Parallel Asynchronous Stochastic Coordinate Descent with Auxiliary Variables

Hsiang-Fu Yu
Amazon

Cho-Jui Hsieh
UCLA

Inderjit S. Dhillon
UT Austin and Amazon

Abstract

The key to the recent success of coordinate descent (CD) in many applications is to maintain a set of auxiliary variables to facilitate efficient single variable updates. For example, the vector of residual/primal variables has to be maintained when CD is applied for Lasso/linear SVM, respectively. An implementation without maintenance is $O(n)$ times slower than the one with maintenance, where n is the number of variables. In serial implementations, maintaining auxiliary variables is only a computing trick without changing the behavior of coordinate descent. However, maintenance of auxiliary variables is non-trivial when there are multiple threads/workers which read/write the auxiliary variables concurrently. Thus, most existing theoretical analysis of parallel CD either assumes vanilla CD without auxiliary variables (which ends up being extremely slow in practice) or limits to a small class of problems. In this paper, we consider a rich family of objective functions where AUX-PCD can be applied. We also establish global linear convergence for AUX-PCD with atomic operations for a general family of functions and perform a complete backward error analysis of AUX-PCD with wild updates, where some updates are not just delayed but lost because of memory conflicts. Our results enable us to provide theoretical guarantees for many practical parallel coordinate descent implementations, which currently lack guarantees (such as the implementation of Shotgun proposed by Bradley et al. 2011, which uses auxiliary variables).

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89. This work is done while all authors were still affliated with UT Austin.

1 Introduction

Stochastic Coordinate descent (CD) is now one of the most widely used algorithms for solving large-scale machine learning problems. At each step, a variable is stochastically selected for update, and the step size for updating this variable is computed by solving a single variable subproblem. Therefore, the efficiency of coordinate descent heavily depends on how efficiently one can construct and solve the univariate subproblem. This aspect has been studied separately for different machine learning problems. For example, efficient update rules have been discussed or implemented in dual CD for linear SVM [7, 21], primal CD for SVM and logistic regression [26, 4], primal Lasso problems [3], and many others.

Interestingly, most successful coordinate descent algorithms/implementations rely on the maintenance of another set of *auxiliary variables*. The use of such auxiliary variables is the key to the success of CD in many machine learning applications recently as these variables cache the latest information required to perform efficient single variable updates. For example, a vanilla dual coordinate descent algorithm for linear SVM requires going through the entire dataset to update a single variable, but by maintaining the primal variable the time complexity can be reduced to the dimensionality of a single data instance. Similar tricks have also been explicitly or implicitly applied in Lasso and logistic regression problems; they lead to faster computation without changing the behavior of coordinate descent. In this paper, we use CD to denote the serial implementation of vanilla stochastic CD and use AUX-CD to denote the serial implementation of stochastic CD with auxiliary variables.

Due to the advance of multi-core computing systems, recent research on CD has largely shifted its focus to parallelization of CD (PCD). Although AUX-CD works perfectly in the serial setting, multi-core parallelization of AUX-CD (AUX-PCD) can be tricky. As multiple threads update the auxiliary variables concurrently it is non-trivial to guarantee the correctness of auxiliary variables and simultaneously achieve good speedups. On the other hand, inconsistency of the auxiliary variables poses new challenges in theoretical analysis. Instead of the more efficient

AUX-PCD, [16, 1, 15, 5] give theoretical guarantees for the vanilla PCD for general problems, but none of them have considered the case with auxiliary variables. [9] discussed an AUX-PCD implementation for a specific smooth problem (the dual formulation of ℓ_2-regularized empirical risk minimization), but their analysis cannot be directly applied to general problems such as Lasso and ℓ_1-regularized Logistic regression. Furthermore, existing convergence analysis only considers inconsistency due to *bounded delayed updates*, while none of them considers the situations that have *missing updates* due to memory conflicts.

In this paper, we consider a rich family of objective functions where vanilla coordinate descent can be accelerated by AUX-CD, which is an efficient version of CD with the maintenance of auxiliary variables. This family captures many existing machine learning problems such as empirical risk minimization (ERM) with either a smooth or non-smooth regularizer. Our contributions are summarized below:

- We study various approaches (*atomic operations* and *wild updates*) to handle the issue of inconsistency of auxiliary variables in AUX-PCD. We also demonstrate that with a simple extension we can parallelize a second-order method such as Newton or proximal Newton with AUX-PCD.
- We bridge the gap between theoretical analysis of parallel coordinate descent and practical parallel implementations. In particular, we establish global linear convergence of AUX-PCD with *atomic operations* to maintain the auxiliary variables.
- We also provide a *complete backward error analysis* to analyze the convergence behavior of AUX-PCD with *wild updates*, where some updates are not just delayed but lost due to memory conflicts. To best of our knowledge, our analysis is the first complete backward error analysis to characterize the quality of the converged solution of AUX-PCD with wild updates.
- We perform experiments to compare the performance of variants of AUX-PCD along with other state-of-the-art algorithms on two non-smooth machine learning problems.

2 Related Work

Stochastic coordinate descent is widely used in several machine learning applications. Most implementations need to maintain another set of auxiliary variables in order to have faster updates. This has been implicitly or explicitly discussed in varied applications, for example, dual linear SVM [7], dual linear logistic regression [25], Lasso [6], primal logistic regressions and SVM [4, 26], and even for nonconex optimization problems such as matrix completion [24]. Using a single thread with sequential updates, the existence of auxiliary variables does not affect the convergence analysis, so the standard analysis for vanilla CD can be directly applied. For example, the linear convergence of CD has been shown in [17].

