SIAM J. SCI. COMPUT.
Vol. 41, No. 1, pp. C1–C27

AVOIDING COMMUNICATION IN PRIMAL AND DUAL BLOCK COORDINATE DESCENT METHODS*

ADITYA DEVARAKONDA†, KIMON FOUNTOULAKIS‡, JAMES DEMMEL§, AND MICHAEL W. MAHONEY†

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 reorganizing 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 show that the communication-avoiding variants reduce the number of synchronizations by a factor of s on distributed-memory parallel machines without altering the convergence rate and attain strong scaling speedups of up to 6.1× over the "standard algorithm" on a Cray XC30 supercomputer.

Key words. primal and dual methods, communication-avoiding algorithms, block coordinate descent, ridge regression

AMS subject classifications. 15A06, 62J07, 65Y05, 68W10

DOI. 10.1137/17M1134433

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, and this gap is continuing to increase.

Communication-avoiding (CA) algorithms are a new class of algorithms that exhibit large speedups on modern parallel architectures through careful algorithmic

*Submitted to the journal's Software and High-Performance Computing section June 13, 2017; accepted for publication (in revised form) October 15, 2018; published electronically January 17, 2019.

http://www.siam.org/journals/sisc/41-1/M113443.html

Funding: The first author's work was supported by a National Science Foundation Graduate Research Fellowship under grant 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 contracts DE-AC02-05CH11231, DE-SC0010200, and DE-SC0008700. This work was supported by Cray, Inc. under grant 47277 and the Defense Advanced Research Projects Agency XDATA program. This research was partially funded by ASPIRE Lab industrial sponsors and affiliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung.

†EECS Department, University of California, Berkeley, Berkeley, CA 94709 (aditya@eecs.berkeley.edu).

‡ICSI and Statistics Department, University of California, Berkeley, Berkeley, CA 94709 (kfount@berkeley.edu, mmahoney@stat.berkeley.edu).

§Mathematics and EECS Department, University of California, Berkeley, Berkeley, CA 94709 (demmel@berkeley.edu).

TABLE 1

Ops (F), Latency (L), Bandwidth (W), and Memory per processor (M) costs comparison along the critical path of classical BCD (Theorem 4.1), BDCD (Theorem 4.2) and communication-avoiding BCD (Theorem 4.6), BDCD (Theorem 4.7) 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 sparse with fdn nonzeros that are uniformly distributed, $0 < f \leq 1$ is the density of X, P is the number of processors, and s is the recurrence unrolling parameter. We assume that the $b \times b$ and $b' \times b'$ Gram matrices computed at each iteration for BCD and BDCD, respectively, are dense.

Summary of Ops and Memory costs

Algorithm	Data layout	Ops cost (F)	Memory cost (M)
BCD	1D-column	$O\left(\frac{Hb^2fn}{P} + Hb^3\right)$	$O\left(\frac{fdn+n}{P} + b^2 + d\right)$
CA-BCD		$O\left(\frac{Hb^2sfn}{P} + Hb^3\right)$	$O\left(\frac{fdn+n}{P} + b^2s^2 + d\right)$
BDCD	1D-row	$O\left(\frac{H'b'^2fd}{P} + H'b'^3\right)$	$O\left(\frac{fdn+d}{P} + b'^2 + n\right)$
CA-BDCD		$O\left(\frac{H'b'^2sfd}{P} + H'b'^3\right)$	$O\left(\frac{fdn+d}{P} + b'^2s^2 + n\right)$

Summary of Communication costs

Algorithm	Data layout	Latency cost (L)	Bandwidth cost (W)
BCD	1D-column	$O\left(H \log P\right)$	$O\left(Hb^2 \log P\right)$
CA-BCD		$O\left(\frac{H}{s} \log P\right)$	$O\left(Hb^2s \log P\right)$
BDCD	1D-row	$O\left(H' \log P\right)$	$O\left(H'b'^2 \log P\right)$
CA-BDCD		$O\left(\frac{H'}{s} \log P\right)$	$O\left(H'b'^2s \log P\right)$

transformations [2]. Much of numerical linear algebra has been reorganized to avoid communication and has led to significant performance improvements over existing state-of-the-art libraries [2, 1, 5, 18, 35, 40]. The results from CA-Krylov subspace methods [5, 15, 18] are particularly relevant to our work. Demmel, Hoemmen, Mohiyuddin, and others [15, 18, 25, 26] introduced 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. Their work extended existing s-step Krylov methods research [38, 11, 12, 20, 39] by combining the matrix powers kernel with extensively modified s-step Krylov methods to avoid communication [5, 15, 18].

We extend the CA technique 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 [28, 32, 41]. Given a sparse dataset $X \in \mathbb{R}^{d\times n}$ where the rows are features of the data and the columns are data points, the block coordinate descent method can compute the regularized or unregularized least-squares solution by iteratively solving a subproblem using a block of b rows of X [28, 32, 41]. This process is repeated until the solution converges to a desired accuracy or until the number of iterations has reached a user-defined limit. If X is distributed (in 1D-row or 1D-column layout) across P processors, then the algorithm communicates at each iteration in order to solve the subproblem. As a result, the running time for such methods is often dominated by communication cost, which increases with P.

There are some frameworks and algorithms that attempt to reduce the communication bottleneck. For example, the CoCoA (communication-efficient distributed dual coordinate ascent) framework [19] 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, but changes the convergence

TABLE 2

Critical path costs of Krylov methods. k is the number of iterations required for Krylov methods to converge to a desired accuracy. We assume a 1D-block row layout if $n < d$ (1D-block column if $n > d$), replicate the $\min(d, n)$-dimensional vectors, and partition the $\max(d, n)$-dimensional vectors.

Ops and Memory costs comparison		
Algorithm	Ops cost (F)	Memory cost (M)
Krylov methods [2]	$O\left(\frac{kfdn}{P}\right)$	$O\left(\frac{fdn}{P} + \min(d,n) + \frac{\max(d,n)}{P}\right)$

Communication costs comparison		
Algorithm	Latency cost (L)	Bandwidth cost (W)
Krylov methods [2]	$O(k \log P)$	$O\left(k \min(d,n) \log P\right)$

behavior. HOGWILD! [31] is a shared-memory, lock-free approach to stochastic gradient descent (SGD) where each processor selects a data point, computes a gradient associated with only that data point, and atomically 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 on shared-memory machines.

In contrast, our results reduce the latency cost (at the expense of additional flops and bandwidth) 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. Our results show that our CA methods attain speedups despite the increase in flops and bandwidth costs. Table 1 summarizes the critical path costs of the algorithms considered in this paper. Hereafter we refer to the primal method as block coordinate descent (BCD) and the dual method as block dual coordinate descent (BDCD). There are several variants of BCD and BDCD in the literature; however, the algorithms we consider can also be referred to as subspace descent methods [29]. The proofs in this paper assume that X is sparse with fdn nonzeros, where $0 < f \leq 1$ is the density of X (i.e., $f = \frac{nnz(X)}{dn}$). We further assume that the nonzeros are uniformly distributed between the rows for BCD and columns for BDCD. Each iteration of BCD samples b rows of X (resp., b' columns of X for BDCD), uniformly at random without replacement. The resulting $b \times n$ (resp., $d \times b'$ for BDCD) sampled matrix contains fbn (resp., $fb'd$ for BDCD) nonzeros. These assumptions simplify our analysis and provide insight into scaling behavior for ideal sparse inputs. We leave extensions of our proofs to general sparse matrices for future work. Other sparsity models like assuming $nnz(X) = \xi n$ where $0 < \xi \leq d$ (i.e., ξ nonzeros per column) or $nnz(X) = \omega d$ where $0 < \omega \leq n$ (i.e., ω nonzeros per row) can also be utilized for the analysis. However, we use the fdn nonzeros with $0 < f \leq 1$ model since it allows for more interpretable bounds and comparison between the primal (BCD) and dual (BDCD) algorithms.

1.1. Contributions. We briefly summarize our contributions:

- We present CA algorithms for BCD and BDCD that *provably* reduce the latency cost by a factor of s.
- We analyze the operational, communication, and storage costs of the classical and our new CA algorithms under two data partitioning schemes and describe their performance tradeoffs.
- We perform numerical experiments to illustrate that the CA algorithms are numerically stable for all choices of s tested.

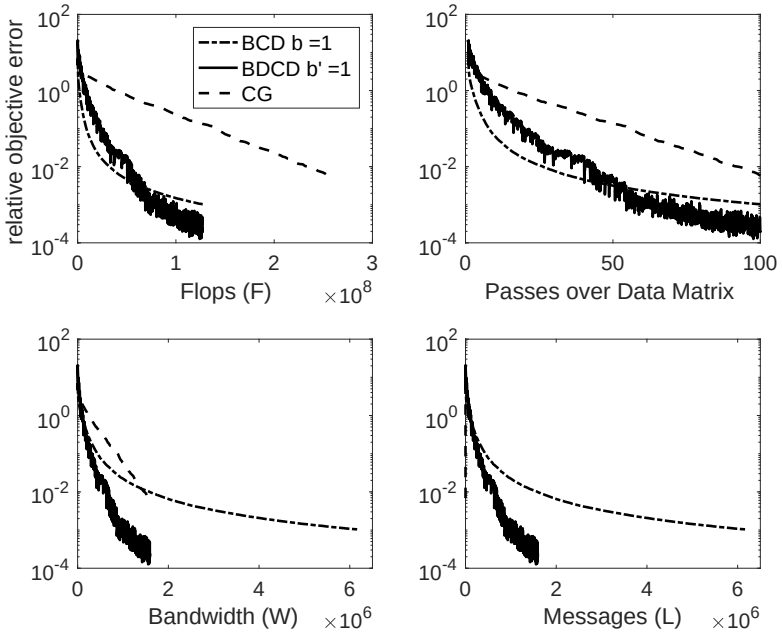


FIG. 1. *Comparison of convergence behavior against algorithm costs of Conjugate Gradients (CG), BCD (with $b = 1$), and BDCD (with $b' = 1$). Convergence is reported in terms of the relative objective error, and the experiments are performed on the news20 dataset ($d = 62061$, $n = 15935$, $nnz(X) = 1272569$) obtained from LIBSVM [10]. We fix the number of CG iterations to $k = 100$, BCD iterations to $H = 100d$, and BDCD iterations to $H' = 100n$.*

- We show performance results to illustrate that the CA algorithms can be up to 6.1× faster than the standard algorithms on up to 1024 nodes of a Cray XC30 supercomputer using message passing interface (MPI).

