PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent

Cho-Jui Hsieh  
Hsiang-Fu Yu  
Inderjit S. Dhillon  
CJHSIEH@CS.UTEXAS.EDU  
ROFYU@CS.UTEXAS.EDU  
INDERJIT@CS.UTEXAS.EDU  
Department of Computer Science, The University of Texas, Austin, TX 78721, USA

Abstract

Stochastic Dual Coordinate Descent (DCD) is one of the most efficient ways to solve the family of $\ell_2$-regularized empirical risk minimization problems, including linear SVM, logistic regression, and many others. The vanilla implementation of DCD is quite slow; however, by maintaining primal variables while updating dual variables, the time complexity of DCD can be significantly reduced. Such a strategy forms the core algorithm in the widely-used LIBLINEAR package. In this paper, we parallelize the DCD algorithms in LIBLINEAR. In recent research, several synchronized parallel DCD algorithms have been proposed, however, they fail to achieve good speedup in the shared memory multi-core setting. In this paper, we propose a family of parallel asynchronous stochastic dual coordinate descent algorithms (PASSCoDe). Each thread repeatedly selects a random dual variable and conducts coordinate updates using the primal variables that are stored in the shared memory. We analyze the convergence properties of DCD when different locking/atomic mechanisms are applied. For implementation with atomic operations, we show linear convergence under mild conditions. For implementation without any atomic operations or locking, we present a novel error analysis for PASSCoDe under the multi-core environment, showing that the converged solution is the exact solution for a primal problem with a perturbed regularizer. Experimental results show that our methods are much faster than previous parallel coordinate descent solvers.

1. Introduction

Given a set of instance-label pairs $(\hat{x}_i, \hat{y}_i)$, $i = 1, \cdots, n$, $\hat{x}_i \in \mathbb{R}^d$, $\hat{y}_i \in \mathbb{R}$, we focus on the following empirical risk minimization problem with $\ell_2$-regularization:

$$\min_{\w \in \mathbb{R}^d} P(\w) := \frac{1}{2} \|\w\|^2 + \sum_{i=1}^n \ell_i(\w^T \x_i),$$

where $\x_i = \hat{y}_i \hat{x}_i$, $\ell_i(\cdot)$ is the loss function and $\| \cdot \|$ is the 2-norm. A large class of machine learning problems can be formulated as the above optimization problem. Examples include Support Vector Machines (SVMs), logistic regression, ridge regression, and many others. Problem (1) is usually called the primal problem, and can be solved by Stochastic Gradient Descent (SGD) (Zhang, 2004; Shalev-Shwartz et al., 2007), second order methods (Lin et al., 2007), or primal Coordinate Descent (CD) algorithms (Chang et al., 2008; Huang et al., 2010).

Instead of solving the primal problem, another class of algorithms solves the following dual problem of (1):

$$\min_{\alpha \in \mathbb{R}^n} D(\alpha) := \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i \x_i \right\|^2 + \sum_{i=1}^n \ell^*_i(-\alpha_i),$$

where $\ell^*_i(\cdot)$ is the conjugate of the loss function $\ell_i(\cdot)$, defined by $\ell^*_i(u) = \max_z (zu - \ell_i(z))$. If we define

$$\w(\alpha) = \sum_{i=1}^n \alpha_i \x_i,$$

then it is known that $\w(\alpha^*) = \w^*$ and $P(\w^*) = -D(\alpha^*)$ where $\w^*$, $\alpha^*$ are the optimal primal/dual solutions respectively. Examples include hinge-loss SVM, square hinge SVM and $\ell_2$-regularized logistic regression.

Stochastic Dual Coordinate Descent (DCD) has become the most widely-used algorithm for solving (2), and it is faster than primal solvers (including SGD) in many large-scale problems. The success of DCD is mainly due to the trick of maintaining the primal variables $\w$ based on the primal-dual relationship (3). By maintaining $\w$ in memory, (Hsieh et al., 2008; Keerthi et al., 2008) showed that the time complexity of each coordinate update can be reduced from $O(\text{nznz})$ to $O(\text{nznz}/n)$, where nznz is the number of nonzeros in the training dataset. Several DCD algorithms for different machine learning problems are currently implemented in LIBLINEAR (Fan et al., 2008) which is now widely used in both academia and industry. The success of DCD has also catalyzed a large body of theoretical studies (Nesterov, 2012; Shalev-Shwartz & Zhang, 2013).
In this paper, we parallelize the DCD algorithm in a shared memory multi-core system. There are two threads of work on parallel coordinate descent. The focus of the first thread is synchronized algorithms, including synchronized CD (Richtárik & Takáč, 2012; Bradley et al., 2011) and synchronized DCD algorithms (Pechyony et al., 2011; Yang, 2013; Jaggi et al., 2014). However, choosing the block size is a trade-off problem between communication and convergence speed, so synchronized algorithms usually suffer from slower convergence. To overcome this problem, the other thread of work focuses on asynchronous CD algorithms in multi-core shared memory systems (Liu & Wright, 2014; Liu et al., 2014). However, none of the existing asynchronous CD algorithms maintains both the primal and dual variables. As a result, the recent asynchronous CD algorithms end up being much slower than the state-of-the-art serial DCD algorithms that maintain both $w$ and $\alpha$, as in the LIBLINEAR software. This leads to a challenging question: how to maintain both primal and dual variables in an asynchronous and efficient way?

