small messages or $W = O\left(\frac{n\ln b}{\ln\ln b}\right)$ and $L = O\left(HP\right)$ for large messages. The all-to-all requires additional storage on each processor of $M = O\left(\frac{bn}{P}\right)$ words. Once the sampled matrix is converted, the BCD algorithm proceeds as in Theorem 1. By combining the cost of the all-to-all over H iterations and the costs from Theorem 1, we obtain the costs for the BCD algorithm with X stored in a 1D-block row layout. Note that the additional storage for the all-to-all does not dominate since $b < d$ by definition. □

Theorem 5. *H' iterations of the Block Dual Coordinate Descent (BDCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block column partitions with a block size b', on P processors along the critical path costs w.h.p.*

$$F = O\left(\frac{H'b'^2d}{P} + H'b'^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + b'^2\right) \textit{words of memory},$$

$$\textit{and } W = O\left(H'b'^2\log P + \frac{H'd\ln b'}{\ln\ln b'}\log P\right) \textit{words moved}, \qquad L = O\left(H'\log P\right) \textit{ for small messages}$$

or for large messages

$$W = O\left(H'b'^2\log P + \frac{H'd\ln b'}{\ln\ln b'}\right) \textit{words moved}, \qquad L = O\left(H'P\right) \textit{messages.}$$

Proof. The BDCD algorithm computes a $b'\times b'$ Gram matrix $\Theta_h = \frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X\mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T\mathbb{I}_h$. A 1D-block column partitioning scheme implies that the Gram matrix computation will be load imbalanced and, therefore, requires an all-to-all to convert the sampled matrix into a 1D-block row layout. A similar cost analysis to the one used in Theorem 4 proves this theorem. □

4.2 Communication-Avoiding Algorithms

In this section, we derive the computation and communication costs of our communication-avoiding BCD and BDCD algorithm under the 1D-block row and 1D-block column data layouts. In both cases we show that our algorithm reduces the communication costs by a factor of s over the classical algorithms. We begin with the CA-BCD algorithm in 1D-block column layout and, then show how this proof extends to CA-BDCD in 1D-block row layout.

Theorem 6. *H iterations of the Communication-Avoiding Block Coordinate Descent (CA-BCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block column partitions with a block size b, on P processors along the critical path costs*

$$F = O\left(\frac{Hb^2ns}{P} + Hb^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + b^2s^2\right) \textit{words of memory},$$

$$\textit{and } W = O\left(Hb^2s\log P\right) \textit{words moved}, \qquad L = O\left(\frac{H}{s}\log P\right) \textit{ messages.}$$

Proof. The CA-BCD algorithm computes the $sb \times sb$ Gram matrix $G = \frac{1}{n}YY^T + \lambda I$ where $Y = \left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]^T X$, solves s $(b \times b)$ linear systems to compute Δw_{sk+j} and updates the vectors w_{sk+s} and α_{sk+s}. Computing the Gram matrix requires that each processor locally compute a $sb \times sb$ block of inner-products and then perform an all-reduce (a reduction and broadcast) to sum the partial blocks. This operation requires $O\left(\frac{s^2b^2n}{P}\right)$ flops, communicates $O\left(s^2b^2\log P\right)$ words, and requires $O\left(\log P\right)$ messages. In order to solve the sub-problem redundantly on all processors, a local copy of the residual $\left(-\lambda\mathbb{I}_{sk+j}^T w_{sk} - \frac{1}{n}\mathbb{I}_{sk+j}^T X\alpha_{sk} + \frac{1}{n}\mathbb{I}_{sk+j}^T Xy\right)$ is required. Computing the residual requires $O\left(\frac{bns}{P}\right)$ flops, and communicates $O\left(sb\log P\right)$ words, in $O\left(\log P\right)$ messages. Once the residual is computed the sub-problem can be solved redundantly on each processor in $O\left(b^3s + b^2s^2\right)$ flops. Finally, the vector updates to w_{sk+s} and α_{sk+s} can be computed without any communication in $O\left(bs + \frac{bns}{P}\right)$ flops on each processor. Since the critical path occurs every $\frac{H}{s}$ iterations (every outer iteration), the algorithm costs $O\left(\frac{Hb^2ns}{P} + Hb^3\right)$ flops, $O\left(Hb^2s\log P\right)$ words, and $O\left(\frac{H}{s}\log P\right)$ messages. Each processor requires enough memory to store w_{sk+j}, G, Δw_{sk+j}, $\mathbb{I}_{sk+j}$ and $\frac{1}{P}$-th of X, α_{sk+j}, and y. Therefore the memory cost of each processor is $d + s^2b^2 + 2sb + \frac{dn+2n}{P} = O\left(\frac{dn}{P} + b^2s^2\right)$ words per processor. □

Theorem 7. *H' iterations of the Communication-Avoiding Block Dual Coordinate Descent (CA-BDCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block column partitions with a block size b', on P processors along the critical path costs*

$$F = O\left(\frac{H'b'^2ds}{P} + H'b'^3\right) \textit{flops}, \qquad M = O\left(\frac{dn}{P} + b'^2s^2\right) \textit{words of memory},$$

$$\textit{and } W = O\left(H'b'^2s\log P\right) \textit{words moved}, \qquad L = O\left(\frac{H'}{s}\log P\right) \textit{ messages.}$$

Proof. The CA-BDCD algorithm computes the $sb' \times sb'$ Gram matrix $G = \frac{1}{\lambda n^2}Y^TY + \frac{1}{n}I$ where $Y = X\left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]$. The 1D-block column partitioning layout ensures that each processor computes a partial $sb' \times sb'$ block of the Gram matrix. A similar cost analysis to Theorem 6 proves this theorem. □

Theorem 8. *H iterations of the Communication-Avoiding Block Coordinate Descent (CA-BCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block row partitions with a block size b, on P processors along the critical path costs w.h.p.*

$$F = O\left(\frac{Hb^2ns}{P} + Hb^3\right) \textit{ flops}, \; M = O\left(\frac{dn}{P} + \frac{bns}{P} + b^2s^2\right) \textit{ words of memory},$$

and $W = O\left(Hb^2s\log P + \frac{Hn\ln sb}{\ln\ln sb}\log P\right)$ *words moved,* $L = O\left(\frac{H}{s}\log P\right)$ *for small messages or for large messages*

$$W = O\left(Hb^2s\log P + \frac{Hn\ln sb}{\ln\ln sb}\right) \textit{ words moved}, \; L = O\left(\frac{H}{s}P\right) \textit{ messages}.$$

Proof. The 1D-block row partitioning scheme implies that the $sb \times sb$ Gram matrix computation may be load imbalanced. Since we randomly select sb rows, some processors may hold multiple chosen rows while some hold none. In order to balance the computational load we perform an all-to-all to convert the $sb \times n$ sampled matrix into the 1D-block column layout. The amount of data moved is bounded by the max-loaded processor, which from Lemma 3, stores $O\left(\frac{\ln sb}{\ln\ln sb}\right)$ rows. This requires $W = O\left(\frac{n\ln sb}{\ln\ln sb}\log P\right)$ and $L = O\left(\log P\right)$ for small messages or $W = O\left(\frac{n\ln sb}{\ln\ln sb}\right)$ and $L = O\left(HP\right)$ for large messages. The all-to-all requires additional storage on each processor of $M = O\left(\frac{sbn}{P}\right)$ words. Once the sampled matrix is converted, the BCD algorithm proceeds as in Theorem 6. By combining the cost of the all-to-all over H iterations and the costs from Theorem 6, we obtain the costs for the CA-BCD algorithm with X stored in a 1D-block row layout. Note that the additional storage for the all-to-all may dominate if $d < bs$. Therefore, bs must be chosen carefully. □