Practical implementation for PCD. In order to be a practical parallel implementation, one must maintain auxiliary variables to perform efficient single variable updates (otherwise the implementation will be extremely slow). For example, the famous Shotgun algorithm for Lasso [3] is described in their paper as PCD without maintaining auxiliary variables for the ease of analysis. However, their real implementation is indeed an AUX-PCD with atomic operations to maintain a set of auxiliary variables.[1] [20] also briefly mentioned atomic operations for maintaining the residual in their paper, but without detailed analysis.

Theoretical analysis for PCD. [18, 5] showed the convergence of a parallel coordinate descent algorithm, where each processor updates a randomly selected block simultaneously. [2] studied asynchronous coordinate updates, but they assume the Hessian is diagonally dominant. [16, 15, 1] present global linear convergence rates under looser conditions, where they assume bounded staleness and that the objective function is essential strongly convex. However, none of these analyses consider the auxiliary variables that have to be maintained during the optimization procedure. Unfortunately, in multi-threaded or distributed systems there will always be staleness when accessing auxiliary variables, so there is still a gap between existing theoretical analysis and practical parallel implementations.

Theoretical analysis for AUX-PCD. Another line of research focuses on AUX-PCD for a special class of problem: the dual form of L_2-regularized empirical risk minimization problems, where primal variables are maintained during the algorithm. [22, 10, 11] consider a variant of AUX-PCD which requires workers to synchronize. The existing work that is closest to our paper is the asynchronous AUX-PCD for the dual form of L_2-regularized ERM introduced in [9]. They explicitly state various choices to maintain auxiliary variables and provide a theoretical guarantee for the atomic update version. However, their analysis is constrained to a limited class of smooth functions. Furthermore, there is a lack of analysis to the convergence behavior of AUX-PCD with wild updates to maintain auxiliary variables.

3 Asynchronous Parallel Coordinate Descent with Auxiliary Variables

Coordinate descent is simple as only a single variable is involved at each step. However, the update may involve all the training samples, so it is often non-trivial to make it efficient. A common trick used in the literature is to *maintain a set of auxiliary variables* [7, 4, 26, 23]. In this paper, we

[1] https://github.com/akyrola/shotgun/

aim to empirically and theoretically study a family of functions where we can apply efficient coordinate descent with auxiliary variables *in parallel*.

3.1 A Family of Functions

We are interested in the function $F(\boldsymbol{z}) : \mathbb{R}^n \to \mathbb{R}$ which can be written in the following form:

$$F(\boldsymbol{z}) = F(\boldsymbol{z}, \boldsymbol{r}), \text{ s.t. } \boldsymbol{r} = Q\boldsymbol{z},$$
$$\text{where } F(\boldsymbol{z}, \boldsymbol{r}) = \sum_{i=1}^{n} f_i(z_i) + D(\boldsymbol{r}), \text{ and } D(\boldsymbol{r}) = \sum_{k=1}^{m} d_k(r_k), \tag{1}$$

where $Q \in \mathbb{R}^{m \times n}$ with $\boldsymbol{q}_i$ as the i-th column and $\bar{\boldsymbol{q}}_k^\top$ as the k-th row, $\boldsymbol{r} \in \mathbb{R}^m$ is a linear transformation of $\boldsymbol{z}$, and $D(\boldsymbol{r})$ is a decomposable function with $d_k(r_k)\ \forall k$. This family covers many functions used in machine learning:

Empirical Risk Minimization. Given a set of pairs $\{(\boldsymbol{x}_k, y_k) : \boldsymbol{x}_k \in \mathbb{R}^f,\ k = 1, \ldots, l\}$ and a regularizer $\mathcal{R}(\boldsymbol{w})$, the ERM problem is as follows:

$$\min_{\boldsymbol{w} \in \mathbb{R}^f} \quad \sum_{k=1}^{l} \ell_k(y_k, \boldsymbol{w}^\top \boldsymbol{x}_k) + \mathcal{R}(\boldsymbol{w}), \tag{2}$$

where ℓ_k is a loss function and $\mathcal{R}(\boldsymbol{w})$ is usually decomposable: $R(\boldsymbol{w}) = \sum_{i=1}^{f} \mathcal{R}_i(w_i)$, such as ℓ_1 and squared ℓ_2 regularization. We can see that by associating each $f_i(\cdot)$ with a $\mathcal{R}_i(\cdot)$, each d_k with a ℓ_k, each $\bar{\boldsymbol{q}}_k = \boldsymbol{x}_k$ and $r_k = \boldsymbol{w}^\top \boldsymbol{x}_k$. As a result, this family of functions includes popular machine learning models such as SVM, logistic regression, and Lasso.

The Dual Form of ℓ_2-regularized ERM. In some situations, it is easier to design an efficient single variable update rule for the dual of (2). In particular, when the ℓ_2-regularizer ($\mathcal{R}(\boldsymbol{w}) = 1/2\|\boldsymbol{w}\|^2$) is used, the dual formulation can be written as follows.

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^l} \quad \frac{1}{2} \boldsymbol{\alpha}^T X X^\top \boldsymbol{\alpha} + \sum_{i=1}^{l} \ell_i^*(-\alpha_i), \tag{3}$$

where $X = [\ldots, y_i \boldsymbol{x}_i, \ldots]^\top$, and ℓ_i^* is the conjugate function of ℓ_i, defined by $\ell_i^*(u) = \max_z(zu - \ell_i(z))$. Note that we use i to index the data instances. If we let $Q = X^\top$, $\boldsymbol{r} = Q\boldsymbol{\alpha}$, and $d_k(r_k) = r_k^2/2$, the function in (3) can be written as $\sum_{k=1}^{f} \frac{r_k^2}{2} + \sum_{i=1}^{l} \ell_i^*(-\alpha_i)$ s.t. $\boldsymbol{r} = Q\boldsymbol{\alpha}$, which clearly belongs to the function family defined by (1).