In this paper, we propose the first asynchronous dual coordinate descent (PASSCoDe) algorithm which addresses the issue of primal variable maintenance in the shared memory multi-core setting. We carefully discuss and analyze three versions of PASSCoDe: PASSCoDe-Lock, PASSCoDe-Atomic, and PASSCoDe-Wild. In PASSCoDe-Lock, convergence is always guaranteed but the overhead for locking makes it even slower than serial DCD. In PASSCoDe-Atomic, the primal-dual relationship (3) is enforced by atomic writes to the shared memory; while PASSCoDe-Wild proceeds without any locking and atomic operations, as a result of which the relationship (3) between primal and dual variables can be violated due to memory conflicts. Our contributions can be summarized below:

- We propose and analyze a family of asynchronous parallelization of the most efficient DCD algorithm: PASSCoDe-Lock, PASSCoDe-Atomic, PASSCoDe-Wild.
- We show linear convergence of PASSCoDe-Atomic under certain conditions.
- We present an error analysis for PASSCoDe-Wild and show that the converged solution is the exact solution of a primal problem with a perturbed regularizer. Therefore the performance is close-to-optimal on most datasets. To best of our knowledge, this is the first attempt to analyze a parallel machine learning algorithm with memory conflicts using backward error analysis, which is a standard tool in numerical analysis (Wilkinson, 1961).
- Experimental results show that our algorithms (PASSCoDe-Atomic and PASSCoDe-Wild) are much faster than existing methods. For example, on the webspam dataset, PASSCoDe-Atomic took 2 seconds and PASSCoDe-Wild took 1.6 seconds to achieve 99% accuracy using 10 threads, while CoCoA (Jaggi et al., 2014) took 11.5 seconds using the same number of threads (LIBLINEAR took 10 seconds using 1 thread to achieve the same accuracy).

2. Related Work

Stochastic Coordinate Descent. Coordinate descent is a classical optimization technique that has been well studied (Bertsekas, 1999; Luo & Tseng, 1992). Recently it has enjoyed renewed interest due to the success of “stochastic” coordinate descent in real applications (Hsieh et al., 2008; Nesterov, 2012). In terms of theoretical analysis, the convergence of (cyclic) coordinate descent has been studied for a long time (Luo & Tseng, 1992; Bertsekas, 1999), while recently (Beck & Tetruashvili, 2013; Wang & Lin, 2014) showed global linear convergence under certain conditions.

Stochastic Dual Coordinate Descent. Many papers (Hsieh et al., 2008; Yu et al., 2011; Shalev-Shwartz & Zhang, 2013) have shown that solving the dual problem using coordinate descent algorithms is faster on large-scale datasets. The success of DCD strongly relies on exploiting the primal-dual relationship (3) to speed up the gradient computation in the dual space. DCD has become the state-of-the-art solver implemented in LIBLINEAR (Fan et al., 2008). In terms of convergence of the dual objective function, standard theoretical guarantees for coordinate descent can be directly applied. Taking a different approach, (Shalev-Shwartz & Zhang, 2013) presented the convergence rate in terms of the duality gap.

Parallel Stochastic Coordinate Descent. In order to conduct coordinate updates in parallel, (Richtárik & Takáč, 2012) studied an algorithm where each processor updates a randomly selected block (or coordinate) simultaneously, and (Bradley et al., 2011) proposed a similar algorithm for $\ell_1$-regularized problems. (Scherrer et al., 2012) studied parallel greedy coordinate descent. However, the above synchronized methods usually face a trade-off in choosing the block size. If the block size is small, the load balancing problem leads to slow running time. If the block size is large, the convergence speed becomes much slower (the algorithm can even diverges). These problems can be resolved by developing an asynchronous algorithm. Asynchronous coordinate descent has been studied by (Bertsekas & Tsitsiklis, 1989), but they require the Hessian to be diagonal dominant in order to establish convergence. Recently, (Liu et al., 2014; Liu & Wright, 2014) proved linear convergence of asynchronous stochastic coordinate descent algorithms under the essential strong convexity condition and a “bounded staleness” condition, where they consider both consistent and inconsistent read models. (Avron et al., 2014) showed linear rate of convergence for the asynchronous randomized Gauss-Seidel updates, which is a special case of coordinate descent on linear systems.

