ASYNCHRONOUS PARALLEL NONCONVEX LARGE-SCALE OPTIMIZATION

L. Cannelli,⋆ F. Facchinei,† V. Kungurtsev,‡ and G. Scutari⋆

ABSTRACT

We propose a novel *parallel asynchronous* algorithmic framework for the minimization of the sum of a smooth (nonconvex) function and a convex (nonsmooth) regularizer. The framework hinges on Successive Convex Approximation (SCA) techniques and on a novel probabilistic model which describes in a unified way a variety of asynchronous settings in a more faithful and exhaustive way with respect to state-of-the-art models. Key features of our framework are: i) it accommodates inconsistent read, meaning that components of the variables may be written by some cores while being simultaneously read by others; ii) it covers in a unified way several existing methods; and iii) it accommodates a variety of parallel computing architectures. Almost sure convergence to stationary solutions is proved for the general case, and iteration complexity analysis is given for a specific version of our model. Numerical results show that our scheme outperforms existing asynchronous ones.

Index Terms— Asynchronous algorithms; big-data; inconsistent read; nonconvex constrained optimization.

1. INTRODUCTION

We consider the constrained minimization problem:

$$\min_{\mathbf{x}\in\mathcal{X}} \quad F(\mathbf{x}) \triangleq f(\mathbf{x}) + G(\mathbf{x}) \tag{1}$$

where f is smooth (not necessarily convex), G is convex (not necessarily smooth), and $\mathcal{X} \triangleq \mathcal{X}_1 \times \cdots \mathcal{X}_N \subseteq \mathbb{R}^n$ is a closed set with a Cartesian product structure. Problem (1) arises in many fields of engineering, including compressed sensing, sensor networks, imaging, machine learning, and genomics. Usually the nonsmooth term G is used to promote some extra structure in the solution, like sparsity.

Many of the aforementioned applications give rise to extremely large problems so that recent years have witnessed a flurry of research activity aimed at developing *parallel* solution methods; some recent works include [1–6]. However, these methods are *synchronous* and therefore do not fully exploit the potential of parallel architectures. Asynchronous parallel methods reduce the idle times of cores, mitigate communication and memory-access congestion, make algorithms more fault-tolerant and have been empirically found, in certain cases, to accelerate convergence with respect to their synchronous counterparts. Studies on asynchronous methods have mainly dealt with convex minimization, see for example [7–18], and nonexpansive fixed-point problems [19–21]. In [22] nonconvex minimization problems are considered, but in the nonconvex setting only smooth problems with convex constraints are analyzed. More recently, [9] and [14] study unconstrained and constrained nonconvex optimization problems, respectively. However, these two papers propose algorithms that require, at each iteration, the global solution of nonconvex subproblems that could be extremely hard to solve and potentially as difficult as the original one.

Asynchronous algorithms produce a sequence $\mathbf{x}^k$ and, at each iteration, a core updates a component $\mathbf{x}_i$, independently from other cores. To perform its update, core c uses a "local estimate", say $\tilde{\mathbf{x}}_c^k$, of $\mathbf{x}^k$. In these asynchronous updates, two features are particularly meaningful, namely: 1) how the component $\mathbf{x}_i$ is chosen at each iteration; and 2) how $\tilde{\mathbf{x}}_c^k$ is formed. The selection of the component to update can be done at random (random selection) or according to schemes that give some kind of guarantees (a case we term pseudo-deterministic selection, see Sec. 2.2). Referring to 2), if $\tilde{\mathbf{x}}_c^k = \mathbf{x}^{k-d^k}$, for some positive integer d^k, the algorithm is asynchronous and it is said to use a "consistent read"; if $(\tilde{\mathbf{x}}_c^k)_i = (\mathbf{x}^{k-d_i^k})_i$, $i = 1, \ldots, N$, where d_i^k are positive integers that may be different for different i (blocks) and c (cores), the algorithm is asynchronous and it is said to use an "inconsistent read" (or to be "lock-free"). The sources of inconsistent read are many and arise from the computational architectures (shared memory multicore architecture, message-passing system, etc.). It is widely accepted that lock-free asynchronous methods with random selection are more efficient and suitable for applications in complex computational architectures.

The main contribution of this paper is a novel parallel asynchronous algorithmic framework for the class of problems (1) that i) uses random (or pseudo-deterministic) selection and inconsistent read; ii) is based on the SCA paradigm: each core updates a block-variable by solving a strongly convex "approximation" of the original (nonconvex) problem; the method is therefore easily implementable and can be tailored to the specific structure of Problem (1) (cf. Sec. 2); iii) can be equipped with global convergence iteration bounds when using fixed stepsizes (diminishing stepsizes can also be used, and asymptotic convergence is proved); and iv) is numerically very efficient and, in our tests, outperforms current asynchronous schemes (cf. Sec. 3). We remark that the proposed scheme is also able to deal with nonconvex constraints, see [23]; however, for the sake of simplicity here we consider only convex constraints.

