Parallel Lasso Screening for Big Data Optimization

Qingyang Li
Arizona State Univ.
Tempe, AZ 85287
qingyang.li@asu.edu

Shuang Qiu
Univ. of Michigan
Ann Arbor, MI 48109
qiush@umich.edu

Shuiwang Ji
Washington State Univ.
Pullman, WA 99164
sji@eecs.wsu.edu

Paul M. Thompson
Univ. of Southern California
Los Angeles, CA 90089
pthomp@usc.edu

Jieping Ye
Univ. of Michigan
Ann Arbor, MI 48109
jpye@umich.edu

Jie Wang
Univ. of Michigan
Ann Arbor, MI 48109
jwangumi@umich.edu

ABSTRACT

Lasso regression is a widely used technique in data mining for model selection and feature extraction. In many applications, it remains challenging to apply the regression model to large-scale problems that have massive data samples with high-dimensional features. One popular and promising strategy is to solve the Lasso problem in parallel. Parallel solvers run multiple cores in parallel on a shared memory system to speedup the computation, while the practical usage is limited by the huge dimension in the feature space. Screening is a promising method to solve the problem of high dimensionality by discarding the inactive features and removing them from optimization. However, when integrating screening methods with parallel solvers, most of solvers cannot guarantee the convergence on the reduced feature matrix. In this paper, we propose a novel parallel framework by parallelizing screening methods and integrating it with our proposed parallel solver. We propose two parallel screening algorithms: Parallel Strong Rule (PSR) and Parallel Dual Polytope Projection (PDPP). For the parallel solver, we proposed an Asynchronous Grouped Coordinate Descent method (AGCD) to optimize the regression problem in parallel on the reduced feature matrix. AGCD is based on a grouped selection strategy to select the coordinate that has the maximum descent for the objective function in a group of candidates. Empirical studies on the real-world datasets demonstrate that the proposed parallel framework has a superior performance compared to the state-of-the-art parallel solvers.

CCS Concepts

•**Information systems** → **Data mining;**

Keywords

Lasso regression, parallel computing, screening rules, coordinate descent, aynchronized coordinate descent

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939859

1. INTRODUCTION

Sparse models with ℓ_1-regularization are widely used to find the linear model of best fit. Many research efforts have been devoted to develop efficient solvers for the ℓ_1-regularized sparse models, such as the Lasso problem [18]. Recent technological innovations lead to huge data collections that keep growing rapidly. As a result, in many applications, running Lasso on huge-scale data sets usually exceeds the computing capacity of a single machine running single-threaded approaches. Parallelizing the learning process for the regression problem has recently drawn a lot of interest. Most of the proposed algorithms are based on Stochastic Coordinate Descent (SCD) [17] to accelerate the whole learning process. Many existing approaches, like Shotgun [6], Parallel Block Coordinate Descent (PBCD) [13] and Thread-Greedy [16, 15], employ multiple-threaded computing by utilizing multiple cores on a shared memory system. However, the curse-of-dimensionality is still a great challenge for large-scale problems. High-dimensional data in feature space means more time spent in the optimization and data synchronization in the multithreading environment.

To address this issue, *screening* is one of highly efficient approaches to solve the high-dimensional problem. Screening pre-identifies inactive features that have zero components in the solution and remove them from the optimization. As a result, we can solve the regression problem on the reduced feature matrix, leading to substantial savings in terms of computation and memory usage. The idea of screening has achieved great success in a large class of ℓ_1-regularized problems [3, 19, 24, 25, 22, 21, 23], such as Lasso regression, Logistic regression, elastic net, multi-task feature learning (MTFL) and more general convex problems.

Parallel screening is a promising strategy to solve the high-dimensional problem in big data optimization. However, the idea of using screening in a multithreading environment has not been investigated, since it is challenging to integrate screening rules with parallel solvers. Most of the published parallel solvers are based on parallelizing SCD to speedup the optimization process. The updating strategy of SCD is to randomly select one coordinate to update in each iteration. Parallelizing SCD allows multi-processors to update the coordinates concurrently without synchronization. Although it might result in the divergence of objective function, parallel solvers can achieve dramatic speedup when optimizing the regression problem in a very high-dimensional feature space. It is shown in [6] that the dimension of fea-

ture space for parallel solvers should be no less than $\frac{P}{2\rho}$ when there are P threads, where ρ denotes the spectral radius of A^TA, and A is the feature matrix. When we integrate screening methods with the state-of-the-art parallel solvers such as Shotgun, PBCD and Asynchronous Stochastic Coordinate Descent (ASYSCD) [11], it can result in the divergence of the objective function since the feature matrix is shrunk to a matrix with a small feature space after applying screening rules on it. Although we can reduce the number of threads to guarantee the convergence, it has a negative effect on the scalability of the parallel method. Since previous parallel solvers cannot satisfy the constraint between the number of threads and feature space, it is essential to develop a parallel solver to optimize the problem in the reduced feature data matrix.

In this paper, we propose a parallel framework to solve the Lasso regression problem on large-scale datasets with huge dimensional feature space. We parallelize screening rules by partitioning the sample and feature space to accelerate the screening process. We propose two parallel safe screening rules: Parallel Strong Rule (PSR) and Parallel Dual Polytope Projection (PDPP). To optimize the regression problem in parallel on the reduced feature matrix, we propose an Asynchronous Grouped Coordinate Descent method (AGCD) to solve the problem of small feature space in the optimization after employing screening rules. In AGCD, we introduce competition strategy to select the candidate coordinates that minimize the objective function with the maximum descent of function value. If the selected coordinate wins the competition in a group of candidates, that coordinate will be updated at this iteration, otherwise the update terminates. The main idea of AGCD is to reduce the frequency to update coordinates and select the most important candidates to update, allowing the solver to converge in a small feature space. It is different with the random selection strategy in most of the parallel solvers.

The main contributions of this study are summarized as follows:

- We propose a parallel framework by integrating the screening methods with parallel solvers to accelerate the whole learning process. To the best of our knowledge, this is the first study to introduce the idea of screening into a parallel framework.
- We propose an asynchronous parallel solver AGCD to guarantee the convergence of optimization on the reduced feature data matrix.
- We evaluate the proposed parallel framework using real world datasets, including the 810 patients of Alzheimer's Disease (AD) collected from North America with approximately 5.9 million features.
- Experimental results demonstrate the effectiveness and efficiency of proposed methods (PSR+AGCD and PDPP +AGCD). PSR+AGCD outperforms other state-of-the-art parallel solvers like PBCD and ASYSCD. When solving the Lasso regression along a sequence of parameter values, PDPP+AGCD obtains a speedup of 130 folds over ASYSCD.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents our proposed parallel framework. Section 4 analyzes the convergence of the proposed methods. Experimental results on the real-world biomedical datasets are reported in Section 5. We conclude the paper in Section 6.

2. RELATED WORK

2.1 Screening

Existing screening methods can be divided into two categories: *Safe Screening Rules* and *Heuristic Screening Rules*. Safe screening rules guarantee that the predicted inactive features have zero coefficients in the solution. In other words, discarding the inactive features in safe screening rules does not sacrifice the accuracy of optimization since the corresponding positions in the solution vector are zero in the ground truth. SAFE [3] is a safe screening method that estimates the dual optimal solution of Lasso. Strong rules [19] are another efficient screening methods based on heuristic screening rules. In most of the cases, strong rules discard more features than SAFE, leading to a substantial speedup. However, strong rules cannot guarantee that discarded features have zero components in the solution. To avoid the incorrectly discarded cases, [19] proposed a method to check the KKT conditions, ensuring the correctness of screening results. To optimize the problem along a sequence of parameter values, Enhanced Dual Polytope Projection (EDPP) [24] is an efficient and effective safe screening method and achieves significant speedup for the Lasso problem.