Parallel Stochastic Dual Coordinate Descent. For solving the one variable subproblem (Eq. (5)) in dual coordinate
descent, each coordinate updates only requires the global primal variable vector \( w \) and one local dual variable \( \alpha_i \), thus algorithms only need to synchronize \( w \). Based on this observation, (Yang, 2013) proposed to update several coordinates or blocks simultaneously and update the global \( w \), while (Jaggi et al., 2014) showed that each block can be solved with other approaches under the same framework. However, both these parallel DCD methods are synchronized algorithms.

To the best of our knowledge, this paper is the first to propose and analyze asynchronous parallel stochastic dual coordinate descent methods. By maintaining a primal solution \( w \) while updating dual variables, our algorithm is much faster than the previous asynchronous coordinate descent methods of (Liu & Wright, 2014; Liu et al., 2014) for solving the dual problem (2). Our algorithms are also faster than synchronized dual coordinate descent methods (Pechyony et al., 2011; Yang, 2013; Jaggi et al., 2014) since the latest value of \( w \) can be accessed by all the threads. In terms of theoretical contribution, the inconsistent read model in (Liu & Wright, 2014) cannot be directly applied to our algorithm because each update on \( \alpha_i \) is based on the shared \( w \) vector. We show linear convergence for PASSCoDe-Atomic, and study the properties of the converged solution for the wild version of our algorithm (without any locking and atomic operations). Our algorithm has been successfully applied to solve the collaborative ranking problem (Park et al., 2015).

3. Algorithms

3.1. Stochastic Dual Coordinate Descent

We first describe the Stochastic Dual Coordinate Descent (DCD) algorithm for solving the dual problem (2). At each iteration, DCD randomly picks a dual variable \( \alpha_i \) and updates it by minimizing the one variable subproblem \( \arg \min \delta_i D(\alpha + \delta e_i) \). Without exploiting the structure of the quadratic term, solving each subproblem requires \( O(\text{nnz}) \) operations, where \( \text{nnz} \) is the total number of nonzero elements in the training data, which can be substantial. However, if \( w(\alpha) \) that satisfies (3) is maintained in memory, the subproblem \( D(\alpha + \delta e_i) \) can be written as

\[
D(\alpha + \delta e_i) = \frac{1}{2} \|w + \delta x_i\|^2 + \ell_i^*(-(\alpha_i + \delta)),
\]

and the optimal solution can be computed by

\[
\delta^* = \arg \min_{\delta} \frac{1}{2} \left( \|w + \delta x_i\|^2 + \frac{1}{\|x_i\|^2} \ell_i^*(-(\alpha_i + \delta)) \right).
\]

Note that all \( \|x_i\| \) can be pre-computed and regarded as constants. For each coordinate update we only need to solve a simple one-variable subproblem, and the main computation is in computing \( w^T x_i \), which requires \( O(\text{nnz}/n) \) time. For SVM problems, the above subproblem has a closed form solution, while for logistic regression problems it has to be solved by an iterative solver (see (Yu et al., 2011) for details). The DCD algorithm, which is part of the popular LIBLINEAR package, is described in Algorithm 1.

**Algorithm 1** Stochastic Dual Coordinate Descent (DCD)

**Input:** Initial \( \alpha \) and \( w = \sum_{i=1}^n \alpha_i x_i \)

1: while not converged do
2: Randomly pick \( i \)
3: Update \( \alpha_i \leftarrow \alpha_i + \Delta \alpha_i \), where
   \[
   \Delta \alpha_i \leftarrow \arg \min_{\delta} \frac{1}{2} \|w + \delta x_i\|^2 + \ell_i^*(-(\alpha_i + \delta))
   \]
4: Update \( w \) by \( w \leftarrow w + \Delta \alpha_i x_i \)
5: end while

3.2. Asynchronous Stochastic Dual Coordinate Descent

To parallelize DCD in a shared memory multi-core system, we propose a family of Parallel ASynchronous Stochastic dual Co-ordinate Descent (PASSCoDe) algorithms. PASSCoDe is very simple but effective. Each thread repeatedly run the updates (steps 2 to 4) in Algorithm 1 using the latest \( w \) and \( \alpha \) stored in a shared memory. The threads do not need to coordinate or synchronize their iterations. The details are shown in Algorithm 2.

Although PASSCoDe is a simple extension of DCD in the shared memory setting, there are many options in terms of locking/atomic operations for each step, and these choices lead to variations in speed and convergence properties, as we will show in this paper.

