SIAM J. SCI. COMPUT.
Vol. 43, No. 2, pp. C154–C176

PARALLEL AND COMMUNICATION AVOIDING LEAST ANGLE REGRESSION*

SWAPNIL DAS†, JAMES DEMMEL†, KIMON FOUNTOULAKIS‡, LAURA GRIGORI§, MICHAEL W. MAHONEY¶, AND SHENGHAO YANG‡

Abstract. We are interested in parallelizing the least angle regression (LARS) algorithm for fitting linear regression models to high-dimensional data. We consider two parallel and communication avoiding versions of the basic LARS algorithm. The two algorithms have different asymptotic costs and practical performance. One offers more speedup and the other produces more accurate output. The first is bLARS, a block version of the LARS algorithm, where we update b columns at each iteration. Assuming that the data are row-partitioned, bLARS reduces the number of arithmetic operations, latency, and bandwidth by a factor of b. The second is tournament-bLARS (T-bLARS), a tournament version of LARS where processors compete by running several LARS computations in parallel to choose b new columns to be added in the solution. Assuming that the data are column-partitioned, T-bLARS reduces latency by a factor of b. Similarly to LARS, our proposed methods generate a sequence of linear models. We present extensive numerical experiments that illustrate speedups up to 4x compared to LARS without any compromise in solution quality.

Key words. communication avoiding, least angle regression, parallel methods

AMS subject classifications. 68W10, 68W15

DOI. 10.1137/19M1305720

1. Motivation and outline. Recently, there has been large growth in data for many applications in statistics, machine learning, and signal processing, which poses the need for powerful computer hardware as well as new algorithms that utilize the new hardware efficiently. Commercial hardware companies started to construct multicore designs because the performance of single central processing units (CPUs) is stagnating due to heat issues, i.e., "the Power Wall" problem [31]. In terms of software and algorithm implementations for processing large-scale data, the increased number of cores might require synchronization among them, which results in data transfer between levels of a memory hierarchy or between CPUs over a network. For this reason the total running time of a parallel algorithm depends on the number of arithmetic operations (computational costs) and the cost of data movement (communication costs). The communication cost includes the "bandwidth cost," i.e., the number of bytes, or more abstractly, number of words, sent among cores for synchronization purposes, and the "latency cost," i.e., the number of messages sent. On modern computer architectures, communicating data often takes much longer than performing a floating-point operation and this gap is continuing to increase [35]. Therefore, it is especially important to design algorithms that minimize communication in order to attain high performance on modern computer architectures.

*Submitted to the journal's Software and High-Performance Computing section December 10, 2019; accepted for publication (in revised form) December 9, 2020; published electronically March 15, 2021. Authors ordered alphabetically.

https://doi.org/10.1137/19M1305720

†Computer Science Division and Department of Mathematics, University of California Berkeley, Berkeley, CA 94720-1776 USA (tracer@berkeley.edu, demmel@berkeley.edu).

‡School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L3G1, Canada (kfountou@uwaterloo.ca, shenghao.yang@uwaterloo.ca).

§Alpines group, INRIA Paris, Paris, 75005, France (laura.grigori@inria.fr).

¶International Computer Science Institute, Department of Statistics, University of California Berkeley, Berkeley, CA 94720 USA (mmahoney@stat.berkeley.edu).

In this paper we will propose two novel parallel and communication avoiding versions of the least angle regression (LARS) algorithm which is a very popular method for sparse linear regression [17]. A plethora of applications in statistics [17], machine learning [29], and signal processing/compressed sensing [4] utilize sparse linear models. To the best of our knowledge, there is no study on parallelizing LARS.

2. Introduction to the problem, existing models, and LARS. Let $A \in \mathbb{R}^{m \times n}$ be a data matrix with m samples and n features. We are concerned with the problem of finding a vector $\mathbf{y} := A\mathbf{x}$ that approximates a given vector $\mathbf{b} \in \mathbb{R}^m$, where vector $\mathbf{y}$ is a linear combination of a few columns/features of the given data matrix A. This means that we are looking for a coefficients vector $\mathbf{x}$ that is sparse, i.e., it has a small number of nonzeros.

Over the years, many algorithms/models to solve this problem have been proposed. In what follows, we review the ones that to the best of our knowledge are the most important. There are two main categories of algorithms/models to solve this problem. The first category consists of algorithms that progressively select a subset of columns/features based on their absolute correlation with the residual vector $\mathbf{y} - \mathbf{b}$. In particular, the classic Forward Selection algorithm in section 8.5 in [40] selects the first column/feature with the largest absolute correlation with the response $\mathbf{b}$. Let us denote the index of the selected column with i, the corresponding column with A_i and the corresponding coefficient with x_i. The next step of the algorithm is to solve a simple linear regression problem

$$\min \ \frac{1}{2}\|A_i x_i - \mathbf{b}\|_2^2.$$

By solving this simple regression problem we obtain the value of the optimal coefficient x_i. The residual $\mathbf{r} := A_i x_i - \mathbf{b}$, which is orthogonal to A_i, is now considered the new response vector for the next iteration. Finally, we orthogonally project the remaining columns in A to A_i. Then we have to repeat this process and find a new column/feature. After k iterations we will have selected k columns, and we use the k columns to solve smaller ordinary regression problem using the response vector $\mathbf{b}$. According to [17], in practice the Forward Selection algorithm might be aggressive in terms of selecting features since other columns might be correlated with the selected column A_i that we ignored. Another algorithm in this category is the Forward Stagewise algorithm [19, 20], which in comparison to Forward Selection is much more cautious since it requires much more steps to converge to a k-sparse model, i.e., k selected columns. More precisely, at each iteration of the Forward Stagewise we select the column that is most correlated with the current residual and increment the corresponding coefficient in the vector $\mathbf{x}$ by a small amount $\pm\epsilon$, where the sign is determined based on the sign of the correlation. The small increment of elements in $\mathbf{x}$ at each iteration is what distinguishes Forward Stagewise and Forward Selection.

The second category of models is optimization based, meaning that we solve a predefined optimization problem to obtain a sparse linear model. There are two subclasses of optimization problems in this category, the first is known as ℓ_1-regularized linear regression or least absolute shrinkage and selection operator (LASSO) [38], the second is ℓ_0-regularized variants. Let us first define the ℓ_1 and ℓ_0 norms and then we will continue by presenting the optimization problems. The ℓ_1 norm of a vector $\mathbf{x}$ is defined as $\|\mathbf{x}\|_1 := \sum_{i=1}^n |x_i|$, while the ℓ_0 norm is defined as $\|\mathbf{x}\|_0 :=$

{number of nonzero elements in $\mathbf{x}$}. Equipped with these definitions we define LASSO:

$$\begin{aligned} &\text{minimize} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 \\ &\text{subject to } \|\mathbf{x}\|_1 \leq \lambda, \end{aligned} \tag{1}$$

where λ is a model parameter. LASSO is a convex optimization problem and can be solved in polynomial time; we discuss several serial and parallel algorithms later in this paper. The LASSO optimization problem is likely to have a set of sparse optimal solutions due to the sparsity inducing ℓ_1-ball constraint. For details we refer the reader to [38]. A nonconvex alternative of LASSO, but with a direct constraint on the sparsity of $\mathbf{x}$, is the ℓ_0-regularized linear regression problem

$$\begin{aligned} &\text{minimize} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 \\ &\text{subject to } \|\mathbf{x}\|_0 \leq \tau, \end{aligned} \tag{2}$$

where τ is a model parameter that bounds the number of nonzeros in $\mathbf{x}$. This is an NP-hard problem. However, we can find local solutions by variants of gradient descent, which we discuss later in this paper.

An important difference between the two approaches, i.e., Forward Selection or Stagewise versus LASSO, is that the former obtains a sequence of solutions $\mathbf{x}_k$ with increasing number of nonzeros, while with the latter we obtain a solution path $\mathbf{x}(\lambda)$. There is a question regarding how those two solution paths differ in terms of the selected features. The LARS algorithm is an algorithmic framework that unifies those two approaches. In particular, the LARS algorithm has been motivated by the Forward Selection and Stagewise algorithms. Therefore, in terms of steps it is similar to those as we will see later, but it is also proved in Theorem 1 in [17] that a certain version of LARS produces a sequence of solutions $\mathbf{x}_k$ that is equivalent to the solution path $\mathbf{x}(\lambda)$. Let us now summarize the steps of the LARS algorithm. This algorithm is discussed in detail in section 6. Similarly to Forward algorithms, at the first iteration of LARS we initialize the algorithm by selecting the column with the largest absolute correlation with vector $\mathbf{b}$. The next step is to update vector $\mathbf{y}$. Instead of solving a simple regression problem like in Forward Selection (which is an aggressive strategy) or making ϵ updates to $\mathbf{x}$ (which is too cautious), we define a vector $\mathbf{u}$ that is equiangular with all previous chosen columns and then we update $\mathbf{y} := \mathbf{y} + \mathbf{u}\gamma$. The step-size $\gamma \in \mathbb{R}$ is set such that the new column to be added in the next iteration has the same correlation with the new residual vector as with all other selected columns so far. This process might sound complicated at first but we will revisit the linear algebra behind these decisions in section 6.

3. Our contributions. Although there are numerous parallel optimization algorithms for ℓ_0- and ℓ_1-regularized regression, we are not aware of any parallel and communication avoiding versions for LARS. To the best of our knowledge, the proposed algorithms are the first parallel versions of LARS that are also communication avoiding. Let us briefly describe the proposed algorithms and the most significant ideas that had to be developed to establish them.

The first method is a block version of LARS (bLARS) which is described in section 7. Instead of adding one feature at each iteration in the solution set, we add b features at a time. By blocking operations and by partitioning the data per row, we are able to show that we decrease the arithmetic, latency, and bandwidth costs by a factor of b. Extensive numerical experiments in section 10 illustrate significant speedups for

bLARS without compromising too much of the quality of the output compared to LARS. In the same section we empirically study the trade-off between the size of b and the quality of the output compared to LARS.

Careful modification of the linear algebra had to be performed in order to successfully generalize LARS to the block case and also guarantee that all steps of the algorithm are well defined. More precisely, LARS has two important properties that we had to relax. The first is that all chosen columns at each iteration have the same absolute correlation with the residual and are also maximally correlated. The second property is that the direction $\mathbf{u}$ is equiangular and also has maximal correlation with the chosen columns. bLARS maintains the property that the chosen columns at each iteration are maximally correlated but they are not equal, meaning that there is no column that has not been selected with larger absolute correlation with the residual than the selected ones. bLARS also relaxes the second property in the sense that $\mathbf{u}$ is not equiangular with all chosen columns but it is maximally correlated, i.e., there is no column that has not been selected with larger correlation than the selected ones. We show that bLARS at each iteration reduces the correlations for all selected columns similarly to LARS. Finally, if we set $b = 1$, then bLARS reduces to LARS.