Theorem 9. *H iterations of the Communication-Avoiding Block Dual Coordinate Descent (CA-BDCD) algorithm with the matrix $X \in \mathbb{R}^{d\times n}$ stored in 1D-block column partitions with a block size b', on P processors along the critical path costs w.h.p.*

$$F = O\left(\frac{H'b'^2ds}{P} + H'b'^3\right) \textit{ flops}, \; M = O\left(\frac{dn}{P} + \frac{b'ds}{P} + b'^2s^2\right) \textit{ words of memory},$$

and $W = O\left(H'b'^2s\log P + \frac{H'n\ln sb'}{\ln\ln sb'}\log P\right)$ *words moved,* $L = O\left(\frac{H'}{s}\log P\right)$ *for small messages or for large messages*

$$W = O\left(H'b'^2s\log P + \frac{H'n\ln sb'}{\ln\ln sb'}\right) \textit{ words moved}, \; L = O\left(\frac{H'}{s}P\right) \textit{ messages}.$$

Proof. The CA-BDCD algorithm computes a $sb' \times sb'$ Gram matrix, G. A 1D-block column partitioning scheme implies that the Gram matrix computation will be load imbalanced and, therefore, requires an all-to-all to convert the sampled matrix into a 1D-block row layout. A similar cost analysis to the one used in Theorem 8 proves this theorem. □

The communication-avoiding variants that we have derived require a factor of s fewer messages than their classical counterparts, at the cost of more flops, bandwidth and memory. The bandwidth increase can largely be ignored since the classical algorithms are not bandwidth-limited. Therefore, increasing the bandwidth by s is unlikely to dominate. However, s must be chosen carefully to balance the additional flops and memory consumption with the reduction in the latency cost. This suggests that if latency is the dominant cost then our communication-avoiding variants can attain a s-fold speedup.

Summary of datasets						
Name	Features (d)	Data Points (n)	NNZ%	σ_{min}	σ_{max}	Source
abalone	8	$4,177$	100	$4.3e{-5}$	$2.3e{+4}$	UCI [27]
news20	$62,061$	$15,935$	0.13	$1.7e{-6}$	$6.0e{+5}$	LIBSVM [26]
a9a	123	$32,651$	11	$4.9e{-6}$	$2.0e{+5}$	UCI [27]
real-sim	$20,958$	$72,309$	0.24	$1.1e{-3}$	$9.2e{+2}$	LIBSVM [29]

Table 3: Properties of the LIBSVM datasets used in our experiments. We report the largest and smallest eigenvalues values[1]of X^TX.

5 Experimental Evaluation

We proved in Section 4 that our communication-avoiding BCD and BDCD algorithms reduce latency (the dominate cost) at the expense of additional bandwidth and computation. In section 5.1 we experimentally show that the communication-avoiding variants are numerically stable and, in section 5.2, we show that the communication-avoiding variants can lead to large modeled speedups on a modern supercomputer using MPI and Apache Spark.

5.1 Numerical Experiments

The algorithm transformations derived in Section 3 require that the CA-BCD and CA-BDCD operate on Gram matrices of size $sb \times sb$ instead of size $b \times b$ every outer iteration. Due to the larger dimensions, the conditioning of the Gram matrix increases and may have an adverse affect on the convergence behavior. We explore this tradeoff between convergence behavior, flops, communication and the choices of b and s for the classical and communication-avoiding algorithms. All experiments were performed in MATLAB version R2015b on a 2.3 GHz Intel i7 machine with 8GB of RAM with datasets obtained from the LIBSVM repository [9]. Datasets were chosen so that all algorithms were tested on a range of shapes, sizes, and condition numbers. Table 3 summarizes the important properties of the datasets tested. For all experiments, we set the regularization parameter to $\lambda = 1000\sigma_{min}$. In practice, λ should be chosen based on metrics like prediction accuracy on the test data (or hold-out data). Smaller values of λ would slow the convergence rate and require more iterations, therefore we choose λ so that our experiments have reasonable running times. We do not explore tradeoffs between λ values, convergence rate and running times in this paper. In order to measure convergence behavior, we plot the relative solution error:

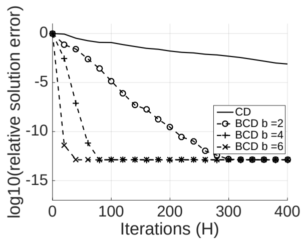
$$\frac{\|w_{opt} - w_h\|_2}{\|w_{opt}\|_2}$$

where w_h is the solution obtained from the coordinate descent algorithms at iteration h and w_{opt} is obtained from conjugate gradients with $tol = 1e{-15}$. We also plot the relative objective error:

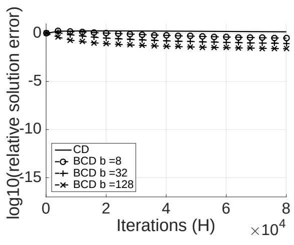
$$\frac{f(X, w_{opt}, y) - f(X, w_h, y)}{f(X, w_{opt}, y)}$$

where $f(X, w, y) = \frac{1}{2n}\|X^T w - y\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$, the primal objective. We use the primal objective to show convergence behavior for BCD, BDCD and their communication-avoiding variants. We explore the tradeoff between the block sizes, b and b', and convergence behavior to test BCD and

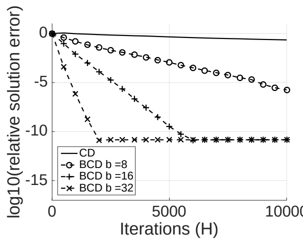
[1]Same as singular values of X^TX, so we use σ_{min} and σ_{max} to avoid overloading definition of λ, the regularization parameter.



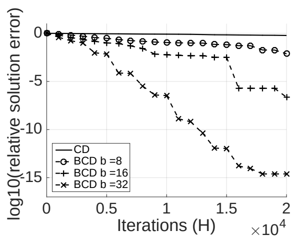
(a) abalone



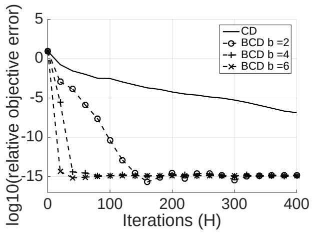
(b) news20



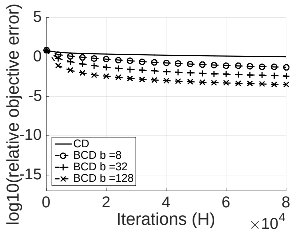
(c) a9a



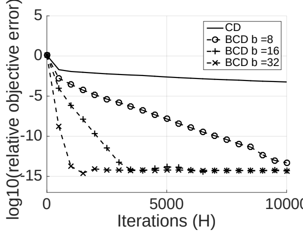
(d) real-sim



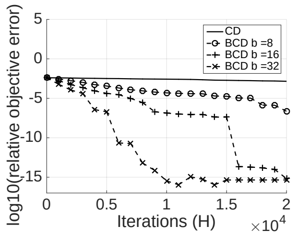
(e) abalone



(f) news20



(g) a9a



(h) real-sim

Figure 2: We compare the convergence of BCD with $b = 1$ (which we refer to as "CD", for coordinate descent, in the plots) and Block Coordinate Descent with $b > 1$ on several machine learning datasets. We show relative solution error (Figures 2a-2d) and objective error (Figures 2e-2h) convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$. The y axes are on $\log_{10}$-scale.

