PASSCoDe: Parallel ASynchronous Stochastic dual Co-ordinate Descent

Cho-Jui Hsieh CJHSIEH@CS.UTEXAS.EDU
Hsiang-Fu Yu ROFUYU@CS.UTEXAS.EDU
Inderjit S. Dhillon 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 ℓ_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 $(\dot{\boldsymbol{x}}_i, \dot{y}_i)$, $i = 1, \cdots, n$, $\dot{\boldsymbol{x}}_i \in \mathbb{R}^d$, $\dot{y}_i \in \mathbb{R}$, we focus on the following empirical risk minimization problem with ℓ_2-regularization:

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

where $\boldsymbol{x}_i = \dot{y}_i \dot{\boldsymbol{x}}_i$, $\ell_i(\cdot)$ 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):

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

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

$$\boldsymbol{w}(\boldsymbol{\alpha}) = \sum\nolimits_{i=1}^{n} \alpha_i \boldsymbol{x}_i, \qquad (3)$$

then it is known that $\boldsymbol{w}(\boldsymbol{\alpha}^*) = \boldsymbol{w}^*$ and $P(\boldsymbol{w}^*) = -D(\boldsymbol{\alpha}^*)$ where $\boldsymbol{w}^*, \boldsymbol{\alpha}^*$ are the optimal primal/dual solutions respectively. Examples include hinge-loss SVM, square hinge SVM and ℓ_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 $\boldsymbol{w}$ based on the primal-dual relationship (3). By maintaining $\boldsymbol{w}$ in memory, (Hsieh et al., 2008; Keerthi et al., 2008) showed that the time complexity of each coordinate update can be reduced from $O(\text{nnz})$ to $O(\text{nnz}/n)$, where nnz 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).

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

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 synchronous 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 $\boldsymbol{w}$ and $\boldsymbol{\alpha}$, 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 ℓ_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 $\boldsymbol{w}$ and one local dual variable α_i, thus algorithms only need to synchronize $\boldsymbol{w}$. Based on this observation, (Yang, 2013) proposed to update several coordinates or blocks simultaneously and update the global $\boldsymbol{w}$, 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 $\boldsymbol{w}$ 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 $\boldsymbol{w}$ 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 α_i is based on the shared $\boldsymbol{w}$ 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 α_i and updates it by minimizing the one variable subproblem $\arg\min_\delta D(\boldsymbol{\alpha} + \delta \boldsymbol{e}_i)$. Without exploiting the structure of the quadratic term, solving each subproblem requires $O(\text{nnz})$ operations, where nnz is the total number of nonzero elements in the training data, which can be substantial. However, if $\boldsymbol{w}(\boldsymbol{\alpha})$ that satisfies (3) is maintained in memory, the subproblem $D(\boldsymbol{\alpha} + \delta \boldsymbol{e}_i)$ can be written as

$$D(\boldsymbol{\alpha} + \delta \boldsymbol{e}_i) = \frac{1}{2}\|\boldsymbol{w} + \delta \boldsymbol{x}_i\|^2 + \ell_i^*(-(\alpha_i + \delta)),$$

and the optimal solution can be computed by

$$\delta^* = \arg\min_\delta \frac{1}{2}(\delta + \frac{\boldsymbol{w}^T \boldsymbol{x}_i}{\|\boldsymbol{x}_i\|^2})^2 + \frac{1}{\|\boldsymbol{x}_i\|^2}\ell_i^*(-(\alpha_i + \delta)).$$

Note that all $\|\boldsymbol{x}_i\|$ 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 $\boldsymbol{w}^T \boldsymbol{x}_i$, 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 $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$

1: **while** not converged **do**
2: Randomly pick i
3: Update $\alpha_i \leftarrow \alpha_i + \Delta\alpha_i$, where

$$\Delta\alpha_i \leftarrow \arg\min_\delta \frac{1}{2}\|\boldsymbol{w} + \delta \boldsymbol{x}_i\|^2 + \ell_i^*(-(\alpha_i + \delta)) \quad (4)$$