3.2 Algorithms

Let us define $T_i(\boldsymbol{z}) = \arg\min_u g(u) := F(\boldsymbol{z} + (u - z_i)\boldsymbol{e}_i)$ to be the operator to obtain the updated variable for the i-th coordinate, where $\boldsymbol{e}_i$ is the i-th column of the identity matrix I_n, and $g(u)$ can be expanded as follows:

$$\begin{aligned} g(u) &:= F(\boldsymbol{z} + (u - z_i)\boldsymbol{e}_i)) \\ &= f_i(u) + D(Q\boldsymbol{z} + (u - z_i)\boldsymbol{q}_i)) + \text{constant} \end{aligned} \tag{4}$$

Efficient Variable Update with Auxiliary Variables. Minimizing (4) usually requires first-order information about $g(u)$. A straightforward procedure costs $O(mn)$ operations to compute $Q\boldsymbol{z}$, which is too expensive for a single variable update. A reformulation with the auxiliary variable $\boldsymbol{r} = Q\boldsymbol{z}$ defined by (1) can be very useful in practice. $g(u)$ can be re-written as

$$g(u) = f_i(u) + D(\boldsymbol{r} + (u - z_i)\boldsymbol{q}_i), \tag{5}$$

where the first-order information of the second term at $u = z_i$ is $\boldsymbol{q}_i^\top \nabla D(\boldsymbol{r})$. Due to the decomposability of $D(\boldsymbol{r}) = \sum_{k=1}^{m} d_k(r_k)$, $\nabla_k D(\boldsymbol{r}) = d_k'(r_k)$, which is the first differential of $d_k(\cdot)$, can be computed in $O(1)$ if r_k is available. Thus, when the entire $\boldsymbol{r}$ is available, $\boldsymbol{q}_t^\top \nabla D(\boldsymbol{r})$ only costs $O(m)$ operations. If $\boldsymbol{q}_i$ is sparse, it only costs $O(\|\boldsymbol{q}_i\|_0)$ time. Furthermore, the maintenance of $\boldsymbol{r}$ can be done by $\boldsymbol{r} \leftarrow \boldsymbol{r} + (u^* - z_i)\boldsymbol{q}_i$, which also only costs $O(\|\boldsymbol{q}_i\|_0)$, where $u^* = T_i(\boldsymbol{z})$.

Note that instead of $\boldsymbol{r}$, it would be better to maintain some simple transformation of $\boldsymbol{r}$ such that the computation of $\nabla D(\boldsymbol{r})$ can be faster. For example, if $d_k(r_k)$ is a squared function such as $d_k(r_k) = (r_k - y_k)^2/2$, then $\nabla D(\boldsymbol{r}) = \boldsymbol{r} - \boldsymbol{y}$, where y_k is the k-th element of a constant vector $\boldsymbol{y} \in \mathbb{R}^m$. Thus, maintaining $\boldsymbol{r} - \boldsymbol{y}$, which has the same cost as maintaining $\boldsymbol{r}$, can save an extra addition when computing $\nabla D(\boldsymbol{r})$. This is the so-called *residual* in the CD approach for Lasso. When the logistic loss is used, $d_k(r_k) = \log(1 + \exp(-r_k))$, both the first and second order information are required to perform the single variable update:

$$d_k'(r_k) = 1/(1 + \exp(r_k)) \text{ and } d_k''(r_k) = d'(r_k)(1 - d'(r_k)).$$

We can see that maintaining $\exp(r_k)$ for all k can save us from expensive exponential operations. The update can also be done using $O(\|\boldsymbol{q}_i\|_0)$ time.

As the first and/or second-order information is obtained through the usage of the auxiliary variable $\boldsymbol{r}$, we can define the following operator $T_i(\boldsymbol{r}, z_i)$:

$$T_i(\boldsymbol{r}, z_i) = \min_u \ f_i(u) + D(\boldsymbol{r} + (u - z_i)\boldsymbol{q}_i). \tag{6}$$

In Algorithm 1, we describe a generic procedure for the single variable update with auxiliary variables.

Parallel Asynchronous Coordinate Updates:
To further accelerate coordinate descent by parallel computation, we describe AUX-PCD in Algorithm 2, which is a simple parallel coordinate descent routine with the operator $T_i(\boldsymbol{r}, z_i)$ described in Algorithm 1. As $T_i(\boldsymbol{r}, z_i)$ modifies both z_i and the auxiliary variable $\boldsymbol{r}$, concurrent access to

the same element of $\boldsymbol{r}$ is possible in Algorithm 2. We will discuss algorithmic issues in this section and leave theoretical issues in Section 4.

As in [9], we discuss the following two approaches[2] to deal with the concurrent access to a shared variable in a multicore setting:

- *Atomic* operations: this approach can be used to achieve better scalability by trading-off the serializability. At each update in $T_i(\boldsymbol{r}, z_i)$, there may be some stale values in $\boldsymbol{r}$. As the atomic operation can guarantee that all the updates will be performed, *the bounded staleness assumption* is usually satisfied by this approach. [3] applied such atomic operations in the coordinate descent for L_1-regularized ERM problems.
- *Wild* access: this approach gives the best scalability in practice. However, some updates might be lost due to conflicts in access, which leads to the difficulty of the theoretical analysis.