The study of parallel asynchronous algorithms using inconsistent read in conjunction with block random selection rules is complex and has been initiated only very recently, [9, 13, 14, 21]. Our probabilistic model is different from and more complex than the one used in the aforementioned papers; we believe that this additional complexity is essential to capture the intricacies of these asynchronous methods and that the analyses in [9, 13, 14, 21], in spite of their pioneering merits, are lacking. Indeed, [9, 13, 14, 21] seem to assume that either $\tilde{\mathbf{x}}_c^k$ is deterministic or (explicitly or implicitly) $\tilde{\mathbf{x}}_c^k$ is *independent* of the story of the algorithm up to iteration k. Since both these assumptions are far from being satisfied in real computing environments, in our view the results in [9, 13, 14, 21] are in jeopardy. We also remark that, to the best of our knowledge, our proposal is the first method for nonconvex problems that can readily be implemented since it requires only the solution of strongly convex subproblems (as opposed to [9, 14]). The only other method sharing this key feature is described in [22, Ch. 7], where gradient projection is used. However gradient projection can be very inefficient in hard problems, can not be generalized in any way to deal with nonconvex constraints and, finally, the method in [22] only deals with smooth problems and uses pseudo-deterministic selection.

2. MODEL AND ALGORITHM

Consider Problem (1) under the following blanket assumptions.

Assumption A.

(A1) Each $\mathcal{X}_i$ is nonempty, closed, and convex (and we partition $\mathbf{x} \in \mathbb{R}^n$ accordingly: $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, with $\mathbf{x}_i \in \mathbb{R}^{n_i}$);

(A2) f is C^1 on an open set containing $\mathcal{X}$;

(A3) $\nabla_{\mathbf{x}_i} f$ is L_f-Lipschitz continuous on $\mathcal{X}_i$;

The authors are listed in alphabetical order.

⋆School of Industrial Engineering, Purdue University, USA; emails: <lcannell,gscutari>@purdue.edu. The work of Cannelli and Scutari was supported by the USA National Science Foundation under Grants CIF 1564044, CAREER Award 1555850, and ONR N00014-16-1-2244.

†Dept. of Computer, Control, and Management Engineering, University of Rome "La Sapienza", Italy; email: facchinei@dis.uniroma1.it.

‡Agent Technology Center, Czech Technical University in Prague, Czech Republic; email: vyacheslav.kungurtsev@fel.cvut.cz.

(A4) G is separable ($G(\mathbf{x}) \triangleq \sum_i g_i(\mathbf{x}_i)$), continuous, convex (possibly nondifferentiable), and Lipschitz continuous on $\mathcal{X}$;

(A5) Problem (1) admits a solution.

The above assumptions are standard and are satisfied by many practical problems. For instance, A3 holds trivially if $\mathcal{X}$ is bounded. Our goal is to design a parallel, asynchronous, lock-free algorithm for Problem (1). Our proposal leverages FLEXA [4], a synchronous parallel method for the solution of Problem (1). Therefore we start with a brief description of FLEXA and then build on it to define our new asynchronous method.

Parallel synchronous updates. In FLEXA, given the current iterate $\mathbf{x}^k$, all blocks $\mathbf{x}_i$ are updated at the same time by solving, for each block i, a convexified problem wherein the nonconvex objective function F is replaced by the following convex surrogate

$$\hat{\mathbf{x}}_i(\mathbf{x}^k) \triangleq \operatorname*{argmin}_{\mathbf{x}_i \in \mathcal{X}_i} \tilde{F}_i(\mathbf{x}_i; \mathbf{x}^k) \triangleq \tilde{f}_i(\mathbf{x}_i; \mathbf{x}^k) + g_i(\mathbf{x}_i), \qquad (2)$$

where $\tilde{f}_i : \mathcal{X}_i \times \mathcal{X} \to \mathbb{R}$ should be regarded as a (simple) strongly convex approximations of f at the current iterate $\mathbf{x}^k$, with respect to $\mathbf{x}_i$, that preserves the first order properties of f in $\mathbf{x}^k$. We require that the following quite natural assumptions be satisfied (we denote by $\nabla \tilde{f}_i$ the partial gradient of $\tilde{f}_i$ with respect to the first argument).

Assumption B (On the surrogate functions).