BDCD stability due to the choice of block sizes. Then, we fix the block sizes and explore the tradeoff between s, the loop-blocking parameter, and convergence behavior to study the stability of the communication-avoiding variants due to choice of s. Finally, for both sets of experiments we also plot the algorithm costs against convergence behavior to illustrate the theoretical performance tradeoffs due to choice of block sizes and choice of s. For the latter experiments we assume that the datasets are partitioned optimally for BCD (1D-block row) and BDCD (1D-block column) and for CA-BCD and CA-BDCD, respectively. We plot the sequential flops cost for all algorithms, ignore the $\log P$ factor for the number of messages and ignore constants in all costs.

5.1.1 Block Coordinate Descent

Recall that the BCD algorithm computes a $b \times b$ Gram matrix and solves a b-dimensional subproblem at each iteration. Therefore, one should expect that as b increases the algorithm converges faster but requires more flops and bandwidth per iteration. So we begin by exploring this block size tradeoff by comparing the convergence behavior and costs of coordinate descent (i.e. $b = 1$) to block coordinate descent with $1 < b < d$. Figure 2 plots the convergence behavior of the datasets in Table 3 in terms of the relative objective error and relative solution error.

Figures 2a and 2e show results for the abalone dataset with $b = 2, 4$, and 6. We observe an inverse relationship between the block size and the number of iterations required to converge. Furthermore, the solution error decreases slower than the objective error, which indicates that traditional convergence criterion may not be suitable for machine learning applications where prediction accuracy or objective error are most important. In particular, we see that for the abalone dataset, there is approximately a 2× difference in the number of iterations required to obtain equal accuracy in the solution and objective. The curves illustrate that b is an important parameter in determining the number of iterations required for convergence to a desired accuracy. For $b = 2$ we see more than 2× reduction in the number of iterations required to attain equal accuracy to coordinate descent. This observation holds comparing $b = 4$ and $b = 2$ but diminishes when comparing $b = 6$ and $b = 4$, which implies that b should be chosen carefully to balance the reduction in iterations with the additional computational cost. The objective is sensitive to small fluctuations in the solution due to round-off error and implies a magnifying effect between the objective error and solution error.

Figures 2b and 2f show the convergence behavior of the news20 dataset with $b = 8, 32$ and 128. Since $d > n$ for this dataset, we expect the BCD algorithm to converge slowly. In particular, we see that the residual error actually increases between iterations 1 to 10^4 and then begins to decrease. Since b rows are randomly selected, some coordinates of the solution may not be updated as frequently. As a result, the solution error continues to grow until all coordinates have been updated. As expected, choosing a larger block size ensures that the solution error decreases in fewer iterations. Similar to the abalone dataset, the objective error is smaller than the solution error. Once again we observe that as block size increases, the algorithm converges faster. Figure 2c and 2g show convergence behavior of the a9a dataset for $b = 8, 16$ and 32. Figure 2d and 2h show the convergence behavior of the real-sim dataset for $b = 8, 16$ and 32. The figures illustrate that $b > 1$ converges faster than $b = 1$ and, that one should choose b to balance the cost of additional flops. In both cases, we observe that the objective error converges faster than the solution error.

Figures 3a, 3e, and 3i show the theoretical flops, bandwidth and messages cost of the BCD algorithm for $b = 1, 2, 4,$ and 6. We obtain the costs from Section 4 but assume that flops are computed sequentially[2], ignore the $\log P$ factor for the latency term and ignore constants for all costs[3]. We observed that the flops cost per iteration increases with b, however, the cost per digit of accuracy remains constant as b increases. This observation also holds for the bandwidth cost, which suggests that with $b > 1$ the abalone dataset converges fast enough to offset the additional flops and bandwidth. The latency cost, however, shows that the number of messages decreases as b increases. Selecting a large block size reduces the latency cost significantly while the other costs remain constant per digit of accuracy gained. For the abalone dataset, b should be large in order to maximize performance.

Figure 3b, 3f, and 3j plot the algorithm costs of the news20 dataset with $b = 1, 8, 32,$ and 128. Unlike the abalone dataset, the flops cost in Figure 3b increases with b. If flops are most important, then this plot suggests that smaller b would reduce flops per digit of accuracy. Figure 3f shows that the same conclusion holds for the bandwidth costs. However, Figure 3j illustrates that a larger block size reduces the latency cost per digit of accuracy. If latency is the dominant cost for the news20 dataset, then Figure 3j encourages the selection of large b. The choice of b implies a bandwidth, flops and latency tradeoff.

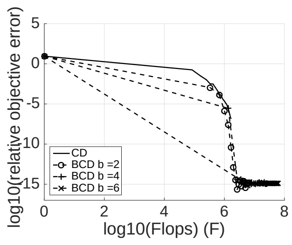
Figures 3c, 3g, and 3k show the BCD costs of the a9a dataset for $b = 1, 8, 16,$ and 32. The flops and bandwidth costs indicate that for $b > 1$ the BCD algorithm is more expensive than for $b = 1$. However, if we ignore the $b = 1$ case, then the flops and bandwidth costs encourage the selection of large b since the cost per digit of accuracy are approximately equal. Figure 3k shows that selecting $b > 1$ reduces the latency cost and encourages the selection of large b. For the a9a dataset, the plots indicate that $b = 1$ reduces flops and bandwidth costs while $b > 1$ reduces the latency cost.

Finally, Figures 3d, 3h, and 3l show the costs of the real-sim dataset with $b = 1, 8, 16,$ and 32. The flops and bandwidth costs are reduced when $b = 1$, while the latency cost is reduced when $b = 32$. Depending on the hardware parameters b should be chosen to balance the computation and communication costs for the real-sim dataset.

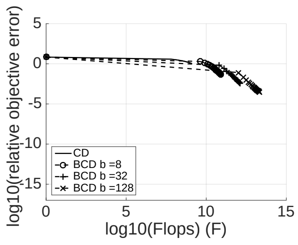
In summary, we observe that for all datasets tested larger block sizes are encouraged on

[2]For example, Flops BCD $= Hb^2n + Hb^3$.

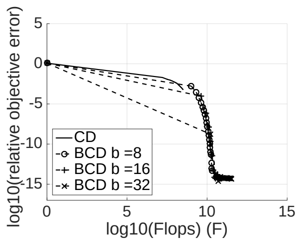
[3]Constants and low-order terms shifts all curves proportionally to the right but does not alter our conclusions significantly.