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 the communication cost model used to analyze our algorithms. Section 3 presents the CA derivations of the BCD and BDCD algorithms. Section 4 analyzes the operational, communication, and storage costs of the classical and CA algorithms under the 1D-block column and 1D-block row data layouts. Section 5 provides numerical and performance experiments which show that the CA algorithms are numerically stable and attain speedups over the standard algorithms. Finally, we conclude in section 6 and describe directions for future work.

2. Background. The regularized least-squares problem can be written as the following optimization problem:

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

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 unregularized ($\lambda = 0$) and regularized ($\lambda > 0$) least-squares problems have been well studied in literature from directly solving the normal equations to

TABLE 3
Relative objective errors of CG, BDCD ($b' = 1$), and BCD ($b = 1$). We normalize the BDCD and BCD iterations to match reported CG iterations. If k is the CG iteration, then BCD performs $H = kd$ and BDCD performs $H' = kn$ iterations.

Relative objective error comparison			
CG iteration	CG error	BDCD error	BCD error
0	6.8735	6.8735	6.8735
1	4.5425	7.8231	1.2826
25	0.5115	0.0441	0.0104
50	0.1326	0.0043	0.0031
75	0.0283	5.0779e-04	0.0016
100	0.0058	1.9346e-04	0.0010

other matrix factorization approaches [14, 3] to Krylov [3, 5, 33] and BCD methods [4, 19, 34, 36, 41]. Tables 1 and 2 summarize the critical path costs of the iterative methods just described.

We briefly summarize the difference between the BCD and BDCD algorithms but defer the derivations to section 3. The BCD algorithm solves the primal minimization problem (1), whereas the BDCD algorithm solves the dual minimization problem:

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

where $\alpha \in \mathbb{R}^n$ is the dual solution vector. The dual problem [34] can be obtained by deriving the convex conjugate of (1) and has the following primal-dual solution relationship:

$$w = \frac{1}{\lambda n} X\alpha. \tag{3}$$

Figure 1 illustrates the tradeoff between convergence behavior and algorithm costs of CG, BCD, and BDCD. We plot the sequential flops cost and ignore the $\log P$ factor for latency. We allow each algorithm to make 100 passes over X and plot 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 CG with a tolerance of 10^{-15}, and w_{alg} is the solution obtained from each iteration of CG, BCD, or BDCD. Since X is not symmetric, CG requires two matrix-vector products at each iteration (one with X and another with X^T). Therefore, the flops cost of CG is twice that of BCD or BDCD. We assume that the two matrix-vector products can be computed with a single pass over X. Table 3 shows a comparison of CG, BDCD, and BCD objective errors after several CG iterations. We see that BDCD and BCD reduce the objective error faster than CG.

If low accuracy suffices, then BCD and BDCD converge faster in terms of flops and passes over X. However, CG is more bandwidth-efficient than BCD (but not BDCD) and is *orders of magnitude* more latency-efficient than BCD and BDCD. This suggests that reducing the latency cost of BCD and BDCD is an important step in making these algorithms competitive.

3. Communication-avoiding primal and dual block coordinate descent. In this section, we rederive 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 derivations of BCD and BDCD lead to recurrences which can be unrolled to derive CA versions of BCD and BDCD, which we will refer to as CA-BCD and CA-BDCD, respectively.

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** $h = 1, 2, \ldots, H$ **do**
3: choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
4: $\mathbb{I}_h = [e_{i_1}, e_{i_2}, \ldots, e_{i_b}]$
5: $\Gamma_h = \frac{1}{n}\mathbb{I}_h^T X X^T \mathbb{I}_h + \lambda \mathbb{I}_h^T \mathbb{I}_h$
6: $\Delta w_h = \Gamma_h^{-1}\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X z_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right)$
7: $w_h = w_{h-1} + \mathbb{I}_h \Delta w_h$
8: $z_h = z_{h-1} + X^T \mathbb{I}_h \Delta w_h$
9: **Output** w_H

3.1. Derivation of block coordinate descent. The minimization problem in (1) can be solved by BCD with the b-dimensional update

$$w_h = w_{h-1} + \mathbb{I}_h \Delta w_h, \tag{4}$$

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

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

with the closed-form solution

$$\Delta w_h = \left(\frac{1}{n}\mathbb{I}_h^T X X^T \mathbb{I}_h + \lambda \mathbb{I}_h^T \mathbb{I}_h\right)^{-1}\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X X^T w_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right). \tag{5}$$

The closed-form solution requires a matrix-vector multiply using the entire data matrix to compute $\frac{1}{n}\mathbb{I}_h^T X X^T w_{h-1}$. However, this can be avoided by introducing the auxiliary variable, $z_h = X^T w_h$, which, by substituting (4), can be rearranged into a vector update of the form

$$z_h = X^T w_{h-1} + X^T \mathbb{I}_h \Delta w_h = z_{h-1} + X^T \mathbb{I}_h \Delta w_h, \tag{6}$$

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

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

In order to make the CA-BCD derivation easier, let us define

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

Then (7) can be rewritten as

$$\Delta w_h = \Gamma_h^{-1}\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X z_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right). \tag{8}$$

This rearrangement leads to the BCD method shown in Algorithm 1. The recurrence in lines 6, 7, and 8 of Algorithm 1 allows us to unroll the BCD recurrences and avoid

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

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. $b \leq d$
for $k = 0, 1, \ldots, \frac{H}{s}$ **do**
  for $j = 1, 2, \ldots, s$ **do**
    choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
    $\mathbb{I}_{sk+j} = [e_{i_1}, e_{i_2}, \ldots, e_{i_b}]$
  let $Y = \left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \ldots, \mathbb{I}_{sk+s}\right]^T X$.
  compute the Gram matrix, $G = \frac{1}{n} YY^T + \lambda I$.
  for $j = 1, 2, \ldots, s$ **do**
    Γ_{sk+j} are the $b \times b$ diagonal blocks of G.
    $$\Delta w_{sk+j} = \Gamma_{sk+j}^{-1}\Bigg(-\lambda \mathbb{I}_{sk+j}^T w_{sk} - \lambda \textstyle\sum_{t=1}^{j-1} \left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t} \Delta w_{sk+t}\right) - \frac{1}{n}\mathbb{I}_{sk+j}^T X z_{sk} - \frac{1}{n}\textstyle\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T X X^T \mathbb{I}_{sk+t}\Delta w_{sk+t}\right) + \frac{1}{n}\mathbb{I}_{sk+j}^T X y\Bigg)$$
    $w_{sk+j} = w_{sk+j-1} + \mathbb{I}_{sk+j}\Delta w_{sk+j}$
    $z_{sk+j} = z_{sk+j-1} + X^T \mathbb{I}_{sk+j}\Delta w_{sk+j}$
Output w_H

communication. We begin by changing the loop index from h to $sk + j$, where k is the outer loop index, s is the recurrence unrolling parameter, and j is the inner loop index. Assume that we are at the beginning of iteration $sk + 1$ and w_{sk} and z_{sk} were just computed. Then Δw_{sk+1} can be computed by

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

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

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

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

$$\Delta w_{sk+j} = \Gamma_{sk+j}^{-1}\Bigg(-\lambda \mathbb{I}_{sk+j}^T w_{sk} - \lambda \sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t}\Delta w_{sk+t}\right) - \frac{1}{n}\mathbb{I}_{sk+j}^T X z_{sk} - \frac{1}{n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T X X^T \mathbb{I}_{sk+t}\Delta w_{sk+t}\right) + \frac{1}{n}\mathbb{I}_{sk+j}^T X y\Bigg) \tag{9}$$

for $j = 1, 2, \ldots, s$. Due to the recurrence unrolling we can defer the updates to w_{sk} and z_{sk} for s steps. Notice that the first summation in (9) 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

Algorithm 3 Block Dual Coordinate Descent (BDCD) Algorithm.

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. $b' \leq n$
Initialize: $w_0 \leftarrow \frac{1}{\lambda n} X\alpha_0$
for $h = 1, 2, \dots, H'$ **do**
  choose $\{i_m \in [n] | m = 1, 2, \dots, b'\}$ uniformly at random without replacement
  $\mathbb{I}_h = \left[e_{i_1}, e_{i_2}, \dots, e_{i_{b'}}\right]$
  $\Theta_h = \frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T\mathbb{I}_h$
  $\Delta\alpha_h = \frac{1}{n}\Theta_h^{-1}\left(-\mathbb{I}_h^T X^T w_{h-1} - \mathbb{I}_h^T\alpha_{h-1} + \mathbb{I}_h^T y\right)$
  $\alpha_h = \alpha_{h-1} + \mathbb{I}_h\Delta\alpha_h$
  $w_h = w_{h-1} + \frac{1}{\lambda n} X\mathbb{I}_h\Delta\alpha_h$
Output α'_H and w'_H

all processors to the same seed for the random number generator. The second summation in (9) computes the Gram-like matrices $\mathbb{I}_{sk+j}^T XX^T\mathbb{I}_{sk+t}$ for $t = 1, \dots, j-1$. Communication can be avoided in this computation by computing the $sb \times sb$ Gram matrix $G = (\frac{1}{n}\left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]^T XX^T\left[+\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right] + \lambda I)$ 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

$$w_{sk+s} = w_{sk} + \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta w_{sk+t}\right), \tag{10}$$

$$z_{sk+s} = z_{sk} + X^T\sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta w_{sk+t}\right). \tag{11}$$

The resulting CA-BCD algorithm is shown in Algorithm 2.