The second method is a tournament block LARS (T-bLARS) method. In this method the data are partitioned per column and distributed to processors. Then each processor calls a modified version of the LARS algorithm on its local data. Each processor can run the modified LARS algorithm for b iterations so that b columns are chosen at termination of the local call to LARS. Using a generalized tree-reduction operation each processor/node sends its chosen columns to the parent node (starting from the bottom of the tree). The parent node calls again the modified LARS algorithm by utilizing only the columns that have been sent from the child nodes. This process repeats until we reach the root node where the final output is used to update the current vector $\mathbf{y}$ and current set of selected columns. By partitioning the data per column (as opposed to per row for block LARS) and using the generalized tree-reduction we allow the nodes to work in parallel in local data and this way we reduce latency by a factor of b. Many of the properties of the LARS algorithm are not satisfied at a global level but some of them are maintained during the local calls to LARS. We discuss details in section 8. In section 10 we show that T-bLARS can be faster than the original LARS without compromising the quality of the output. Similarly to bLARS, we study the tradeoff between speed and quality of output as we vary parameter b and the number of processors.

4. Literature review for parallel models and methods. The dependence of the running time of parallel methods on communication requirements gave a totally new perspective on how to efficiently parallelize existing algorithms. Communication avoiding algorithms became a very popular subject of study and it has been demonstrated that such algorithms exhibit large speedups on modern, distributed-, and shared-memory parallel architectures through careful algorithmic modifications [3]. Many iterative methods for linear systems and matrix decomposition algorithms have been reorganized to avoid communication, which has led to significant performance improvements over existing state-of-the-art libraries [2, 3, 6, 21, 36, 41].

The origins of communication avoiding algorithms lie in the s-step conjugate gradients method [39] by Van Rosendale and in the work of Chronopoulos on parallel iterative methods for linear systems [8]. More precisely, Chronopoulos and Gear developed s-step methods for symmetric linear systems [9, 10], Chronopoulos and Swanson developed s-step methods for unsymmetric linear systems [11], and Kim and

Chronopoulos developed an s-step nonsymmetric Lanczos method [23]. Furthermore, Demmel et al. [15, 21, 24, 25] 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. Finally, Carson, Demmel, and Hoemmen developed communication avoiding Krylov subspace methods [6, 15, 21] by combining the matrix powers kernel and s-step methods.

The above results are mainly focused on iterative methods for least-squares and linear systems. Our focus in this paper is sparse linear regression where we require the coefficients of the model to be sparse. As is mentioned in section 2 there are two categories of methods that can solve this problem efficiently. The first is LARS-type algorithms. To the best of our knowledge, there are no studies on parallelizing LARS. However, we will see in section 7 that the computational bottleneck for LARS is computing matrix-vector products. Therefore, a straightforward approach for parallelizing LARS is to make use of parallel matrix-vector products. There are numerous works on parallelizing matrix-vector product calculations [32]. In our experiments in section 10 we do compare the two proposed methods with a LARS implementation that uses parallel matrix-vector products. Similarly, the proposed bLARS algorithm in section 7 relies on matrix-matrix products which can also be efficiently parallelized [32]. The proposed T-bLARS algorithm divides the problem into smaller problems that are solved in parallel, and then we aggregate the results by allowing processors to compete. This strategy is similar to [14] for parallel QR and LU algorithms, where pivoting is performed in parallel by using a generalized tree reduction operation. Although we also use a generalized tree-reduction operation, at each leaf of the tree we perform a LARS operation and not a pivoting operation. Additionally, we modify a crucial part of the LARS algorithm, i.e., the calculation of the step-size, to guarantee that all steps are well defined. Details are discussed in section 8.

Recently, there have been numerous works regarding parallel optimization algorithms. ℓ_1-regularization problems often appear in statistics [17], machine learning [29], and signal processing/compressed sensing [4] where there is a vast amount of data available, i.e., matrix A has millions if not billions of samples and features. Large scale problems are the main reason for the resurgence in methods with computationally inexpensive iterations. Many modern first-order methods meet the previous goal. For instance, for ℓ_1-regularized least-squares problems coordinate descent methods can have up to n times less computational complexity per iteration than methods which use full gradient steps while at the same time coordinate descent methods achieve very fast progress to optimality [28, 34, 42]. However, it is shown in [16] that the running time for such methods is often dominated by communication cost which increases with the number of processors. In the same work [16] the authors show how to avoid communication for an s-step accelerated proximal block coordinate descent and demonstrate up to 5x speedup compared to parallelized alternatives. Moreover, there are parallel accelerated and proximal coordinate descent methods [18] that do not use the s-step technique but allow coordinate updates to happen without synchronization. For example, HOGWILD! [33] 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. Finally, there are some frameworks and algorithms that attempt to reduce the communication bottleneck by reducing the number of iterations. For example, the CoCoA framework [22] reduces communication by performing coordinate descent on locally stored data points on each processor and intermittently communicating by summing or averaging the local solutions. Regarding

ℓ_0-regularization, there are not many works in terms of parallel methods; a notable work is that of Needell and Woolf [27]. In this paper the authors suggest an asynchronous parallel and stochastic greedy algorithm, where multiple processors asynchronously update a vector in shared memory containing information on the estimated coefficients vector $\mathbf{x}$. Finally, one could also easily parallelize gradient-based methods for ℓ_1- and ℓ_0-regularization by parallelizing the computation of the gradients which relies on matrix-vector products.

Note that parallel optimization based methods aim at solving a single instance of ℓ_1- or ℓ_0-regularized least-squares, i.e., they produce a single sparse linear model. In this paper we are interested in algorithms that produce a sequence of sparse linear models.

5. Preliminaries and assumptions.

5.1. Preliminaries. Capital letters denote matrices, lowercase bold letters denote vectors, lowercase letters denote scalars, and hollow letters denote sets. We denote with $\mathbf{0}_m$ a vector of zeros of length m. Subscript k denotes the kth iteration of the algorithm. The set of positive integers is denoted by $\mathbb{Z}_+$. We use $[\cdot]_{set}$, to denote a function with a vector as an input that returns a subvector which corresponds to the indices in the subscript set. A^T denotes the transpose of a matrix. We denote with A_{set} the concatenation of columns of matrix A with indices in the subscript set. We denote the complement of a set by using the superscript c. We use the function $sign(\cdot)$ to denote the sign function which is applied componentwise if the input is a vector. We use the convention that $sign(0) = 0$. We use $\|\cdot\|_\infty$ to denote the infinity norm, i.e., maximum absolute component of the input, and define $\|\cdot\|_{\infty,k}$ to be the sum of k largest absolute components of the input. We define $abs(\cdot)$ as the absolute function which is often applied componentwise. We define the function $\max^b(\cdot)$ and $\arg\max^b(\cdot)$ as the bth maximum of the input vector and the indices of the b largest components of the input vector, respectively. If the input vector has less than b components, then the latter functions overwrite b to be the length of the input vector. We define $\min^b(\cdot)$ and $\arg\min^b(\cdot)$ similarly. The function $\min^+(\cdot)$ returns the minimum positive value. The symbol $\emptyset$ denotes the empty set. We denote the simple multiplication of two scalars a and b by $a \cdot b$. By log we denote the logarithm with base 2.

5.2. Assumptions. For simplicity, we assume that the columns of matrix A have unit ℓ_2 norm, and that matrix A is full-rank. For bLARS, we also assume that every b columns are linearly independent. However, minor modifications to the algorithms can be done to bypass these assumptions. We assume that the communication cost includes the "bandwidth cost," i.e., the number of words, sent among cores for synchronization purposes, and the "latency cost," i.e., the number of messages sent.

6. Least angle regression.

In this section we review the LARS algorithm. LARS is shown in Algorithm 1. The termination criterion in step 2 of Algorithm 1 is arbitrary; one can choose other criteria such as a lower bound on the maximum absolute correlation $\|\mathbf{c}_k\|_\infty$; see [17]. Let us explain the first iteration of the algorithm. Let us assume that at the 0th iteration we have response $\mathbf{y}_0$, residual vector $\mathbf{r}_0 = \mathbf{b} - \mathbf{y}_0$, correlation vector $\mathbf{c}_0 := A^T\mathbf{r}_0$, and maximum absolute correlation $c_0 := \max|\mathbf{c}_0|$. The algorithm starts by choosing all columns that have maximum absolute correlation

$$\mathbb{I}_0 := \{i \in [n] \mid |[\mathbf{c}_0]_i| = c_0\}. \tag{3}$$

The next decision step is how to set $\mathbb{I}_1$ and $\mathbf{y}_1$ using $\mathbb{I}_0$ and $\mathbf{y}_0$. We will define the update as $\mathbf{y}_1 := \mathbf{y}_0 + \mathbf{u}_0\gamma_0$. This implies that we will have to define the vector

$\mathbf{u}_0$ and the step-size γ_0. Let us start with the definition of $\mathbf{u}_0$. LARS defines $\mathbf{u}_0$ as a unit-length vector that is equiangular with signed columns in matrix A with index in $\mathbb{I}_0$. It is easy to see that $\mathbf{u}_0 := A_{\mathbb{I}_0}(A_{\mathbb{I}_0}^T A_{\mathbb{I}_0})^{-1} sign([\mathbf{c}_0]_{\mathbb{I}_0}) c_0 h_0$, where $h_0 := \|A_{\mathbb{I}_0}(A_{\mathbb{I}_0}^T A_{\mathbb{I}_0})^{-1} sign([\mathbf{c}_0]_{\mathbb{I}_0}) c_0\|_2^{-1}$, satisfies the requirements. This means that $A_{\mathbb{I}_0}^T \mathbf{u}_0 = sign([\mathbf{c}_0]_{\mathbb{I}_0}) c_0 h_0$, which in turn implies that subject to sign changes and because the columns of $A_{\mathbb{I}_0}$ and $\mathbf{u}_0$ are unit-length then $\mathbf{u}_0$ is equiangular with all columns in $\mathbb{I}_0$, with cosine $\pm c_0 h_0$. To define γ_0 and to update $\mathbb{I}_1$ based on γ_0 we will need first to understand how the update rule $\mathbf{y}_1 := \mathbf{y}_0 + \mathbf{u}_0\gamma_0$ affects the correlation vector $\mathbf{c}_1$ as a function of γ_0. For this we will make use of the auxiliary vector $\mathbf{a}_0 := A^T \mathbf{u}_0$, and we will use a different step-size γ_j for each element j. In particular, we have that $[\mathbf{c}_1]_j(\gamma_j) = A_j^T(\mathbf{b} - \mathbf{y}_0 - \mathbf{u}_0\gamma_j) = [\mathbf{c}_0]_j - [\mathbf{a}_0]_j\gamma_j \ \forall j \in \mathbb{I}_0^c$ and

$$[\mathbf{c}_1]_j(\gamma_j) = sign([\mathbf{c}_0]_j)(1 - \gamma_j h_0) c_0 \ \forall j \in \mathbb{I}_0. \tag{4}$$