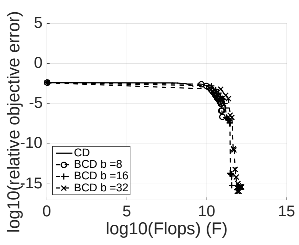
(a) abalone



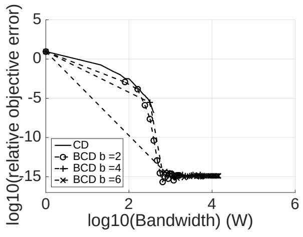
(b) news20



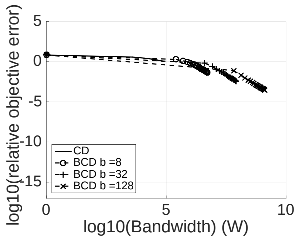
(c) a9a



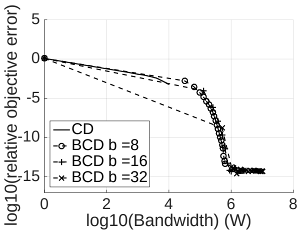
(d) real-sim



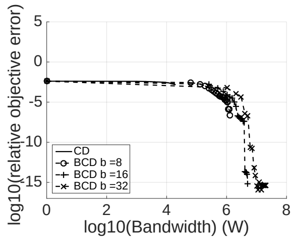
(e) abalone



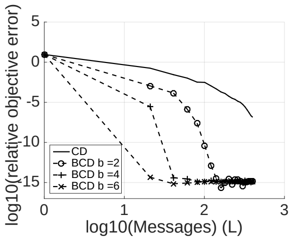
(f) news20



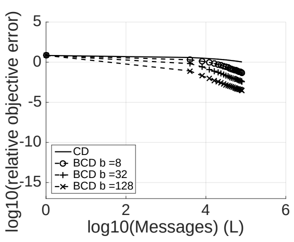
(g) a9a



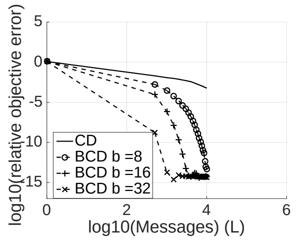
(h) real-sim



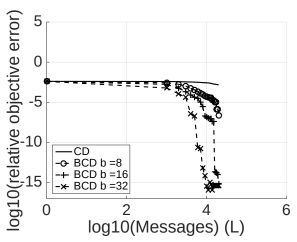
(i) abalone



(j) news20



(k) a9a



(l) real-sim

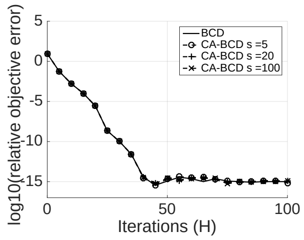
Figure 3: We compare the algorithm costs of BCD with $b = 1$ (which we refer to as "CD", for coordinate descent, in the plots) and BCD with $b > 1$ on several machine learning datasets. Flops cost (Figures 3a-3d), bandwidth cost (Figures 3e-3h), and messages cost (Figures 3i-3l) versus convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$. The objective error is first measured at iteration $H = 0$ and then re-computed at regular intervals. This interval accounts for the large gap between the first and second data points in the plots.

machines where latency is the dominant cost. However, in general, our results show that b should be chosen to balance the flops, bandwidth and latency costs.

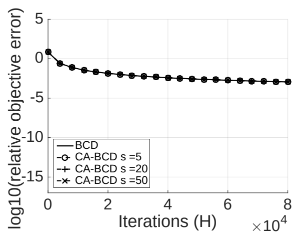
5.1.2 Communication-Avoiding Block Coordinate Descent

Our derivation of the CA-BCD algorithm showed that by unrolling the iteration loop we can reduce the latency cost of the BCD algorithm by a factor of s. However, this comes at the cost of computing a larger $sb \times sb$ Gram matrix whose condition number is larger than the $b \times b$ Gram matrix computed in the BCD algorithm. The larger condition number implies that the CA-BCD algorithm may not be stable for $s > 1$ due to round-off error. Therefore, we begin by experimentally showing the convergence behavior of the CA-BCD algorithm on the datasets in Table 3.

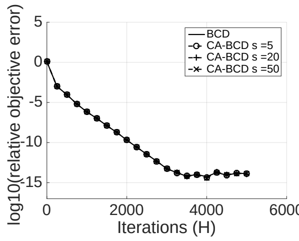
Figures 4a and 4e show the convergence behavior of the abalone dataset in terms of the objective error (Figure 4a) and the solution error (Figure 4e) with $s = 5, 20$, and 100. We fix the block size to $b = 4$ for both BCD and CA-BCD so that only s varies. We observe that the CA-BCD convergence for all values of s matches the convergence of BCD in terms of the objective error and the solution error. We see some deviation in the objective error for less than 10^{-14}



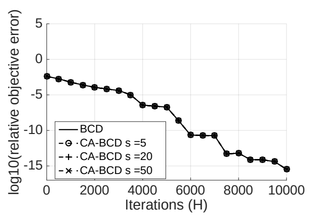
(a) abalone



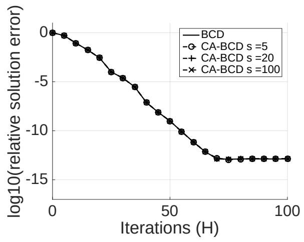
(b) news20



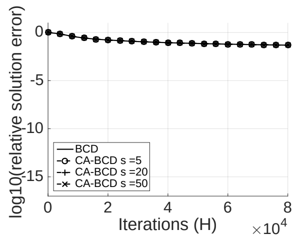
(c) a9a



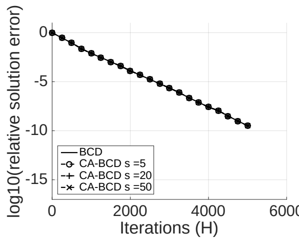
(d) real-sim



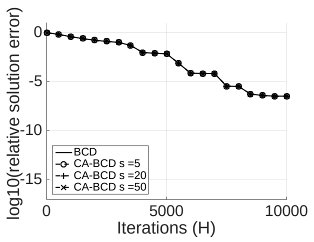
(e) abalone



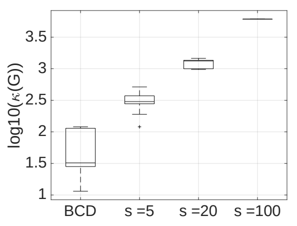
(f) news20



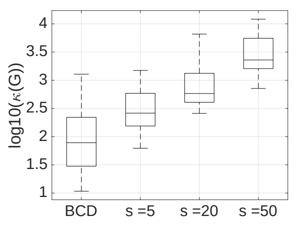
(g) a9a



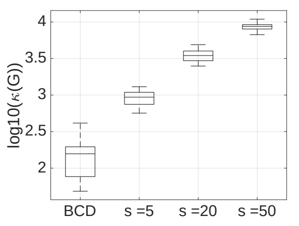
(h) real-sim



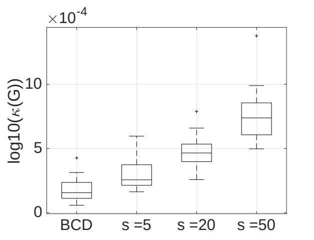
(i) abalone



(j) news20



(k) a9a



(l) real-sim

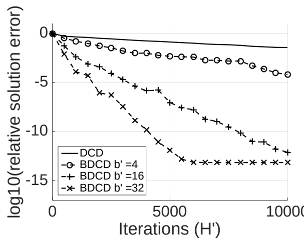
Figure 4: We compare the convergence behavior of BCD and CA-BCD with various values of s on several machine learning datasets. Relative objective error (Figures 4a-4d), relative solution error (Figures 4e-4h), and Gram matrix condition numbers (Figures 4i-4l) versus convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$: abalone with $b = 4$, news20 with $b = 64$, a9a with $b = 16$, and real-sim with $b = 32$. The y axes are on $\log_{10}$-scale. Figure 4l contains values very close to 10^0, therefore the y-axis is scales by 10^{-4} to show separation for various values of s.

accuracy, which can be attributed to small-magnitude error once the solution error is less than 10^{-12}. Figure 4i plots the statistics of the Gram matrix condition number over all iterations. We see that the Gram matrix condition number increases with s but is reasonably small. In other words, the Gram matrix is well-conditioned even in the $s = H = 100$ setting where CA-BCD performs a single-pass over the data and communicates only once.

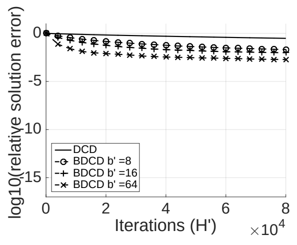
Figures 4b and 4f show the convergence behavior of the news20 dataset with $s = 5, 20, 50$, and $b = 64$. We see that BCD and CA-BCD converge slowly since the news20 dataset is ill-conditioned. Aside from the slow convergence, CA-BCD for all settings of s matches the convergence of BCD. Figure 4j shows that the Gram matrix condition numbers do not increase drastically with s and indicate that larger values of s will be stable. Since $s > 50$ is likely to be numerically stable, the maximum choice of s is limited by the algorithm costs tradeoffs rather than numerical stability.