2.2 Parallel Solvers

After we get the reduced feature matrix from screening rules, we can apply different solvers to optimize it, such as Stochastic Gradient Descent (SGD) [26], FISTA [1], ADMM [2] and SCD [17, 12, 14]. When we consider solvers in a multithread environment, a lot of parallel solvers were proposed based on the SCD. Shotgun [6] is a parallel coordinate descent method which allows multiple processors to update the coordinates concurrently. PBCD [13] described a method for the convex composite optimization problem. In this method, all the processors update randomly selected coordinates or blocks synchronously at each iteration. The method in [9, 16, 15] are based on the greedy coordinate descent method. Recently, asynchronous parallel methods are proposed to accelerate the updating process. ASYSCD [11] proved the linear convergence of asynchronous SCD solver under the essential strong convexity condition. PASSCoDe [4] is an asynchronous Stochastic Dual Coordinate Descent(SDCD) method for solving the dual problem. Parallel SDCA [20] is an asynchronous parallel solver based on the Stochastic Dual Coordinate Ascent (SDCA) method. Both PassCoDe and Parallel SDCA focus on the ℓ_2-regularized models.

3. PROPOSED PARALLEL FRAMEWORK

In this section, we present the proposed parallel framework based on a shared memory model with multiple processors. Our parallel framework is composed of two main procedures:

1. Identify the inactive features by the parallel screening rules and remove inactive features from optimization.

2. Solve the Lasso regression on the reduced feature matrix in parallel.

In step one, we parallelize screening rules to identify and

discard inactive features, significantly accelerating the whole learning process. We propose two parallel screening rules: PSR and PDPP.

In step two, we propose an asynchronous parallel solver AGCD to solve the Lasso regression on the reduced data matrix in parallel.

3.1 Problem Formulation

In this paper, we consider the following ℓ_1-regularized minimization problem:

$$\min_{x\in\mathbb{R}^N} F(x) = \frac{1}{2}\|Ax-y\|_2^2 + \lambda\|x\|_1, \tag{1}$$

where A is the design matrix and $A \in \mathbb{R}^{M\times N}$, $y \in \mathbb{R}^M$ is the response vector and x is the sparse model we need to learn. λ is the regularization parameter and $\lambda > 0$.

3.2 Parallel SAFE and Parallel Strong Rules

SAFE [3] is an efficient safe screening method. SAFE discards the ith entry of x when the following rule holds:

$$|A_i^T y| < \lambda - \|A_i\|_2\|y\|_2\frac{\lambda_{\max}-\lambda}{\lambda_{\max}}, \tag{2}$$

where $\lambda_{\max} = \max_i |A_i^T y|$, A_i denotes the ith column of A.

Strong rules [19] are another efficient screening methods based on heuristic screening method. In strong rules, the ith feature will be discarded when satisfies the following equation:

$$|A_i^T y| < 2\lambda - \lambda_{\max}. \tag{3}$$

The calculation of $\lambda_{\max}$ follows the same way in SAFE.

For large-scale problems, it is necessary to parallelize the learning process. To speedup the learning process, we parallelize screening rules in a multithreading environment. Suppose there are P processors, we partition the feature space into P parts. The jth processor holds a subset S_j of feature space where $S_j \subseteq S$ and $S = \{1, 2, ..., N\}$. To average the performance of parallel solvers, each thread holds N/P coordinates and the partition is non-overlapped. We summarize the Parallel SAFE rule (PSAFE) in algorithm 1.

At the beginning, every processor generates the index set S_j and $S_j \in \mathbb{R}^{N/P}$. Let us define some notations here. E is a vector and $E \in \mathbb{R}^{N/P}$. $[W]_{S_j}$ indicates the collection of ωth element in W where $\omega \in S_j$ if W is a vector. When W represents a matrix, $[W]_{S_j}$ denotes the collection of ωth column of W and $\omega \in S_j$. Since $\lambda_{\max} = \max_i |A_i^T y|$, we first need to compute $E = A^T b$ firstly. To achieve this on P processors, we partition the computation into P parts. Every processor performs $[E]_{S_j} = [A]_{S_j}^T y$ in parallel. The time complexity is reduced from O(MN) to O(MN/P) since no synchronization is needed between processors. E is stored as a global variable and can be accessed by all the processors after updated. Then we get $\lambda_{\max}$ by $\|E\|_\infty$. From the 6th line to the 11th line in algorithm 1, every processor performs screening rules on its own index set S_j to select the active features. Since A and b are global variables, all the processors are able to calculate σ and τ in parallel without synchronization. In the end, we get the selected index set I and reduced feature matrix $\widetilde{A}$. Suppose I has $\widetilde{N}$ elements of true values, the dimension of $\widetilde{A}$ is $\mathbb{R}^{M\times\widetilde{N}}$. The original optimization problem (1) can be reformulated as:

$$\min_{\widetilde{x}} \frac{1}{2}\|\widetilde{A}\widetilde{x}-y\|_2^2 + \lambda\|\widetilde{x}\|_1 : \widetilde{x} \in \mathbb{R}^{\widetilde{N}}. \tag{4}$$

Algorithm 1 Parallel SAFE rule (PSAFE)

Input:
A, b and λ.
Output:
$\widetilde{A}$ and the selected index set I.

1: **Initialize:** $I = \mathbf{0} \in \mathbb{R}^N$.
2: **In parallel** on P processors.
3: Generate the index set S_j for the jth processor.
4: Compute the λ_{max}: $[E]_{S_j} = [A]_{S_j}^T y$, $\lambda_{\max} = \|E\|_\infty$.
5: Get the norm of response y: $\sigma = \|y\|_2$.
6: **for** each element i in the set of S_j **do**
7: Get the norm of the ith column of A: $\tau = \|A_i\|_2$.
8: **if** $|A_i^T y| \geq \lambda - \tau * \sigma * \frac{\lambda_{\max}-\lambda}{\lambda_{\max}}$ **then**
9: $I_i =$ **true**, select ith column of A into $\widetilde{A}$.
10: **end if**
11: **end for**

After we obtain the solution vector $\widetilde{x}^*$ for problem (4), we can recover x^* by I and $\widetilde{x}^*$.

The implementation of PSR in parallel follows the same way of PSAFE. The only difference between PSR and PSAFE is that the computation of $\|A_i\|_2$ and $\|y\|_2$ in equation (2) is not needed in PSR. We employ the same partition strategy and parallel technique to parallelize the strong rules in (3). We skip the details of implementation for brevity.