3.2. Derivation of block dual coordinate descent. The solution to the primal problem (1) can also be obtained by solving the dual minimization problem shown in (2) with the primal-dual relationship shown in (3). The dual problem (2) can be solved using BCD, which iteratively solves a subproblem 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 $\alpha_h \in \mathbb{R}^n$:

$$\alpha_h = \alpha_{h-1} + \mathbb{I}_h\Delta\alpha_h. \tag{12}$$

Here 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 (2), $\Delta\alpha_h$ is the solution to a minimization problem in $\mathbb{R}^{b'}$ as desired:

$$\underset{\Delta\alpha_h\in\mathbb{R}^{b'}}{\arg\min}\ \frac{1}{2\lambda n^2}\left\|X\alpha_{h-1} + X\mathbb{I}_h\Delta\alpha_h\right\|_2^2 + \frac{1}{2n}\left\|\alpha_{h-1} + \mathbb{I}_h\Delta\alpha_h - y\right\|_2^2. \tag{13}$$

Finally, due to (3) we obtain the primal vector update for $w_h \in \mathbb{R}^d$:

$$w_h = w_{h-1} + \frac{1}{\lambda n} X\mathbb{I}_h\Delta\alpha_h. \tag{14}$$

From (12), (13), and (14) we obtain a BCD algorithm which solves the dual minimization problem. Henceforth, we refer to this algorithm as block dual coordinate

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

1: **Input**: $X = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$, $H' > 1$, $\alpha_0 \in \mathbb{R}^n$, $b' \in \mathbb{Z}_+$ s.t. $b' \leq n$

2: **Initialize**: $w_0 \leftarrow \frac{1}{\lambda n} X\alpha_0$

3: **for** $k = 0, 1, \ldots, \frac{H'}{s}$ **do**

4: **for** $j = 1, 2, \ldots, s$ **do**

5: choose $\{i_m \in [n] | m = 1, 2, \ldots, b'\}$ uniformly at random without replacement

6: $\mathbb{I}_{sk+j} = \left[e_{i_1}, e_{i_2}, \ldots, e_{i_{b'}}\right]$

7: let $Y = X\left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \ldots, \mathbb{I}_{sk+s}\right]$.

8: compute the Gram matrix, $G' = \frac{1}{\lambda n^2} Y^T Y + \frac{1}{n} I$.

9: **for** $j = 1, 2, \ldots, s$ **do**

10: Θ_{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 $w_{H'}$

descent (BDCD). Note that by setting $b' = 1$ we obtain the SDCA (stochastic dual coordinate ascent) algorithm [34] with the least-squares loss function.

The optimization problem (13) which computes the solution along the chosen coordinates has the closed form

(15)
$$\Delta\alpha_h = \left(\frac{1}{\lambda n^2}\mathbb{I}_h^T X^T X \mathbb{I}_h + \frac{1}{n}\mathbb{I}_h^T \mathbb{I}_h\right)^{-1}\left(\frac{-1}{\lambda n^2}\mathbb{I}_h^T X^T X \alpha_{h-1} - \frac{1}{n}\mathbb{I}_h^T \alpha_{h-1} + \frac{1}{n}\mathbb{I}_h^T y\right).$$

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,

(16)
$$\Delta\alpha_h = \frac{1}{n}\Theta_h^{-1}\left(-\mathbb{I}_h^T X^T w_{h-1} - \mathbb{I}_h^T \alpha_{h-1} + \mathbb{I}_h^T y\right),$$

and obtain the BDCD algorithm shown in Algorithm 3. The recurrence in lines 7, 8, and 9 of Algorithm 3 allows us to unroll the BDCD recurrences and avoid communication. We begin by changing the loop index from h to $sk + j$, where k is the outer loop index, s is the recurrence unrolling parameter, and j is the inner loop index. 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 recurrences for w_{sk+1} and α_{sk+1} we can analogously to (9) show by induction that

$$\Delta\alpha_{sk+j} = \frac{1}{n}\Theta_{sk+j}^{-1}\Bigg(- \mathbb{I}_{sk+j}^T X^T w_{sk} - \frac{1}{\lambda n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T X^T X \mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) - \mathbb{I}_{sk+j}^T \alpha_{sk} - \sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right) + \mathbb{I}_{sk+j}^T y\Bigg) \tag{17}$$

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 primal and dual solution vectors from the previous outer iteration. Since the solution vector updates require communication, the recurrence 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

$$w_{sk+s} = w_{sk} + \frac{1}{\lambda n} X \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right), \tag{18}$$

$$\alpha_{sk+s} = \alpha_{sk} + \sum_{t=1}^{s}\left(\mathbb{I}_{sk+t}\Delta\alpha_{sk+t}\right). \tag{19}$$

The resulting CA-BDCD algorithm is shown in Algorithm 4.

4. Analysis of algorithms. From the derivations in section 3, we can observe that BCD and BDCD 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 BCD and BDCD under two data partitioning schemes: 1D-block row (feature partitioning) and 1D-block column (data point partitioning). In both cases, we derive the associated operational, storage, and communication costs. We perform a similar analysis of the CA variants to illustrate that we provably avoid communication and describe tradeoffs. Since X is sparse, the analysis of the computational cost includes passes over the sparse data structure instead of just the floating-point operations associated with the sparse matrix–sparse matrix multiplication (i.e., Gram matrix computation). Therefore, our analysis gives bounds on the local operations for each processor. We begin in section 4.1 with the analysis of the BCD and BDCD algorithms and then analyze our new, CA variants in section 4.2. As discussed in section 1, we can use other sparsity models instead of our fdn nonzeros model. We assume an idealized sparse matrix with uniform distribution of the nonzeros between rows (for BCD) and between columns (for BDCD). Due to this assumption, we can easily change from the fdn model to the $nnz = \xi n$ and $nnz = \omega d$ models. Replacing fd with ξ (for BCD) and fn with ω (for BDCD) gives bounds for the different models. We also note that practical values of b need to be quite small in order to be competitive with Krylov subspace methods

(see Figure 1). Therefore, we assume that b is small enough that the Gram matrix is most efficiently computed through the use of dot-products. For large values of b tighter bounds for the Gram matrix computations can be obtained by analyzing the matrix-product. The matrix-product bounds have an additional factor of f for the Gram matrix computations. Since $0 < f \leq 1$, this bound is tighter than the one obtained by assuming dot-products.

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 4.1. *H iterations of the 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 cost*

$$F = O\left(\frac{Hb^2fn}{P} + Hb^3\right) \text{ ops, } M = O\left(\frac{fdn+n}{P} + b^2 + d\right) \text{ words of memory.}$$

Communication costs

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

Proof. The BCD algorithm computes a $b\times b$ Gram matrix, Γ_h, solves a $b\times b$ linear system to obtain Δw_h, and updates the vectors w_h and z_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. Since the $b \times n$ submatrix $\mathbb{I}_h^T X$ has bfn nonzeros, the parallel Gram matrix computation ($\mathbb{I}_h^T X X^T \mathbb{I}_h$) requires $O(\frac{b^2fn}{P})$ operations (there are b^2 elements of the Gram matrix, each of which depends on fn nonzeros) and communicates $O\left(b^2 \log P\right)$ words, with $O\left(\log P\right)$ messages. In order to solve the subproblem redundantly on all processors, a local copy of the residual is required. Computing the residual requires $O(\frac{bfn}{P})$ operations and communicates $O\left(b \log P\right)$ words, in $O\left(\log P\right)$ messages. Once the residual is computed, the subproblem can be solved redundantly on each processor in $O\left(b^3\right)$ flops. Finally, the vector updates to w_h and z_h can be computed without any communication in $O(b + \frac{bfn}{P})$ flops on each processor. The critical path costs of H iterations of this algorithm are $O(\frac{Hb^2fn}{P} + Hb^3)$ 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, $\mathbb{I}_h$, $\frac{1}{P}$th columns of X, and $\frac{1}{P}$th elements of z_h and y. Therefore, the memory cost of each processor is $d + b^2 + 2b + \frac{fdn+2n}{P} = O(\frac{fdn+n}{P} + b^2 + d)$ words per processor. ∎

If $\frac{fn}{P} > b$, then computing the Gram matrix dominates solving the subproblem. Furthermore, the storage cost of X dominates the cost of the Gram matrix.

THEOREM 4.2. *H' iterations of the 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 cost*

$$F = O\left(\frac{H'b'^2fd}{P} + H'b'^3\right) \text{ ops, } M = O\left(\frac{fdn+d}{P} + b'^2 + n\right) \text{ words of memory.}$$

Communication costs

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

Proof. The proof is similar to that of Theorem 4.1 with an appropriate change of variables for BDCD. ∎

If X is stored in a 1D-block row layout, then each processor stores a disjoint subset of the features of X. Since BCD selects b features at each iteration, 1D-block row partitioning could lead to load imbalance. In order to avoid load imbalance, we repartition the chosen b features into a 1D-block column layout and proceed by using the 1D-block column BCD algorithm. Repartitioning the b features requires communication, so we begin by bounding the maximum number of features assigned to a single processor. The bandwidth cost of repartitioning is bounded by the processor with maximum load (i.e., maximum number of features). These bounds only hold with high probability since the features are chosen uniformly at random. To attain bounds on the bandwidth cost, we assume that each sampled row of X has fn nonzeros.

LEMMA 4.3. *Given a matrix $X \in \mathbb{R}^{d\times n}$ and P processors such that each processor stores $\Theta(\lfloor \frac{d}{P} \rfloor)$ features, if b features are chosen uniformly at random, then the worst-case maximum number of features, $\eta(b,P)$, assigned to a single processor with high probability (w.h.p.) is*

$$\eta(b,P) = \begin{cases} O\left(\frac{b}{P} + \sqrt{\frac{b\log P}{P}}\right) & \text{if } b > P\log P, \\ O\left(\frac{\log b}{\log\log b}\right) & \text{if } b = P, \\ O\left(\frac{\log P}{\log \frac{P}{b}}\right) & \text{if } b < \frac{P}{\log P}. \end{cases}$$

Proof. This is the well-known generalization of the balls and bins problem introduced by Gonnet [17] and extended by Mitzenmacher [24] and Raab and Steger [30].

A similar result holds for BDCD with X stored in a 1D-block column layout.