**Algorithm 2** Parallel Asynchronous Stochastic dual Co-ordinate Descent (PASSCoDe)

**Input:** Initial \( \alpha \) and \( w = \sum_{i=1}^n \alpha_i x_i \)

Each thread repeatedly performs the following updates:

- step 1: Randomly pick \( i \)
- step 2: Update \( \alpha_i \leftarrow \alpha_i + \Delta \alpha_i \), where
  \[
  \Delta \alpha_i \leftarrow \arg \min_{\delta} \frac{1}{2} \|w + \delta x_i\|^2 + \ell_i^*(-(\alpha_i + \delta))
  \]
- step 3: Update \( w \) by \( w \leftarrow w + \Delta \alpha_i x_i \)

Note that the \( \Delta \alpha_i \) obtained by subproblem (5) is exactly the same as (4) in Algorithm 1 if only one thread is involved. However, when there are multiple threads, the \( w \) vector may not be the latest one since some other threads may not have completed the writes in step 3. We now present three variants of DCD.

PASSCoDe-Lock. To ensure \( w = \sum_i \alpha_i x_i \) for the latest \( \alpha \), we have to lock the “active” variables between step 1 and 2 in Algorithm 2:

- step 1.5: lock variables in \( N_i := \{ w_i \mid (x_i)_i \neq 0 \} \).

The locks are then released after step 3. With this locking mechanism, PASSCoDe-Lock is serializable, i.e., there
Table 1. Scaling of PASSCoDe algorithms. We present the run time (in seconds) for each algorithm on the rcv1 dataset with 100 sweeps, and the speedup of each method over the serial DCD algorithm (2x means it is two times faster than the serial algorithm).

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Lock</th>
<th>Atomic</th>
<th>Wild</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>98.05s / 0.27x</td>
<td>15.28s / 1.75x</td>
<td>14.08s / 1.90x</td>
</tr>
<tr>
<td>4</td>
<td>106.11s / 0.25x</td>
<td>8.35s / 3.20x</td>
<td>7.61s / 3.50x</td>
</tr>
<tr>
<td>10</td>
<td>114.35s / 0.25x</td>
<td>3.86s / 8.91x</td>
<td>3.39s / 7.45x</td>
</tr>
</tbody>
</table>

Table 2. The performance of PASSCoDe-Wild using \( \hat{w} \) or \( \hat{w} \) for prediction. Results show that \( \hat{w} \) yields much better prediction accuracy, which justifies our theoretical analysis in Section 4.2.

<table>
<thead>
<tr>
<th># threads</th>
<th>( \hat{w} ) Prediction Accuracy (%)</th>
<th>( \hat{w} ) Prediction Accuracy (%)</th>
<th>LIBLINEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>news20</td>
<td>97.1 / 96.4</td>
<td>97.1 / 96.4</td>
<td>97.1</td>
</tr>
<tr>
<td>4</td>
<td>97.2 / 93.3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>97.2 / 93.3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>covtype</td>
<td>67.8 / 38.0</td>
<td>67.8 / 38.0</td>
<td>66.3</td>
</tr>
<tr>
<td>4</td>
<td>97.7 / 97.5</td>
<td>97.7 / 97.4</td>
<td>97.7</td>
</tr>
<tr>
<td>8</td>
<td>97.7 / 97.4</td>
<td>97.7 / 97.4</td>
<td>97.4</td>
</tr>
<tr>
<td>rcv1</td>
<td>99.1 / 93.1</td>
<td>99.1 / 93.1</td>
<td>99.1</td>
</tr>
<tr>
<td>4</td>
<td>89.8 / 87.7</td>
<td>89.8 / 87.7</td>
<td>88.8</td>
</tr>
<tr>
<td>8</td>
<td>89.8 / 87.7</td>
<td>89.8 / 87.7</td>
<td>88.8</td>
</tr>
<tr>
<td>webspam</td>
<td>89.8 / 87.7</td>
<td>89.8 / 87.7</td>
<td>88.8</td>
</tr>
<tr>
<td>kddb</td>
<td>89.8 / 87.7</td>
<td>89.8 / 87.7</td>
<td>88.8</td>
</tr>
</tbody>
</table>

is a sequence update such that serial DCD generates the same solution as PASSCoDe-Lock. Unfortunately, threads will spend too much time to update due to the locks, so PASSCoDe-Lock is very slow compared to the non-locking version (and even slower than the serial version of DCD). See Table 1 for details.

PASSCoDe-Atomic. The above locking scheme is to ensure that each thread updates \( \alpha_i \) based on the latest \( w \) values. However, as shown in (Niu et al., 2011; Liu & Wright, 2014), the effect of using slightly stale values is usually small in practice. Therefore, we propose an atomic algorithm called PASSCoDe-Atomic that avoids locking all the variables in \( N_i \) simultaneously. Instead, each thread just reads the current \( w \) values from memory without any locking. In practice (see Section 5) we observe that the convergence speed is not significantly affected by using these “unlocked” values of \( w \). However, to ensure that the limit point of the algorithm is still the global optimizer of (1), the equation \( w^* = \sum \alpha_i^* x_i \) has to be maintained. Therefore, we apply the following “atomic writes” in step 3:

step 3: For each \( j \in N(i) \)