Application to Parallelize Second-Order Methods. Second-order methods are known to enjoy faster convergence than first-order methods such as CD or stochastic gradient descent in terms of iteration complexity. However, the cost per iteration of second-order methods is usually much higher as it requires n^2 entries in the Hessian to construct a quadratic approximation and obtain a Newton direction $\boldsymbol{s}^*$. Exploiting the structure of the Hessian matrix, some recent works successfully apply CD with auxiliary variables to solve the quadratic approximation with the explicit construction of Hessian matrix, such as GLMNET for ℓ_1-regularized logistic regression [27] and QUIC for ℓ_1-regularized Gaussian graphical model learning [8]. As a natural application of AUX-PCD, we can parallelize these inner coordinate descent solvers to speedup these second order methods. Due to the space limit, we leave the details in Appendix A, and the experimental results are presented in Section 5.

4 Theoretical Analysis

In this section, we establish theoretical convergence for parallel asynchronous stochastic coordinate descent with auxiliary variables. We make the following assumptions and notations:

- Either $F(\boldsymbol{z})$ is a L_{max}-smooth function or $f_i(\cdot)$ can be non-smooth but $\bar{L}$-Lipschitz continuous.
- $d_k(\cdot)$ is L-Lipschitz continuously differentiable.
- Eq (5) is σ-strongly convex.
- $M = \|Q\|_F = 1$ and $R_{max} = \max_i \|\boldsymbol{q}_i\| \leq 1$

Note that if $\|Q\|_F \neq 1$, we can always consider an equivalent scaled problem with $Q^{\text{new}} = Q\|Q\|^{-1}$, and $f_i^{\text{new}}(z_i) = f_i(\|Q\|_F z_i)$, where $\bar{L}^{\text{new}} = \|Q\|\bar{L}$. Similar to [9], we assume that $F(\boldsymbol{z})$ satisfies the following property:

Definition 1. *$F(\boldsymbol{z})$ admits the global error bound if there is a constant κ such that*

$$\|\boldsymbol{z} - P_S(\boldsymbol{z})\| \leq \kappa \|T(\boldsymbol{z}) - \boldsymbol{z}\|, \forall \boldsymbol{z} \tag{7}$$

where $P_S(\cdot)$ is the Euclidean projection to the set of optimal solutions, and $T : \mathbb{R}^n \to \mathbb{R}^n$ is the operator with the T_i defined in Section 3.2.

We assign a virtual global counter for the total number of updates, and $i(j)$ denotes the index of coordinate selected at the j-th update. Let $\{\boldsymbol{z}^j\}$ denote the sequence generated by AUX-PCD and $\{\hat{\boldsymbol{r}}^j\}$ to denote the auxiliary vector used in Step 2 of Algorithm 2 for the j-th update.

4.1 AUX-PCD with Atomic Operations

We first consider AUX-PCD with *atomic operations* to update auxiliary variables. This implies the updates to $\boldsymbol{r}$ might be delayed but definitely not lost. We establish a linear convergence rate for AUX-PCD when the delay is bounded.

Here we give the formal definition. We define $\Delta z_j = T_{i(j)}(\hat{\boldsymbol{r}}^j, z^j_{i(j)}) - z^j_{i(j)} = z^{j+1}_{i(j)} - z^j_{i(j)}$ be the step performed at the j-th update. Let $\mathcal{Z}^j$ be the set containing all the updates to $\boldsymbol{r}$ until step j:

$$\mathcal{Z}^j = \{(t,k) \mid t < j, k \in N(i(t))\}, \tag{8}$$

where $N(i)$ is the nonzero elements in $\boldsymbol{q}_i$. The set $\mathcal{U}^j$ is the updates that have been written to the auxiliary vector $\hat{\boldsymbol{r}}$ at iteration j, so

$$\begin{aligned} \text{Observed auxiliary vector: } \hat{\boldsymbol{r}}^j &= \boldsymbol{r}^0 + \sum_{(t,k)\in\mathcal{U}^j} Q_{k,i(t)} \Delta z_t \boldsymbol{e}_k \\ \text{Real auxiliary vector: } \boldsymbol{r}^j &= \boldsymbol{r}^0 + \sum_{(t,k)\in\mathcal{Z}^j} Q_{k,i(t)} \Delta z_t \boldsymbol{e}_k, \end{aligned} \tag{9}$$

where $\boldsymbol{e}_k$ is the k-th column of the identity matrix. We also make a **bounded delay assumption**: the updates to $\boldsymbol{r}$ have a maximum delay τ, which can be formally written as

$$\mathcal{Z}^{j-\tau} \subseteq \mathcal{U}^j \subseteq \mathcal{Z}^j. \tag{10}$$

Based on the above bounded delay assumption, we show the linear convergence rate of our algorithm in the following theorem:

Theorem 1. *Assume a function $F(\boldsymbol{z})$ in the family defined by (1) admits a global error bound from the beginning. Assume there is a constant $\bar{L}$ such that all f_i are $\bar{L}$-Lipschitz*

[2] As shown in [9], the third approach—locking the variables before access—is even slower than the serial implementation, so we omit the discussion here.

Algorithm 1 Single variable update: $T_i(\boldsymbol{r}, z_i)$

Input: selected index i, current z_i and $\boldsymbol{r}$

1: Obtain u^* by solving (6) with the required first/second order information computed from $\boldsymbol{r}$.
2: $\Delta z_i \leftarrow u^* - z_i$
3: $z_i \leftarrow u^*$
4: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \Delta z_i \boldsymbol{q}_i$

continuous. If the upper bound of the staleness τ is small enough such that the following two conditions hold:

$$(3(\tau+1)^2 e\bar{M})/\sqrt{n} \le 1, \quad \textit{and} \quad \frac{2Lc_0}{\sigma(1-2c_0)} \le 1, \tag{11}$$

where $\bar{M} = \sigma^{-1}L\left(R_{max}^2 + 2R_{max}\right)$, *and* $c_0 = \frac{\tau^2\sigma^{-2}L^2R_{max}^2e^2}{n}$ *then Algorithm 2 with atomic operations has a global linear convergence rate in expectation, that is, there is a constant $c_2 > 0$ and $\eta < 1$ such that*

$$\begin{aligned}&\mathbb{E}\left[\!\left[F(\boldsymbol{z}^{j+1}) - F^* + c_2\left\|T(\boldsymbol{z}^j) - \boldsymbol{z}^j\right\|\right]\!\right]\\ \le&\eta\left(\mathbb{E}\left[\!\left[F(\boldsymbol{z}^j) - F^* + c_2\left\|T(\boldsymbol{z}^{j-1}) - \boldsymbol{z}^{j-1}\right\|\right]\!\right]\right),\end{aligned} \tag{12}$$

where F^* is the minimum of $F(\boldsymbol{z})$.