THEOREM 4.4. *H iterations of the 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 cost*

$$F = O\left(\frac{Hb^2fn}{P} + Hb^3\right) \text{ ops}, \ M = O\left(\frac{fdn+n}{P} + b^2 + d\right) \text{ words of memory}.$$

For small messages, communication costs w.h.p.

$$W = O\left(\left(b^2 + \eta(b,P)fn\right)H\log P\right) \text{ words moved}, \ L = O\left(H\log P\right) \text{ messages}.$$

For large messages, communication costs w.h.p.

$$W = O\left(Hb^2\log P + H\eta(b,P)fn\right) \text{ words moved}, \ L = O\left(HP\right) \text{ messages}.$$

Proof. The 1D-block row partitioning scheme implies that the $b\times b$ Gram matrix, Γ_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 4.3, stores $O\left(\eta(b,P)\right)$ rows w.h.p. in the worst case. This requires $W = O\left(\eta(b,P)fn\log P\right)$ and $L = O\left(\log P\right)$ for small messages or $W = O\left(\eta(b,P)fn\right)$ and $L = O\left(HP\right)$ for large messages. The all-to-all requires additional storage on each processor of $M = O(\frac{bfn}{P})$ words. Once the sampled matrix

is converted, the BCD algorithm proceeds as in Theorem 4.1. By combining the cost of the all-to-all over H iterations and the costs from Theorem 4.1, we obtain the costs for the BCD algorithm with X stored in a 1D-block row layout. ∎

The additional storage for the all-to-all does not dominate since $b < d$ by definition.

THEOREM 4.5. *H' iterations of the 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 cost w.h.p.*

$$F = O\left(\frac{H'b'^2fd}{P} + H'b'^3\right) \text{ ops, } M = O\left(\frac{fdn+d}{P} + b'^2 + n\right) \text{ words of memory.}$$

For small messages, communciation costs w.h.p.

$$W = O\left(\left(b'^2 + \eta(b',P)fd\right)H'\log P\right) \text{ words moved, } L = O\left(H'\log P\right) \text{ messages.}$$

For large messages, communication costs w.h.p.

$$W = O\left(H'b'^2\log P + H'\eta(b',P)fd\right) \text{ words moved, } L = O\left(H'P\right) \text{ messages.}$$

Proof. Cost analysis similar to Theorem 4.4 proves this theorem. ∎

4.2. Communication-avoiding algorithms. In this section, we derive the computation, storage, and communication costs of our CA-BCD and CA-BDCD algorithms under the 1D-block row and 1D-block column data layouts. In both cases we show that our algorithms reduce the latency costs by a factor of s but increase flops and bandwidth by the same factor. Our experimental results will show that this tradeoff can lead to speedups. 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 4.6. *H iterations of the 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 cost*

$$F = O\left(\frac{Hb^2sfn}{P} + Hb^3\right) \text{ ops, } M = O\left(\frac{fdn+n}{P} + b^2s^2 + d\right) \text{ words of memory.}$$

Communication costs

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

Proof. The CA-BCD algorithm computes the $sb \times sb$ Gram matrix, $G = \frac{1}{n}YY^T + \lambda I$, where $Y = \left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \ldots, \mathbb{I}_{sk+s}\right]^T X$, solves s $(b \times b)$ linear systems to compute Δw_{sk+j}, and updates the vectors w_{sk+s} and z_{sk+s}. Computing the Gram matrix requires that each processor locally compute an $sb \times sb$ block of inner-products and then perform an all-reduce (a reduction and broadcast) to sum the partial blocks. This operation requires $O(\frac{b^2s^2fn}{P})$ operations (there are s^2b^2 elements of the Gram matrix, each of which depends on fn nonzeros), communicates $O\left(s^2b^2\log P\right)$ words, and requires $O\left(\log P\right)$ messages. In order to solve the subproblem redundantly on all

processors, a local copy of the residual is required. Computing the residual requires $O(\frac{bsfn}{P})$ flops and communicates $O(sb \log P)$ words in $O(\log P)$ messages. Once the residual is computed, the subproblem can be solved redundantly on each processor in $O\left(b^3 s + b^2 s^2\right)$ flops. Finally, the vector updates to w_{sk+s} and z_{sk+s} can be computed without any communication in $O(bs + \frac{bsfn}{P})$ flops on each processor. Since the critical path occurs every $\frac{H}{s}$ iterations (every outer iteration), the algorithm costs $O(\frac{Hb^2sfn}{P} + Hb^3)$ flops, $O\left(Hb^2 s \log P\right)$ 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}, $\mathbb{I}_{sk+j}$, $\frac{1}{P}$th columns of X, and $\frac{1}{P}$th elements of z_{sk+j} and y. Therefore, the memory cost of each processor is $d + s^2b^2 + 2sb + \frac{fdn+2n}{P} = O(\frac{fdn+n}{P} + b^2s^2 + d)$ words per processor. ∎

THEOREM 4.7. *H' iterations of the CA-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 cost*

$$F = O\left(\frac{H'b'^2sfd}{P} + H'b'^3\right) \text{ ops}, \; M = O\left(\frac{fdn+d}{P} + b'^2s^2 + n\right) \text{ words of mem.}$$

Communication costs

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

Proof. The proof is similar to that of Theorem 4.6 with an appropriate change of variables for CA-BDCD. ∎

Now we analyze the operational and communication costs of CA variants of the 1D-block row BCD and 1D-block column BDCD algorithms.

THEOREM 4.8. *H iterations of the 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 cost*

$$F = O\left(\frac{Hb^2sfn}{P} + Hb^3\right) \text{ ops}, \; M = O\left(\frac{(d+bs)fn+n}{P} + b^2s^2 + d\right) \text{ words.}$$

For small messages, communication costs w.h.p.

$$W = O\left(\left(b^2 s + \eta(sb, P) fn\right) H \log P\right) \text{ words moved}, L = O\left(\frac{H}{s} \log P\right) \text{ messages.}$$

For large messages, communication costs w.h.p.

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

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 4.3, stores $O(\eta(sb, P))$ rows w.h.p. in the worst case. This requires $W = O(\eta(sb, P) fn \log P)$ and $L = O(\log P)$ for small messages

TABLE 4
Properties of the LIBSVM datasets used in our experiments. We report the largest and smallest singular values (same as the eigenvalues) of X^TX.

Summary of datasets						
Name	Features (d)	Data Points (n)	NNZ%	σ_{min}	σ_{max}	Source
news20	62,061	15,935	0.13	1.7e−6	6.0e+5	LIBSVM [21]
a9a	123	32,561	11	4.9e−6	2.0e+5	UCI [22]
real-sim	20,958	72,309	0.24	1.1e−3	9.2e+2	LIBSVM [23]

or $W = O(\eta(sb, P)fn)$ and $L = O(HP)$ for large messages. The all-to-all requires additional storage on each processor of $M = O(\frac{bsfn}{P})$ words. Once the sampled matrix is converted, the BCD algorithm proceeds as in Theorem 4.6. By combining the cost of the all-to-all over H iterations and the costs from Theorem 4.6, 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, b and s must be chosen carefully.

THEOREM 4.9. *H iterations of the 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 cost*

$$F = O\left(\frac{H'b'^2sfd}{P} + H'b'^3\right) \text{ ops}, M = O\left(\frac{(n+b's)fd+d}{P} + b'^2s^2 + n\right) \text{ words}.$$

For small messages, communication costs w.h.p.

$$W = O\left(\left(b'^2s + \eta(sb', P)fd\right)H'\log P\right) \text{ words moved}, L = O\left(\frac{H'}{s}\log P\right) \text{ msgs}.$$

For large messages, communication costs w.h.p.

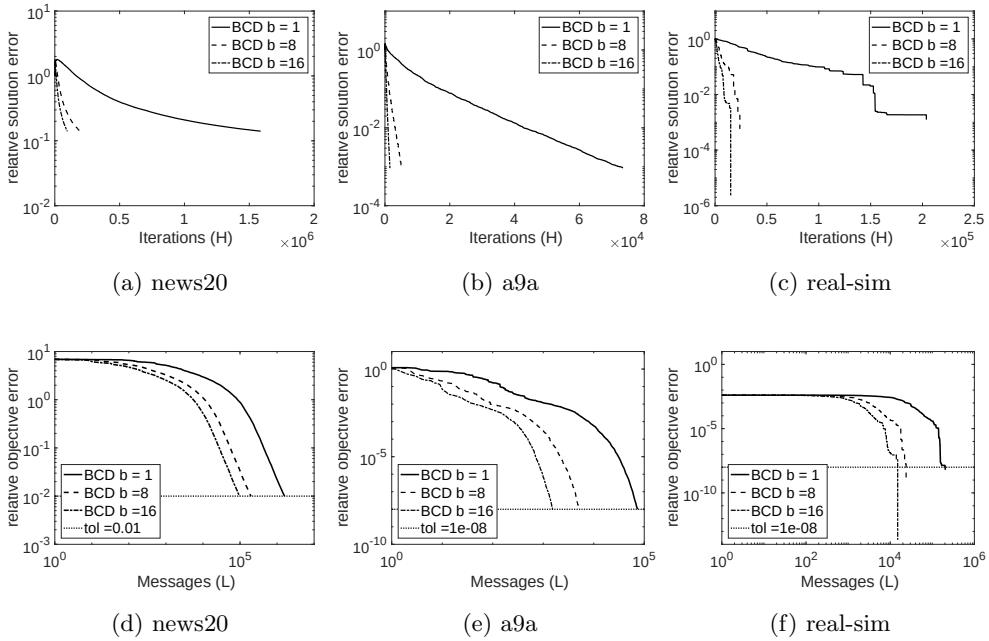
$$W = O\left(H'b'^2s\log P + H'\eta(sb', P)fd\right) \text{ words moved}, \ L = O\left(\frac{H'}{s}P\right) \text{ messages}.$$

Proof. A similar cost analysis to Theorem 4.8 proves this theorem. ∎

The CA variants that we have derived require a factor of s fewer messages than their classical counterparts, at the cost of more computation, bandwidth, and memory. This suggests that s must be chosen carefully to balance the additional costs with the reduction in the latency cost.

5. Experimental evaluation. We proved in section 4 that the CA-BCD and CA-BDCD algorithms reduce latency (the dominant cost) at the expense of additional bandwidth and computation. The recurrence unrolling we propose may also affect the numerical stability of CA-BCD and CA-BDCD since the sequences of computations and vector updates are different. In section 5.1 we experimentally show that the CA variants are numerically stable (in contrast to some CA-Krylov methods [5, 6, 7, 8, 9, 18]), and, in section 5.2, we show that the CA variants can lead to large speedups on a Cray XC30 supercomputer using MPI.