Equation (4) uses $[\mathbf{a}_0]_{\mathbb{I}_0} = A_{\mathbb{I}_0}^T \mathbf{u}_0 = sign([\mathbf{c}_0]_{\mathbb{I}_0}) c_0 h_0$ and that vector $[\mathbf{c}_0]_{\mathbb{I}_0}$ has components of magnitude equal to c_0 since it satisfies the definition in (3). Notice that if $\gamma_j = 1/h_0$, then $[\mathbf{c}_1]_j(\gamma_j) = 0 \ \forall j \in \mathbb{I}_0$, which means that the least-squares problem is minimized with respect to the chosen columns in $\mathbb{I}_0$. Although tempting, this is not the goal of LARS since this is an aggressive strategy similar to Forward Selection. As we increase γ_j from 0 to $1/h_0$ the absolute correlations in $\mathbb{I}_0$ are decreased *identically*; see (4). This is because the absolute correlations for the columns in $\mathbb{I}_0$ are equal. However, the absolute correlations in $\mathbb{I}_0^c$ might increase or decrease. LARS' goal is to find a column in $\mathbb{I}_0^c$ whose absolute correlation becomes equal to the maximum absolute correlation as we increase γ_0. To find such a column, we need to find γ_j for each $j \in \mathbb{I}_0^c$ such that

$$c_0(1 - \gamma_j h_0) = |[\mathbf{c}_0]_j - \gamma_j [\mathbf{a}_0]_j|. \tag{5}$$

Such γ_j will guarantee that column $j \in \mathbb{I}_0^c$ has the same absolute correlation as the columns with index in $\mathbb{I}_0$. It remains to check if (5) has a solution. It has two solutions, out of which we keep the minimum positive one

$$\gamma_j := \min{}^+ \left(\frac{c_0 - [\mathbf{c}_0]_j}{c_0 h_0 - [\mathbf{a}_0]_j}, \frac{c_0 + [\mathbf{c}_0]_j}{c_0 h_0 + [\mathbf{a}_0]_j} \right).$$

Out of all γ_j where $j \in \mathbb{I}_0^c$, we choose the one with the minimum value $\gamma_0 := \min_{j \in \mathbb{I}_0^c} \gamma_j$. Note that the minimum step-size γ_0 corresponds to the column(s) in $\mathbb{I}_0^c$ that will be the first to have the same *maximal* absolute correlation as the columns in $\mathbb{I}_0$. Then LARS updates the set of selected columns as $\mathbb{I}_1 := \mathbb{I}_0 \cup \{\arg\min_{j \in \mathbb{I}_0^c} \gamma_j\}$. The chosen column is the column with the least-angle which is where LARS gets its name from. Finally, having the step-size γ_0 we update the response $\mathbf{y}_1 := \mathbf{y}_0 + \gamma_0 \mathbf{u}_0$.

It is easy to show that our claims above hold for any iteration k. Therefore, it is easy to show that LARS guarantees that $\mathbb{A}_k \subset \mathbb{A}_{k+1}$ and $|\mathbb{A}_k| = |\mathbb{A}_{k+1}| + 1 \ \forall k$. Moreover, LARS decreases the maximum absolute correlation c_k until it finally is equal to zero for $k = \min(m, n)$. Furthermore, the columns in $\mathbb{A}_k$ have maximum absolute correlations $\forall k$. Therefore, using (4) we see that LARS decreases $\|\mathbf{c}_k\|_\infty$ at each iteration. Furthermore, note that LARS also decreases $\|\mathbf{c}_k\|_{\infty,k} :=$ sum of k largest absolute components; as we will see later this is a property that bLARS generalizes for the $k \cdot b$ largest components.

7. Parallel bLARS. In this section, we describe one iteration of bLARS (without going into any details about parallelism), and then we explain how we can parallelize bLARS.

Algorithm 1. LARS.

Initialize $k := 0$, $\mathbf{y}_k := \mathbf{0}_m$, $\mathbf{r}_k := \mathbf{b}$, $\mathbf{c}_k := A^T\mathbf{r}_k$, $i := \arg\max |\mathbf{c}_k|$, $c_k := \max |\mathbf{c}_k|$, $\mathbb{I}_k := \{i\}$, $t \leq \min(m, n)$
while $|\mathbb{I}_k| \leq t$ **do**
  $\mathbf{u}_k := A_{\mathbb{I}_k}(A_{\mathbb{I}_k}^T A_{\mathbb{I}_k})^{-1} sign([\mathbf{c}_k]_{\mathbb{I}_k}) h_k c_k$, where $h_k := \|A_{\mathbb{I}_k}(A_{\mathbb{I}_k}^T A_{\mathbb{I}_k})^{-1} sign([\mathbf{c}_k]_{\mathbb{I}_k}) c_k\|_2^{-1}$
  $\gamma_j := \min^+ \left(\frac{c_k - [\mathbf{c}_k]_j}{c_k h_k - [\mathbf{a}_k]_j}, \frac{c_k + [\mathbf{c}_k]_j}{c_k h_k + [\mathbf{a}_k]_j} \right) \forall j \in \mathbb{I}_k^c$, where $\mathbf{a}_k := A^T\mathbf{u}_k$
  $\gamma_k := \min_{j \in \mathbb{I}_k^c} \gamma_j$, $i := \arg\min_{j \in \mathbb{I}_k^c} \gamma_j$, $\mathbb{I}_{k+1} := \mathbb{I}_k \cup \{i\}$
  $\mathbf{y}_{k+1} := \mathbf{y}_k + \mathbf{u}_k \gamma_k$
  $\mathbf{c}_{k+1} := A^T\mathbf{r}_{k+1}$, where $\mathbf{r}_{k+1} := \mathbf{b} - \mathbf{y}_{k+1}$
  $c_k := \max |\mathbf{c}_k|$
  $k := k + 1$
end while
Return $\mathbb{I}_k$, $\mathbf{y}_k$

Let us assume that at the 0th iteration of bLARS we have response $\mathbf{y}_0$, residual vector $\mathbf{r}_0 = \mathbf{b} - \mathbf{y}_0$, correlation vector $\mathbf{c}_0 := A^T\mathbf{r}_0$, and the bth maximum correlation $c_0 := \max^b |\mathbf{c}_0|$. The algorithm chooses all columns that have larger or equal absolute correlation than the maximum bth absolute correlation $\mathbb{I}_0 = \{i \in [n] \mid |[\mathbf{c}_0]_i| \geq c_0\}$. Similarly to LARS, we define the update as $\mathbf{y}_1 := \mathbf{y}_0 + \mathbf{u}_0\gamma_0$, but the decision rules for selecting $\mathbf{u}_0$, γ_0 and updating $\mathbb{I}_0$ and $\mathbf{y}_0$ are different. bLARS defines $\mathbf{u}_0$ as $\mathbf{u}_0 := A_{\mathbb{I}_0}(A_{\mathbb{I}_0}^T A_{\mathbb{I}_0})^{-1}[\mathbf{c}_0]_{\mathbb{I}_0} h_0$ and $h_0 := \|A_{\mathbb{I}_0}(A_{\mathbb{I}_0}^T A_{\mathbb{I}_0})^{-1}[\mathbf{c}_0]_{\mathbb{I}_0}\|_2^{-1}$. This means that $\mathbf{u}_0$ is a unit-length vector that satisfies $A_{\mathbb{I}_0}^T\mathbf{u}_0 = [\mathbf{c}_0]_{\mathbb{I}_0} h_0$, instead of $A_{\mathbb{I}_0}^T\mathbf{u}_0 = \text{sign}([\mathbf{c}_0]_{\mathbb{I}_0}) c_0 h_0$ for LARS. Note that $\mathbf{u}_0$ is not guaranteed to be equiangular to the chosen columns in $\mathbb{I}_0$. This is because $[\mathbf{c}_0]_{\mathbb{I}_0}$ is not guaranteed to have components with equal value. On the contrary, LARS guarantees that all components of $[\mathbf{c}_0]_{\mathbb{I}_0}$ are equal to the maximum absolute correlation. However, bLARS still guarantees that there is no column that has not been selected with absolute correlation larger than the bth maximum absolute correlation. Similarly to LARS, we will make use of the auxiliary vector $\mathbf{a}_0 := A^T\mathbf{u}_0$, but we will use different step-sizes γ_j for each element j. In particular, we have that $[\mathbf{c}_1]_j(\gamma_j) = A_j^T(\mathbf{b} - \mathbf{y}_0 - \mathbf{u}_0\gamma_j) = [\mathbf{c}_0]_j - [\mathbf{a}_0]_j\gamma_j \ \forall j \in \mathbb{I}_0^c$, where $\mathbb{I}_0^c$ is the complement of $\mathbb{I}_0$, and

$$[\mathbf{c}_1]_j(\gamma_j) = [\mathbf{c}_0]_j(1 - \gamma_j h_0) \ \forall j \in \mathbb{I}_0. \tag{6}$$

The last equality uses $[\mathbf{a}_0]_{\mathbb{I}_0} = A_{\mathbb{I}_0}^T\mathbf{u}_0 = [\mathbf{c}_0]_{\mathbb{I}_0} h_0$. This is different from LARS which uses $[\mathbf{a}_0]_{\mathbb{I}_0} = \text{sign}([\mathbf{c}_0]_{\mathbb{I}_0}) c_0 h_0$. This means that as we increase γ_j, LARS decreases the absolute correlations identically, but bLARS decreases the absolute correlations with the same rate but not identically. However, bLARS still guarantees that if $\gamma_j = 1/h_0$, then $[\mathbf{c}_1]_j(\gamma_j) = 0 \ \forall j \in \mathbb{I}_0$, which means that the least-squares problem is minimized with respect to the chosen columns in $\mathbb{I}_0$. Furthermore, bLARS still guarantees that as we increase γ_j from 0 to $1/h_0$ the absolute correlations in $\mathbb{I}_0$ are decreased (see (6)), but the absolute correlations in $\mathbb{I}_0^c$ might increase or decrease. bLARS goal is to find b columns in $\mathbb{I}_0^c$ for which their absolute correlations become greater than or equal to the minimum absolute correlation of columns in $\mathbb{I}_0$ as we increase γ_0. To find such a column we need to find γ_j for each $j \in \mathbb{I}_0^c$ such that

$$c_0(1 - \gamma_j h_0) = |[\mathbf{c}_0]_j - \gamma_j [\mathbf{a}_0]_j|. \tag{7}$$

Using the definition of c_0, such γ_j will guarantee that column $j \in \mathbb{I}_0^c$ has the same absolute correlation as the column with index $i \in \mathbb{I}_0$ that satisfies $i = \arg\max^b |\mathbf{c}_0|$. Equation (7) has two solutions; we keep the minimum positive solution

$$\gamma_j := \min^+ \left(\frac{c_0 - [\mathbf{c}_0]_j}{c_0 h_0 - [\mathbf{a}_0]_j}, \frac{c_0 + [\mathbf{c}_0]_j}{c_0 h_0 + [\mathbf{a}_0]_j} \right).$$

Out of all γ_j where $j \in \mathbb{I}_0^c$ we choose the one with the minimum bth value $\gamma_0 := \min_{j \in \mathbb{I}^c}^b \gamma_j$. Note that the bth minimum step-size γ_0 corresponds to the column(s) in $\mathbb{I}_0^c$ that will be the bth to have the same absolute correlation with the column in $\mathbb{I}_0$ with the minimum absolute correlation. Then bLARS updates $\mathbb{I}_1 := \mathbb{I}_0 \cup \{b$ columns with $\gamma_j \geq \gamma_0\}$. Note that bLARS decreases $\|\mathbf{c}_k\|_{\infty,k\cdot b} :=$ sum of $k \cdot b$ largest absolute components, compared to LARS which decreases sum of k largest absolute components. It is easy to see that by setting $b = 1$, then bLARS is equivalent to LARS.