See Appendix C for a detailed proof for Theorem 1 and Appendix B for a simplified version with a smooth assumption on the entire $F(\cdot)$. Note that c_2 is a numerical constant that depends on the problem. L is the Lipschitz constant for $d_k(\cdot)$ and σ is strong convexity constant for the single variable problem (5). $\bar{M}$ plays a similar role as the condition number of the entire problem. It is not hard to see that condition (11) can be satisfied when τ is small enough or n is large enough.

We highlight some differences of our results and proofs to the ones in [9]:

- We have established convergence for a rich family of problems including both smooth and non-smooth functions, while the analysis in [9] only applies to the dual form of ℓ_2-regularized problems.
- We derive an upper bound for the expansion for the operator $T_i(\boldsymbol{r}, s)$ defined in (6) for general f_i and d_k. In the dual formulation of ℓ_2-regularized ERM, this operator is equivalent to a proximal operator so non-expansiveness is guaranteed.

4.2 Backward Error Analysis for AUX-PCD with Wild Updates

Next we discuss the behavior of Algorithm 2 without atomic operations to maintain auxiliary variables. It was observed that AUX-PCD with wild updates achieves the best scaling performance in [9], but since the writes can be overwritten, the algorithm fails to maintain the relationship $\boldsymbol{r}^j = Q\boldsymbol{z}^j$. Thus it cannot converge to the exact solution of the original problem. However, intuitively if the ratio of memory conflicts is low, AUX-PCD with wild updates should still converge to a "reasonable" solution. This is observed in practice, and here we provide a *complete backward error analysis* to formally characterize the convergence properties of the wild variant of AUX-PCD. Our backward error analysis for AUX-PCD with wild updates includes three parts:

Algorithm 2 AUX-PCD: Parallel Stochastic Coordinate Descent with Auxiliary Variables

Input: Initialize $\boldsymbol{z}^0$ and $\boldsymbol{r} = \sum_{i=1}^n z_i^0 \boldsymbol{q}_i$

1: Each thread repeatedly performs the following updates *in parallel*:
2: step 1: Randomly select i
3: step 2: Run Algorithm 1 to update z_i and maintain $\boldsymbol{r}$

- **B1:** model the source of *errors* ,
- **B2:** prove that the output of the algorithm is the *exact* solution of a "perturbed" problem when the errors are bounded, and
- **B3:** derive the bound on the errors accumulated in the algorithm.

B1 (Modeling the Source of Errors): For simplicity, we consider the situation where F is smooth and assume that all the write operations in Algorithm 1 can be overwritten by another thread with the same probability θ. Therefore,

$$r_k^{j+1} = r_k^j + Q_{k,i(j)}\Delta z_j \delta_k^j \ \ \forall k$$

where δ_k^j is a random variable with probability θ to be 0 and $1-\theta$ to be 1. Note that the missing rate is usually very small ($\theta \ll 1$). For example, consider the perfect case where nonzero features are distributed uniformly in the entire dataset (i.e., $\|\boldsymbol{q}_0\|_0 = \cdots = \|\boldsymbol{q}_d\|_0$) and two threads independently pick (k_1, i_1), (k_2, i_2) and conduct the update $r_{k_1} \leftarrow r_{k_1} + \delta_1 Q_{k_1,i_1}$ and $r_{k_2} \leftarrow r_{k_2} + \delta_2 Q_{k_2,i_2}$ simultaneously. The conflict write happens only when $k_1 = k_2$, which happens with probability $1/m$. This suggests that for ERM problems (2) the confliction rate decreases as the number of samples increases. In the left plot of Figure 1, we perform the experiments on the dataset TFIDF-2006 with various duplicated copies. AUX-PCD (known as CDN in the Lasso literature) with wild updates converges to a more close solution as the number of duplications increases, which confirms that the conflict rate θ is inverse proportional to m, which is the number of instances in (2).

Following the notation in (8), we define the set $\hat{\mathcal{Z}}^j$ to be all the updates that are not missing until step j:

$$\hat{\mathcal{Z}}^j = \{(t,k) \mid t < j, k \in N(i(t)), \delta_k^t = 1\} \subseteq \mathcal{Z}^j,$$

and the updates in this set will be eventually done (with a delay up to τ), so

$$\hat{\mathcal{Z}}^{j-\tau} \subseteq \mathcal{U}^j \subseteq \hat{\mathcal{Z}}^j.$$

Now $\boldsymbol{r}^j$ defined in (9) will not be equal to $Q\boldsymbol{z}^j$. The $\boldsymbol{\epsilon}^j$ is defined to be the error caused by missing updates, where

$$\boldsymbol{\epsilon}^j = Q\boldsymbol{z}^j - \boldsymbol{r}^j.$$

B2. (Exact Solution to a Perturbed Problem.) Assume $\boldsymbol{z}^j$ converges to $\bar{\boldsymbol{z}}$, which may not be the optimal solution for $F(\boldsymbol{z})$. However, we show $\bar{\boldsymbol{z}}$ is the solution of a perturbed problem if $\boldsymbol{\epsilon}^j$ converges to some vector $\boldsymbol{\epsilon}^\infty$:

Theorem 2. *If $\{\boldsymbol{\epsilon}^j\}$ converges to $\boldsymbol{\epsilon}^\infty$ and $\bar{\boldsymbol{z}}$ is a limit point of $\{\boldsymbol{z}^j\}$, then*

- *$\bar{\boldsymbol{z}}$ is a minimizer of the following "perturbed" problem:*

$$\operatorname*{argmin}_{\boldsymbol{z}} \Big\{ \sum_{i=1}^{n} f_i(z_i) + D(Q\boldsymbol{z} - \boldsymbol{\epsilon}^\infty) \Big\} := F^\infty(\boldsymbol{z})$$

- *Furthermore, the distance between real and computed solution can be bounded by*

$$\|\bar{\boldsymbol{z}} - \boldsymbol{z}^*\| \le \kappa\sigma^{-1}L\|\boldsymbol{\epsilon}^\infty\|,$$

where $\boldsymbol{z}^*$ is the optimal solution for the original objective function F.

Note that the only difference between perturbed problem F^∞ and original problem F is the argument of the second part, and when $\boldsymbol{\epsilon}^\infty$ is small they will be similar. The derivation for the Wild algorithm in [9] is a special case of this analysis; however, they fail to show the existence and boundedness of $\boldsymbol{\epsilon}^\infty$, which makes the story incomplete.

B3. (Bounding the Errors) In the following, we complete the last part of backward error analysis by showing $\boldsymbol{\epsilon}^\infty$ exists and is bounded, which is surprisingly a difficult task. To our knowledge there is no previous analysis for doing this. Our idea is based on a new notion of "linear convergence" for this process. Note that due to lost updates, the objective function is also perturbed at each iteration. We define the perturbed objective function at j-th iterate based on the current error $\boldsymbol{\epsilon}^j$: $F^j(\boldsymbol{z}) := \sum_{i=1}^n f_i(z_i) = D(Q\boldsymbol{z} - \boldsymbol{\epsilon}^j)$. Theorem 3 establishes a new type of linear convergence:

Theorem 3. *Let $\boldsymbol{z}^{*(j+1)}, \boldsymbol{z}^{*(j)}$ be the optimal solution for F^{j+1}, F^j respectively. Assume F is L_{max}-smooth and σ_{min} strongly convex. Assume the conditions (11) in Theorem 1 hold. There exists $\bar{\eta} < 1$ such that the sequence $\{\boldsymbol{z}^j\}$ generated by the wild algorithm satisfies*

$$\mathbb{E}\Big[F^{j+1}(\boldsymbol{z}^{j+1}) - F^{j+1}(\boldsymbol{z}^{*(j+1)})\Big] \le \bar{\eta}\mathbb{E}\Big[F^j(\boldsymbol{z}^j) - F^j(\boldsymbol{z}^{*(j)})\Big], \tag{13}$$

if $\bar{c}_0 + \theta\bar{c}_1 < 1$, where $\bar{c}_0$ and $\bar{c}_2$ are positive numerical constants, and θ is the conflict rate defined earlier (See Appendix D.3 for details).

With the linear convergence stated in Theorem 3, we can show that $\{\boldsymbol{\epsilon}^j\}$ converges.

Theorem 4. *If (13) holds, then $\{\boldsymbol{\epsilon}^j\}$ converges. In particular, we have*

$$\mathbb{E}[\|\boldsymbol{\epsilon}^\infty\|] \le \theta R_{max}\sqrt{F(\boldsymbol{z}^0) - F^*}\Big(\frac{1}{1-\sqrt{\bar{\eta}}}\Big)\sqrt{\frac{1}{2\sigma}},$$

where $\bar{\eta} \in (0,1)$ is the linear convergence rate in Theorem 3.

5 Experimental Results

The superior performance of the parallel asynchronous coordinate descent scheme has been demonstrated for the dual form of ℓ_2-regularized ERM problems [9]. In this paper, we are interested in the empirical performance for the proposed approach for the primal formulation of ERM with non-smooth regularization. In particular, we conduct experiments for Lasso and ℓ_1-regularized logistic regression.

We consider three datasets: kddb, webspam for classification and TFIDF-2006 for regression, and kddb. See Table 1 for the detailed information. All the experiments are performed on an Intel multi-core dual-socket machine with 20 cores and 256 GB memory. In particular, the machine are equipped with two Intel Xeon E5-2680 v2 Ivy Bridge sockets with 10 CPUs cores per socket.

5.1 ℓ_1-regularized Least Squares (Lasso)

Compared Methods

- parallel CDN: a parallel version of CDN [26] for Lasso problem. It is well-known as Shotgun [3], which is possibly the most cited work for parallel coordinate descent. Although the auxiliary variable maintenance is not mentioned explicitly in that paper, the Shotgun package implements CDN with residual maintenance. However, Shotgun does not include random coordinate selection in its Lasso implementation, which might lead to slower convergence. To have a fair comparison, we implement parallel CDN with random coordinate selection. As mentioned before, there will be two variants of parallel CDN: one with *atomic* operations (which is also used in Shotgun implementation) and the other one with *wild* updates.
- Thread-Greedy-b: a parallel thread-greedy coordinate descent proposed in [19]. As there is no publicly available code, we implement this method following the description in [20]. We also adopt the technique proposed in [10] to reduce the step size to 1/#threads to improve convergence.