5.1. Numerical experiments. The algorithm transformations derived in section 3 require that CA-BCD and CA-BDCD operate on Gram matrices of size $sb \times sb$

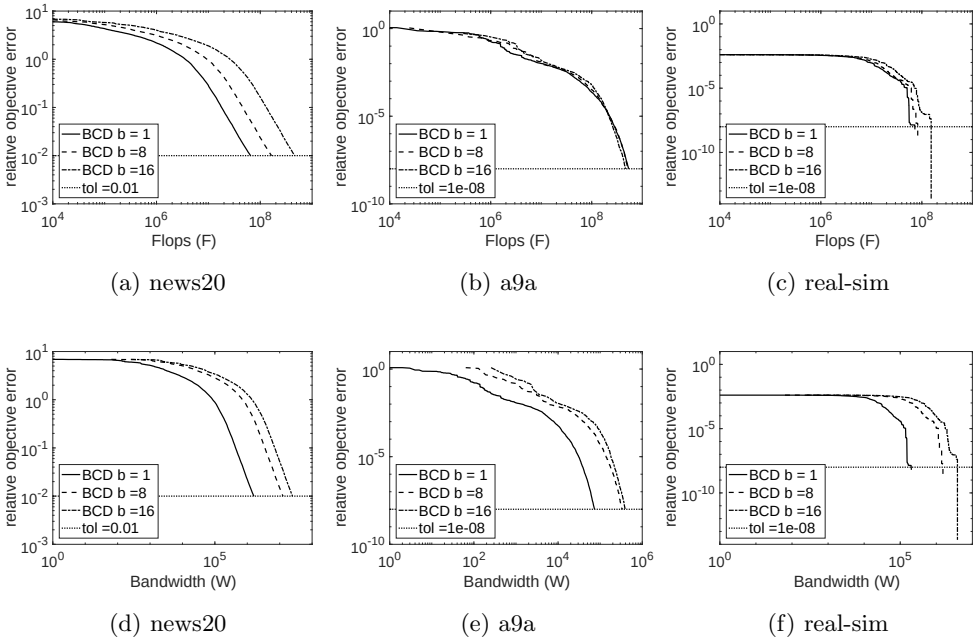


(a) news20 (b) a9a (c) real-sim

(d) news20 (e) a9a (f) real-sim

FIG. 2. *We compare the convergence behavior of BCD for several block sizes, b, such that* $1 \leq b < d$ *on several machine learning datasets. We show relative solution error (top row, Figures* 2a–2c*) and objective error (bottom row, Figure* 2d–2f*) convergence plots with* $\lambda = 1000\sigma_{min}$*. We fix the objective error tolerance for news*20 *to* $1e{-}2$ *and* $1e{-}8$ *for a*9*a and real-sim. The x-axis for Figures* 2d–2f *show the number of messages required on a* $\log_{10}$ *scale. The x-axis is also equivalent to the number of iterations (modulo* $\log_{10}$ *scale).*

instead of size $b \times b$ every outer iteration. Due to the larger dimensions, the condition number 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 standard and CA algorithms. All numerical stability experiments were performed in MATLAB version R2016b on a 2.3 GHz Intel i7 machine with 8GB of RAM with datasets obtained from the LIBSVM repository [10]. Datasets were chosen so that all algorithms were tested on a range of shapes, sizes, and condition numbers. Table 4 summarizes the important properties of the datasets tested. For all experiments, we set the regularization parameter to $\lambda = 1000\sigma_{min}$. The regularization parameter reduces the condition numbers of the datasets and allows the BCD and BDCD algorithms to converge faster. 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 among 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 ob-



(a) news20 (b) a9a (c) real-sim

(d) news20 (e) a9a (f) real-sim

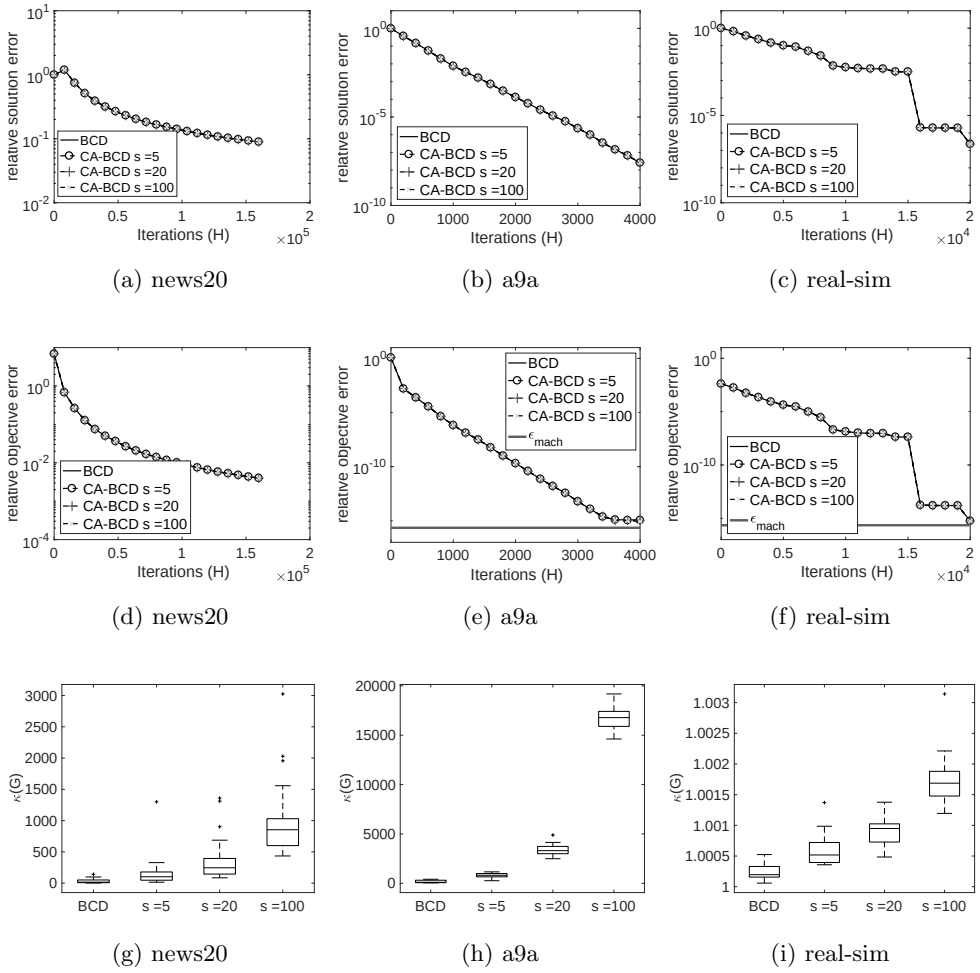
FIG. 3. *We compare the convergence behavior of BCD for several block sizes, b, such that* $1 \leq b < d$ *on several machine learning datasets. Flops cost (top row, Figures* 3a–3c*) and bandwidth cost (middle row, Figures* 3d–3f*) versus convergence with* $\lambda = 1000\sigma_{min}$.

jective to show convergence behavior for BCD, BDCD, and their CA variants. We explore the tradeoff between the block sizes, b and b', and convergence behavior to test BCD and BDCD stability due to the choice of block sizes. Then, we fix the block sizes and explore the tradeoff between s, the recurrence unrolling parameter, and convergence behavior to study the stability of the CA variants. 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 in 1D-block column for BCD and 1D-block row for BDCD. We plot the sequential flops cost for all algorithms, ignore the $\log P$ factor for the number of messages, and ignore constants. We obtain the Gram matrix computation cost from the SuiteSparse [13] routine `ssmultsym`.[1]

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 the block size versus convergence behavior tradeoff for BCD with $1 \leq b < d$.

Figure 2 shows the convergence behavior of the datasets in Table 4 in terms of the relative solution error (Figures 2a–2c) and relative objective error (Figures 2d–2f). The x-axis for the latter figures are on a $\log_{10}$ scale. Note that the number of messages is equivalent to the number of iterations, since BCD communicates every

[1]Symbolically executes the sparse matrix–sparse matrix multiplication and reports an estimate of the flops cost (counting multiplications and additions).



(a) news20 (b) a9a (c) real-sim

(d) news20 (e) a9a (f) real-sim

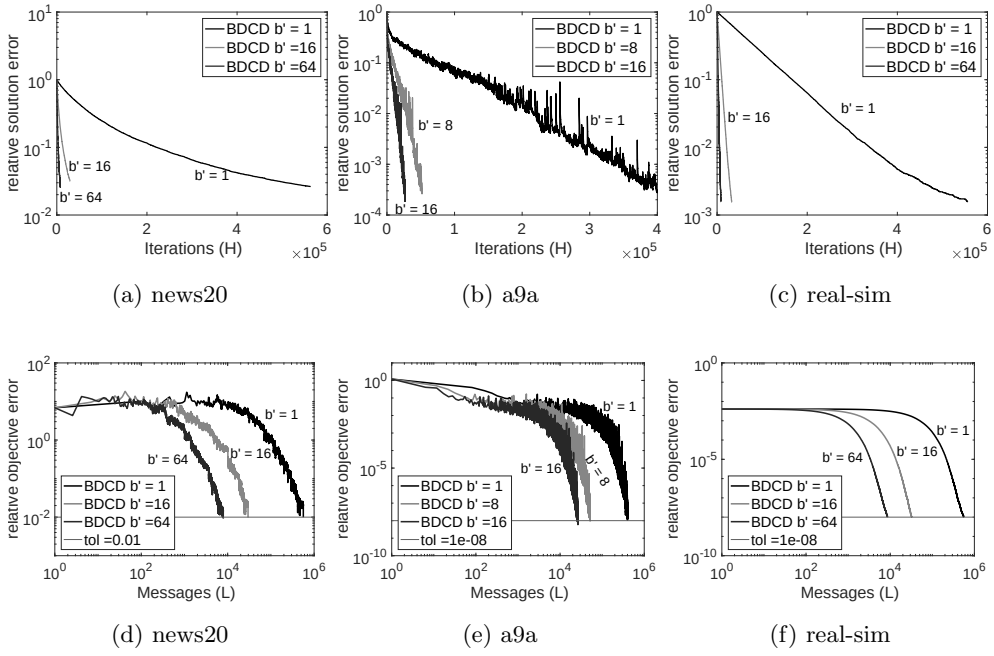
(g) news20 (h) a9a (i) real-sim

FIG. 4. *We compare the convergence behavior of BCD and CA-BCD with several values of* s*. Relative solution error (top row, Figures* 4a–4c*), relative objective error (middle row, Figures* 4d–4f*), and statistics of the Gram matrix condition numbers (bottom row, Figures* 4g–4i*) versus convergence. The block size for each dataset is set to* $b = 16$*. The boxplots (Figures* 4g–4i*) use standard MATLAB convention* [37].

iteration. We observe that the convergence rates for all datasets improve as the block sizes increase.