3.3 Parallel Dual Polytope Projection

In many machine learning applications, the optimal value of the regularization parameter λ is unknown. To tune the value of λ, commonly used methods such as cross validation needs to solve the Lasso problem along a sequence of parameter values $\lambda_0 > \lambda_1 > ... > \lambda_\kappa$, which can be very time-consuming. A sequential version of strong rules was proposed in [19] by utilizing the information of optimal solutions in the previous parameter. Suppose we have already obtained the solution vector $x(\lambda_{k-1})^*$ at λ_{k-1} where the integer $k \in [1, \kappa]$, the sequential strong rule rejects the ith feature at λ_k when the following rule holds:

$$|A_i^T(y - Ax(\lambda_{k-1})^*| < 2\lambda_k - \lambda_{k-1}. \tag{5}$$

Although the sequential strong rule is able to predict a large proportion of inactive features, it might mistakenly discard active features that have nonzero components in the solution. We need to check the KKT conditions to guarantee the correctness of the predicted results.

EDPP [24] is a highly efficient safe screening method that estimates the dual problem and geometric properties of Lasso regression, achieving significant speedups for real-world applications. The implementation details of EDPP is available on the GitHub[1]. We omit the introduction of EPDD for brevity. Based on the partition strategy and parallel technology employed on PSAFE and PSR, we propose a parallel safe screening rules, known as the Parallel Dual Polytope Projection (PDPP), to quickly identify and discard inactive features parallelly in a sequence of parameters.

To parallelize the screening rules, we need to partition both the feature space and sample space. In Section 2.1, this is done in the feature space, and we follow a similar

[1]http://dpc-screening.github.io/lasso.html

way to partition it in the sample space. Before introducing the details about the proposed algorithm, we first introduce notations in the paper. As we discussed in Section 2.1, we use $[W]_{S_j}$ to indicate the collection of elements of W in the index set S_j if W denotes a vector. When W is a matrix, we use the $[W]_{S_j}$ to represent the corresponding columns of W in the index set S_j. We use the same notations in PDPP. Suppose there are P processors, we partition the sample space into P parts. The jth processor holds an index set T_j of sample space, where $T_j \subseteq T$ and $T = \{1, 2, ..., M\}$. Every subset T_j has M/P elements and there is no overlap among them. When W denotes a vector, $\{W\}_{T_j}$ indicates the collection of every ωth elements from W in the index set T_j where $\omega \in T_j$. When W is a data matrix, $\{W\}_{T_j}$ denotes the collection of every ωth rows in W where $\omega \in T_j$. To take $\{A\}_{T_j}$ as an example, we extract columns of A in the index set T_j to construct it. So the dimension of $\{A\}_{T_j}$ is $\mathbb{R}^{\frac{M}{P} \times N}$. We summarize the PDPP method in algorithm 2.

Algorithm 2 Parallel Dual Polytope Projection (PDPP)

1: **In parallel** on P processors
2: Generate the S_j and T_j for the jth processor.
3: Compute the λ_{max}: $[E]_{S_j} = [A]^T_{S_j} y$, $\lambda_{\max} = \|E\|_\infty$.
4: $\phi = argmax_i |E|$, $v = A_\phi$, v is the ϕth column of A.
5: Let $\lambda_0 \in (0, \lambda_{\max}]$ and $\lambda \in (0, \lambda_0]$.
6: **if** $\lambda = \lambda_{\max}$ **then**
7: $\{\theta(\lambda)\}_{T_j} = \frac{\{y\}_{T_j}}{\lambda_{\max}}$.
8: **else**
9: $\{\theta(\lambda)\}_{T_j} = \frac{\{y\}_{T_j} - \{A\}_{T_j} x(\lambda)^*}{\lambda}$.
10: **end if**
11: **if** $\lambda_0 = \lambda_{\max}$ **then**
12: $\{v_1(\lambda_0)\}_{T_j} = sign(v^T y)\{v\}_{T_j}$.
13: **else**
14: $\{v_1(\lambda_0)\}_{T_j} = \frac{\{y\}_{T_j}}{\lambda_0} - \{\theta(\lambda_0)\}_{T_j}$.
15: **end if**
16: $\{v_2(\lambda, \lambda_0)\}_{T_j} = \frac{\{y\}_{T_j}}{\lambda} - \{\theta(\lambda_0)\}_{T_j}$.
17: $\alpha = \frac{<v_1(\lambda_0), v_2(\lambda, \lambda_0)>}{\|v_1(\lambda_0)\|_2^2}$.
18: $\{v_2^\perp(\lambda, \lambda_0)\}_{T_j} = \{v_2(\lambda, \lambda_0)\}_{T_j} - \alpha\{v_1(\lambda_0)\}_{T_j}$.
19: Given $\lambda_{\max} = \lambda_0 > ... > \lambda_\kappa$, for $k \in [1, \kappa]$, we make a prediction of screening on λ_k if $x(\lambda_{k-1})^*$ is known:
20: $[w]_{S_j} = [A]^T_{S_j}(\theta(\lambda_{k-1}) + \frac{1}{2} v_2^\perp(\lambda_k, \lambda_{k-1}))$
21: **for** every element i in the set of S_j **do**
22: **if** $w_i < 1 - \frac{1}{2}\|v_2^\perp(\lambda_k, \lambda_{k-1})\|_2 \|A_i\|_2$ **then**
23: Discard the i th column from A.
24: **end if**
25: **end for**

In PDPP, all the P processors perform the computation in parallel. Firstly, the jth processor generates the corresponding index set S_j and T_j by the method we discussed previously. Then we follow the same way in PSAFE and PSR to calculate the $\lambda_{\max}$ in parallel. The dimensions of $\theta(\lambda)$, $v_1(\lambda_0)$, $v_2(\lambda, \lambda_0)$, $v_2^\perp(\lambda, \lambda_0)$ are $\mathbb{R}^M$. In PDPP, we employ partition strategy in sample space on these variables: $\{\theta(\lambda)\}_{T_j}$, $\{v_1(\lambda_0)\}_{T_j}$, $\{v_2(\lambda, \lambda_0)\}_{T_j}$ and $\{v_2^\perp(\lambda, \lambda_0)\}_{T_j}$. As a result, the computation of these variables is performed in parallel. From line 19 to line 25 in algorithm 2, we employ PDPP on a sequence of parameter values: $\lambda_{\max} = \lambda_0 > ... > \lambda_\kappa$. When performing the screening rule on λ_k and $k \in [1, \kappa]$, we need to compute w firstly, where $w \in \mathbb{R}^N$. We perform the computation of w based on the partition strategy in the feature space:

$$[w]_{S_j} = [A]^T_{S_j}(\theta(\lambda_{k-1}) + \frac{1}{2} v_2^\perp(\lambda_k, \lambda_{k-1})). \quad (6)$$

Finally, for each element i in the index set S_j, we will identify the ith entry of $x(\lambda_k)^*$ to be zero if the following rule holds:

$$w_i \geq 1 - \frac{1}{2}\|v_2^\perp(\lambda_k, \lambda_{k-1})\|_2 \|A_i\|_2. \quad (7)$$

The calculation of $\|v_2^\perp(\lambda_k, \lambda_{k-1})\|_2$ and $\|A_i\|_2$ in (7) is similar to the calculation of $\|y\|_2$ and $\|A_i\|_2$ in algorithm 1.

Overall, the time complexity of PDPP is O(MN/P). Regardless of the calculation and updating on the vector variables, the calculation of these variables is dominant: w, $\{\theta(\lambda)\}_{T_j}$ and $\|A_i\|_2$ where $i \in S_j$. The calculation of all these variables can be parallelized by the partition strategy in PDPP without synchronization among processors.