Update \( w_j \leftarrow w_j + \Delta \alpha_i(x)_{ij} \) atomically

PASSCoDe-Atomic is much faster than PASSCoDe-Lock as shown in Table 1 since the atomic write for a single variable is much faster than locking all the variables. However, the convergence of PASSCoDe-Atomic cannot be guaranteed by tools used in the past, but we observe empirical convergence. To bridge this gap between theory and practice, we prove linear convergence of PASSCoDe-Atomic under certain conditions in Section 4.
The update \( \Delta \) component selected at the PASSCoDe-Lock \( \sum \) be the sequence generated by our algorithms, and \( 2 \) of Algorithm 2, and the index \( i \) the Appendix. We assign a global counter \( \text{DCD} \) ical core and allocate data in its local memory. Note that the current OpenMP does not support this functionality for thread affinity; a library such as libnuma can be used to enforce thread affinity.

4. Convergence Analysis

In this section we formally analyze the convergence properties of PASSCoDe. Note that all the proofs can be found in the Appendix. We assign a global counter \( j \) for all the asynchronous updates based on the time of completion of step 2 (Eq. (5)) of Algorithm 2, and the index \( i(j) \) denotes the component selected at the \( j \)-th update. Let \( \{\alpha^1, \alpha^2, \ldots\} \) be the sequence generated by our algorithms, and

\[
\Delta\alpha_j = \alpha^{j+1}_{i(j)} - \alpha^{j}_{i(j)}.
\]

The update \( \Delta\alpha_j \) at iteration \( j \) is obtained by solving

\[
\Delta\alpha_j \leftarrow \min_{\delta} \frac{1}{2} \|\hat{w}^j + \delta x_{i(j)}\|^2 + \ell_{i(j)}(-\alpha_{i(j)} + \delta),
\]

where \( \hat{w}^j \) is the current \( w \) in the memory. We use \( w^j = \sum_i \alpha_i^j x_i \) to denote the “accurate” \( w \) at iteration \( j \).

In PASSCoDe-Lock, \( w^j = \hat{w}^j \) is ensured by locking the variables. However, in PASSCoDe-Atomic and PASSCoDe-Wild, \( \hat{w}^j \neq w^j \) because some of the updates would not have been written into the shared memory. To capture this phenomenon, we define \( Z^j \) to be the set of all “updates to \( w \)” before iteration \( j \):

\[
Z^j := \{(t,k) \mid t < j, k \in N(i(t))\},
\]

where \( N(i(t)) := \{|u| X_{i(t),u} \neq 0\} \) is the set of all nonzero features in \( x_{i(t)} \). We define \( U^j \subseteq Z^j \) to be the updates that have already been written into \( \hat{w}^j \). Therefore, we have

\[
\hat{w}^j = w^0 + \sum_{(t,k) \in U^j} (\Delta\alpha_t) x_{i(t),k} e_k.
\]

4.1. Linear Convergence of PASSCoDe-Atomic

In PASSCoDe-Atomic, we assume all the updates before the \((j - \tau)\)-th iteration have been written into \( \hat{w}^j \), therefore,

**Assumption 1.** The set \( U^j \) satisfies \( Z^{j-\tau} \subseteq U^j \subseteq Z^j \).

Now we define some constants used in our theoretical analysis. Note that \( X \in \mathbb{R}^{n \times d} \) is the data matrix, and we use \( \hat{X} \in \mathbb{R}^{n \times d} \) to denote the normalized data matrix where each row is \( \hat{x}_i = x_i^T / \|x_i\| \). We then define

\[
M_i = \max_{S \subseteq [d]} \| \sum_{i \in S} \hat{X}_{:,i} x_{i,i} \|, \quad M = \max_i M_i,
\]

where \([d] := \{1, \ldots, d\}\) is the set of all the feature indices, and \( \hat{X}_{:,i} \) is the \( t \)-th column of \( \hat{X} \). We also define \( R_{\text{min}} = \min_i \|x_i\|^2, R_{\text{max}} = \max_i \|x_i\|^2, \) and \( L_{\text{max}} \) to be the Lipschitz constant such that \( \|\nabla D(\alpha_1) - \nabla D(\alpha_2)\| \leq L_{\text{max}} \|\alpha_1 - \alpha_2\| \) for all \( \alpha_1, \alpha_2 \) within the level set \( \{\alpha \mid D(\alpha) \leq D(\alpha^0)\} \). We assume that \( R_{\text{max}} = 1 \) and there is no zero training sample, so \( R_{\text{min}} > 0 \).

To prove the convergence of asynchronous algorithms, we first show that the expected step size does not increase super-linearly by the following Lemma 1.