Figure 3 shows the convergence behavior (in terms of the objective error) versus flops and bandwidth costs for each dataset. From these results, we observe that BCD with $b = 1$ is more flops- and bandwidth-efficient, whereas $b > 1$ is more latency-efficient (from Figures 2d–2f). This indicates the existence of a tradeoff between BCD's convergence rate (which depends on the block size) and hardware-specific parameters (like flops rate, memory/network bandwidth, and latency).

5.1.2. Communication-avoiding block coordinate descent. Our derivation of the CA-BCD algorithm showed that by unrolling the vector update recurrences we can reduce the latency cost of the BCD algorithm by a factor of s. However, this



(a) news20 (b) a9a (c) real-sim

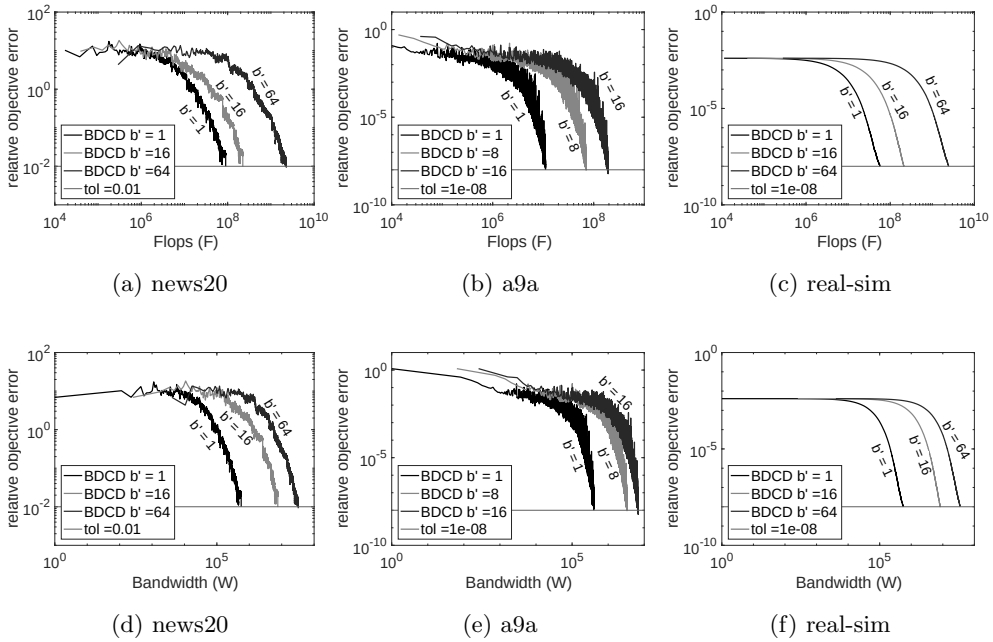
(d) news20 (e) a9a (f) real-sim

FIG. 5. *We compare the convergence behavior of BDCD for several block sizes, b', such that $1 \leq b' < n$. We show relative solution error (top row, Figures* 5a–5c*) and objective error (bottom row, Figures* 5d–5f*) convergence plots with $\lambda = 1000\sigma_{min}$. The x-axis for Figures* 5d–5f *shows the number of messages required on a* $\log_{10}$ *scale. The x-axis is also equivalent to the number of iterations (modulo* $\log_{10}$ *scale). In Figure* 5f *the unlabeled curve is* $b' = 64$.

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. We begin by experimentally showing the convergence behavior of the CA-BCD algorithm on the datasets in Table 4 with fixed block sizes of $b = 16$ for news20, a9a, and real-sim, respectively.

Figure 4 compares the convergence behavior of BCD and CA-BCD for $s > 1$. We plot the relative solution error, relative objective error, and statistics of the Gram matrix condition numbers. The convergence plots indicate that CA-BCD shows almost no deviation from the BCD convergence. While the Gram matrix condition numbers increase with s for CA-BCD, those condition numbers are not so large as to significantly alter the numerical stability. Figures 4e and 4f show that the objective error converges very close to ϵ_{mach}. The well-conditioning of the real-sim dataset in addition to the regularization and small block size (relative to d) makes the Gram matrices almost perfectly conditioned. Based on these results, it is likely that the factor of s increase in flops and bandwidth will be the primary bottleneck.

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'-dimensional 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



(a) news20 (b) a9a (c) real-sim

(d) news20 (e) a9a (f) real-sim

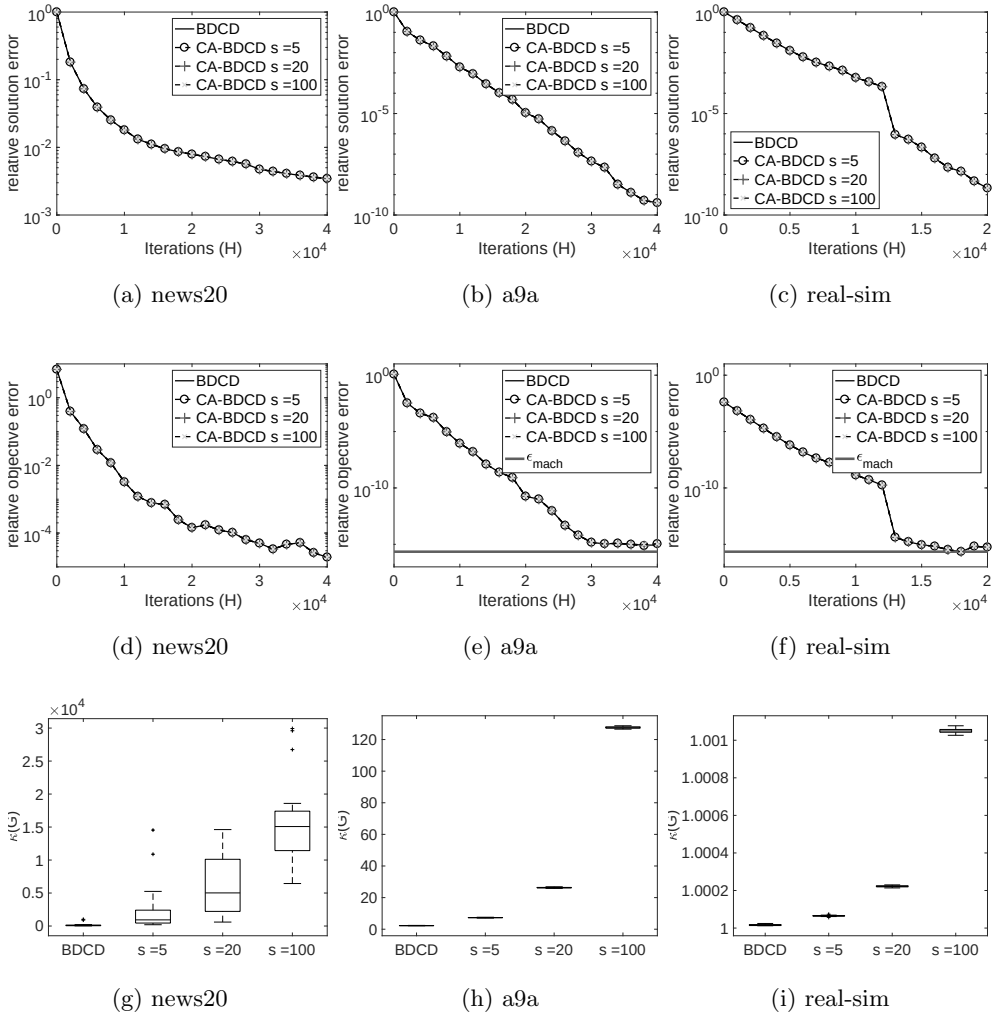
FIG. 6. *We compare the convergence behavior of BDCD for several block sizes, b', such that $1 \leq b' < n$. Flops cost (top row, Figures* 6a–6c*) and bandwidth cost (middle row, Figures* 6d–6f*) versus convergence with $\lambda = 1000\sigma_{min}$.*

explore this tradeoff space by comparing the convergence behavior (solution error and objective error) and algorithm costs for BDCD with $1 \leq b' < n$.

Figure 5 shows the convergence behavior on the datasets in Table 4 for various block sizes and measures the relative solution error (Figures 5a–5c) and relative objective error (Figures 5d–5f). Similar to BCD, as the block sizes increase the convergence rates of each dataset improve. However, unlike BCD, the objective error does not immediately decrease for some datasets (news20 and a9a). This is expected behavior since BDCD minimizes the dual objective (see section 3.2) and obtains the primal solution vector, w_h, by taking linear combinations of b' columns of X and w_{h-1}. This also accounts for the nonmonotonic decrease in the primal objective and primal solution errors.

Figure 6 shows the convergence behavior (in terms of the objective error) versus flops and bandwidth costs of BDCD for the datasets and block sizes tested in Figure 5. We see that small block sizes are more flops- and bandwidth-efficient, while large block sizes are latency-efficient (from Figures 5d–5f). Due to this tradeoff, it important to select block sizes that balance these costs based on machine-specific parameters.

5.1.4. Communication-avoiding block dual coordinate descent. The CA-BDCD algorithm avoids communication in the dual problem by unrolling the vector update recurrences by a factor of s. This allows us to reduce the latency cost by computing a larger $sb' \times sb'$ Gram matrix instead of a $b' \times b'$ Gram matrix 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 4.