The parallel bLARS algorithm is shown in Algorithm 2. This algorithm is presented in great detail since this demonstrates our implementation. We assume that the data matrix A and any vector/set of length/cardinality m are partitioned across processors, i.e., each processor holds m/P components, where P is the number of processors, and we assume for simplicity that m/P is an integer. More complicated two-dimensional partitions could be used [5, 30] and may potentially improve communication cost, but we use row partition for simplicity and leave more sophisticated partitioning methods for future work. The main computational kernels of the algorithm are matrix-matrix and matrix-vector products, which we can parallelize efficiently using Message Passing Protocol (MPI) collective routines for reduction [37]. We also make use of collective routines for broadcasting data [37]. In our numerical experiments in section 10, we use parallel bLARS with $b = 1$ as parallel LARS.

7.1. Asymptotic costs for parallel bLARS and LARS. In what follows we examine the asymptotic costs of each step of parallel bLARS in Algorithm 2. The asymptotic costs of parallel LARS are obtained by setting $b = 1$. We also comment when a step is executed only by the master processor, by all processors independently, or in parallel with synchronization. We model the running time of an algorithm by considering both arithmetic and communication costs. In particular, we model the running time of an algorithm as a sum of three terms as

$$\gamma F + \alpha L + \beta W,$$

where γ, α, and β are hardware parameters for time per arithmetic operation, time per message sent, and time per word moved, respectively. F, L, and W are algorithm parameters for the number of arithmetic operations to be executed, number of messages to be sent, and number of words to be moved, respectively. We choose the α-β model to measure communication of algorithms for simplicity. More refined models exist like the LogP [12] and LogGP [1] models.

We assume that matrix A is a dense matrix. Step 1 requires $O(m/P)$ operations for initialization of $\mathbf{y}_0$ and $\mathbf{r}_0$ in parallel with no communication. Step 2 requires computing $\mathbf{c}_k$ which is equal to $A^T\mathbf{r}_k$. This operation can be performed in parallel with synchronization in $O(mn/P)$ operations, $n \log P$ words, and $\log P$ messages, using a binary tree reduction algorithm in [37]. The result of step 2 is reduced to the master processor. Step 3 is performed by the master processor and it costs $O(n)$ operations using Introspective Selection [26]. Step 4 is performed in parallel with synchronization, which requires $O(b^2m/P)$ operations, $b^2 \log P$ words, and $\log P$ messages using binary

Algorithm 2. Parallel bLARS for row-partitioned data.

1: Initialize $b \in \mathbb{Z}_+$, $t \leq \min(m,n) \in \mathbb{Z}_+$, $k := 0$, $\mathbf{y}_k := \mathbf{0}_m$, $\mathbf{r}_k := \mathbf{b}$ in parallel without synchronization.
2: Compute $\mathbf{c}_k := A^T\mathbf{r}_k$ in parallel using reduction.
3: $c_k := \max^b |\mathbf{c}_k|$, $\mathbb{I}_k := \{i \in [n] \mid |[\mathbf{c}_k]_i| \geq c_k\}$.
4: Compute $G_k := A_{\mathbb{I}_k}^T A_{\mathbb{I}_k}$ in parallel using a reduction.
5: Compute L_k, the Cholesky factor of G_k.
6: **while** $|\mathbb{I}_k| < t$ **do**
7: $\quad \mathbf{s}_k := [\mathbf{c}_k]_{\mathbb{I}_k}$, $\mathbf{q}_k := (L_k L_k^T)^{-1}\mathbf{s}_k$
8: $\quad h_k := (\mathbf{s}_k^T\mathbf{q}_k)^{-1/2}$, $\mathbf{w}_k := \mathbf{q}_k h_k$
9: $\quad$ The master processor broadcasts $\mathbf{w}_k$.
10: $\quad$ Compute $\mathbf{u}_k := A_{\mathbb{I}_k}\mathbf{w}_k$ in parallel, no communication is required.
11: $\quad$ Compute $\mathbf{a}_k := A^T\mathbf{u}_k$ in parallel using a reduction.
12: $\quad \gamma_j := \min^+\left(\frac{c_k - [\mathbf{c}_k]_j}{c_k h_k - [\mathbf{a}_k]_j}, \frac{c_k + [\mathbf{c}_k]_j}{c_k h_k + [\mathbf{a}_k]_j}\right)\ \forall j \in \mathbb{I}_k^c$
13: $\quad \gamma_k := \min_{j \in \mathbb{I}_k^c}^b \gamma_j$
14: $\quad \mathbb{B} := \arg\min_{j \in \mathbb{I}_k^c}^b \gamma_j$ (note this returns b indices)
15: $\quad \mathbb{I}_{k+1} := \mathbb{I}_k \cup \mathbb{B}$
16: $\quad$ The master processor broadcasts γ_k to all processors.
17: $\quad$ Compute $\mathbf{y}_{k+1} := \mathbf{y}_k + \mathbf{u}_k\gamma_k$ in parallel, no communication is required.
18: $\quad [\mathbf{c}_{k+1}]_j := [\mathbf{c}_k]_j(1 - \gamma_k h_k)\ \forall j \in \mathbb{I}_k$ and $[\mathbf{c}_{k+1}]_j = [\mathbf{c}_k]_j - \gamma_k[\mathbf{a}_k]_j\ \forall j \in \mathbb{I}_k^c$
19: $\quad c_{k+1} := c_k(1 - \gamma_k h_k)$
20: $\quad$ Compute $A_{\mathbb{I}_k}^T A_{\mathbb{B}}$ and $A_{\mathbb{B}}^T A_{\mathbb{B}}$ in parallel using a reduction.
21: $\quad H_{k+1} := L_k^{-1} A_{\mathbb{I}_k}^T A_{\mathbb{B}}$
22: $\quad$ Solve $\Omega_{k+1}^T\Omega_{k+1} = A_{\mathbb{B}}^T A_{\mathbb{B}} - H_{k+1}^T H_{k+1}$ subject to Ω_{k+1} being a lower triangular matrix.
23: $\quad L_{k+1} := \begin{bmatrix} L_k & \mathbf{0}_{k,b} \\ H_{k+1} & \Omega_{k+1} \end{bmatrix}$
24: $\quad k := k + 1$
25: **end while**
26: Return $\mathbb{I}_k$, $\mathbf{y}_k$

tree reduction. Step 5 is executed by the master processor, which costs $O(b^3)$ operations. Step 7 is executed by the master processor, which costs $O(|\mathbb{I}_k|)$ operations to compute $\mathbf{s}_k := [\mathbf{c}_k]_{\mathbb{I}_k}$. Since $|\mathbb{I}_k| = b(k+1)$, this requires $O(bk+b)$ operations. Moreover, step 7 requires an additional $O(b^2(k+1)^2)$ operations to compute $\mathbf{q}_k := (L_k L_k^T)^{-1}\mathbf{s}_k$, which is also executed by the master processor. Step 8 costs $O(bk+b)$ operations and it is executed by the master processor. In step 9, $\mathbf{w}_k$ has to be broadcast to each processor from the master processor, which costs $b(k+1)\log P$ words and $\log P$ messages using a broadcast algorithm from [37]. Step 10 is computed in parallel without synchronization in $O(b(k+1)m/P)$ operations, i.e., each processor multiplies its own part of the vector $A_{\mathbb{I}_k}$ with $\mathbf{w}_k$. Step 11 is executed in parallel with synchronization, which requires $O(mn/P)$ operations, $n\log P$ words, and $\log P$ messages using a reduction. The result of step 11 is reduced to the master processor. Step 12 is executed by the master processor, which requires $O(|\mathbb{I}_k^c|)$ operations and is upper bounded by $O(n)$ operations in worst-case since $|\mathbb{I}_k^c| \leq n$. Steps 13 and 14 are executed by the master processor and they require in worst-case $O(n)$ operations using Introspective Selection. Step 15 is executed by the master processor and costs $O(b)$ operations. In Step 16 the

TABLE 1

Running time costs for parallel bLARS in Big O notation. The running time costs of LARS can be obtained by setting $b = 1$. The first column shows the number of step(s) of the algorithm. The second, third, and fourth columns show the number of operations, the number of words, and the number of messages, respectively, that are required by bLARS to output a solution with t columns/features.

Step(s)	Arithmetic operations (F)	Words (W)	Messages (L)
1	$\frac{m}{P}$	-	-
2	$\frac{mn}{P}$	$n \log P$	$\log P$
3	n	-	-
4	$\frac{b^2 m}{P}$	$b^2 \log P$	$\log P$
5–8	$\frac{t^3}{b}$	-	-
9	-	$\frac{t^2}{b} \log P + t \log P$	$\frac{t}{b} \log P$
10	$\frac{t^2 m}{bP}$	-	-
11	$\frac{tmn}{bP}$	$\frac{tn}{b} \log P$	$\frac{t}{b} \log P$
12–14	$\frac{tn}{b}$	-	-
15	t	-	-
16	-	$\frac{t}{b} \log P$	$\frac{t}{b} \log P$
17–19	$\frac{tm}{bP} + \frac{tn}{b}$	-	-
20	$\frac{t^2 m}{P} + \frac{tbm}{P}$	$t^2 \log P + tb \log P$	$\frac{t}{b} \log P$
21–23	$t^3 + t^2 b$	-	-
Total (assuming $t \gg b$)	$\frac{tmn}{bP} + \frac{tn}{b} + \frac{t^2 m}{P} + t^3$	$\frac{tn}{b} \log P + t^2 \log P$	$\frac{t}{b} \log P$

step-size γ is broadcast to all processors from the master processor in $\log P$ words and $\log P$ messages. Step 17 is executed in parallel without synchronization and requires $O(m/P)$ operations. Steps 18 and 19 are executed by the master processor and require $O(n)$ operations. Step 20 is executed in parallel with synchronization and requires $O(b^2 km/P + b^2 m/P)$ operations, $O(b^2 k \log P + b^2 \log P)$ words, and $2 \log P$ messages. The result of Step 20 is reduced to the master processor. Step 21 is executed by the master processor and requires $O(b^3 k^2)$ operations since L_k is a lower triangular matrix. Step 22 is executed by the master processor and requires $O(b^3 k + b^2)$ operations. Step 23 is executed by the master processor and requires $O(b^2 k + b^2)$ operations. Notice that if we want to obtain t columns using LARS, then we need to run the algorithm for $t - 1$ iterations. Therefore, if we want to obtain t columns using bLARS, then we need to run the algorithm for $(t - 1)/b$ iterations. By using this and the above costs for each step we summarize in Table 1 the asymptotic costs of bLARS and LARS for obtaining a solution with t columns. Assuming that $t \gg b$, which means that we want to output many more columns than b, then we observe in Table 1 that by using bLARS we reduce by a factor of b all major computational and communication costs compared to LARS.