Our results on both TFIDF-2006 and kddb are shown in Figure 1. The suffix digit of each legend denotes the number of threads used for the corresponding method. The result of serial CDN is also included for reference. The y-axis

Table 1: Data statistics. #nonzero per feature is corresponding to the $\|\boldsymbol{q}_i\|_0$, which is the average time cost per coordinate update for the primal formulation of ERM problems.

	# instances		# features	avg #nonzero	avg #nonzero	range of
	training	test		per instance	per feature	$\boldsymbol{y}$
TFIDF-2006	16,087	3,308	150,360	1241.4	132.8	$[-7.90, -0.52]$
kddb	19,264,097	748,401	29,890,095	29.4	18.9	$\{0, 1\}$
webspam	315,000	35,000	16,609,143	3727.7	62.84	$\{0, 1\}$

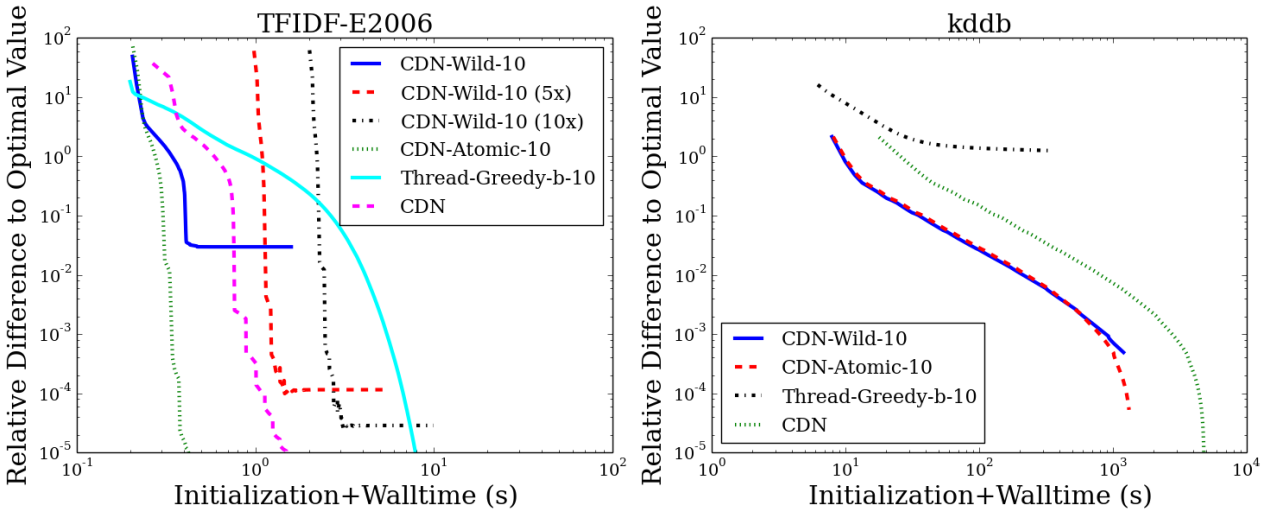

Figure 1: Experimental results for Lasso. Both x-axis and y-axis are in the log scale.

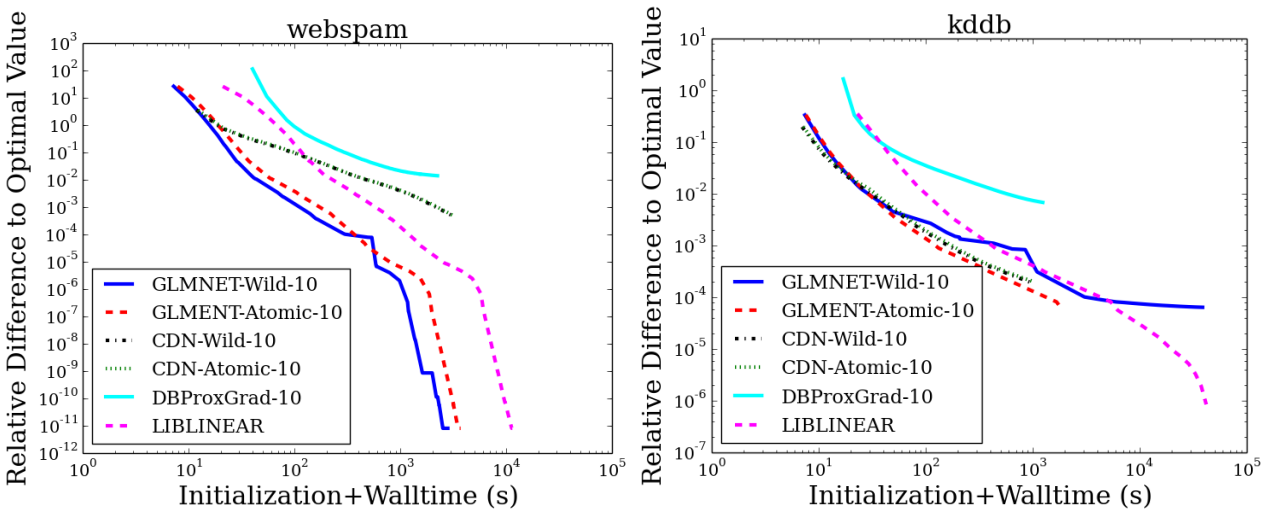

Figure 2: Experimental results for L1-regularized Logistic Regression. Both x-axis and y-axis are in the log scale.

is log-scale of the relative difference to the optimal objective function value, $(F(\boldsymbol{z}) - F(\boldsymbol{z}^*))/F(\boldsymbol{z}^*)$, and x-axis is the computation time in log scale. We have the following observations:

- CDN-Atomic is an order-of-magnitude faster than Thread-Greedy on both datasets. Although it is contrary to the finding in [19], it can be explained by the stochastic selection of coordinate.
- Without the atomic operations, some updates of auxiliary variables of CDN-Wild might be missing. If the number of missing updates is too large, the converged solution might be different, which can be observed on the TFIDF-2006 dataset.
- As discussed in Section 4.2, if all the nonzero entires are uniformly distributed over n features, the conflict rate is $1/m$ where m is number of training samples. To verify this, we duplicate the dataset by 5 times and 10 times (5x, 10x), and we observe that CDN-Wild converges to a much better solution when we increase the number of samples, indicating a smaller conflict rate.