(a) news20 (b) a9a (c) real-sim

(d) news20 (e) a9a (f) real-sim

(g) news20 (h) a9a (i) real-sim

FIG. 7. *We compare the convergence behavior of BDCD and CA-BDCD with several values of s. Relative solution error (top row, Figures* 7a–7c*), relative objective error (middle row, Figures* 7d–7f*), and statistics of the Gram matrix condition numbers (bottom row, Figures* 7g–7i*) versus convergence. The block sizes for each dataset are news*20 *with* $b' = 64$*, a9a with* $b' = 16$*, and real-sim with* $b' = 64$*.*

Figure 7 compares the convergence behavior of BDCD and CA-BDCD for $s > 1$ with block sizes of $b' = 64, 16$, and 64 for the news20, a9a, and real-sim datasets, respectively. The results indicate that CA-BDCD is numerically stable for all tested values of s on all datasets. While the condition numbers of the Gram matrices increase with s, the numerical stability is not significantly affected. The well-conditioning of the real-sim dataset in addition to the regularization and small block size (relative to n) make the Gram matrices almost perfectly conditioned.

5.2. Performance experiments. In section 5.1 we showed tradeoffs between convergence behavior and algorithm costs for several datasets. In this section, we explore the performance tradeoffs of standard versus CA variants on datasets obtained

TABLE 5
LIBSVM datasets used in our performance experiments.

Algorithm	Name	Features (d)	Data Points (n)	NNZ%	residual tolerance (tol)
BCD	a9a	123	32561	11	1e-2
	covtype	54	581012	22	1e-1
	mnist8m	784	8100000	25	1e-1
BDCD	news20	62061	15935	0.13	1e-2
	e2006	150360	3308	0.93	1e-2
	rcv1	47236	3000	0.17	1e-3

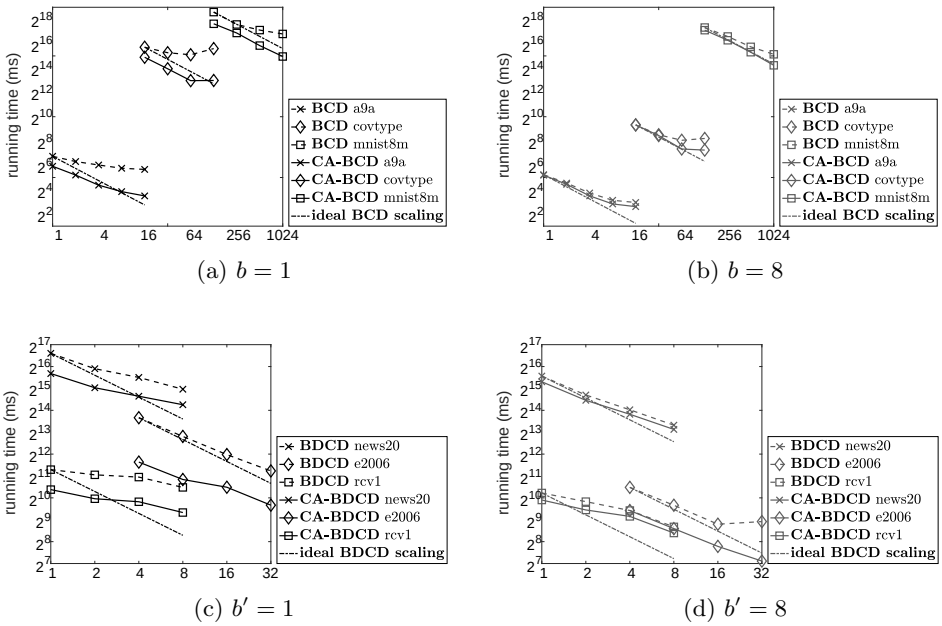
from LIBSVM [10]. We implemented these algorithms in C/C++ using Intel MKL for (sparse and dense) BLAS routines and MPI [16] for parallel processing. While sections 4 and 5.1 assumed dense data for the theoretical analysis and numerical experiments, our parallel implementation stores the data in compressed sparse row (CSR) format. We used a Cray XC30 supercomputer ("Edison") at NERSC [27] to run our experiments on the datasets shown in Table 5. We used a 1D-column layout for datasets with $n > d$ and a 1D-row layout for $n < d$. We ensured that the parallel file I/O was load-balanced (i.e., each processor read roughly equal bytes) and found that the nonzero entries were reasonably well balanced.[2] We constrain the running time of (CA-)BCD and (CA-)BDCD by fixing the residual tolerance for each dataset to the values described in Table 5. We ran many of these datasets for smaller tolerances of $1e-8$ and found that our conclusions did not significantly change.

Section 5.2.1 compares the strong scaling behavior of the standard BCD and BDCD algorithms against their CA variants, section 5.2.2 shows the running time breakdown to illustrate the flops versus communication tradeoff, and section 5.2.3 compares the speedups attained as a function of the number of processors, block size, and recurrence unrolling parameter, s.

5.2.1. Strong scaling. All strong scaling experiments were conducted with one MPI process per processor (flat-MPI) with one warm-up run and three timed runs. Each data point in Figure 8 represents the maximum running time over all processors averaged over the three timed runs. For each dataset in Figure 8 we plot the BCD running times, the fastest CA-BCD running times for $s \in \{2, 4, 8, 16, 32\}$, and the ideal scaling behavior. We show the scaling behavior of all datasets for $b \in \{1, 8\}$ to illustrate how the CA-BCD speedups are affected by the choice of block size, b. When the BCD algorithm is entirely latency dominated (i.e., Figure 8a), CA-BCD attains speedups of 3.6× (mnist8m), 4.5× (a9a), and 6.1× (covtype). When the BCD algorithm is flops and bandwidth dominated (i.e., Figure 8b), CA-BCD attains modest speedups of 1.2× (a9a), 1.8× (mnist8m), and 1.9× (covtype). The strong scaling behavior of the BDCD and CA-BDCD algorithms is shown in Figures 8c and 8d. CA-BDCD attains speedups of 1.6× (news20), 2.2× (rcv1), and 2.9× (e2006) when latency dominates and 1.1× (news20), 1.2× (rcv1), and 3.4× (e2006) when flops and bandwidth dominate. The e2006 dataset achieves greater speedup for $b = 8$ than $b = 1$. This is due to machine noise, which caused larger latency times for $b = 8$.

While we did not experiment with weak scaling, we can observe from our analysis (in section 4) that the BCD and BDCD algorithms achieve perfect weak scaling (in theory). It is likely that the CA-BCD and CA-BDCD algorithms would attain weak scaling speedups by reducing the latency cost by a factor of s, if latency dominates.

[2]For datasets with highly irregular sparsity structure, additional load balancing is likely required, but we leave this for future work.



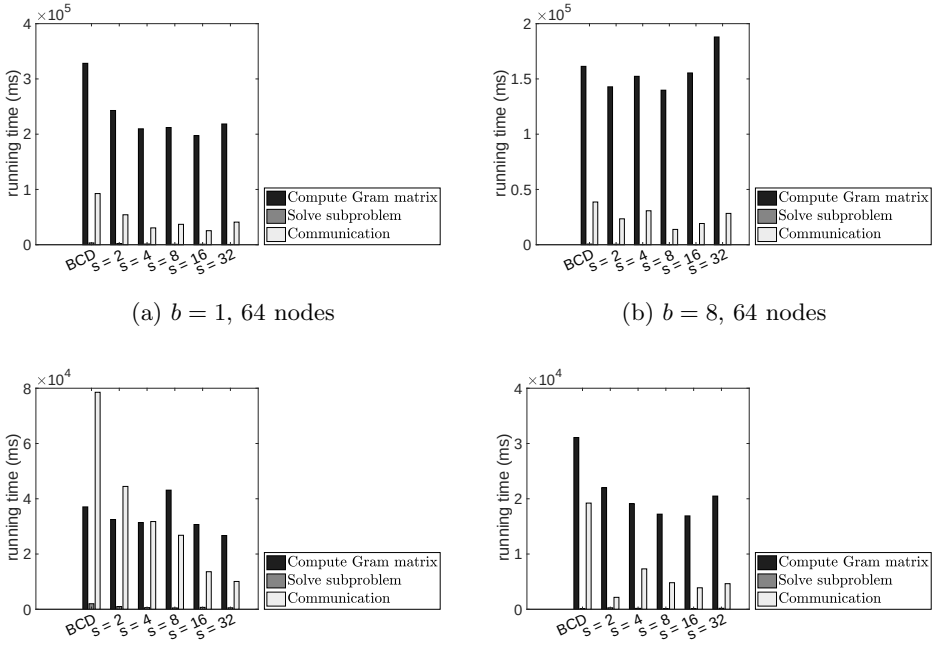
(a) $b = 1$ (b) $b = 8$

(c) $b' = 1$ (d) $b' = 8$

FIG. 8. *Strong scaling results for (CA-)BCD (top row, Figures* 8a–8b*) and (CA-)BDCD (bottom row, Figures* 8c–8d*). Ideal strong scaling behavior for BCD and BDCD to illustrate the performance improvements of the CA-variants.*

5.2.2. Running time breakdown. Figure 9 shows the running time breakdown of BCD and CA-BCD for $s \in \{2, 4, 8, 16, 32\}$ on the mnist8m dataset. We plot the breakdowns for $b \in \{1, 8\}$ at scales of 64 nodes and 1024 nodes to illustrate CA-BCD tradeoffs for different flops versus communication ratios. Figures 9a and 9b show the running time breakdown at 64 nodes for $b = 1$ and $b = 8$, respectively. In both cases flops dominate and speedup for CA-BCD is from faster flops. CA-BCD with $s > 1$ increases the computational intensity and achieves higher flops performance through the use of BLAS-3 GEMM operations. Flops scaling continues until CA-BCD becomes CPU-bound. For $b = 8$, memory-bandwidth is saturated at $s < 8$. For $s \geq 8$ CA-BCD becomes CPU-bound and does not attain any speedup over BCD. Since communication is more bandwidth dominated, less communication speedup is expected. On 1024 nodes (Figures 9c and 9d), where latency costs are more dominant, CA-BCD attains larger communication and overall speedups. These experiments suggest that appropriately chosen values of s can attain large speedups when latency dominates.