Figures 4c and 4g illustrate that CA-BCD with $s = 5, 20, 50$, and $b = 16$ for the a9a dataset is numerically stable. Larger values of s are possible given that the largest condition number is $O\left(10^4\right)$.

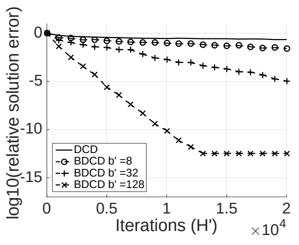
Figures 4d and 4h show that the same conclusion holds for the real-sim dataset with $b = 32$. In summary, our experiments show that CA-BCD has the same convergence behavior as BCD on



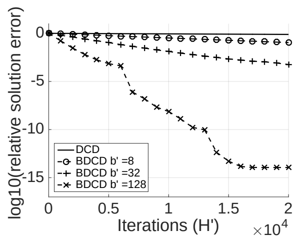
(a) abalone



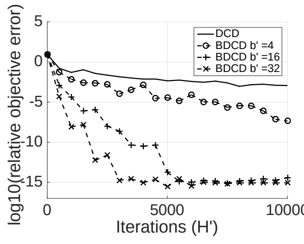
(b) news20



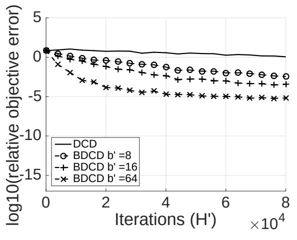
(c) a9a



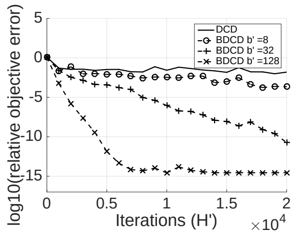
(d) real-sim



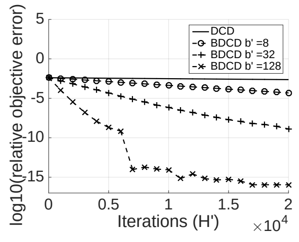
(e) abalone



(f) news20



(g) a9a



(h) real-sim

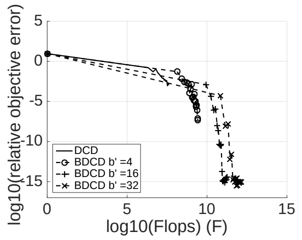
Figure 5: We compare the convergence of BDCD with $b' = 1$ (which we refer to as "DCD", for dual coordinate descent, in the plots) and BDCD with $b' > 1$ on several machine learning datasets. Relative solution error (Figures 5a-5d) and objective error (Figures 5e-5h) convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$. The y axes are on $\log_{10}$-scale.

all datasets tested. Although s can be large, practical values of s should be determined based on algorithm costs and corresponding hardware parameters.

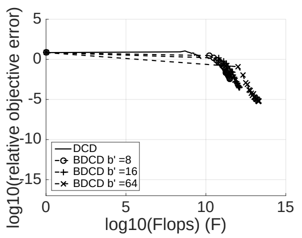
5.1.3 Block Dual Coordinate Descent

The BDCD algorithm solves the dual of the regularized least-squares problem by computing a $b' \times b'$ Gram matrix obtained from the columns of X (instead of the rows of X for BCD) and solves a b-dimension subproblem at each iteration. Similar to BCD, we expect that as b' increases, the BDCD algorithm converges faster at the cost of more flops and bandwidth. We explore this tradeoff space by comparing the convergence behavior and algorithm costs of dual coordinate descent (i.e. BDCD with $b' = 1$) to BDCD with $1 < b' < n$. We perform experiments on the datasets in Table 3 for various block sizes and measure the relative objective error and relative solution error.

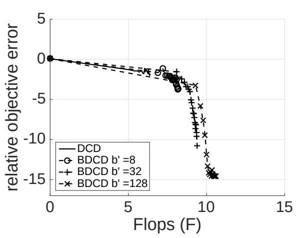
Figure 5a and 5e plot the solution error and objective error, respectively, of the BDCD algorithm on the abalone dataset with $b' = 1, 4, 16$, and 32. As b' increases the BDCD algorithm converges faster in both the solution error and the objective error. Similar to the BCD algorithm, there exists an approximately $2\times$ difference in the solution error and the objective error. However, unlike the BCD algorithm, BDCD requires $O(100)$ times more iterations to converge to the same level of accuracy. This is due to the fact that BDCD chooses b' data points at each iteration from the dataset. Since $n > d$ for the abalone dataset, BDCD requires a much larger block size in order to converge to the same accuracy as BCD, if the number of iterations are fixed. Figure 5b and 5f show the solution error and objective error for the news20 dataset with $b' = 1, 8, 16$, and 64. As b' increases the BDCD algorithm converges proportionally faster per iteration and, in comparison to BCD, attains a higher accuracy in both the solution and the objective for a fixed number of iterations. BDCD attains a higher accuracy in fewer iterations due to the shape of the new20 dataset. Since $n < d$, BDCD selects a proportionally larger block size than BCD, which leads to more updates to each coordinate of the solution vector. This suggests that by



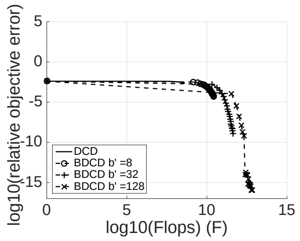
(a) abalone



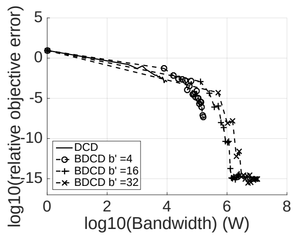
(b) news20



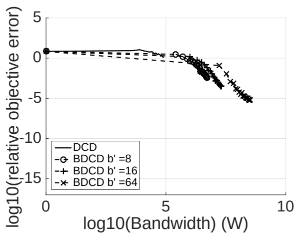
(c) a9a



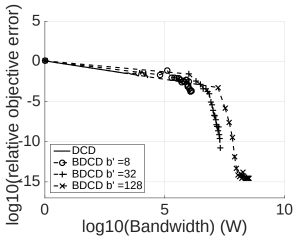
(d) real-sim



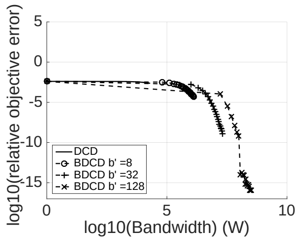
(e) abalone



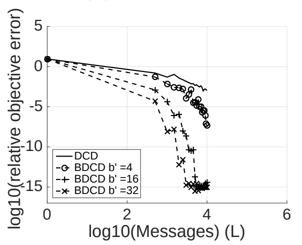
(f) news20



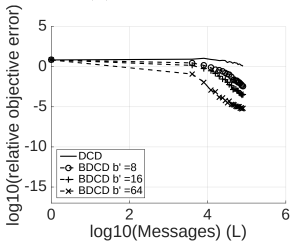
(g) a9a



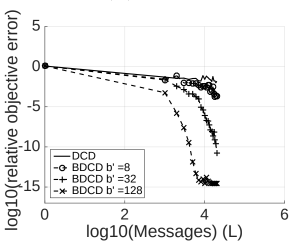
(h) real-sim



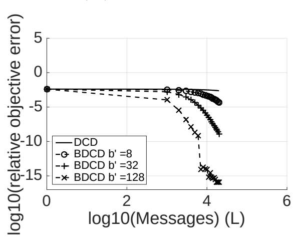
(i) abalone



(j) news20



(k) a9a

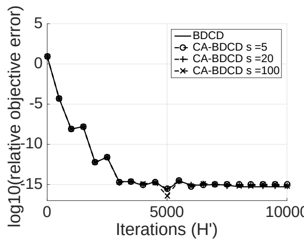


(l) real-sim

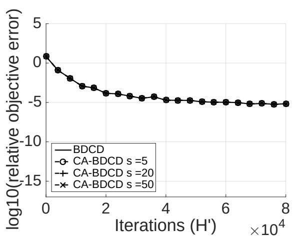
Figure 6: We compare the algorithm costs of BDCD with $b' = 1$ (which we refer to as "DCD", for dual coordinate descent, in the plots) and BDCD with $b' > 1$ on several machine learning datasets. Flops cost (Figures 6a-6d), bandwidth cost (Figures 6e-6h), and messages cost (Figures 6i-6l) versus convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$. The x and y axes are on $\log_{10}$-scale. The objective error is first measured at iteration $H' = 0$ and then re-computed at regular intervals. This interval accounts for the large gap between the first and second data points in the plots.