4: Update $\boldsymbol{w}$ by $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\alpha_i \boldsymbol{x}_i$
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 $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ 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 Coordinate Descent (*PASSCoDe*)

Input: Initial $\boldsymbol{\alpha}$ and $\boldsymbol{w} = \sum_{i=1}^n \alpha_i \boldsymbol{x}_i$

Each thread repeatedly performs the following updates:
step 1: Randomly pick i
step 2: Update $\alpha_i \leftarrow \alpha_i + \Delta\alpha_i$, where

$$\Delta\alpha_i \leftarrow \arg\min_\delta \frac{1}{2}\|\boldsymbol{w} + \delta \boldsymbol{x}_i\|^2 + \ell_i^*(-(\alpha_i + \delta)) \quad (5)$$

step 3: Update $\boldsymbol{w}$ by $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\alpha_i \boldsymbol{x}_i$

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 $\boldsymbol{w}$ 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 $\boldsymbol{w} = \sum_i \alpha_i \boldsymbol{x}_i$ for the latest $\boldsymbol{\alpha}$, we have to lock the "active" variables between step 1 and 2 in Algorithm 2:

step 1.5: lock variables in $N_i := \{w_t \mid (\boldsymbol{x}_i)_t \neq 0\}$.

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).

Number of threads	Lock	Atomic	Wild
2	98.03s / 0.27x	15.28s / 1.75x	**14.08s / 1.90x**
4	106.11s / 0.25x	8.35s / 3.20x	**7.61s / 3.50x**
10	114.43s / 0.23x	3.86s / 6.91x	**3.59s / 7.43x**

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

	# threads	Prediction Accuracy (%) by $\hat{\boldsymbol{w}}$	$\bar{\boldsymbol{w}}$	LIBLINEAR
news20	4	97.1	96.1	97.1
	8	97.2	93.3	
covtype	4	67.8	38.0	66.3
	8	67.6	38.0	
rcv1	4	97.7	97.5	97.7
	8	97.7	97.4	
webspam	4	99.1	93.1	99.1
	8	99.1	88.4	
kddb	4	88.8	79.7	88.8
	8	88.8	87.7	

is a update sequence 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 α_i based on the latest $\boldsymbol{w}$ 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 $\boldsymbol{w}$ values from memory without any locking. In practice (see Section 5) we observe that the convergence speed is not significantly effected by using these "unlocked" values of $\boldsymbol{w}$. However, to ensure that the limit point of the algorithm is still the global optimizer of (1), the equation $\boldsymbol{w}^* = \sum_i \alpha_i^* \boldsymbol{x}_i$ has to be maintained. Therefore, we apply the following "atomic writes" in step 3:

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

Update $w_j \leftarrow w_j + \Delta\alpha_i(\boldsymbol{x}_i)_j$ **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.

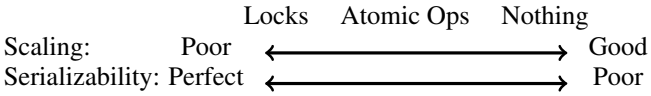


Figure 1. Spectrum for the choice of mechanism to avoid memory conflicts for *PASSCoDe*.

PASSCoDe-Wild. Finally, we consider Algorithm 2 without any locks and atomic operations. The resulting algorithm, *PASSCoDe-Wild*, is faster than *PASSCoDe-Atomic* and *PASSCoDe-Lock* and can achieve almost linear speedup. However, due to the memory conflicts in step 3, some of the "updates" to $\boldsymbol{w}$ could be over-written by other threads. As a result, the $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{\alpha}}$ do not satisfy Eq (3):

$$\hat{\boldsymbol{w}} \neq \bar{\boldsymbol{w}} := \sum\nolimits_i \hat{\alpha}_i \boldsymbol{x}_i, \tag{6}$$

where $\hat{\boldsymbol{w}}, \hat{\boldsymbol{\alpha}}$ are the primal and dual variables output by the algorithm, and $\bar{\boldsymbol{w}}$ defined in (6) is computed from $\hat{\boldsymbol{\alpha}}$. It is easy to see that $\hat{\boldsymbol{\alpha}}$ is not the optimal solution of (2). Hence, in the prediction phase it is not clear whether one should use $\hat{\boldsymbol{w}}$ or $\bar{\boldsymbol{w}}$. In Section 4 we show that $\hat{\boldsymbol{w}}$ is actually the optimal solution of a perturbed primal problem (1), where the loss function is the same and the regularization term is slightly perturbed. As a result, the prediction should be done using $\hat{\boldsymbol{w}}$, and this also yields much better accuracy in practice, as shown in Table 2.