3.4 Asynchronous Grouped Coordinate Descent

To address the challenge we discussed in section 2.2, we propose a novel parallel solver, called Asynchronous Grouped Coordinate Descent (AGCD), to solve the Lasso regression on the reduced feature matrix. Rather than randomly selecting coordinates or blocks to update asynchronously among threads, AGCD adopts a grouped selection strategy; that is, chooses the candidate that minimizes the objective function with the most descent to update among a group of coordinates. The details of AGCD are given in algorithm 3.

In AGCD, there are two global variables d and R to store where $d \in \mathbb{R}^{\widetilde{N}}$ and $R \in \mathbb{R}^M$. We initialize d to be zero and R to be $-y$. In each iteration, every processor randomly picks up a coordinate i from $\{1, 2, ..., \widetilde{N}\}$ to estimate and update. The calculation of the gradient for the ith coordinate, which is denotes as $g(\widetilde{x})_i$, can be written as:

$$g(\widetilde{x})_i = \widetilde{A}_i^T(\widetilde{A}\widetilde{x} - y). \quad (8)$$

To make it more efficient, the calculation of (8) can be decomposed into the following steps:

Step1: Calculate the gradient: $g(\widetilde{x})_i = \widetilde{A}_i^T R$ and get $\Delta\widetilde{x}_i$.

Step2: Update R : $R = R + \Delta\widetilde{x}_i \widetilde{A}_i^T$.

Since R is initialized as $-y$, R stores the information of $\widetilde{A}_i^T(\widetilde{A}x - y)$ by following the above updating rules. To calculate $\Delta\widetilde{x}_i$, we apply soft thresholding function [17] to get the proximal gradient for $\widetilde{x}_i$. The definition of soft thresholding operator Γ is given by :

$$\Gamma_\varphi(x) = sign(x)(|x| - \varphi). \quad (9)$$

In algorithm 3, L denotes the Lipschitz constant. For SCD [17, 12, 14], the Lipschitz constant is set to be $\|A_i\|_2^2$ when updating the ith coordinate. Since SCD randomly picks only one coordinate to update, the problem has a closed-form solution in each iteration. When considering a multithreading environment, the way to calculate L is different. PBCD [13] employs Expectation Maximization (EM) to get an approximation model on L but it depends on the sparsity of the

Algorithm 3 Asynchronous Grouped Coordinate Descent

Input:

$\widetilde{A}$, b and λ.

Output:

The solution vector $\widetilde{x}$ for the problem (4)

Initialize: $\widetilde{x} = d = \mathbf{0} \in \mathbb{R}^{\widetilde{N}}$ and $R = -b$.
while not converged **do**
  In parallel on P processors
    Randomly pick i from the index set $\{1, 2, ..., \widetilde{N}\}$.
    Compute the ith gradient: $g(\widetilde{x})_i = \widetilde{A}_i^T R$.
    Get $\Delta\widetilde{x}_i$: $\Delta\widetilde{x}_i = \Gamma_{\lambda/L}(\widetilde{x}_i - \frac{g(\widetilde{x})_i}{L}) - \widetilde{x}_i$.
    $d_i = \lambda(|\widetilde{x}_i| - |\widetilde{x}_i + \Delta\widetilde{x}_i|) - g(\widetilde{x})_i \Delta x_i - \frac{1}{2}\|\widetilde{A}_i\|_2^2 \Delta x_i^2$.
    for $t = (i - \omega/2)$ **to** $(i + \omega/2)$ **do**
      if $d_i < d_t$ **then**
        Return and perform the next iteration.
      end if
    end for
    Update $\widetilde{x}_i$: $\widetilde{x}_i = \widetilde{x}_i + \Delta\widetilde{x}_i$.
    Update R: $R = R + \Delta\widetilde{x}_i \widetilde{A}_i^T$.
    Update d_i by the same way from 5th to 7th line.
end while

feature matrix. In this paper, we employ the same method in [11] to get the Lipschitz constant from the Hassian matrix.

$$L = \max_{i=1,2,...,\widetilde{N}} [\nabla^2 F(\widetilde{x})]_{ii}. \tag{10}$$

For the Lasso problem, L can be calculated by:

$$L = \max_{i=1,2,...,\widetilde{N}} \|A_i\|_2^2. \tag{11}$$

3.5 Grouped Selection Strategy

The strategy of selecting potential coordinates in AGCD is to evaluate the descent of objective function for the selected candidate. If the selected coordinate i wins the competition in a group of candidates, x_i will be updated by $\Delta\widetilde{x}_i$. Otherwise this selection fails and the update in this iteration is terminated. In a multithreading environment, all the processors perform this process in parallel.

Then we need to estimate the descent of objective function for a specific coordinate. The descent of the objective function is stored and updated in d. For the ith coordinate, suppose that we have already obtained $\Delta\widetilde{x}_i$ and $g(\widetilde{x})_i$, d_i can be estimated by the following equation:

$$d_i = F(\widetilde{x}) - F(\widetilde{x} + \Delta\widetilde{x}_i e_i). \tag{12}$$

where e_i is a unit vector in the ith coordinate. When considering the Lasso problem, d_i can be rewritten as the following formula:

$$\lambda(|\widetilde{x}_i| - |\widetilde{x}_i + \Delta\widetilde{x}_i|) + \frac{1}{2}(\|\widetilde{A}\widetilde{x} - y\|_2^2 - \|\widetilde{A}\widetilde{x} + \widetilde{A}_i^T \Delta\widetilde{x}_i - y\|_2^2). \tag{13}$$

Finally, d_i can be calculated by:

$$d_i = \lambda(|\widetilde{x}_i| - |\widetilde{x}_i + \Delta\widetilde{x}_i|) - \frac{1}{2}\Delta\widetilde{x}_i{}^2 \|\widetilde{A}_i\|_2^2 - \Delta\widetilde{x}_i \widetilde{A}_i^T R, \tag{14}$$

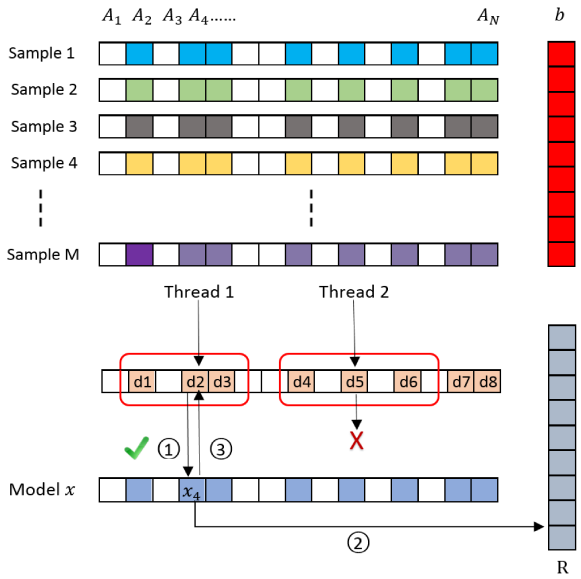

Figure 1: Illustration of AGCD with two threads. The white blocks in each sample represent inactive features discarded by screening rules. Thread 1 chooses the 2nd active feature to evaluate. Firstly, d_2 is updated and evaluated in a group of $\{d_1, d_2, d_3\}$. d_2 wins the competition and we follow the three steps to update x_4, R and d_2. Thread 2 selects the 5th candidate. However, it fails in the competition and update is terminated for thread 2 in this iteration.

where $\widetilde{A}_i^T R$ equals to $g(\widetilde{x})_i$ that has already been calculated previously. The value of $\|\widetilde{A}_i\|_2^2$ is also obtained when calculating the Lipschitz constant L in equation (11). Thus, the time complexity to update d_i is $O(1)$.