**Lemma 1.** If \( \tau \) is small enough such that

\[
(6 \tau (\tau + 1)^2 e M) / \sqrt{n} \leq 1,
\]

then PASSCoDe-Atomic satisfies the following inequality:

\[
E(\|\alpha^{j-1} - \alpha^j\|^2) \leq \rho E(\|\alpha^j - \alpha^{j+1}\|^2),
\]

where \( \rho = (1 + 6 \tau (\tau + 1)^2 e M)^2 \).

We use a similar technique as in (Liu & Wright, 2014) to prove this lemma with two major differences:

- Their “inconsistent read” model assumes \( \hat{w}^j = \sum_i \alpha_i x_i \) for some \( \alpha \). However, in our case \( \hat{w}^j \) may not be written in this form due to incomplete updates in step 3 of Algorithm 2.
- In (Liu & Wright, 2014), each coordinate is updated by \( \gamma \nabla f(\alpha) \) with a fixed step size \( \gamma \). We consider the case that each subproblem (4) is solved exactly.

To show linear convergence, we assume the objective function satisfies the following property:

**Definition 1.** The objective function (2) admits a global error bound if there is a constant \( \kappa \) such that for all \( \alpha \),

\[
\|\alpha - P_S(\alpha)\| \leq \kappa \|T(\alpha) - \alpha\|,
\]

where \( P_S(\cdot) \) is the Euclidean projection to the set \( S \) of optimal solutions, and \( T : \mathbb{R}^n \rightarrow \mathbb{R}^n \) is defined by

\[
T_1(\alpha) = \arg\min_u D(\alpha + (u - \alpha_t) e_t) \quad \forall t = 1, \ldots, n,
\]

where \( e_t \) is the standard \( t \)-th unit vector. We say that the algorithm satisfies a global error bound from the beginning if (9) holds for all iterates \( \{\alpha^j\}_{j=1,2,\ldots} \) generated by the algorithm.

This definition is a generalized version of Definition 6 in (Wang & Lin, 2014). We show that many machine learning problems satisfy this assumption (see Appendix 7).

**Theorem 1.** Support Vector Machines (SVM) with hinge loss or square hinge loss satisfy a global error bound (9). DCD for the \( \ell_2 \)-regularized logistic regression satisfies a global error bound from the beginning.

Next we explicitly state the linear convergence guarantee for PASSCoDe-Atomic.
Theorem 2. Assume the objective function (2) admits a global error bound from the beginning and the Lipschitz constant $L_{max}$ is finite in the level set. If (7) holds and
\[ 1 \geq \frac{2L_{max}}{R_{min}} \left( 1 + \frac{\epsilon T M}{\sqrt{n}} \right) \left( \frac{\tau^2 M^2 e^2}{n} \right) \]
then PASSCoDe-Atomic has a global linear convergence rate in expectation, that is,
\[ E[D(\alpha^{j+1})] - D(\alpha^*) \leq \eta \left( E[D(\alpha^j)] - D(\alpha^*) \right) , \quad (10) \]
where $\alpha^*$ is the optimal solution and
\[ \eta = \frac{1 - \kappa}{L_{max}} \left( 1 - \frac{2L_{max}}{R_{min}} \left( 1 + \frac{\epsilon T M}{\sqrt{n}} \right) \left( \frac{\tau^2 M^2 e^2}{n} \right) \right). \]

4.2. Error Analysis for PASSCoDe-Wild

In PASSCoDe-Wild, assume the sequence $\{\alpha^j\}$ converges to $\hat{\alpha}$ and $\{w^j\}$ converges to $\hat{w}$. Now we show that the dual solution $\hat{\alpha}$ and the corresponding primal variables $\hat{w} = \sum_{i=1}^{n} \hat{a}_i x_i$ are actually the dual and primal solutions of a perturbed problem:

**Theorem 3.** $\hat{\alpha}$ is the optimal solution of a perturbed dual problem
\[ \hat{\alpha} = \arg \min_{\alpha} D(\alpha) - \sum_{i=1}^{n} \alpha_i e^T x_i, \quad (11) \]
and $\hat{w} = \sum_{i=1}^{n} \hat{a}_i x_i$ is the solution of the corresponding primal problem:
\[ \hat{w} = \arg \min_{w} \frac{1}{2} w^T w + \sum_{i=1}^{n} \ell_i((w - \epsilon)^T x_i), \quad (12) \]
where $\epsilon \in \mathbb{R}^d$ is given by $\epsilon = \hat{w} - \hat{w}$.