We summarize the behavior of the three algorithms in Figure 1. Using locks, the algorithm *PASSCoDe-Lock* is serializable but very slow (even slower than the serial *DCD*). In the other extreme, the wild version without any locking and atomic operation has very good speedup, but the behavior can be different from serial *DCD*. Theoretically, we provide a convergence guarantee for *PASSCoDe-Atomic*, and show that *PASSCoDe-Wild* converges to the solution with the same loss function with a perturbed regularizer.

3.3. Implementation Details

Deadlock Avoidance. Without a proper implementation, a deadlock can arise in *PASSCoDe-Lock* because a thread needs to acquire all the locks associated with N_i. A simple way to avoid deadlock is by associating an ordering for all the locks such that each thread follows the same ordering to acquire the locks.

Random Permutation. In LIBLINEAR, the random sampling (step 2) of Algorithm 1 is replaced by the index from a random permutation, such that each α_i can be selected in n steps instead of $n \log n$ steps in expectation. Random permutation can be easily implemented asynchronously for Algorithm 2 as follows: Initially, given p threads, $\{1, \ldots, n\}$ is randomly partitioned into p blocks. Then, each thread can asynchronously generate the random permutation on its own block of variables.

Shrinking Heuristic. For loss such as hinge and squared-hinge, the optimal $\boldsymbol{\alpha}^*$ is usually sparse. Based on this property, a shrinking strategy was proposed by (Hsieh et al.,

2008) to further speed up *DCD*. This heuristic is also implemented in LIBLINEAR. The idea is to maintain an active set by skipping variables which tend to be fixed. This heuristic can also be implemented in Algorithm 2 by maintaining an active set for each thread.

Thread Affinity. The memory design of most modern multi-core machines is *non-uniform memory access (NUMA)*, where a core has faster memory access to its local memory socket. To reduce possible latency due to the remote socket access, we should bind each thread to a physical 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 $\{\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \dots\}$ 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

$$\Delta\alpha_j \leftarrow \arg\min_{\delta} \frac{1}{2}\|\hat{\boldsymbol{w}}^j + \delta \boldsymbol{x}_{i(j)}\|^2 + \ell^*_{i(j)}(-(\alpha_{i(j)} + \delta)),$$

where $\hat{\boldsymbol{w}}^j$ is the current $\boldsymbol{w}$ in the memory. We use $\boldsymbol{w}^j = \sum_i \alpha^j_i \boldsymbol{x}_i$ to denote the "accurate" $\boldsymbol{w}$ at iteration j.

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

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

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

$$\hat{\boldsymbol{w}}^j = \boldsymbol{w}^0 + \sum\nolimits_{(t,k)\in\mathcal{U}^j} (\Delta\alpha_t) X_{i(t),k} \boldsymbol{e}_k.$$

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 $\hat{\boldsymbol{w}}^j$, therefore,

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

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 $\bar{X} \in \mathbb{R}^{n\times d}$ to denote the normalized data matrix where each row is $\bar{\boldsymbol{x}}_i^T = \boldsymbol{x}_i^T / \|\boldsymbol{x}_i\|$. We then define

$$M_i = \max_{S\subseteq[d]} \|\sum_{t\in S} \bar{X}_{:,t} X_{i,t}\|, \quad M = \max_i M_i,$$

where $[d] := \{1, \dots, d\}$ is the set of all the feature indices, and $\bar{X}_{:,t}$ is the t-th column of $\bar{X}$. We also define $R_{min} = \min_i \|\boldsymbol{x}_i\|^2$, $R_{max} = \max_i \|\boldsymbol{x}_i\|^2$, and L_{max} to be the Lipschitz constant such that $\|\nabla D(\boldsymbol{\alpha}_1) - \nabla D(\boldsymbol{\alpha}_2)\| \leq L_{max}\|\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2\|$ for all $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$ within the level set $\{\boldsymbol{\alpha} \mid D(\boldsymbol{\alpha}) \leq D(\boldsymbol{\alpha}^0)\}$. We assume that $R_{max} = 1$ and there is no zero training sample, so $R_{min} > 0$.