8. Tournament block least angle regression. In this section we will present tournament block LARS (T-bLARS), a variation of LARS where b columns are selected at each iteration using a generalized reduction on a binary tree. Like bLARS, T-bLARS requires a lot of nontrivial modifications in order to maintain some properties of the original algorithm which we discuss in detail below. In comparison to parallel LARS and bLARS, for T-bLARS we assume that the data matrix A *column-partitioned*, i.e., each processor holds n/P columns, where P is the number of processors and we assume that n/P is an integer. Furthermore, we assume that vectors of length m or n or sets with cardinality at most m or n can be stored locally.

Let us now describe one iteration of T-bLARS. Let us assume that at the lth

iteration we have response $\mathbf{y}_l$ and we have selected columns $\mathbb{I}_l$. Furthermore, let us assume that $P = 2$, i.e., two processors. Each processor gets n/P columns, which we denote with index sets $\mathbb{I}_{v_1}$ and $\mathbb{I}_{v_2}$. T-bLARS requires running a modified version of LARS (mLARS), which we discuss later, as a reduction on a binary tree. For a visual explanation, see Figure 1. The algorithm starts at the bottom of the tree by calling mLARS for each node in parallel. Nodes v_1 and v_2 return candidate columns with indices in the sets $\mathbb{B}_{v_1}$ and $\mathbb{B}_{v_2}$, respectively. Columns $\mathbb{B}_{v_1} \cup \mathbb{B}_{v_2}$ are sent to node v_3, which is the parent of v_1 and v_2. Finally, the node v_3 calls mLARS using columns in $\mathbb{I}_l \cup \mathbb{B}_{v_1} \cup \mathbb{B}_{v_2}$ which returns the new response $\mathbf{y}_{l+1}$ and index set $\mathbb{I}_{l+1}$. Then this process is repeated. Details are provided in Algorithm 3.

$$\mathbb{B}_{v_1} \leftarrow \mathrm{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{I}_{v_1}) \quad \mathbb{B}_{v_2} \leftarrow \mathrm{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{I}_{v_2})$$

$$\mathbb{I}_{l+1}, \mathbf{y}_{l+1} \leftarrow \mathrm{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{B}_{v_1} \cup \mathbb{B}_{v_2})$$

FIG. 1. *Binary tree for one iteration of T-bLARS. The nodes at the bottom of the tree communicate columns in* $\mathbb{B}_{v_1}$ *and* $\mathbb{B}_{v_2}$.

Modified LARS. We mentioned that each node calls a modified version of LARS Algorithm 4. Let us now comment on this algorithm and why LARS needs to be modified in order for T-bLARS to be a well-defined algorithm. The problem is caused due to the fact that each processor on any level of the binary tree runs mLARS independently of other processors and on data that might not overlap. This may result in violation of a basic rule of LARS, which is that *there is no column that has not been selected with larger absolute correlation than the current known maximum absolute correlation* c_k.

Similarly to LARS, mLARS chooses one column at each iteration. Each call to mLARS operates on the columns with indices in $\mathbb{I}_\nu \cup \mathbb{I}_l$, where ν is the index of the node in the binary tree and $\mathbb{I}_l$ is the set of indices of columns that have been selected at the lth iteration of T-bLARS. If $\mathbb{I}_l$ does not include any index with maximum absolute correlation among the indices in $\mathbb{I}_\nu \cup \mathbb{I}_l$, then (5) might not have a nonnegative solution. This affects the step-size calculation, which for LARS is computed by solving (5) with the constraint that $\gamma \geq 0$. To guarantee that a meaningful step-size is calculated at each iteration of mLARS, we propose using stepLARS in Procedure 1. Briefly, stepLARS detects violations to the above basic rule of LARS. If it detects a violation, it checks if (5) still has a nonnegative solution and sets γ_k appropriately. If it cannot resolve it ((5) does not have a nonnegative solution), then it sets $\gamma_k = 0$. By setting $\gamma_k = 0$ we guarantee that the response $\mathbf{y}_k$ is not updated in the current iteration. Setting γ_k to a positive value would be a "mistake" since as we show in step 14 of stepLARS Procedure 1, this would result in decreasing the current known maximum correlation c_k of mLARS but at the same time it *increases* the absolute correlation of columns that violate the LARS property. This makes violation of the LARS property even larger.

If $\gamma_k = 0$, then mLARS at step 18 adds the column with the largest absolute correlation that also violates the LARS property in the set of selected columns. This decision guarantees that a violation will not happen again during the execution of mLARS. This is because similarly to LARS, mLARS guarantees that once c_k is maximal, then it will remain like this for all iterations, which ensures that (5) always

has at least one nonnegative solution. More details are described in mLARS Algorithm 4 and Procedure 1.

Algorithm 3. T-bLARS.

1: Initialize $l := 0$, $\mathbf{y}_l := \mathbf{0}_m$, $t \in \mathbb{Z}_+$, $b \in \mathbb{Z}_+$, $L_l = 0$, where L_l is the Cholesky factor.
2: Initialize $\mathbb{I}_l = \emptyset$
3: **while** $|\mathbb{I}_l| < t$ **do**
4: **for** all levels of the tree from bottom to the root **do**
5: **if** at the bottom of the tree **then**
6: Let $\mathbb{I}_v$ be the columns of node v in the tree. For all nodes v in the current level of the binary tree call $\mathbb{B}_v \leftarrow \text{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{I}_v, L_l)$.
7: **else if** not at root of the binary tree **then**
8: Let $\mathbb{B}_v$ be the columns selected by child nodes of v. For all nodes v in the current level of the binary tree call $\tilde{\mathbb{B}}_v \leftarrow \text{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{B}_v, L_l)$, where $\tilde{\mathbb{B}}_v$ are the selected b columns out of $\mathbb{B}_v$.
9: Send columns $\tilde{\mathbb{B}}_v$ for each node v to the processor of the parent node of v.
10: **else**
11: $\mathbf{y}_{l+1}, \mathbb{I}_{l+1}, \mathbb{B}_{l+1}, L_{l+1} \leftarrow \text{mLARS}(b, \mathbf{y}_l, \mathbb{I}_l \cup \mathbb{B}_v, L_l)$
12: Broadcast selected columns with index in $\mathbb{B}_{l+1}$, $\mathbf{y}_{l+1}$, and L_{l+1} to all processors. Note that we only communicate the part of L_{l+1} that gets updated by the root node.
13: **end if**
14: **end for**
15: $l := l + 1$
16: **end while**
17: Return $\mathbb{I}_l$, $\mathbf{y}_l$

8.1. Asymptotic costs for parallel implementation of T-bLARS. In this subsection we examine the asymptotic costs for T-bLARS Algorithm 3. We start with the asymptotic costs of mLARS Algorithm 4, which is used by T-bLARS at every iteration.

Before we compute the asymptotic costs for mLARS we have to bound the cardinality of some sets. The cardinality $\mathbb{I}_v$ is bounded by $|\mathbb{I}_v| \leq n/P$. Let l be the lth iteration of T-bLARS, and let $\mathbb{I}_l$ be the current selected columns of T-bLARS. Then $|\mathbb{I}_l| \leq lb$. Assuming that we are on the kth iteration of mLARS, then $|\mathbb{I}_k| \leq |\mathbb{I}_l| + b \leq lb + b$, and $|\mathbb{I}_k \cup \tilde{\mathbb{I}}_v| \leq lb + b + n/P$ for all k if node v is at the bottom of the tree, i.e., $\tilde{\mathbb{I}}_v := \mathbb{I}_v$, otherwise $|\mathbb{I}_k \cup \tilde{\mathbb{I}}_v| \leq lb + b + 2b$ for all k because node v not at the bottom of the tree, i.e., $\tilde{\mathbb{I}}_v := \mathbb{B}_v$. The cardinality of $\tilde{\mathbb{I}}_v \backslash \mathbb{I}_k$ is bounded by n/P if v is a leaf node because $|\tilde{\mathbb{I}}_v \backslash \mathbb{I}_k| \leq |\tilde{\mathbb{I}}_v| = |\mathbb{I}_v| \leq n/P$, or otherwise bounded by $2b$ because $|\tilde{\mathbb{I}}_v| = |\mathbb{B}_v| \leq 2b$. Using these bounds we will compute the asymptotic costs of each step of mLARS. Note that there is no parallelism for each individual run of mLARS. Therefore, we only report results for arithmetic operations.

Step 3 costs $O(m)$ operations. Step 4 costs $O(mn/P + mlb + mb)$ at leaf node and $O(mlb + 3mb)$ otherwise. Step 5 costs $O(lb + b)$. Step 7 costs $O(n/P + lb + b + m)$ at leaf node and $O(lb + 3b + m)$ otherwise. Step 10 costs $O(lb^2 + b^2)$. Step 11 costs $O(b(lb + b)^2)$. Steps 12–13 cost $O(lb^2 + b^2)$. Step 14 costs $O(mlb^2 + mb^2)$. Step 15 costs $O(bmn/P + mlb^2 + mb^2)$ at leaf node and $O(mlb^2 + 3mb^2)$ otherwise. Steps 16–18 cost $O(bn/P)$ at leaf node and $O(2b^2)$ otherwise. Step 19 costs $O(m)$. Steps 20–21 cost $O(bn/P + lb^2 + b^2)$ at leaf node and $O(lb^2 + 3b^2)$ otherwise. Step 22 costs $O(lb^2 + b^2)$.

Procedure 1. Step-size for modified LARS (stepLARS).

Input: c_k, h_k, $\mathbf{c}_k$, $\mathbf{a}_k$ and an index j
if $c_k \geq |[\mathbf{c}_k]_j|$ **then**
  if $[\mathbf{c}_k]_j$ and $[\mathbf{a}_k]_j$ have the same sign **then**
    Equation $c_k(1-\gamma h_k) = |[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ has at least one positive solution, we select the minimum positive one $\gamma := \min^+\left(\frac{c_k-[\mathbf{c}_k]_j}{c_k h_k-[\mathbf{a}_k]_j}, \frac{c_k+[\mathbf{c}_k]_j}{c_k h_k+[\mathbf{a}_k]_j}\right)$.
  else
    Equation $c_k(1-\gamma h_k) = |[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ has one positive solution that is $\gamma := \frac{c_k-|[\mathbf{c}_k]_j|}{c_k h_k+|[\mathbf{a}_k]_j|}$.
  end if

  if $[\mathbf{c}_k]_j$ and $[\mathbf{a}_k]_j$ have the same sign and $[\mathbf{c}_k]_j h_k \leq [\mathbf{a}_k]_j$ **then**
    Equation $c_k(1-\gamma h_k) = |[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ has one positive solution that is $\gamma := \frac{c_k-|[\mathbf{c}_k]_j|}{c_k h_k-|[\mathbf{a}_k]_j|}$.
  else if $[\mathbf{c}_k]_j$ and $[\mathbf{a}_k]_j$ have the same sign and $[\mathbf{c}_k]_j h_k > [\mathbf{a}_k]_j$ **then**
    Equation $c_k(1-\gamma h_k) = |[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ does not have a positive solution. But as γ increases $c_k(1-\gamma h_k)$ and $|[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ decrease; therefore, we set γ to its maximum value $\gamma := 1/h_k$.
  else
    Equation $c_k(1-\gamma h_k) = |[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ does not have a positive solution. In this case, as γ increases $|[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j|$ increases and $c_k(1-\gamma h_k)$ decreases. Therefore, we set $\gamma := 0$, which subject to $\gamma \geq 0$ minimizes the error $|[\mathbf{c}_k]_j - \gamma[\mathbf{a}_k]_j| - c_k(1-\gamma h_k)$.
  end if
end if
Return γ

Step 23 costs $O(mlb^2 + mb^2)$. Step 24 costs $O(b(lb+b)^2)$. Steps 25–26 cost $O(lb^2+b^2)$. For t columns we need to run T-bLARS for t/b iterations and each iteration makes $\log P$ parallel calls to mLARS which results in

$$t/b \cdot (\text{arithmetic cost of mLARS at leaf node})$$
$$+\, t/b \cdot (\text{arithmetic cost of mLARS at nonleaf node}) \cdot \log P$$

total operations. Therefore, in Big O notation T-bLARS requires

$$F = O\left(\frac{tmn}{P} + \frac{tmn}{bP} + \left(t^2 m + t^3\right)\log P\right)$$

operations. Communication occurs $\log P$ times because of the binary tree and another $\log P$ times to broadcast data from the root node to the rest of the nodes. Therefore, T-bLARS requires

$$L = 2\frac{t}{b}\log P$$

messages. Each node (except of the root) communicates bm words for columns in $\mathbb{B}$. Therefore, the execution of the binary tree requires $tm \log P$ words. Broadcasting data from the root node to the rest of the nodes at step 12 costs a total of

$$W = O\left(\left(tm + \frac{tm+t^2}{b} + tb\right)\log P\right)$$

Algorithm 4. Modified least angle regression (mLARS).