The proof is in Appendix 9.1. Note that $\epsilon$ is the error caused by the memory conflicts. From Theorem 3, $\hat{w}$ is the optimal solution of the “biased” primal problem (12), however, in (12) the actual model that fits the loss function should be $\tilde{w} = w - \epsilon$. Therefore after the training process we should use $\tilde{w}$ to predict, which is the $w$ we maintained during the parallel coordinate descent updates. Replacing $w$ by $w - \epsilon$ in (12), we have the following corollary:

**Corollary 1.** $\hat{w}$ computed by PASSCoDe-Wild is the solution of the following perturbed primal problem:
\[ \hat{w} = \arg \min_{w} \frac{1}{2} (w + \epsilon)^T (w + \epsilon) + \sum_{i=1}^{n} \ell_i((w + \epsilon)^T x_i) \quad (13) \]
The above corollary shows that the computed primal solution $\hat{w}$ is actually the exact solution of a perturbed problem (where the perturbation is on the regularizer). This strategy (of showing that the computed solution to a problem is the exact solution of a perturbed problem) is inspired by the backward error analysis technique commonly employed in numerical analysis (Wilkinson, 1961)\(^1\). Bounding the size of the perturbation is a topic of future research.

---

\(1\) H. Wilkinson received the Turing Award in 1970, partly for his work on backward error analysis.

Table 3. Data statistics. $\bar{n}$ is the number of test instances. $\bar{d}$ is the average nnz per instance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$\bar{n}$</th>
<th>$d$</th>
<th>$\bar{d}$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>news20</td>
<td>16,000</td>
<td>3,996</td>
<td>1,355,191</td>
<td>455.5</td>
<td>2</td>
</tr>
<tr>
<td>covtype</td>
<td>500,000</td>
<td>81,012</td>
<td>54</td>
<td>11.9 0.0625</td>
<td>1</td>
</tr>
<tr>
<td>rcv1</td>
<td>677,399</td>
<td>20,242</td>
<td>4,723,67</td>
<td>73.2</td>
<td>1</td>
</tr>
<tr>
<td>webspam</td>
<td>280,000</td>
<td>70,000</td>
<td>16,609,143</td>
<td>3727.7</td>
<td>1</td>
</tr>
<tr>
<td>kdddb</td>
<td>19,264,097</td>
<td>748,401</td>
<td>29,890,095</td>
<td>29.4</td>
<td>1</td>
</tr>
</tbody>
</table>

5. Experimental Results

We conduct several experiments and show that the proposed methods PASSCoDe-Atomic and PASSCoDe-Wild have superior performance compared to other state-of-the-art parallel coordinate descent algorithms. We consider five datasets: news20, covtype, rcv1, webspam, and kdddb. Detailed information is shown in Table 3. To have a fair comparison, we implement all methods in C++ using OpenMP as the parallel programming framework. All the experiments are performed on an Intel multi-core dual-socket machine with 256 GB memory. Each socket is associated with 10 computation cores. We explicitly enforce that all the threads use cores from the same socket to avoid inter-socket communication. Our code is available in [http://www.cs.utexas.edu/~rofuyu/exp-codes/passcode-icml15-exp/](http://www.cs.utexas.edu/~rofuyu/exp-codes/passcode-icml15-exp/). We focus on solving SVM with hinge loss in the experiments, but the algorithms can be applied to other objective functions.

Serial Baselines.

- **DCD:** we implement Algorithm 1. Instead of sampling with replacement, a random permutation is used to enforce random sampling without replacement.
- **LIBLINEAR:** we use the implementation in [http://www.csie.ntu.edu.tw/~cjlin/liblinear](http://www.csie.ntu.edu.tw/~cjlin/liblinear). This implementation is equivalent to DCD with the shrinking strategy.

Compared Parallel Implementation.

- **PASSCoDe:** We implement the proposed three variants of Algorithm 2 using DCD as the building block: Wild, Atomic, and Lock.
- **CoCoA:** We implement a multi-core version of CoCoA (Jaggi et al., 2014) with $\beta_K = 1$ and DCD as its local dual method.
- **AsySCD:** We follow the description in (Liu & Wright, 2014; Liu et al., 2014) to implement AsySCD with step length $\gamma = \frac{1}{2}$ and the shuffling period $p = 10$ as suggested in (Liu et al., 2014).

5.1. Convergence in terms of iterations.

The primal objective function value is used to determine the convergence. Note that we still use $P(\tilde{w})$ for PASSCoDe-Wild, although the true primal objective should be (13). As long as $\tilde{w}^T \epsilon$ remains small enough, the trends of (13) and
\(P(\hat{w})\) are similar.

Figures 2(a), 3(a), 5(a), 4(a), 6(a) show the convergence results of \textit{PASSCoDe-Wild}, \textit{PASSCoDe-Atomic}, \textit{CoCoA}, and \textit{AsySCD} with 10 threads in terms of the number of iterations. The result for \textit{LIBLINEAR} is also included for reference. We make the following observations:

- Convergence of three \textit{PASSCoDe} variants are almost identical and very close to the convergence behavior of serial \textit{LIBLINEAR} on three large sparse datasets (\textit{rcv1}, \textit{websap}, and \textit{kddb}).
- \textit{PASSCoDe-Wild} and \textit{PASSCoDe-Atomic} converge significantly faster than \textit{CoCoA}.
- On \textit{covtype}, a more dense dataset, all three algorithms (\textit{PASSCoDe-Wild}, \textit{PASSCoDe-Atomic}, and \textit{CoCoA}) have slower convergence.