5.2 ℓ_1-regularized logistic regression (L1LR)

Compared Methods.

- CDN-Wild and CDN-Atomic: similar to the Lasso experiments, we include both parallel versions of CDN for ℓ_1-regularized logistic regression. Note that CDN-Atomic will be the same as the L1-regularized logistic regression implementation in Shotgun.
- Parallel newGLMNET: As mentioned in Section A, AUX-PCD can be applied to parallelize a second-order method. Here, we embed both CDN-Wild and CDN-Atomic in newGLMNET, which is a proximal Newton method for L1LR [27].
- Delayed Block Proximal Gradient (DBProxGrad): this is an asynchronous proximal gradient descent method for L1LR proposed in [13]. Although it is designed for the distributed computation, it is shown to be significantly faster than other parallel solvers for L1LR[3]. We use the public code available in `https://github.com/dmlc/parameter_server` to generate the results for DBProxGrad. As this package is not designed for the multi-core setting, it would utilize all the computational resource in the single machine. We follow the suggestion from the author [12] to use 10 servers and 10 workers on the single machine we used.

Figure 2 shows the results for kddb and webspam. We only report the result of DBProxGrad on kddb as there are some issues for the parameter sever implementation to generate the results for webspam. The y-axis is the relative difference the optimal solution in log-scale, while the x-axis is the computation time in log scale. We make the following observations:

- GLMNET-Wild seems to converge to a similar solution as other solvers. It is expected as the quadratic approximation is re-constructed for each Newton iteration, so the errors due to the wild updates are not accumulated as CDN-Wild does for Lasso.
- On webspam, both GLMNET-Wild and GLMNET-Atomic are faster than parallel CDN. The gap becomes more significant as the time proceeds. This can be explained by the faster final convergence behavior of GLMNET, which is a second-order method.
- On kddb, both parallel versions of CDN have similar performance as GLMNET-Atomic. GLMNET-Wild is slight slower than GLMNET-Atomic in the later stage. We found that this is due to the lengthy inner CDN iterations in GLMNET-Wild, which can be explained by CDN-Wild not meeting the inner stopping condition of GLMNET. We plan to investigate this phenomenon in the future.
- As DBProxGrad adopts proximal operator with a fix-step size without any line search, final convergence of DBProxGrad is not stable, which can be observed on kddb. Apart from that DBProxGrad is slower than other parallel methods on this multi-core setting.

6 Conclusions

In this paper, we consider a family of objective functions where parallel coordinate descent with auxiliary variables (AUX-PCD) can be applied. We also establish theoretical guarantees for AUX-PCD. With atomic operations, we provide global linear convergence of AUX-PCD for a rich family of objective functions, while for AUX-PCD with wild updates, we develop backward error analysis to analyze the convergence behavior. Our results provide theoretical guarantees for many practical parallel coordinate descent implementations (such as the implementation of Shotgun [3] which uses auxiliary variables).

Acknowledgement

This research was partially supported by NSF grants IIS-1546452 and CCF-1564000. CJH was supported by NSF grant RI-1719097, Intel and AITRICKS.

References

[1] H. Avron, A. Druinsky, and A. Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. In *IEEE International Parallel and Distributed Processing Symposium*, 2014.

[3] See Appendix B of [14]

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[3] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML)*, pages 321–328, 2011.

[4] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale L2-loss linear SVM. *JMLR*, 9:1369–1398, 2008.

[5] O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[6] J. H. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.

[7] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

[8] C.-J. Hsieh, M. A. Sustik, P. Ravikumar, and I. S. Dhillon. Sparse inverse covariance matrix estimation using quadratic approximation. In *Advances in Neural Information Processing Systems (NIPS) 24*, 2011.

[9] C.-J. Hsieh, H. F. Yu, and I. S. Dhillon. PASSCoDe: Parallel ASynchronous Stochastic dual Coordinate Descent. In *International Conference on Machine Learning(ICML),*, 2015.

[10] M. Jaggi, V. Smith, M. Takáč, J. Terhorst, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *NIPS*. 2014.

[11] C.-P. Lee and D. Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *ICML*, 2015.

[12] M. Li, 2015. Private communication.

[13] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[14] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661–670, 2014.

[15] J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. 2014.

[16] J. Liu, S. J. Wright, C. Re, and V. Bittorf. An asynchronous parallel stochastic coordinate descent algorithm. In *ICML*, 2014.

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

[18] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Math. Program.*, 2012. Under revision.

[19] C. Scherrer, M. Halappanavar, A. Tewari, and D. Haglin. Scaling up coordinate descent algorithms for large l_1 regularization problems. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1407–1414, 2012.

[20] C. Scherrer, A. Tewari, M. Halappanavar, and D. Haglin. Feature clustering for accelerating parallel coordinate descent. In *NIPS*, 2012.

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

[22] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, 2013.

[23] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, 2012.

[24] H. F. Yu, C. J. Hsieh, S. Si, and I. S. Dhillon. Parallel matrix factorization for recommender systems. In *Knowledge and Information Systems(KAIS)*, May 2013.

[25] H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, October 2011.

[26] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR*, 11:3183–3234, 2010.

[27] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved GLMNET for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.