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 τ is small enough such that*

$$(6\tau(\tau+1)^2 eM)/\sqrt{n} \leq 1, \tag{7}$$

then PASSCoDe-Atomic satisfies the following inequality:

$$E(\|\boldsymbol{\alpha}^{j-1} - \boldsymbol{\alpha}^j\|^2) \leq \rho E(\|\boldsymbol{\alpha}^j - \boldsymbol{\alpha}^{j+1}\|^2), \tag{8}$$

where $\rho = (1 + \frac{6(\tau+1)eM}{\sqrt{n}})^2$.

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

- Their "inconsistent read" model assumes $\hat{\boldsymbol{w}}^j = \sum_i \dot{\alpha}_i \boldsymbol{x}_i$ for some $\dot{\boldsymbol{\alpha}}$. However, in our case $\hat{\boldsymbol{w}}^j$ 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 $\gamma\nabla_t f(\boldsymbol{\alpha})$ with a fixed step size γ. 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 κ such that for all $\boldsymbol{\alpha}$,*

$$\|\boldsymbol{\alpha} - P_S(\boldsymbol{\alpha})\| \leq \kappa\|T(\boldsymbol{\alpha}) - \boldsymbol{\alpha}\|, \tag{9}$$

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

$$T_t(\boldsymbol{\alpha}) = \arg\min_u \; D(\boldsymbol{\alpha} + (u - \alpha_t)\boldsymbol{e}_t) \quad \forall t = 1, \dots, n,$$

where $\boldsymbol{e}_t$ 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 $\{\boldsymbol{\alpha}^j\}_{j=1,2,\dots}$ 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 ℓ_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*

$$1 \geq \frac{2L_{max}}{R_{min}}\left(1+\frac{e\tau M}{\sqrt{n}}\right)\left(\frac{\tau^2M^2e^2}{n}\right)$$

then PASSCoDe-Atomic has a global linear convergence rate in expectation, that is,

$$E[D(\boldsymbol{\alpha}^{j+1})]-D(\boldsymbol{\alpha}^*) \leq \eta\left(E[D(\boldsymbol{\alpha}^j)]-D(\boldsymbol{\alpha}^*)\right), \quad (10)$$

where $\boldsymbol{\alpha}^*$ is the optimal solution and

$$\eta = 1-\frac{\kappa}{L_{max}}\left(1-\frac{2L_{max}}{R_{min}}\left(1+\frac{e\tau M}{\sqrt{n}}\right)\left(\frac{\tau^2M^2e^2}{n}\right)\right).$$

4.2. Error Analysis for *PASSCoDe-Wild*

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

Theorem 3. *$\hat{\boldsymbol{\alpha}}$ is the optimal solution of a perturbed dual problem*

$$\hat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha}} \; D(\boldsymbol{\alpha}) - \sum\nolimits_{i=1}^n \alpha_i \boldsymbol{\epsilon}^T \boldsymbol{x}_i, \quad (11)$$

and $\bar{\boldsymbol{w}} = \sum_i \hat{\alpha}_i \boldsymbol{x}_i$ is the solution of the corresponding primal problem:

$$\bar{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \; \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum\nolimits_{i=1}^n \ell_i((\boldsymbol{w}-\boldsymbol{\epsilon})^T\boldsymbol{x}_i), \quad (12)$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^d$ is given by $\boldsymbol{\epsilon} = \bar{\boldsymbol{w}} - \hat{\boldsymbol{w}}$.