After d_i is updated, there is a competition between the ith coordinate and a group of ω candidates. If d_i wins, $\widetilde{x}_i$ will be updated. Otherwise the update of the ith coordinate in this iteration is terminated. Bradley et al.[6] showed that the number of updated coordinates is at most $\widetilde{N}/2\rho$ in one iteration where ρ is the spectral radius of $\widetilde{A}^T\widetilde{A}$. AGCD divides the feature space into $\widetilde{N}/\omega$ groups, which means there are at most $\widetilde{N}/\omega$ coordinates to be updated in each iteration. Therefore, we set the size of group ω to be 2ρ in AGCD. If ω is larger than $\widetilde{N}$, ω is set to be $\widetilde{N}$. The way we set the candidate group is to select ω number of coordinates that are close to the ith coordinate. Specifically, the index in the group starts from $i - \omega/2$ and ends at $i + \omega/2$. If $i - \omega/2 \leq 0$, it starts from 1 to ω. When $i + \omega/2 \geq \widetilde{N}$, it is from $i - \omega$ to i. If the ith coordinate wins the competition, the update is performed by the following three steps:

Step1: Update $\widetilde{x}_i$: $\widetilde{x}_i + \Delta\widetilde{x}_i$.

Step2: Update R : $R = R + \Delta\widetilde{x}_i \widetilde{A}_i^T$.

Step3: Update d_i by equation (8), (9) and (14).

Fig. 1 illustrates the process of group selection and asynchronous updating flowchart with two threads. Although d_i has already been updated at the beginning, AGCD still needs to perform Step 3 in the above updating rules because we intend to minimize the effects of un-updated winners' d_i to the competition of other candidates. We still take the example in Figure 1 to illustrate this. After the 2nd coordinate

wins the competition, AGCD performs Step 1 and Step 2 to update $\widetilde{x}_i$ and R. However, d_i is not the current descent of object function since x_i and R have changed. In the next iteration, if AGCD selects the 1st coordinate to evaluate, x_1 might still not be updated since $d_2 > d_1$ in the last iteration. Although d_1 is updated in the next iteration, x_1 still has a lower chance to be updated since x_2 is the winner last time and d_2 is the "winning distance". Because of asynchronous characteristic of AGCD, it is not guaranteed that all the d_i are updated to the newest one. AGCD makes all the winners' d_i updated to newest value to minimize the effects of winners to competitions of other candidates.

3.6 Discussion

We apply atomic operations to avoid the synchronization among threads when updating x_i, R and d_i. Since x, R and d_i are global variables, it is necessary to add locks on these shared variables when multiple threads attempt to update them simultaneously. However, updating a single variable and locking all the variables is not an efficient strategy since all the other threads have to wait for one thread to finish its job. We employ atomic operations to write the global variables atomically without any locks. [11] and [4] have observed empirical convergence when applying "atomic writes" on updating the shared variables.

AGCD adopts a grouped selection strategy to update the solution vector by choosing the candidate that minimizes the objective function with the most descent. In the random selection strategy used in parallel SCD solvers, a number of processors update the solution vector asynchronously, which is more likely to result in the divergence of the optimization problem in a small feature space. In AGCD, the update of solution variables is not as frequent as in the random selection strategy. The selected coordinate has to beat a group of candidates to get the chance to update. In fact, the selected coordinates will not be updated in most of the iterations since it failed in the competition. However, this does not mean that the computation spent in a failed candidate is a waste of time. Although x_i is not updated, it updates the descent value d_i for that coordinate. Suppose x_i is updated, it means that R is changed. As a result, all the elements in d should be updated by (14). Therefore, updating d concurrently is critical to guarantee the accurate result of competition in the grouped selection strategy.

4. CONVERGENCE ANALYSIS

In this section, we analyze the convergence of the proposed parallel framework. PSAFE and PDPP are safe screening rules and it is guaranteed that all the discarded features have zero coefficients in the solution. PSR is a heuristic screening method but we can ensure the correctness of result by checking the KKT condition. AGCD can be safely applied on the reduce feature matrix $\widetilde{A}$ to optimize the problem (4). Therefore, the proposed parallel framework will work if we prove the convergence of AGCD.

We follow the same way in [17] to rewrite the objective function (1) into an equivalent problem with a twice-differentiable regularizer:

$$\min_{\hat{x}} \frac{1}{2}\sum_{j=1}^{M}(\hat{a}_j^T\hat{x} - b_j)^2 + \lambda\sum_{i=1}^{2N}\hat{x}_i : \hat{x} \in \mathbb{R}^{2N}, \qquad (15)$$

where a_j denotes the jth row of A and the feature space is duplicated as: $\hat{a}_j = [a_j; -a_j]$ and $\hat{a}_j \in \mathbb{R}^{2N}$. Once the optimal solution $\hat{x}^*$ of equation (15) is obtained, we can recover the solution vector x^* of (1) by $x_i^* = \hat{x}_{d+i}^* - \hat{x}_i^*$. We denote the objective function $F(x)$ equal to (15) in the convergence analysis part.

Definition 1. Let $F(x) : \mathbb{R}^{2N} \to \mathbb{R}$ be a convex function. Assume that there exists $\beta > 0$, for all x and Δx updated in parallel, we have the following rule:

$$F(x + \Delta x) \leq F(x) + \Delta x^T \nabla F(x) + \frac{\beta}{2}\Delta x^T A^T A \Delta x.$$

We denote $\beta = 1$ for the square loss function and $\beta = \frac{1}{4}$ for the logistic loss function. Let $\hat{d} = [\hat{d}_1, \hat{d}_2, ...\hat{d}_{2N}]$ represent the potential candidates updated by (12). Δx denotes the collective update of x in one iteration. Δx_i is equal to zero when $\hat{d}_i$ fails the competition where $i \in (1, 2N)$.

When there is only one coordinate updated at the same time, we have $\Delta x = (\Delta x_i)e_i$ and e_i is a unit vector in the ith coordinate. The process of optimization is the same as the sequential coordinate descent when one coordinate is updated at each iteration. It was shown in [17] that the sequential coordinate descent converges by the following bound:

$$E[F(x^{(K)})] - F(x^*) \leq \frac{N(\beta\|x^*\|_2^2 + 2F(x^{(0)}))}{K+1}, \qquad (16)$$

where $F(x^{(K)})$ is the output after K iterations. The convergence analysis of AGCD is the same as sequential coordinate descent if only one coordinate is updated in each iteration.