5.2. Efficiency

**Timing.** To have a fair comparison, we include both initialization and computation into the timing results. For \textit{DCD}, \textit{PASSCoDe}, and \textit{CoCoA}, initialization requires one pass over the entire data matrix (which is \(O(nnz(X))\)) to compute \(||x_i||\) for each instance. In the initialization stage, \textit{AsySCD} requires \(O(n \times nnz(X))\) time and \(O(n^2)\) space to form and store the Hessian matrix \(Q\) for (2). Thus, we only have results on \textit{news20} for \textit{AsySCD} as all other datasets are too large for \textit{AsySCD} to fit \(Q\) in even 256 GB memory. Note that we also parallelize the initialization part for each algorithm in our implementation to have a fair comparison.

Figures 2(b), 3(b), 5(b), 4(b), 6(b) show the primal objective values in terms of time and Figures 2(c), 3(c), 5(c), 4(c), 6(c) shows the accuracy in terms of time. Note that the x-axis for \textit{news20}, \textit{covtype}, and \textit{rcv1} is in log-scale. We have the following observations:

- From Figures 2(b) and 2(c), we can see that \textit{AsySCD} is orders of magnitude slower than other approaches including parallel methods and serial reference (\textit{AsySCD} using 10 cores takes 0.4 seconds to run 10 iterations, while all the other parallel approaches take less than 0.14 seconds, and \textit{LIBLINEAR} takes less than 0.3 seconds). In fact, \textit{AsySCD} is still slower than other methods even when initialization time is excluded. This is expected because \textit{AsySCD} is a parallel version of a standard coordinate descent method, which is known to be much slower than \textit{DCD} for (2). Since \textit{AsySCD} runs out of memory for all the other larger datasets, we do not show the results in other figures.
- In most cases, both \textit{PASSCoDe} approaches outperform \textit{CoCoA}. In Figure 6(c), \textit{kddb} shows better accuracy performance in the early stages which can be explained by the ensemble nature of \textit{CoCoA}. In the long term, it still converges to the accuracy obtained by \textit{LIBLINEAR}.
- For all datasets, \textit{PASSCoDe-Wild} is slightly faster than \textit{PASSCoDe-Atomic}. Given the fact that both methods show similar convergence in terms of iterations, this phenomenon can be explained by the effect of atomic operations. We observe that more dense the dataset, the larger the difference between \textit{PASSCoDe-Wild} and \textit{PASSCoDe-Atomic}.

5.3. Speedup

We are interested in the following evaluation criterion:

\[
\text{speedup} := \frac{\text{time taken by the target method with } p \text{ threads}}{\text{time taken by the best serial reference method}}
\]

This criterion is different from \textit{scaling}, where the denominator is replaced by “time taken for the target method with single thread.” Note that a method can have perfect scaling but very poor speedup. Figures 2(d), 3(d), 5(d), 4(d), 6(d) shows the speedup results, where 1) \textit{DCD} is used as the best serial reference; 2) the shrinking heuristic is turned off for all \textit{PASSCoDe} and \textit{DCD} to have fair comparison; 3) the initialization time is excluded from the computation of speedup.

- \textit{PASSCoDe-Wild} has very good speedup performance compared to other approaches. It achieves a speedup of about 6 to 8 using 10 threads on all the datasets.
- From Figure 2(d), we can see that \textit{AsySCD} hardly has any “speedup” over the serial reference, although it is shown to have almost linear scaling (Liu et al., 2014; Liu & Wright, 2014).

6. Conclusions and Future Work

In this paper, we present a family of parallel asynchronous stochastic dual coordinate descent algorithms in the shared memory multi-core setting, where each thread repeatedly selects a random dual variable and conducts coordinate updates using the primal variables that are stored in the shared memory. We analyze the convergence properties when different locking/atomic mechanisms are used. For the setting with atomic updates, we show linear convergence under certain conditions. For the setting without any lock or atomic write, which achieves the best speedup, we present an error analysis to show that the primal variable obtained by the algorithm is the exact solution for a primal problem with a perturbed regularizer. Bounding the size of this perturbation is a topic of future research. Experimental results show that our algorithms are much faster than previous parallel coordinate descent solvers.

Acknowledgements

This research was supported by NSF grants CCF-1320746 and CCF-1117055. C.-J.H also acknowledges support from an IBM PhD fellowship. H.-F.Y also acknowledges support from an Intel PhD fellowship.
PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent

(a) Convergence
(b) Objective
(c) Accuracy
(d) Speedup

Figure 2. news20 dataset

Figure 3. covtype dataset

Figure 4. webspam dataset
References