Input: number of columns $b \in \mathbb{Z}_+$, response $\tilde{\mathbf{y}}$, column index sets $\tilde{\mathbb{I}}_0 \cup \tilde{\mathbb{I}}_v$ (third input), and Cholesky factor $\tilde{L}$ (forth input)
Initialize: $k := 0$, $\mathbb{B} := \emptyset$, $L_k := \tilde{L}$, $\mathbb{I}_k := \tilde{\mathbb{I}}_0$
$\mathbf{r}_k := \mathbf{b} - \tilde{\mathbf{y}}$
$\mathbf{c}_k := A^T_{\mathbb{I}_k \cup \tilde{\mathbb{I}}_v} \mathbf{r}_k$
$c_k := \max |[\mathbf{c}_k]_{\mathbb{I}_k}|$. Note that we abuse notation here for $[\mathbf{c}_k]_{\mathbb{I}_k}$. Since $\mathbf{c}_k \in \mathbb{R}^{|\mathbb{I}_k \cup \tilde{\mathbb{I}}_v|}$ and by usual convention, its components are indexed from 1 to $|\mathbb{I}_k \cup \tilde{\mathbb{I}}_v|$, which might not overlap with the indices in $\mathbb{I}_k$. We assume that the components of $\mathbf{c}_k$ are indexed using the indices in $\mathbb{I}_k \cup \tilde{\mathbb{I}}_v$. We use this abuse of notation at other steps of this algorithm because it simplifies notation.
if $\mathbb{I}_k = \emptyset$ **then**
  $c_k := \max |[\mathbf{c}_k]|$, $\mathbb{I}_k := \{\arg\max |\mathbf{c}_k|\}$, $L_k = (A^T_{\mathbb{I}_k} A_{\mathbb{I}_k})^{1/2}$.
end if
while $|\mathbb{I}_k| < |\tilde{\mathbb{I}}_0| + b$ **do**
  $\mathbf{s}_k := [\mathbf{c}_k]_{\mathbb{I}_k}$
  $\mathbf{q}_k := (L_k L_k^T)^{-1} \mathbf{s}_k$
  $h_k := (\mathbf{s}_k^T \mathbf{q}_k)^{-1/2}$
  $\mathbf{w}_k := \mathbf{q}_k h_k$
  $\mathbf{u}_k := A_{\mathbb{I}_k} \mathbf{w}_k$
  $\mathbf{a}_k := A^T_{\mathbb{I}_k \cup \tilde{\mathbb{I}}_v} \mathbf{u}_k$
  $\gamma_j \leftarrow \text{stepLARS}(c_k, h_k, \mathbf{c}_k, \mathbf{a}_k, j)$ $\forall j \in \tilde{\mathbb{I}}_v \backslash \mathbb{I}_k$
  If there are γ_j that are equal to zero, set γ_k to zero. Otherwise, set γ_k to the minimum nonzero γ_j.
  If there are γ_j that are equal to zero, set i to the jth column with the largest $|[\mathbf{c}_k]_j|$. Otherwise, set i to the jth column with the minimum nonzero γ_j.
  $\mathbf{y}_{k+1} := \mathbf{y}_k + \mathbf{u}_k \gamma_k$
  $[\mathbf{c}_{k+1}]_j := [\mathbf{c}_k]_j (1 - \gamma h_k)$ $\forall j \in \mathbb{I}_k$, and $[\mathbf{c}_{k+1}]_j = [\mathbf{c}_k]_j - \gamma [\mathbf{a}_k]_j$ $\forall j \in \tilde{\mathbb{I}}_v \backslash \mathbb{I}_k$
  $\mathbb{I}_{k+1} := \mathbb{I}_k \cup \{i\}$, $\mathbb{B} := \mathbb{B} \cup \{i\}$
  $c_{k+1} := \max |[\mathbf{c}_{k+1}]_{\mathbb{I}_{k+1}}|$
  Compute $A^T_{\mathbb{I}_k} A_i$ and $A_i^T A_i$.
  $\mathbf{l}_{k+1} := L_k^{-1} A^T_{\mathbb{I}_k} A_i$
  $\omega_{k+1} := (A_i^T A_i - \mathbf{l}_{k+1}^T \mathbf{l}_{k+1})^{1/2}$
  $L_{k+1} := \begin{bmatrix} L_k & \mathbf{0}_k \\ \mathbf{l}_{k+1} & \omega_{k+1} \end{bmatrix}$
  $k := k + 1$
end while
Return $\mathbf{y}_k$, $\mathbb{I}_k$, $\mathbb{B}$, L_k

words.

9. Comparison of asymptotic costs. In this section, we compare the asymptotic costs of parallel LARS, bLARS, and T-bLARS. The results are shown in Table 2. Note that parallel bLARS becomes faster than parallel LARS for $b > 1$. Parallel bLARS and T-bLARS have similar latency costs. However, an important difference is that the number of words for parallel bLARS depends on the number of columns n while the number of words for T-bLARS depends on the number of rows m. This is due

TABLE 2
Asymptotic costs for parallel LARS, bLARS, and T-bLARS. Here, t is the required number of columns to be output by all algorithms. We assume that $t \gg b$ and that matrix A is dense.

Method	Arithmetic operations	Words communicated	Messages
LARS	$\frac{tmn}{P} + \frac{t^2m}{P} + tn + t^3$	$tn \log P + t^2 \log P$	$t \log P$
bLARS	$\frac{tmn}{bP} + \frac{tn}{b} + \frac{t^2m}{P} + t^3$	$\frac{tn}{b} \log P + t^2 \log P$	$\frac{t}{b} \log P$
T-bLARS	$\frac{tmn}{P} + \frac{tmn}{bP} + (t^2m + t^3) \log P$	$(tm + \frac{tm}{b} + tb) \log P + \frac{t^2}{b} \log P$	$\frac{t}{b} \log P$

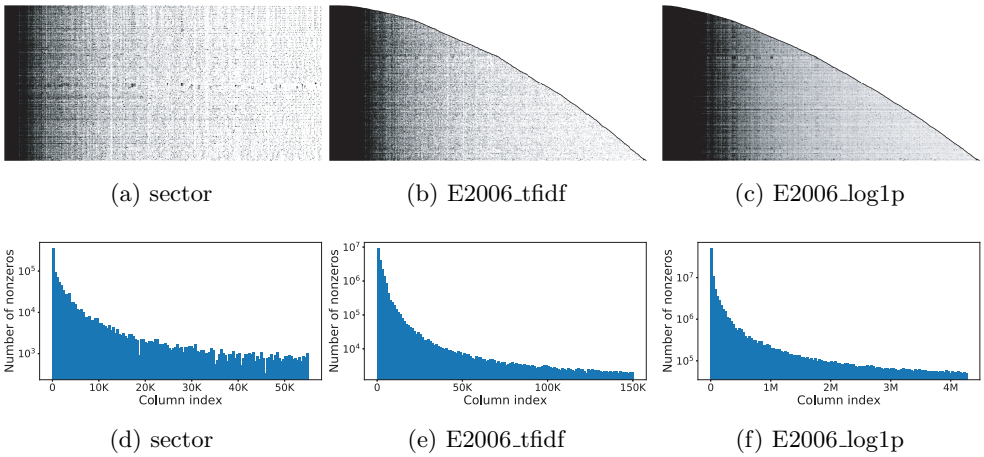
to the fact that for parallel bLARS we partition the data per row, while for T-bLARS we partition the data per column. Therefore, in the high-dimensional regression setting where $n \gg m$, T-bLARS requires communicating far fewer words than bLARS. We compare the two methods empirically in section 10.

We note that even though the results in Table 2 are obtained by assuming matrix A is dense, the complexity bounds trivially extend to sparse matrices as long as we have balanced partitions, i.e., the local sparse matrices stored at different processors should have similar number of nonzero entries. In the balanced sparse case, we simply replace mn with the number of nonzeros $\text{nnz}(A)$ and obtain the arithmetic complexity for all methods. The communication costs stay the same. In section 10 we use balanced partition to deal with sparse matrices.

10. Empirical performance. This section contains two parts. First, we evaluate and compare the solution quality of bLARS and T-bLARS for a range of block sizes b and processors P. Second, we present a comprehensive list of plots that demonstrate both overall speedups and more detailed running time breakdowns from increasing b and P. We carry out the experiments on four regression datasets summarized in Table 3. The data matrices for sector and E2006 are sparse and columnwise unbalanced, i.e., the distribution of nonzeros per column is skewed (Figure 2). In order to balance the computation workload on all processors, for T-bLARS we distribute the columns of these sparse matrices so that the partitioned columns at each processor have roughly the same number of nonzeros. Other column partitioning could also be used. We discuss the effect of column partition on solution quality of T-bLARS in the next subsection. For comparison purposes we limit both algorithms to collect the first 75 columns. We implemented the code in Python and used the optimized mpi4py library [13]. The code is run on a computer cluster with distributed memory. Each node in the cluster comes with 2 x Intel E5-2683 v4, 128 GB of RAM.