When there are more than one candidates winning the competition at the same time, we summarize the main analysis in the following theorem:

THEOREM 1. *Let x^* be the solution of F(x) and $x^{(K)}$ be the output of AGCD after K iterations. Let P be the number of processors and Φ denotes the maximum number of candidates to be updated in one iteration. Suppose $F(x)$ satisfies the assumption of Definition 1; let $\epsilon = \frac{(\Phi-1)(\rho-1)}{2N-1} < 1$ and ρ be the spectral radius of A^TA. We have*

$$E[F(x^{(K)})] - F(x^*) \leq \frac{N(\beta\|x^*\|_2^2 + \frac{2}{1-\epsilon}F(x^{(0)}))}{(K+1)\Phi}.$$

PROOF. We use a similar technique in [6] to prove this theorem. Let Θ denote the index set that collects the winners of competitions in one iteration and $\Phi = |\Theta|$. Based on the assumption of Definition 1, we have

$$\begin{aligned}
&E_\Theta[F(x + \Delta x) - F(x)] \\
&\leq E_\Theta[\Delta x^T \nabla F(x) + \frac{\beta}{2}\Delta x^T A^T A \Delta x] \\
&= \Phi E_i[\Delta x_i \nabla F(x)_i + \frac{\beta}{2}(\Delta x_i)^2] \\
&+ \frac{\beta}{2}\Phi(\Phi-1)E_{i,j:i\neq j}[\Delta x_i (A^TA)_{i,j}\Delta x_j] \\
&= \Phi E_i[\Delta x_i \nabla F(x)_i + \frac{\beta}{2}(1 + \frac{(\Phi-1)(\rho-1)}{2N-1})(\Delta x_i)^2] \\
&= \Phi E_i[\Delta x_i \nabla F(x)_i + \frac{\beta}{2}(1+\epsilon)(\Delta x_i)^2].
\end{aligned}$$

Let $\rho = \max_{\mu:\mu^T\mu=1}\mu^T(A^TA)\mu$. $E_{i,j:i\neq j}[\Delta x_i (A^TA)_{i,j}\Delta x_j]$ is bounded by ρ in terms of $E_i[\Delta(x_i)^2]$ where i is chosen uniformly at random from $\{1, ..., 2N\}$. The rest of proof follows the same way in Shotgun [6]'s convergence analysis to obtain the result of Theorem 1. We omit it for brevity. □

Table 1: A comparison of PDPP+AGCD and EDPP+SLEP along a sequence of 100 parameter values on 0.5 million ADNI dataset

EDPP+SLEP					PDPP+AGCD				
No. in the sequence	λ/λmax	nnz after screening	nnz in the solution	Objective function	No. in the sequence	λ/λmax	nnz after screening	nnz in the solution	Objective function
1	1.0	0	0	373.0	1	1.0	0	0	373.0
6	0.95	9	4	372.9657	6	0.95	9	4	372.9657
12	0.9	17	8	372.6708	12	0.9	17	8	372.6708
17	0.85	26	16	372.0295	17	0.85	26	16	372.0295
23	0.8	52	26	370.5095	23	0.8	52	26	370.5095
28	0.75	82	46	368.3644	28	0.75	82	46	368.3644
34	0.7	133	69	364.2990	34	0.7	133	69	364.2990
39	0.65	182	103	359.3397	39	0.65	182	103	359.3397
45	0.6	267	134	351.0384	45	0.6	267	134	351.0384
50	0.55	351	169	341.9249	50	0.55	351	169	341.9249
56	0.5	447	216	327.9936	56	0.5	447	216	327.9936
61	0.45	551	260	313.6343	61	0.45	551	260	313.6343
67	0.4	717	307	292.8738	67	0.4	717	307	292.8738
72	0.35	867	364	272.4728	72	0.35	867	364	272.4728
78	0.3	1104	419	244.0223	78	0.3	1104	418	244.0223
84	0.25	1423	473	210.9855	84	0.25	1423	473	210.9855
89	0.2	1878	508	179.7974	89	0.2	1878	508	179.7974
94	0.15	2745	571	145.1732	94	0.15	2745	569	145.1732
100	0.1	6065	650	98.9118	100	0.1	6070	651	98.9118

In Shotgun, it was shown that the number of processors should satisfy $P \leq P^* \approx \frac{N}{2\rho}$ and the experiment demonstrates that Shotgun diverges as P exceeds P^*. As we discussed in section 3.5, the size of each group is set to be $\omega = 2\rho$. The maximum number of updated coordinates in one iteration is $\Phi = \frac{N}{\omega}$ which satisfy the above constraint. In the real cases, the number of updated candidates is much smaller than P since most of the updates happen in the calculation of d. In the grouped selection strategy of AGCD, each coordinate has a low probability to be updated among ω candidates. As a result, P can be equal to or larger than $\frac{N}{2\rho}$ in the real application of AGCD. We demonstrate it in the experiment that AGCD encountered the cases that the number of processors is larger than the number of active features while AGCD still converged to the optimal value.

5. EXPERIMENTAL RESULTS

In this section, we conduct several experiments to evaluate the convergence and speedup of the proposed framework on the following four data sets: ADNI[2], MNIST [7], rcv1 [8] and news20 [5]. The Alzheimer's Disease NeuroimagingInitiative (ADNI) is a real biomedical dataset collected from neuroimaging and genomic data from elderly individuals across North America, including 809 patients of Alzheimer's disease with 5,906,152 features, involving a 80 GB feature matrix with 42 billion nonzeros. For MNIST, rcv1 and news20, we use the training dataset obtained from LIBSVM data set repository[3] to construct the feature data matrices and response vectors. We compare our method with the state-of-the-art algorithms like PBCD [13] and ASYSCD [11]. All the experiments are carried out on an Intel (R) Xeon (R) 48-core machine with 2.50 GHZ processors and 256 GB of globally addressable memory. We employ OpenMP as the parallel framework and all the methods are implemented in C++ for fair comparisons.

5.1 Accuracy Evaluation

In this experiment, we examine the accuracy of solution vectors in the proposed method. We perform PDPP+AGCD along a sequence of 100 parameter values equally spaced on the linear scale of $\lambda/\lambda_{\max}$ from 0.1 to 1. To make a comparison, we perform EDPP using SLEP [10] as the solver on the same sequence. In SLEP, we force the "LeastR" function to run 500 iterations. AGCD also executes 500 iterations using 48 threads. Experiments are conducted on ADNI data sets. We choose the volume of the right pallidum as the response, including 747 samples by removing samples without labels. The volumes of brain regions are extracted from each subject's T1 MRI scan using Freesurfer[4]. We randomly select 0.5 million features from ADNI to construct the feature matrix and normalize the matrix using the "zscore" function in Matlab . The experimental result is shown in Table 1.

We report the result of 20 parameter values from 100 parameters. The first column in both methods is the position of the parameter in the sequence. The third column shows the remaining number of features $\widetilde{N}$ after applying screening rules. Table 1 shows that the optimal value obtained by the PDPP+AGCD and the number of nonzero in the solution is the same as that of EDPP+SLEP. When $\lambda/\lambda_{\max}$ is higher than 0.8, the remaining features after screening is less than the number of threads. However, PDPP+AGCD is still able to converge to the optimal value.