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

Corollary 1. *$\hat{\boldsymbol{w}}$ computed by PASSCoDe-Wild is the solution of the following perturbed primal problem:*

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{2}(\boldsymbol{w}+\boldsymbol{\epsilon})^T(\boldsymbol{w}+\boldsymbol{\epsilon}) + \sum_{i=1}^n \ell_i(\boldsymbol{w}^T\boldsymbol{x}_i) \quad (13)$$

The above corollary shows that the computed primal solution $\hat{\boldsymbol{w}}$ 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] J. H. Wilkinson received the Turing Award in 1970, partly for his work on backward error analysis

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

	n	$\tilde{n}$	d	$\bar{d}$	C
news20	16,000	3,996	1,355,191	455.5	2
covtype	500,000	81,012	54	11.9	0.0625
rcv1	677,399	20,242	47,236	73.2	1
webspam	280,000	70,000	16,609,143	3727.7	1
kddb	19,264,097	748,401	29,890,095	29.4	1

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 kddb. 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/`. 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`. 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 $P(\hat{\boldsymbol{w}})$ for *PASSCoDe-Wild*, although the true primal objective should be (13). As long as $\hat{\boldsymbol{w}}^T\boldsymbol{\epsilon}$ remains small enough, the trends of (13) and

$P(\hat{\boldsymbol{w}})$ are similar.

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

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

5.2. Efficiency

Timing. To have a fair comparison, we include both initialization and computation into the timing results. For *DCD*, *PASSCoDe*, and *CoCoA*, initialization requires one pass over the entire data matrix (which is $O(nnz(X))$) to compute $\|\boldsymbol{x}_i\|$ for each instance. In the initialization stage, *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 news20 for *AsySCD* as all other datasets are too large for *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 news20, covtype, and rcv1 is in log-scale. We have the following observations:

- From Figures 2(b) and 2(c), we can see that *AsySCD* is orders of magnitude slower than other approaches including parallel methods and serial reference (*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 LIBLINEAR takes less than 0.3 seconds). In fact, *AsySCD* is still slower than other methods even when initialization time is excluded. This is expected because *AsySCD* is a parallel version of a standard coordinate descent method, which is known to be much slower than *DCD* for (2). Since *AsySCD* runs out of memory for all the other larger datasets, we do not show the results in other figures.
- In most cases, both *PASSCoDe* approaches outperform *CoCoA*. In Figure 6(c), kddb shows better accuracy performance in the early stages which can be explained by the ensemble nature of *CoCoA*. In the long term, it still converges to the accuracy obtained by LIBLINEAR.
- For all datasets, *PASSCoDe-Wild* is slightly faster than *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 *PASSCoDe-Wild* and *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 *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) *DCD* is used as the best serial reference; 2) the shrinking heuristic is turned off for all *PASSCoDe* and *DCD* to have fair comparison; 3) the initialization time is excluded from the computation of speedup.

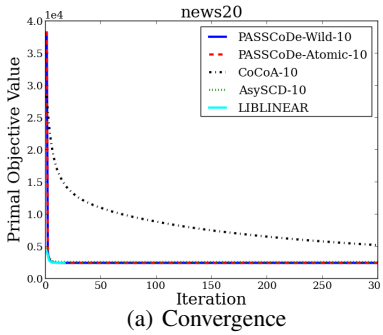
- *PASSCoDe-Wild* has very good speedup performance compared to other approaches. It achieves a speedup of about about 6 to 8 using 10 threads on all the datasets.
- From Figure 2(d), we can see that *AsySCD* hardly has any "speedup" over the serial reference, although it is shown to have almost linear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

6. Conclusions and Future Work

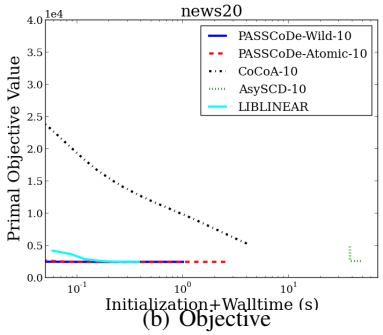
In this paper, we present a family of parallel asynchronous stochastic dual coordinate descent algorithms in the shared memory multi-core setting, where each thread repeatedly selects a random dual variable and conducts coordinate updates using the primal variables that are stored in the shared memory. 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