TABLE 3
Properties of the datasets that we consider. $\text{nnz}(A)$ *denotes the number of nonzeros in matrix A. Consequently, the fourth column gives the (relative) sparsity of A. These regression datasets can be downloaded from* [7] *as part of the LIBSVM Data package. The E*2006 *and Year datasets are the three largest regression datasets in LIBSVM.*

Dataset	m	n	$\text{nnz}(A)/mn$
sector	6412	55197	0.003
YearPredictionMSD	463715	90	1.00
E2006_log1p	16087	4272227	0.001
E2006_tfidf	16087	150360	0.008



(a) sector (b) E2006_tfidf (c) E2006_log1p

(d) sector (e) E2006_tfidf (f) E2006_log1p

FIG. 2. *Sparsity pattern and distribution of nonzeros for sparse datasets sector and E*2006. *The histograms* (d)–(f) *are drawn on* 128 *equally spaced bins.*

10.1. Solution quality. We use two metrics to measure solution quality. One metric is, for a given parameter b, the value of the ℓ_2-norm of the residual vector versus the number of columns added at each iteration (Figure 3). For the second metric, since LARS is primarily used for column selection in regression, we treat the columns selected by LARS as the ground truth, and we use precision in column selection to measure performance, i.e., we compare the percentage of columns selected by bLARS and T-bLARS that overlap with the columns selected by LARS (Figure 4).

Observe that T-bLARS is overall more successful in terms of both data fitting and column selection. The ℓ_2-norm of the residual produced by T-bLARS is nearly identical to that of LARS on all datasets and for all choices of b and P. On the other hand, bLARS has higher residuals as b increases. For column selection, we see a decrease in precision for both methods when $b > 1$, but in most settings T-bLARS recovers more columns than bLARS. In particular, the precision of bLARS keeps dropping quickly as b increases, while on three out of four datasets the precision of T-bLARS goes up again for larger b. This makes sense because for T-bLARS, the larger the block size is, the more columns will be sent from leaf nodes to nonleaf nodes to choose from.

For bLARS, how rows are partitioned among processors does not affect the columns selected by the algorithm. For T-bLARS, different column partitions can lead to different tournaments at nonleaf nodes and thus cause T-bLARS to select different columns at the root node. Figure 5 shows a range of precision results for T-bLARS over 10 random partitions of columns into $P = 128$ processors. We observe that T-bLARS still has a higher precision than bLARS in most cases. Determining the best column partitions that would yield the highest precision for T-bLARS in terms of column selection is interesting both in theory and in practice, but it is beyond the scope of this work.

10.2. Speedup. We show the speedup trends in Figure 6. Note that for $P = b = 1$, the speedup factor for T-bLARS is not identically 1.0 because T-bLARS performs more matrix-vector products than LARS in this parameter setting. For example, T-bLARS recomputes $\mathbf{c}_k$ repeatedly due to iterative call to mLARS (step 4), while in LARS, the vector $\mathbf{c}_k$ is computed only once and updated iteratively.

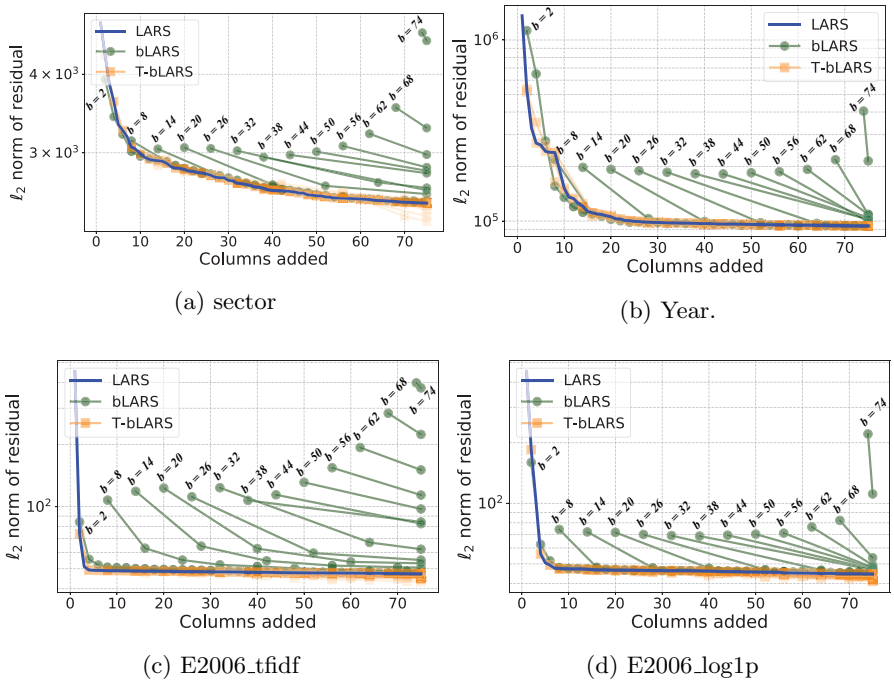
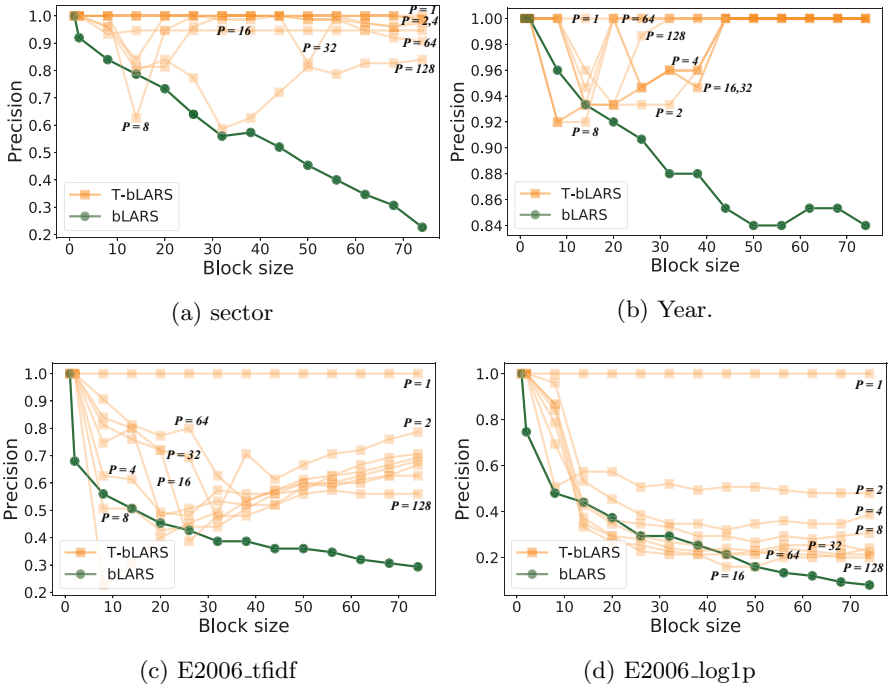


FIG. 3. *ℓ_2-norm of residuals. For T-bLARS each line corresponds to a setting of P and b. We do not show all legends for T-bLARS to ease readability; most settings give similar quality. For bLARS each line corresponds to a different b. Note that P does not affect the quality of bLARS. Seventy-five columns were chosen for all experiments.*

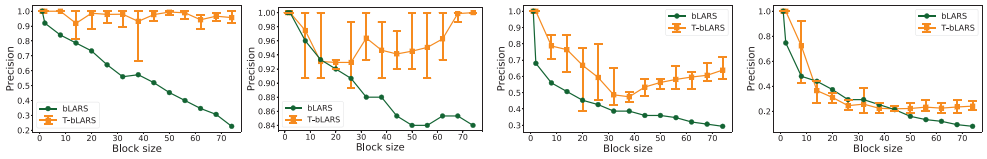
Overall, bLARS enjoys much higher speedups across all datasets. When the data is not very high-dimensional, i.e., not in the regime $n \gg m$, the total running time of bLARS scales with both P and b as predicted by the asymptotic costs analysis. The largest dataset E2006_log1p has far more columns than rows, and bLARS slows down when we increase the number of processors beyond 4. On the other hand, apart from E2006_log1p, T-bLARS does not seem to have a good speedup on other datasets. In order to understand what causes the speedups or the slow-downs, in Figure 7 (resp., Figure 8) we fix b (resp., P) and vary P (resp., b) and show how the major components of the total running time scales. For arithmetic operations, we plot the time spent on performing matrix-matrix and matrix-vector products and the time spent on computing the step-size γ separately, as both the cost analysis (cf. Table 1) and subsequent plots show that these are the computation bottlenecks. There is only a very small fraction of the total time spent on other computations, e.g., scalar multiplications, array initializations, and Cholesky factorization and inversion of small-size matrices, so we do not plot all of them explicitly. Note that the binary tree reduction in T-bLARS has $\log P$ serial levels: for a column to become a winner at the root, it has to go through $\log P$ number of competitions sequentially. Therefore, once the candidate columns are selected at leaf nodes and competitions start at nonleaf nodes, there will always be some nodes waiting for the root to broadcast the final winners before starting the next iteration. For T-bLARS we include this wait time in the running time breakdown plots. We estimated the wait time using the average computation time per competition at nonleaf nodes times the number of levels in the



(a) sector (b) Year.

(c) E2006_tfidf (d) E2006_log1p

FIG. 4. *Precision in column selection. For both bLARS and T-bLARS each line corresponds to a setting of P. Note that different P's give rise to different row partitions for bLARS and different column partitions for T-bLARS. Row partitions do not affect the precision of bLARS.*

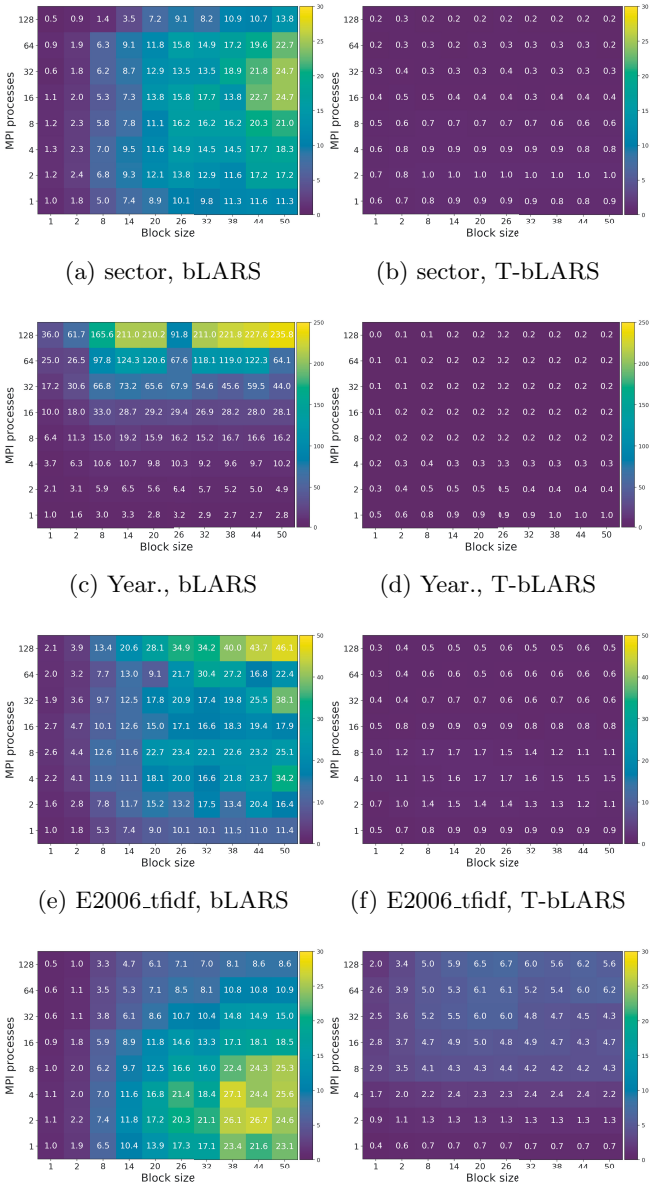


(a) sector (b) Year. (c) E2006_tfidf (d) E2006_log1p

FIG. 5. *Effects of column partitions on the precision of column selection for T-bLARS. We fix $P = 128$ and run T-bLARS on 10 random column partitions. The bars for each b show the minimum and maximum precisions over the 10 runs, and the line in the middle connects the mean.*

tree.