selecting block sizes that are proportional to the dimensions of the matrix, the primal and dual methods can converge to the same accuracy for a fixed number of iterations (modulo the effects of randomized sampling). Figures 5c and 5g show the solution and objective error for the a9a dataset with $b' = 1, 8, 32$, and 128. As b' increases the algorithm converges faster than for $b' = 1$. However, the objective and solution errors fluctuate since BDCD performs block updates on a vector in the large dimension. Therefore, selecting b' proportional to n is important in ensuring that BDCD converges quickly. Finally, Figures 5d and 5h show the convergence behavior for the real-sim dataset with $b' = 1, 8, 32$, and 128. Similar to the other datasets as b' increases, the BDCD algorithm converges faster.

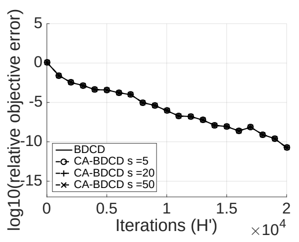
Figure 6 illustrates the computation and communication costs of the BDCD algorithm against the objective error. Figures 6a, 6e, and 6i show the flops, bandwidth and messages cost of the abalone dataset with $b' = 1, 4, 16$, and 32. The flops and bandwidth cost plots indicate that small values of b' are required if flops or bandwidth are the dominant terms. However, if the latency cost dominates, then large values of b' are preferable. In general, b' should be selected to balance the flops, bandwidth, latency costs and desired accuracy. In comparison to the BCD



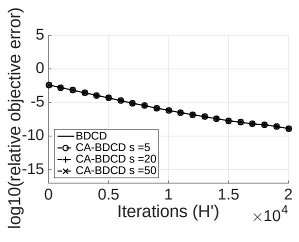
(a) abalone



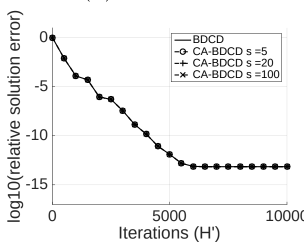
(b) news20



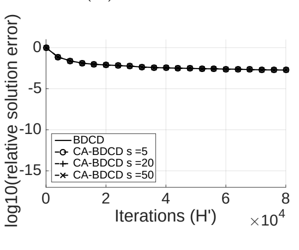
(c) a9a



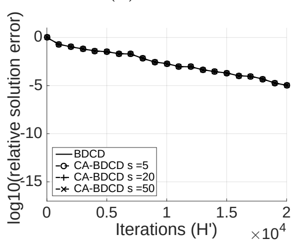
(d) real-sim



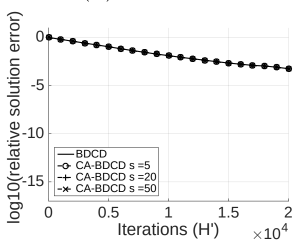
(e) abalone



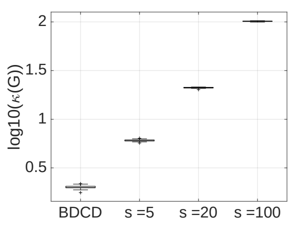
(f) news20



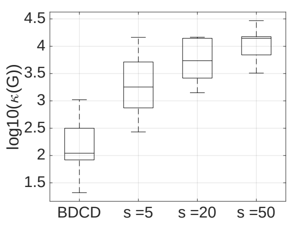
(g) a9a



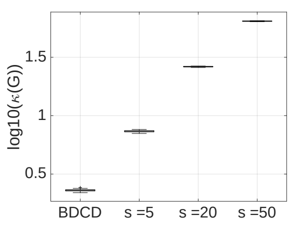
(h) real-sim



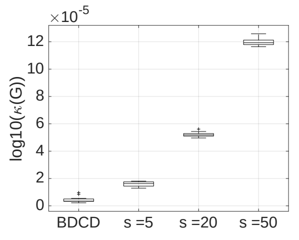
(i) abalone



(j) news20



(k) a9a



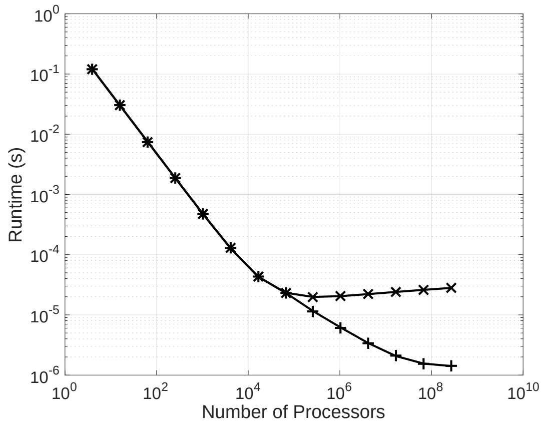
(l) real-sim

Figure 7: We compare the convergence behavior of BDCD and CA-BDCD with various values of s on several machine learning datasets. Relative objective error (Figures 7a-7d), relative solution error (Figures 7e-7h), and Gram matrix condition numbers (Figures 7i-7l) versus convergence plots of datasets in Table 3 with $\lambda = 1000\sigma_{min}$: abalone with $b' = 32$, news20 with $b' = 64$, a9a with $b' = 32$ and real-sim with $b' = 32$. The y axes are on $\log_{10}$-scale. Figure 7l contains values very close to 10^0, therefore the y-axis is scales by 10^{-5} to show separation for various values of s.

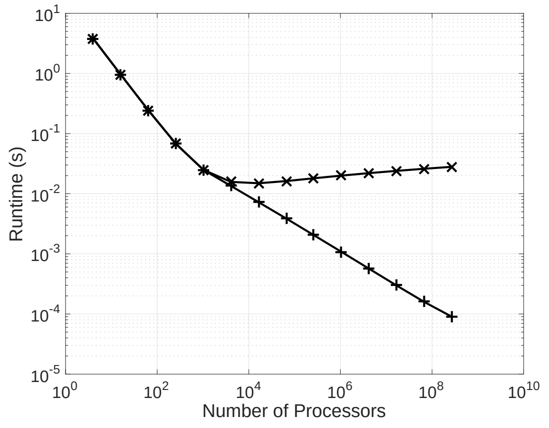
costs (in Figure 3), the BDCD algorithm requires orders of magnitude more flops, bandwidth and messages to converge to the same accuracy. The remaining figures exhibit the same tradeoffs between the computation and communication costs. However, when compared to the costs of the BCD algorithm, the best algorithm depends on the shape of the dataset and the choice of block size. Furthermore, due to the fully randomized block selection (i.e. we do not cycle through blocks of coordinates), the shape of the dataset affects the convergence of the primal and dual methods, since coordinate descent in the large dimension will, for this random sampling scheme, require more iterations.