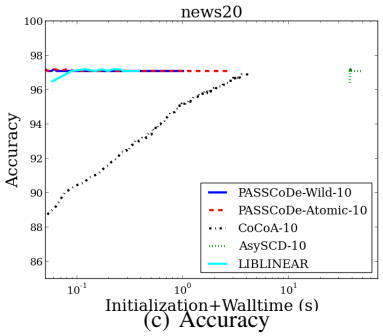
This research was supported by NSF grants CCF-1320746 and CCF-1117055. C.-J.H also acknowledges support from an IBM PhD fellowship. H.-F.Y also acknowledges support from an Intel PhD fellowship.



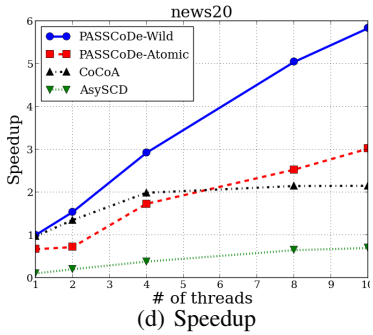
(a) Convergence



(b) Objective

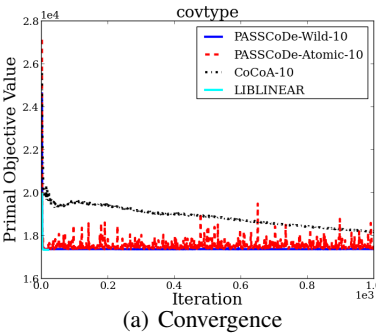


(c) Accuracy

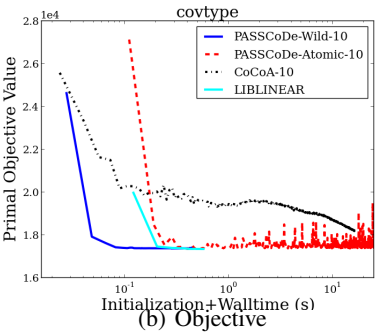


(d) Speedup

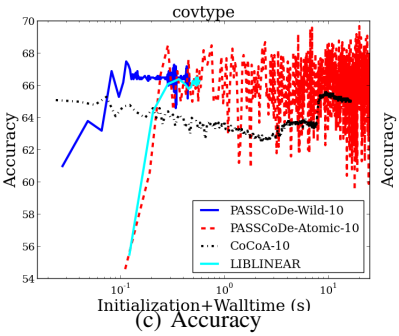
Figure 2. news20 dataset



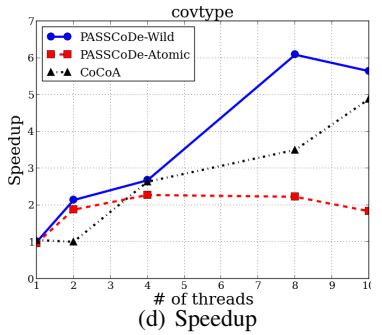
(a) Convergence



(b) Objective

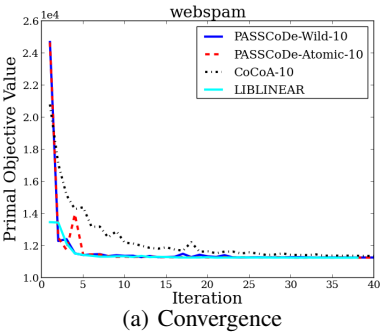


(c) Accuracy

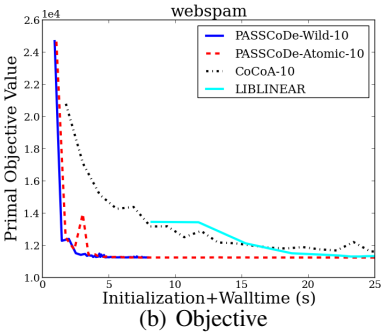


(d) Speedup

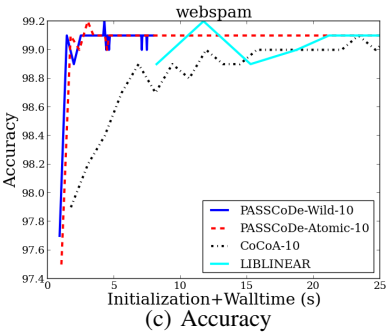
Figure 3. covtype dataset



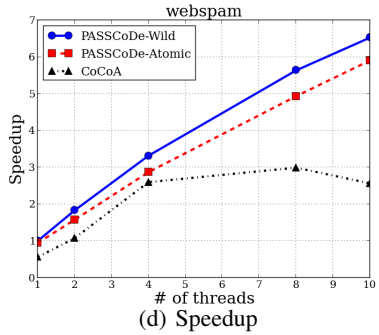
(a) Convergence



(b) Objective



(c) Accuracy



(d) Speedup

Figure 4. webspam dataset

References

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

Beck, Amir and Tetruashvili, Luba. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

Bertsekas, Dimitri P. *Nonlinear Programming*. Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

Bertsekas, Dimitri P. and Tsitsiklis, John N. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

Boser, Bernhard E., Guyon, Isabelle, and Vapnik, Vladimir. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152. ACM Press, 1992.

Bradley, Joseph K., Kyrola, Aapo, Bickson, Danny, and Guestrin, Carlos. Parallel coordinate descent for l1-regularized loss minimization. In *ICML*, 2011.

Chang, Kai-Wei, Hsieh, Cho-Jui, and Lin, Chih-Jen. Coordinate descent method for large-scale L2-loss linear SVM. *Journal of Machine Learning Research*, 9:1369–1398, 2008.

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, and Lin, Chih-Jen. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Hsieh, Cho-Jui, Chang, Kai-Wei, Lin, Chih-Jen, Keerthi, S. Sathiya, and Sundararajan, Sellamanickam. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

Huang, Fang-Lan, Hsieh, Cho-Jui, Chang, Kai-Wei, and Lin, Chih-Jen. Iterative scaling and coordinate descent methods for maximum entropy. *Journal of Machine Learning Research*, 11:815–848, 2010.