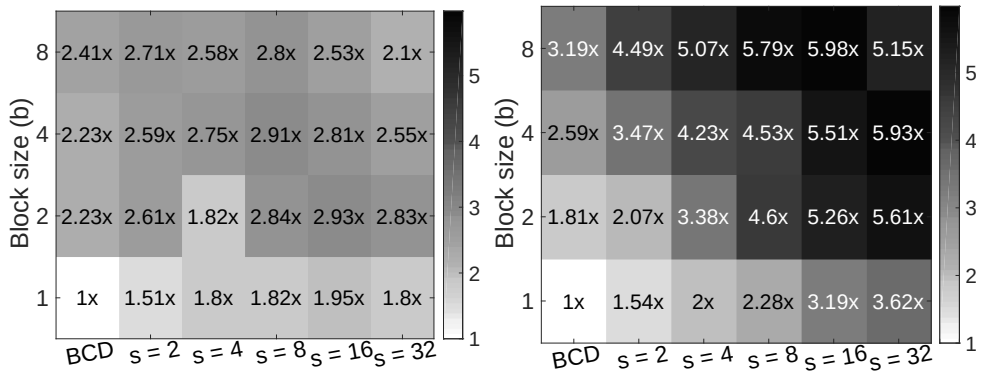
5.2.3. Speedup comparison. Figure 10 summarized the speedups attainable on the mnist8m dataset at 64 nodes and 1024 nodes for several combinations of block sizes (b) and recurrence unrolling values (s). We normalize the speedups to BCD with $b = 1$. At small scale (Figure 10a) we see speedups of 1.95× to 2.91× since flops and bandwidth are the dominant costs. The speedup for larger block sizes is due to faster convergence (i.e., fewer iterations and messages) and due to the use of BLAS-3 matrix-matrix operations. At large scale, when latency dominates (Figure 10b), we observe greater speedups of 3.62× to 5.98×. Overall, CA-BCD is fastest for all block



(a) $b = 1$, 64 nodes (b) $b = 8$, 64 nodes

(c) $b = 1$, 1024 nodes (d) $b = 8$, 1024 nodes

FIG. 9. *Running time breakdown for the mnist8m dataset for* $b \in \{1, 8\}$ *at scales of* 64 *and* $1,024$ *nodes. We plot the fastest timed run for each algorithm and setting.*



(a) Speedup of mnist8m on 64 nodes. (b) Speedup of mnist8m on 1024 nodes.

FIG. 10. *Speedups achieved for CA-BCD on mnist8m for various settings of* b *and* s*. We show speedups for* 64 *and* $1,024$ *nodes.*

sizes and at all scales tested.

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 in some settings, BCD and BDCD methods may converge faster than traditional Krylov methods—especially when the solution does not require high accuracy. We analyzed the computation, communication, and storage costs of the classical and communication-avoiding variants under two partitioning schemes. Our experiments showed that CA-BCD and CA-BDCD are numerically stable algorithms for all values of s tested and experimentally showed the tradeoff between algorithm parameters and convergence. Finally, we showed that the communication-avoiding variants can attain large speedups of up to 6.1× on a Cray XC30 supercomputer using MPI.

While CA-BCD and CA-BDCD appear to be stable, numerical analysis of these methods would be interesting directions for future work. Extending the CA technique to other algorithms (SGD, L-BFGS, Newton's method, etc.), regularization (LASSO, Elastic-net, etc.), and loss functions (SVM, logistic, etc.) would be particularly interesting.

REFERENCES

[1] G. Ballard, *Avoiding Communication in Dense Linear Algebra*, Ph.D. thesis, EECS Department, University of California, Berkeley, Berkeley, CA, 2013.

[2] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, *Communication lower bounds and optimal algorithms for numerical linear algebra*, Acta Numer., 23 (2014), pp. 1–155, https://doi.org/10.1017/S0962492914000038.

[3] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996, https://doi.org/10.1137/1.9781611971484.

[4] L. Bottou, *Large-scale machine learning with stochastic gradient descent*, in Proceedings of Computation Statistics, Springer, New York, 2010, pp. 177–186, https://doi.org/10.1007/978-3-7908-2604-3_16.

[5] E. Carson, *Communication-Avoiding Krylov Subspace Methods in Theory and Practice*, Ph.D. thesis, EECS Department, University of California, Berkeley, Berkeley, CA, 2015.

[6] E. Carson and J. Demmel, *A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 22–43, https://doi.org/10.1137/120893057.

[7] E. Carson and J. W. Demmel, *Accuracy of the s-step Lanczos method for the symmetric eigenproblem in finite precision*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 793–819, https://doi.org/10.1137/140990735.

[8] E. Carson, N. Knight, and J. Demmel, *Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods*, SIAM J. Sci. Comput., 35 (2013), pp. S42–S61, https://doi.org/10.1137/120881191.

[9] E. Carson, N. Knight, and J. Demmel, *An efficient deflation technique for the communication-avoiding conjugate gradient method*, Electron. Trans. Numer. Anal., 43 (2014), pp. 125–141.

[10] C.-C. Chang and C.-J. Lin, *LIBSVM: A library for support vector machines*, ACM Trans. Intelligent Syst. Tech., 2 (2011), pp. 1–27, https://doi.org/10.1145/1961189.1961199.

[11] A. Chronopoulos and C. Gear, *On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy*, Parallel Comput., 11 (1989), pp. 37–53, https://doi.org/10.1016/0167-8191(89)90062-8.

[12] A. Chronopoulos and C. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168, https://doi.org/10.1016/0377-0427(89)90045-9.

[13] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar, *A survey of direct methods for sparse linear systems*, Acta Numer., 25 (2016), pp. 383–566, https://doi.org/10.1017/S0962492916000076.

[14] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239, https://doi.org/10.1137/080731992.

[15] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding Communication in Computing Krylov Subspaces*, Tech. report UCB/EECS-2007-123, EECS Department, University of California, Berkeley, Berkeley, CA, 2007.
[16] M. P. I. Forum, *MPI: A Message-Passing Interface Standard*, University of Tennessee, Knoxville, TN, 1994.
[17] G. H. Gonnet, *Expected length of the longest probe sequence in hash code searching*, J. ACM, 28 (1981), pp. 289–304, https://doi.org/10.1145/322248.322254.
[18] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, Berkeley, Berkeley, CA, 2010.
[19] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, *Communication-efficient distributed dual coordinate ascent*, in Proceedings of the 27th International Conference on Neural Information Processing Systems, Montreal, Canada, 2014, pp. 3068–3076.
[20] S. Kim and A. Chronopoulos, *An efficient nonsymmetric Lanczos method on parallel vector computers*, J. Comput. Appl. Math., 42 (1992), pp. 357–374, https://doi.org/10.1016/0377-0427(92)90085-C.
[21] K. Lang, *NewsWeeder: Learning to filter netnews*, in Proceedings of the 12th International Machine Learning Conference, Tahoe City, CA, 1995, https://doi.org/10.1016/B978-1-55860-377-6.50048-7.
[22] M. Lichman, *UCI Machine Learning Repository*, 2013, http://archive.ics.uci.edu/ml.
[23] A. McCallum, *SRAA: Simulated/Real/Aviation/Auto UseNet Data*, https://people.cs.umass.edu/~mccallum/data.html.
[24] M. D. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, Ph.D. thesis, EECS Department, University of California, Berkeley, Berkeley, CA, 1996.
[25] M. Mohiyuddin, *Tuning Hardware and Software for Multiprocessors*, Ph.D. thesis, EECS Department, University of California, Berkeley, Berkeley, CA, 2012.
[26] 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, ACM, New York, 2009, 36, https://doi.org/10.1145/1654059.1654096.
[27] NERSC, *NERSC Edison Configuration*, http://www.nersc.gov/users/computational-systems/edison/configuration/.
[28] Y. Nesterov, *Efficiency of coordinate descent methods on huge-scale optimization problems*, SIAM J. Optim., 22 (2012), pp. 341–362, https://doi.org/10.1137/100802001.
[29] Z. Qu, P. Richtárik, M. Takáč, and O. Fercoq, *SDNA: Stochastic dual Newton ascent for empirical risk minimization*, in Proceedings of the 33rd International Conference on Machine Learning, Volume 48, New York, NY, 2016, pp. 1823–1832, http://dl.acm.org/citation.cfm?id=3045390.3045583.
[30] M. Raab and A. Steger, *"Balls into Bins"—A Simple and Tight Analysis*, in Randomization and Approximation Techniques in Computer Science, Springer, New York, 1998, pp. 159–170, https://doi.org/10.1007/3-540-49543-6_13.
[31] 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, Granada, Spain, 2011, pp. 693–701.
[32] P. Richtárik and M. Takáč, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Math. Program., 144 (2014), pp. 1–38, https://doi.org/10.1007/s10107-012-0614-z.
[33] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003, https://doi.org/10.1137/1.9780898718003.
[34] S. Shalev-Shwartz and T. Zhang, *Stochastic dual coordinate ascent methods for regularized loss*, J. Mach. Learn. Res., 14 (2013), pp. 567–599.
[35] E. Solomonik, *Provably Efficient Algorithms for Numerical Tensor Algebra*, Ph.D. thesis, EECS Department, University of California, Berkeley, Berkeley, CA, 2014.
[36] M. Takáč, P. Richtárik, and N. Srebro, *Distributed Mini-Batch SDCA*, preprint, https://arxiv.org/abs/1507.08322, 2015.
[37] The MathWorks, *Box Plots*, https://www.mathworks.com/help/stats/box-plots.html.
[38] J. Van Rosendale, *Minimizing Inner Product Data Dependencies in Conjugate Gradient Iteration*, IEEE Computer Society, Los Alamitos, CA, 1983.
[39] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 152–163, https://doi.org/10.1137/0909010.

[40] 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 Proceedings of the International Parallel and Distributed Processing Symposium, Phoenix, AZ, 2014, pp. 1149–1158, https://doi.org/10.1109/IPDPS.2014.119.

[41] S. J. Wright, *Coordinate descent algorithms*, Math. Program., 151 (2015), pp. 3–34, https://doi.org/10.1007/s10107-015-0892-3.