We make some comments about Figures 7 and 8. First, both bLARS and T-bLARS reduce the time spent on matrix-vector products as we increase either P or b. The speedup of bLARS mainly comes from the speedup of matrix-vector products. Second, bLARS spent smaller fraction of total time on communication when the data matrix is a tall $m \gg n$, e.g., YearPredictionMSD; T-bLARS spent a smaller fraction of total time on communication when the data matrix is fat $n \gg m$. This is expected because the number of words communicated for bLARS increases with n and is independent of m, while the number of words communicated for T-bLARS increases with m and is independent of n. Third, we didn't see a good speedup of T-bLARS for sector, YearPredictionMSD, and E2006_tfidf, because T-bLARS spent a large fraction of time on serial reduction in the binary tree, which overweighs the reduction in time for matrix-vector products. On the other hand, the wait time for serial tournaments for



(a) sector, bLARS (b) sector, T-bLARS

(c) Year., bLARS (d) Year., T-bLARS

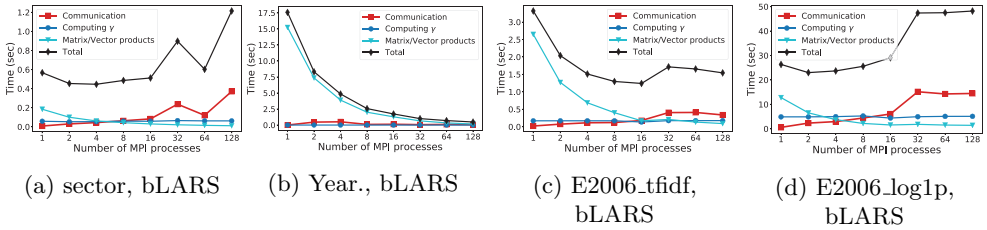
(e) E2006_tfidf, bLARS (f) E2006_tfidf, T-bLARS

(g) E2006_log1p, bLARS (h) E2006_log1p, T-bLARS

FIG. 6. *Total speedup.*

E2006_log1p took relatively much less time, so T-bLARS obtains good speedups. In general, one can expect T-bLARS to have a good speedup when the "wait time" is much less than parallel computation times (e.g., matrix-vector products) at leaf nodes. Our implementation of T-bLARS uses sparse data structures for computations at leaf nodes (to reduce memory requirement) and dense data structures for computations at nonleaf nodes (to reduce overheads). This has put T-bLARS in a slight disadvantage when dealing with sparse data as many arithmetic operations at nonleaf nodes will be unnecessary. We thus expect T-bLARS to achieve better speedups (than the 6x on

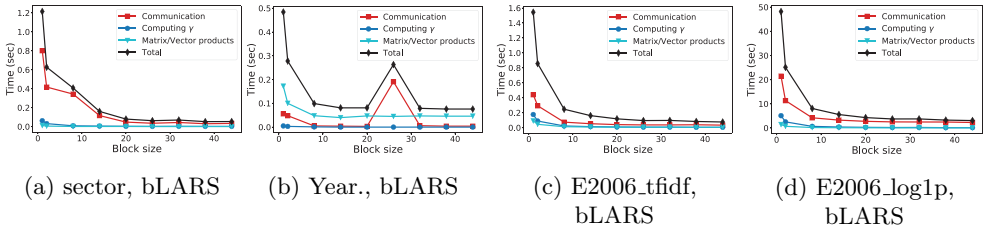
E2006_log1p) on dense and high-dimensional data where $n \gg m$. Finally, Figure 8 shows that the communication cost of both bLARS and T-bLARS tends to decrease as b increases, which is also expected according to Table 2.



(a) sector, bLARS (b) Year., bLARS (c) E2006_tfidf, bLARS (d) E2006_log1p, bLARS

(e) sector, T-bLARS (f) Year., T-bLARS (g) E2006_tfidf, T-bLARS (h) E2006_log1p, T-bLARS

FIG. 7. *Running time breakdown. We fix $b = 1$ and vary P. The pattern is similar for other b.*



(a) sector, bLARS (b) Year., bLARS (c) E2006_tfidf, bLARS (d) E2006_log1p, bLARS

(e) sector, T-bLARS (f) Year., T-bLARS (g) E2006_tfidf, T-bLARS (h) E2006_log1p, T-bLARS

FIG. 8. *Running time breakdown. We fix $P = 128$ and vary b. The pattern is similar for other P.*

Our experiments indicate that there is a tradeoff between bLARS and T-bLARS. On one hand, bLARS is well suited for row-partitioned data and can achieve speedups up to two orders of magnitude. However, the amazing speedup of bLARS comes at the expense of solution quality. One the other hand, while T-bLARS is generally slower than bLARS, it has lower residual norms and on average selects columns more accurately than bLARS. For example, for E2006_log1p, T-bLARS achieves 4x speedup ($P = 64$, $b = 2$) while correctly selecting 100% columns, bLARS only obtains 2x speedup for $b = 2$ and has a precision below 80%. Even though bLARS has up to 27x speedup ($P = 4$, $b = 38$) for E2006_log1p, in this setting bLARS only correctly

recovers around 30% columns that would have been selected by LARS.

11. Conclusions. The two parallel and communication avoiding methods we have introduced, bLARS and T-bLARS, present valuable methods of LARS that provide higher performance of speed than LARS can normally give. The choice between the two comes down to what priorities and expectations the user has from the solutions generated from these algorithms, e.g., be it higher speed or more resilient accuracy.

REFERENCES

[1] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, *LogGP: Incorporating long messages into the logP model for parallel computation*, J. Parallel Distrib. Comput., 44 (1997), pp. 71–79.

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

[3] 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.

[4] E. J. Candés, J. Romberg, and T. Tao, *Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information*, IEEE Trans. Inf. Theory, 52 (2006), pp. 489–509.

[5] L. Cannon, *A Cellular Computer to Implement the Kalman Filter Algorithm*, Ph.D. thesis, Montana State University, Bozeman, MT, 1969.

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

[7] C.-C. Chang and C.-J. Lin, *LIBSVM: A library for support vector machines*, ACM Trans. Intell. Syst. Technol., 2 (2011), 27,; software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[8] A. T. Chronopoulos, *A Class of Parallel Iterative Methods Implemented on Multiprocessors*, Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

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

[10] A. T. Chronopoulos and C. W. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

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

[12] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, T. Subramonian, and R. von Eicken, *LogP: Towards a realistic model of parallel computation*, in Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 1993, pp. 1–12.

[13] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, *Parallel distributed computing using Python*, Adv. Water Res., 34 (2011), pp. 1124–1139, https://doi.org/10.1016/j.advwatres.2011.04.013.

[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/10.1137/090760969.

[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, CA, 2007.

[16] A. Devarakonda, K. Fountoulakis, J. Demmel, and M. Mahoney, *Avoiding synchronization in first-order methods for sparse convex optimization*, in Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium, 2018, pp. 409–418.

[17] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, *Least angle regression*, Ann. Statist., 32 (2004), pp. 407–499.

[18] O. Fercoq and P. Richtárik, *Accelerated, parallel, and proximal coordinate descent*, SIAM J. Optim., 25 (2015), pp. 1997–2023, https://doi.org/10.1137/130949993.

[19] T. Hastie, J. Taylor, R. Tibshirani, and G. Walther, *Forward stagewise regression and the monotone lasso*, Electron. J. Statist., 1 (2007), pp. 1–29.

[20] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag, New York, 2001.
[21] M. Hoemmen, *Communication-avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, Berkeley, CA, 2010.
[22] 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, NIPS'14, MIT Press, Cambridge, MA, 2014, pp. 3068–3076.
[23] S. K. Kim and A. T. Chronopoulos, *An efficient nonsymmetric Lanczos method on parallel vector computers*, J. Comput. Appl. Math., 42 (1992), pp. 357–374.
[24] M. Mohiyuddin, *Tuning Hardware and Software for Multiprocessors*, Ph.D. thesis, EECS Department, University of California, Berkeley, CA, 2012.
[25] 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, ACM, New York, 2009, 36.
[26] D. R. Musser, *Introspective sorting and selection algorithms*, Softw. Pract. Exper., 27 (1997), pp. 983–993.
[27] D. Needell and T. Woolf, *An asynchronous parallel approach to sparse recovery*, in Proceedings of the Information Theory and Applications Workshop (ITA), IEEE, 2017, pp. 1–5.
[28] Yu. 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] A. Y. Ng, *Feature selection, L1 vs. L2 regularization, and rotational invariance*, in Proceedings of the Twenty-first International Conference on Machine Learning, 2004, 78.
[30] N. Park, B. Hong, and V. K. Prasanna, *Tiling, block data layout, and memory hierarchy performance*, IEEE Trans. Parallel Distrib. Syst., 14 (2003), pp. 640–654.
[31] D. A. Patterson and J. L. Hennessy, *Organization and Design: The Hardware/Software Interface*, Morgan Kaufman, Burlington, MA, 2013.
[32] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, New York, 2004.
[33] B. Recht, C. Ré, S. Wright, and F. Niu, *HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent*, in Proceedings of the 24th International Conference on Neural Information Processing Systems, Curran Associates, 2011, pp. 693–701.
[34] 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.
[35] J. Shalf, S. Dosanjh, and J. Morrison, *Exascale computing technology challenges*, in Proceedings of the International Conference on High Performance Computing for Computational Science - VECPAR 2010, Lecture Notes in Comput. Sci. 6449, Springer, Berlin, Heidelberg, 2010, pp. 1–25.
[36] E. Solomonik, *Provably Efficient Algorithms for Numerical Tensor Algebra*, Ph.D. thesis, EECS Department, University of California, Berkeley, CA, 2014.
[37] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in MPICH*, Int. J. High Perform. Comput. Appl., 19 (2005), pp. 49–66.
[38] R. Tibshirani, *Regression shrinkage and selection via the lasso*, J. Roy. Statist. Soc. Ser. B, 58 (1996), pp. 267–288.
[39] J. Van Rosendale, *Minimizing inner product data dependencies in conjugate gradient iteration*, in Proceedings of the International Conference on Parallel Processing, ICPP'83, IEEE, Los Alamitos, CA, 1983, pp. 44–46.
[40] S. Weisberg, *Applied Linear Regression*, John Wiley & Sons, New York, 1980.
[41] 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 28th International Symposium on Parallel and Distributed Processing, IEEE, 2014, pp. 1149–1158.
[42] S. J. Wright, *Coordinate descent algorithms*, Math. Program., 151 (2015), pp. 3–34.