5.2 Convergence Comparison

In this experiment, we evaluate the convergence property of the proposed parallel methods. We conduct the experiment on PSR+AGCD and compare with state-of-the-art parallel solvers: PBCD and ASYSCD. We choose two dif-

[2]http://www.adni-info.org

[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

[4]http://freesurfer.net/

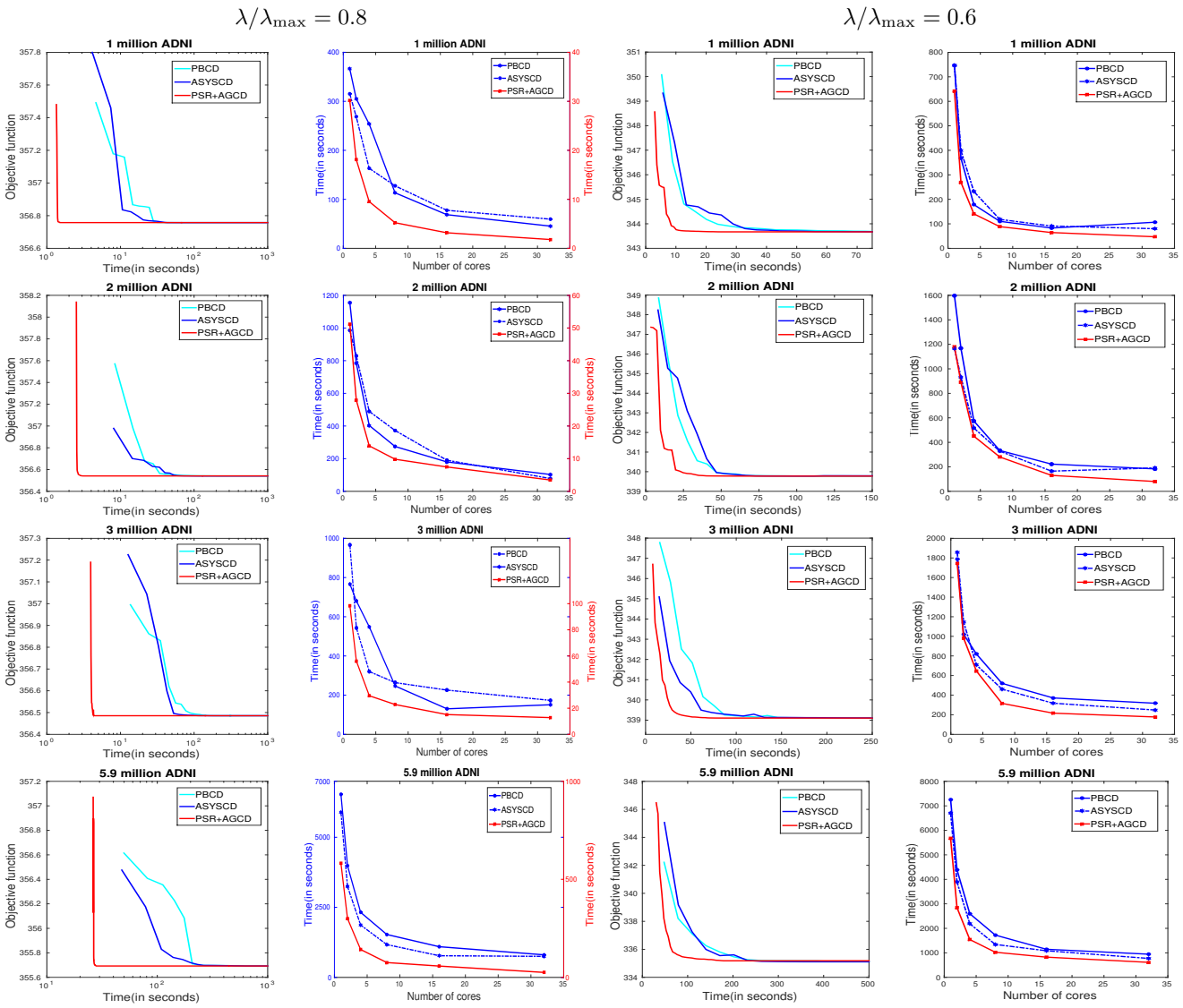

Figure 2: Convergence comparison between PBCD, ASYSCD and PSR+AGCD

ferent λ value: $0.8\lambda_{\max}$ and $0.6\lambda_{\max}$ to estimate the convergence of above methods. To prevent PSR from discarding active features, we check the KKT condition to ensure the correctness of screening results. To estimate the scalability of above algorithms, the number of cores is varied from 1 to 32: 1, 2, 4, 8, 16 and 32. For different number of cores, we show the time of optimization that solvers converged to the same optimal values with 48 cores. We evaluate the efficiency of parallel solvers on four ADNI dataset: ADNI_1m, ADNI_2m, ADNI_3m and ADNI_5.9m where feature dimension is varied from 1 million to 5.9 million. We choose the volume of hippocampus in patients as the response, including 717 samples from the original dataset. We show details of data sets in Table 2 and results of comparison in Fig. 2.

In the first and third columns of Fig. 2, we evaluate the convergence in terms of time using 48 cores. Note that we use log-scale in x-axis when $\lambda = 0.8\lambda_{\max}$. As observed from the figure, the objective function in PSR+AGCD converged faster than other solvers since most of inactive features are discarded. Most of time is spent in the screening part. When $\lambda = 0.6\lambda_{\max}$, only part of inactive features are discarded by screening but PSR+AGCD still has superior performances.

Table 3: Data statistics

Dataset	Number of features	Number of samples
MNIST	780	60000
news20	62061	15935
rcv1	47236	15564
ADNI_1m	1000000	717
ADNI_2m	2000000	717
ADNI_3m	3000000	717
ADNI_5.9m	5906152	717

The second and fourth columns of Fig. 2 show the time of different solvers that converged to the same optimal value of 48 cores when varying the number of cores from 1 to 32. Note that we use two y-axis when λ equals to $0.8\lambda_{\max}$. PBCD and ASYSCD use the left y-axis while PSR+AGCD uses the right one. PSR+AGCD outperforms the other solvers when varying the number of cores and the running time of PSR+AGCD is reduced when there are more cores.

Table 2: Efficiency comparison along a sequence of parameter values.

Dataset: MNIST								
The number of λ	Method	Time spent in different number of cores (in minutes)						
		1	2	4	8	16	32	48
100	ASYSCD	131.08	76.25	62.78	39.17	26.7	23.33	25.94
	PDPP+AGCD	6.37	3.61	2.43	1.49	1.02	0.87	1.03
	Speedup	**20.58**	**21.12**	**25.84**	**26.29**	**26.178**	**26.82**	**25.18**
200	ASYSCD	283.43	182.36	116.86	61.96	45.76	38.37	48.87
	PDPP+AGCD	10.9	6.25	3.96	1.96	1.48	1.17	1.53
	Speedup	**26.01**	**29.18**	**29.51**	**31.61**	**30.92**	**32.79**	**31.94**
400	ASYSCD	571.46	364.21	238.65	129.24	92.58	71.71	88.37
	PDPP+AGCD	17.14	11.07	7.13	3.86	2.53	1.76	2.47
	Speedup	**33.34**	**32.90**	**33.47**	**33.48**	**36.59**	**40.74**	**35.78**
Dataset: news20								
The number of λ	Method	Time spent in different number of cores (in minutes)						
		1	2	4	8	16	32	48
100	ASYSCD	2736.52	1491.26	762.57	403.06	265.71	194.27	214.48
	PDPP+AGCD	40.22	20.69	10.60	5.59	3.67	2.50	2.72
	Speedup	**68.04**	**72.07**	**71.93**	**72.03**	**72.40**	**77.67**	**78.97**
200	ASYSCD	5528.48	2813.78	1525.03	804.68	552.83	358.35	374.48
	PDPP+AGCD	64.40	30.07	16.33	8.47	5.69	3.63	3.92
	Speedup	**85.84**	**93.56**	**93.38**	**95.38**	**97.02**	**98.77**	**95.63**
400	ASYSCD	10437.35	5480.47	2812.21	1565.12	1057.30	662.792	597.35
	PDPP+AGCD	120.38	64.31	29.05	15.88	10.42	6.67	6.05
	Speedup	**86.70**	**85.21**	**96.79**	**98.54**	**101.51**	**99.47**	**98.79**
Dataset: ADNI_2m								
The number of λ	Method	Time spent in different number of cores (in minutes)						
		1	2	4	8	16	32	48
100	ASYSCD	3205.13	1892.34	1105.21	756.23	435.64	324.81	372.31
	PDPP+AGCD	48.34	23.63	12.21	8.28	5.34	4.03	4.05
	Speedup	**66.30**	**80.08**	**90.52**	**91.33**	**81.58**	**80.59**	**91.93**
200	ASYSCD	5614.21	3217.91	1821.87	1345.21	826.51	692.87	684.25
	PDPP+AGCD	63.32	34.07	18.81	12.99	7.44	5.79	5.76
	Speedup	**88.66**	**94.44**	**96.86**	**103.56**	**111.09**	**119.67**	**116.77**
400	ASYSCD	11275.34	6328.35	3812.87	2315.34	1521.54	1296.54	1125.61
	PDPP+AGCD	95.42	52.17	33.10	18.18	11.06	9.57	8.74
	Speedup	**118.16**	**119.57**	**115.19**	**127.35**	**137.57**	**135.47**	**128.79**