Jaggi, Martin, Smith, Virginia, Takáč, Martin, Terhorst, Jonathan, Hofmann, Thomas, and Jordan, Michael I. Communication-efficient distributed dual coordinate ascent. In *NIPS*. 2014.

Keerthi, S. S., Sundararajan, S., Chang, K.-W., Hsieh, C.-J., and Lin, C.-J. A sequential dual method for large scale multi-class linear SVMs. In *KDD*, 2008.

Lin, Chih-Jen, Weng, Ruby C., and Keerthi, S. Sathiya. Trust region Newton method for large-scale logistic regression. In *ICML*, 2007.

Liu, J. and Wright, S. J. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. 2014. URL http://arxiv.org/abs/1403.3862.

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

Luo, Zhi-Quan and Tseng, Paul. On the convergence of coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

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

Niu, Feng, Recht, Benjamin, Ré, Christopher, and Wright, Stephen J. HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pp. 693–701, 2011.

Park, D., Neeman, J., Zhang, J., Sanghavi, S., and Dhillon, I. S. Preference completion: Large-scale collaborative ranking from pairwise comparison. In *ICML*, 2015.

Pechyony, Dmitry, Shen, Libin, and Jones, Rosie. Solving large scale linear svm with distributed block minimization. In *NIPS 2011 Workshop on Big Learning: Algorithms, Systems, and Tools for Learning at Scale*. 2011.

Richtárik, Peter and Takáč, Martin. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 2012. Under revision.

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

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: primal estimated sub-gradient solver for SVM. In *ICML*, 2007.

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

Wang, Po-Wei and Lin, Chih-Jen. Iteration complexity of feasible descent methods for convex optimization. *Journal of Machine Learning Research*, 15:1523–1548, 2014.

Wilkinson, J. H. Error analysis of direct methods of matrix inversion. *Journal of the ACM*, 1961.

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

Yu, Hsiang-Fu, Huang, Fang-Lan, and Lin, Chih-Jen. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2): 41–75, October 2011.

Zhang, Tong. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, 2004.