5.1.4 Communication-Avoiding Block Dual Coordinate Descent

The CA-BDCD algorithm avoids communication in the dual problem by unrolling the iteration loop by a factor of s. This allows us to reduce the latency cost computing a larger $sb' \times sb'$ Gram matrix whose condition number is larger than the $b' \times b'$ Gram matrix computed in the BDCD algorithm. The larger condition number implies that the CA-BDCD algorithm may not be stable, so we begin by experimentally showing the convergence behavior of the CA-BCD algorithm on the datasets in Table 3.



(a) Cori Strong Scaling



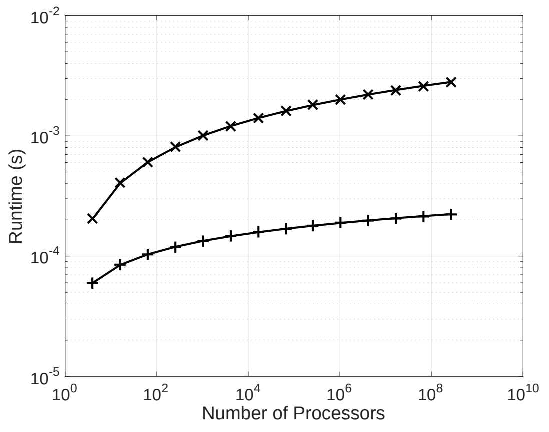
(b) Spark Strong Scaling

Figure 8: Strong scaling plots of BCD ($\times$) and CA-BCD ($+$) with block size, $b = 4$, on NERSC Cori using MPI with $d = 1024, n = 2^{35}$(left) and using Spark with $d = 1024, n = 2^{40}$ (right). Ideal strong scaling, in both plots, would be $2\times$ decrease in running time with $2\times$ increase in number of processors (P).

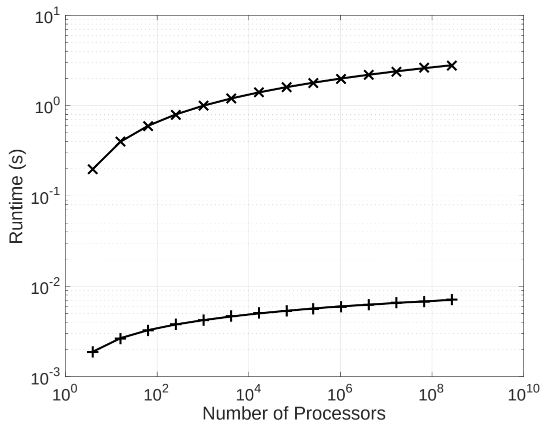
Figures 7a and 7e show the convergence behavior of the abalone dataset in terms of the objective error (Figure 7a) and the solution error (Figure 7e) with $s = 5, 20$, and 100. We fix the block size to $b' = 32$ for both BDCD and CA-BDCD. The plots illustrate that the CA-BDCD convergence for all values of s matches the convergence of BCD in terms of the objective error and the solution error. We see some fluctuation in the objective error for less than 10^{-14} accuracy, which can be attributed to small fluctuations in the solution error. Figure 7i shows the statistics of the Gram matrix condition number over all iterations. From this plot we see that the Gram matrix condition number increases with s but is only $O(10^2)$ for $s = 100$. This suggests that s can be much larger without adversely affecting the stability of our communication-avoiding algorithm. Figures 7b and 7f show the convergence behavior of the news20 dataset with $s = 5, 20, 50$, and $b' = 64$. We see that although BDCD and CA-BDCD converge slowly, their convergence curves match for all choices of s. Figure 7j shows that the Gram matrix condition numbers do not increase drastically with s and indicate that larger values of s will be stable. Figures 7c, 7g, and 7k show that CA-BDCD with $s = 5, 20, 50$, and $b' = 32$ is numerically stable for all tested values of s on the a9a dataset. Much larger values of s are possible given that the largest condition number is $O\left(10^{1.8}\right)$. Figures 7d, 7h, and 7l show that the same conclusion holds for the real-sim dataset with $s = 5, 20, 50$, and $b' = 32$. In conclusion, our experiments show that CA-BDCD has the same convergence behavior as BDCD on all datasets and values of s tested. Although s can be large, practical values of s should be determined based on algorithm costs and corresponding hardware parameters.

5.2 Modeled Performance Experiments

We have shown that the communication-avoiding variants are numerically stable and reduce the number of messages sent by a factor of s. Our results in the previous section indicate that s can be quite large in theory without altering the convergence behavior. However, s cannot be too large in practice due to the additional flops and bandwidth costs. In this section, we model the running time of the BCD and CA-BCD algorithms with $b = 4$ using results from Section 4. We use the running time model introduced in Section 2, $T = \gamma F + \alpha L + \beta W$, to obtain strong and



(a) Cori Weak Scaling



(b) Spark Weak Scaling

Figure 9: Weak scaling plots of BCD ($\times$) and CA-BCD ($+$) with block size, $b = 4$, on NERSC Cori using MPI (left) and using Spark (right) with $d = 1024$, $\frac{n}{P} = 2^{11}$ for both. Ideal weak scaling, in both plots, would be constant running time for $2\times$ increase in the number of processors (P).

weak scaling speedups along the critical path of the BCD and CA-BCD algorithms.

We assume that each processor has infinite local memory and can execute each flop at peak machine rate. In the parallel setting, we assume that communication costs dominate the local flops on each processor. Our experiments consider two programming models, MPI and Spark on Cori[4], the latest supercomputing architecture at NERSC. The Cori supercomputer has hardware parameters: $\gamma = 8 \cdot 10^{-13}$, $\alpha = 1 \cdot 10^{-6}$ and $\beta = 1.3 \cdot 10^{-10}$[1]. We assume that MPI can run at the hardware peak. For Spark we assume that the flops and bandwidth rates are not altered, but increase the latency to $\alpha = 1 \cdot 10^{-3}$ due to the scheduling and centralization overhead incurred while performing tree reductions in Spark [20].

We begin with strong scaling and assume that the dataset is dense and partitioned appropriately so that flops are load balanced (i.e. 1D-column partitioned). For the MPI model we use a problem size with $n = 2^{35}$ columns and assume d is large enough to at least select b rows without replacement. For the Spark experiments we increase the problem size to $n = 2^{40}$ columns. In both cases, we perform our experiments with $P \in \{2^2 \ldots 2^{28}\}$. Figure 8 shows the strong scaling results using MPI and Spark on Cori. In Figure 8a we see that BCD initially scales perfectly since the algorithm is dominated by flops. However, as the local problem size decreases communication dominates. On the other hand, CA-BCD matches the scaling of BCD initially – since flops dominate, $s = 1$ is the best choice – and continues to scale past BCD. In Figure 8b, BCD becomes communication-dominated much more quickly due to the additional Spark overheads. However, CA-BCD continues to scale almost linearly for the entire processor range. The best speedups we attained were $14\times$ for MPI and $165\times$ for Spark with $s = 40$ and $s = 600$, respectively.

Figure 9 shows the weak scaling results using MPI and Spark on Cori with the same processor range. The problem size is fixed so that $\frac{n}{P} = 2^{11}$ and assume once again that d is at least large enough to select b rows without replacement. In Figure 9a we see that initially when P is small and flops dominate, the BCD and CA-BCD gap is small. However, as P increases and communication dominates, the gap widens. For all processor counts, CA-BCD is faster than BCD. In Figure 9b, we see a much larger initial gap due to the communication overhead. This gap continues to widen as the number of processors are increased. In summary, the best speedups

[4]`http://www.nersc.gov/users/computational-systems/cori`

we attained were 12× for MPI and 396× for Spark with $s = 25$ and $s = 750$, respectively.