5.3 Time Efficiency

The advantage of the proposed parallel framework is to solve the Lasso problem along a sequence of parameter values. In this experiment, we perform PDPP+AGCD along a sequence of parameter values equally spaced on the linear scale of λ/λmax from 0.1 to 1. We vary the length of parameter sequences as 100, 200 and 400. As a comparison, ASYSCD is performed on the same sequence. The experiment is conducted at three different data sets: MNIST, news20 and ADNI_2m. Detailed information about data sets is in Table 2. To evaluate the scalability of both methods, we vary the number of cores as: 1, 2, 4, 8, 16, 32 and 48. The result of comparison is presented in Table 3.

Table 3 shows that more parameters lead to higher speedup for PDPP+AGCD compared to ASYSCD. PDPP+AGCD achieved 137 folds speedup in ADNI_2m dataset, 101 folds in news20, and 40 folds in MNIST over ASYSCD with 400 parameters. When using more cores, speedups of our method tend to increase. Thus, in terms of speedup, PDPP+AGCD favors more cores and sequences with more parameters.

5.4 Scalability

To estimate the scalability of proposed parallel methods, we perform PSR+AGCD and PDPP+AGCD on 1, 2, 4, 8, 16, 32 and 48 cores to observe the speedup. We give the definition of speedup by the following criterion:

$$\text{speedup} = \frac{\text{time spent on } P \text{ processors}}{\text{time spent on a single processor}}.$$

In this experiment, we employ both methods on 5.9 million ADNI and rcv1 data sets, respectively. PDPP+AGCD is carried out along a 100 linear-scale sequence of parameter values from 0.1 to 1. For PSR+AGCD, we set λ to be $0.8\lambda_{\max}$ in the optimization. Fig. 3 presents the result. PDPP+AGCD is more scalable than PSR+AGCD and achieves approximate 17 and 11.5 folds speedup with 48 cores in ADNI_5.9m and rcv1 data sets, respectively.

6. CONCLUSIONS

This paper proposes a parallel framework to solve the Lasso problem on huge dimensional datasets. We introduce screening rules into a parallel platform to discard the inactive features before optimization, accelerating the whole learning process significantly. Then the problem boils down to solve Lasso on a multithreading environment with a small feature space. A grouped selection strategy is proposed to select the candidates that minimize the objective function with the largest descent. Experiments demonstrate the efficiency and effective of proposed methods. For future works, we plan to extend our framework to a distributed platform.

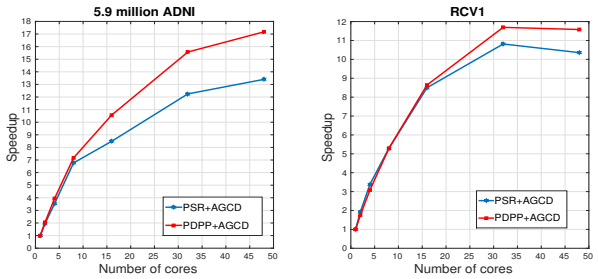

Figure 3: Speedup of proposed methods on 5.9 million ADNI and rcv1 data sets respectively

7. ACKNOWLEDGMENTS

This work was supported in part by research grants from NIH (R01 LM010730 and RF1AG051710) and NSF (IIS-0953662 and III-1421057).

8. REFERENCES

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[3] L. E. Ghaoui, V. Viallon, and T. Rabbani. Safe feature elimination for the lasso and sparse supervised learning problems. *Pacific Journal of Optimization*, (8):667–698, 2012.

[4] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. In *Proceedings of the 31st International Conference on Machine Learning (ICML-15)*, 2015.

[5] S. S. Keerthi, K. Duan, S. K. Shevade, and A. N. Poo. A fast dual algorithm for kernel logistic regression. *Machine learning*, 61(1-3):151–165, 2005.

[6] A. Kyrola, D. Bickson, C. Guestrin, and J. K. Bradley. Parallel coordinate descent for l_1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 321–328, 2011.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[9] Y. Li and S. Osher. Coordinate descent optimization for l^1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

[10] J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009.

[11] J. Liu, S. Wright, C. Re, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 469–477, 2014.

[12] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[13] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pages 1–52.

[14] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[15] C. Scherrer, M. Halappanavar, A. Tewari, and D. Haglin. Scaling up coordinate descent algorithms for large regularization problems. In *Proceedings of the 29st International Conference on Machine Learning (ICML-12)*, 2012.

[16] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems*, pages 28–36, 2012.

[17] S. Shalev-Shwartz and A. Tewari. Stochastic methods for l_1-regularized loss minimization. *The Journal of Machine Learning Research*, 12:1865–1892, 2011.

[18] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[19] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(2):245–266, 2012.

[20] K. Tran, S. Hosseini, L. Xiao, T. Finley, and M. Bilenko. Scaling up stochastic dual coordinate ascent. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1185–1194. ACM, 2015.

[21] J. Wang, Q. Li, S. Yang, W. Fan, P. Wonka, and J. Ye. A highly scalable parallel algorithm for isotropic total variation models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 235–243, 2014.

[22] J. Wang and J. Ye. Two-layer feature reduction for sparse-group lasso via decomposition of convex sets. In *Advances in Neural Information Processing Systems*, pages 2132–2140, 2014.

[23] J. Wang and J. Ye. Safe screening for multi-task feature learning with multiple data matrices. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[24] J. Wang, J. Zhou, P. Wonka, and J. Ye. Lasso screening rules via dual polytope projection. In *Advances in Neural Information Processing Systems*, pages 1070–1078, 2013.

[25] Y. Wang, Z. J. Xiang, and P. J. Ramadge. Lasso screening with a small regularization parameter. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3342–3346. IEEE, 2013.

[26] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st international conference on Machine learning (ICML-04)*, page 116. ACM, 2004.