We have shown in this section that block versions of coordinate descent and dual coordinate descent are preferable due to faster convergence and smaller latency costs. We showed the tradeoffs between the primal and dual methods as a function of dataset shape and block size on convergence behavior and algorithm costs. We experimentally showed that the new communication-avoiding variants are numerically-stable for all choices of s tested on all datasets and showed that they reduce the latency cost by s. Finally, we illustrated modeled speedups of 14× (strong scaling) and 12× (weak scaling) on a model problem using MPI and 165× (strong scaling) and 396× (weak scaling) using Spark on a current supercomputer.

6 Conclusion and Future Work

In this paper, we have shown how to extend the communication-avoiding technique of CA-Krylov subspace methods to block coordinate descent and block dual coordinate descent algorithms in machine learning. We showed that when considering passes over the dataset, BCD and BDCD methods converge faster than traditional Krylov methods and analyzed the computation, communication and storage costs of the classical and communication-avoiding variants. Our experiments showed that CA-BCD and CA-BDCD are numerically stable algorithms for all values of s tested, show the tradeoff between algorithm parameters and convergence. We also showed that the communication-avoiding variants can attain large modeled speedups on a modern supercomputer using MPI or Spark.

There are several open questions and directions for future work yet to be explored. Distributed-memory MPI and Spark implementations of the algorithms described and comparison to existing parallel BCD and BDCD methods is our most immediate goal. BCD and BDCD methods are especially important when applied to solving the kernel ridge regression problem where standard Krylov methods are more expensive. The algorithms developed in this work can also be applied to the kernelized regression problem, but we leave this for future work. Our communication-avoiding technique may apply to many other iterative machine learning algorithms. In particular, extensions to non-linear loss functions, BFGS, L-BFGS and other quasi-Newton methods would be particularly interesting.

Acknowledgements

We thank Grey Ballard, Penporn Koanantakool and Evangelos Georganas for helpful comments and discussions. AD is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 1106400. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract Nos. DE-AC02-05CH11231, DE-SC0010200, and DE-SC0008700. This work is supported by Cray, Inc. and the Defense Advanced Research Projects Agency XDATA program. Research partially funded by ASPIRE Lab industrial sponsors and affiliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] NERSC Cori configuration. `http://www.nersc.gov/users/computational-systems/cori/configuration/`.

[2] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model for parallel computation. *Journal of parallel and distributed computing*, 44(1):71–79, 1997.

[3] G. Ballard. *Avoiding Communication in Dense Linear Algebra.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2013.

[4] G. Ballard, E. Carson, J. Demmel, M Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 2014.

[5] Ã. Björck. *Numerical Methods for Least Squares Problems.* Society for Industrial and Applied Mathematics, 1996.

[6] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of Computation Statistics*, pages 177–186. Springer, 2010.

[7] J. Bruck, C-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, 1997.

[8] E. Carson. *Communication-Avoiding Krylov Subspace Methods in Theory and Practice.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2015.

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

[10] A. T. Chronopoulos and C. D. Swanson. Parallel iterative s-step methods for unsymmetric linear systems. *Parallel Computing*, 22(5):623–641, 1996.

[11] A.T. Chronopoulos and C.W. Gear. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Computing*, 11(1):37 – 53, 1989.

[12] A.T. Chronopoulos and C.W. Gear. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25(2):153 – 168, 1989.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, and T. Subramonian, R.and Von Eicken. *LogP: Towards a realistic model of parallel computation*, volume 28. ACM, 1993.

[14] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-avoiding parallel and sequential QR and LU factorizations. *SIAM Journal of Scientific Computing*, 2008.

[15] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical Report UCB/EECS-2007-123, EECS Department, University of California, Berkeley, Oct 2007.

[16] Message Passing Interface Forum. MPI: A message-passing interface standard, 1994.

[17] K. Fountoulakis and J. Gondzio. Performance of First- and Second-Order Methods for L1-Regularized Least Squares Problems. *ArXiv e-prints*, March 2015.

[18] K. Fountoulakis and R. Tappenden. Robust Block Coordinate Descent. *ArXiv e-prints*, July 2014.

[19] S. H. Fuller and L. I. Millett. Computing performance: Game over or next level? *Computer*, (1):31–38, 2011.

[20] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, P. Sharma, J. Yang, J. Demmel, J. Harrell, V. Krishnamurthy, M. W. Mahoney, and Prabhat. Matrix Factorization at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies. *ArXiv e-prints*, July 2016.

[21] G. H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM (JACM)*, 28(2):289–304, 1981.

[22] S. L. Graham, M. Snir, and C. A. Patterson. *Getting up to speed : the future of supercomputing*. National Academies Press, Washington, DC, 2005.

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

[24] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. Technical report, September 2014.

[25] S.K. Kim and A.T. Chronopoulos. An efficient nonsymmetric Lanczos method on parallel vector computers. *Journal of Computational and Applied Mathematics*, 42(3):357 – 374, 1992.

[26] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Machine Learning Conference*, 1995.

[27] M. Lichman. UCI machine learning repository, 2013.

[28] J. Mareček, P. Richtárik, and M. Takáč. Distributed block coordinate descent for minimizing partially separable functions. In *Numerical Analysis and Optimization*, pages 261–288. Springer, 2015.

[29] A. McCallum. SRAA: Simulated/real/aviation/auto usenet data. `https://people.cs.umass.edu/~mccallum/data.html`.

[30] M. D. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, EECS Department, University of California, Berkeley, 1996.

[31] M. Mohiyuddin. *Tuning Hardware and Software for Multiprocessors*. PhD thesis, EECS Department, University of California, Berkeley, May 2012.

[32] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 36:1–36:12, New York, NY, USA, 2009. ACM.

[33] M. Raab and A. Steger. "balls into bins" – a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer, 1998.

[34] B. Recht, C. Ré, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[35] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1):1–38, 2014.

[36] Y. Saad. *Iterative methods for sparse linear systems.* SIAM, 2003.

[37] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[38] E. Solomonik. *Provably efficient algorithms for numerical tensor algebra.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2014.

[39] J. Stamper, A. Niculescu-Mizil, S. Ritter, G.J. Gordon, and K.R. Koedinger. Algebra 2008-2009 from challenge data set. In *KDD Cup 2010 Educational Data Mining Challenge*, 2010.

[40] M. Takáč, P. Richtárik, and N. Srebro. Distributed mini-batch SDCA. *CoRR*, abs/1507.08322, 2015.

[41] R. Thakur and W. D. Gropp. Improving the performance of MPI collective communication on switched networks. November 2002.

[42] R. Thakur and W. D. Gropp. Improving the performance of collective operations in mpich. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 257–267. Springer, 2003.

[43] J. Van Rosendale. *Minimizing inner product data dependencies in conjugate gradient iteration.* IEEE Computer Society Press, Silver Spring, MD, Jan 1983.

[44] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

[45] S. Williams, M. Lijewski, A. Almgren, B. Van Straalen, E. Carson, N. Knight, and J. Demmel. s-step Krylov subspace methods as bottom solvers for geometric multigrid. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1149–1158. IEEE, 2014.

[46] S. J. Wright. Coordinate descent algorithms. *Math. Program.*, 151(1):3–34, June 2015.

[47] H.-F. Yu, H.-Y. Lo, H.-P. Hsieh, J.-K. Lou, T. G. Mckenzie, J.-W. Chou, P.-H. Chung, C.-H. Ho, C.-F. Chang, J.-Y. Weng, E.-S. Yan, C.-W. Chang, T.-T. Kuo, P. T. Chang, C. Po, C.-Y. Wang, Y.-H Huang, Y.-X. Ruan, Y.-S. Lin, S.-D. Lin, H.-T. Lin, and C.-J. Lin. Feature engineering and classifier ensemble for kdd cup 2010. In *JMLR Workshop and Conference Proceedings*, 2011.

[48] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.

[49] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603. 2010.