- **(B1)** $\tilde{f}_i(\bullet; \mathbf{x}^k)$ is $c_{\tilde{f}}$-strongly convex and continuously differentiable on $\mathcal{X}_i$ for all $\mathbf{x}^k \in \mathcal{X}$;
- **(B2)** $\nabla \tilde{f}_i(\mathbf{x}_i^k; \mathbf{x}^k) = \nabla_{\mathbf{x}_i} f(\mathbf{x}^k)$, for all $\mathbf{x}^k \in \mathcal{X}$;
- **(B3)** $\nabla \tilde{f}_i(\mathbf{x}_i^k; \bullet)$ is Lipschitz continuous on $\mathcal{X}$, for all $\mathbf{x}_i^k \in \mathcal{X}_i$, with a Lipschitz constant which is independent of i and k.

A wide array of examples of surrogate functions $\tilde{f}_i$ satisfying Assumption B can be found in [4]. Here we only note that one can always choose as a surrogate the linearization $\tilde{f}_i(\mathbf{x}_i; \mathbf{x}^k) = \nabla_{\mathbf{x}_i} f(\mathbf{x}^k)^T (\mathbf{x}_i - \mathbf{x}_i^k) + \tilde{c}_i \|\mathbf{x}_i - \mathbf{x}_i^k\|_2^2$, where $\tilde{c}_i$ is a positive constant.

Using $\hat{\mathbf{x}}_i(\mathbf{x}^k)$ defined in (2), the synchronous parallel update of each block $\mathbf{x}_i^k$ to $\mathbf{x}_i^{k+1}$ is performed taking a step $\gamma^k > 0$ from $\mathbf{x}_i^k$ along the direction $\hat{\mathbf{x}}_i(\mathbf{x}^k) - \mathbf{x}_i^k$, which reads

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \gamma^k \left(\hat{\mathbf{x}}_i(\mathbf{x}^k) - \mathbf{x}_i^k \right), \quad \forall i = 1, \dots, N. \qquad (3)$$

The step-size γ^k can be chosen according to classical diminishing rules with an additional safeguard, see Sec. 2.2 for the precise conditions. We will also consider rules that do not drive the stepsize to zero; which are useful to obtain iteration complexity results.

Parallel asynchronous updates: core-based model. We break now the synchronization enforced in the parallel updates (3) by allowing the cores to update the block-components $\mathbf{x}_i$ in an independent and asynchronous fashion, without any form of locking. Our asynchronous model, at a core level, is described in Algorithm 1, and termed Asynchronous FLexible ParallEl Algorithm (AsyFLEXA). At any time, a core c can update a block-component $\mathbf{x}_i$ of $\mathbf{x}^k$ chosen at random in the set $\mathcal{N} \triangleq \{1, \dots, N\}$, thus generating the vector $\mathbf{x}^{k+1}$. To update block i, core c: i) reads the inconsistent iterate $\tilde{\mathbf{x}}_c^k$; ii) computes $\hat{\mathbf{x}}_i(\tilde{\mathbf{x}}_c^k)$ solving the strongly convex optimization problem (2); iii) reads the current iterate $\mathbf{x}_i^k$, which may be different from what it cached, $(\tilde{\mathbf{x}}_c^k)_i$, because other cores might have updated $\mathbf{x}$ in the interim; and iv) writes $\mathbf{x}_i^{k+1}$ according to (4). Note that we do not restrict the $(p_c^k)_i$ to any specific distribution (cf. Assumption C).

Parallel asynchronous updates: global model. As pointed out also in [22, Ch. 6] the analysis of asynchronous algorithms is a challenging task. Our approach to analyze Algorithm 1 is to build a "global description" of the the method, where all updates performed by the cores are seen as indistinguishable. The global model is formally given by Algorithm 2. At this higher level, an iteration $k \to k+1$ is defined as the time at which a block i is updated by some core using an inconsistent read $\mathbf{x}^{k-\mathbf{d}} \triangleq (\mathbf{x}_1^{k-d_1}, \dots, \mathbf{x}_N^{k-d_N})$, where $\mathbf{d} \triangleq (\mathbf{d}_i)_{i=1}^N$ is the vector of delays. Note that we are no longer interested in which core has performed the aforementioned update (we thus omitted the dependence on c). Of course different cores would perform different updates because they would be using different inconsistent vectors $\mathbf{x}^{k-\mathbf{d}}$. Roughly speaking, this source of randomness is captured by our model assuming that the pair index-delay $(i, \mathbf{d})$ used at each iteration k to update $\mathbf{x}^k$ is a realization of a random vector $\underline{\boldsymbol{\omega}}^k \triangleq (\underline{i}^k, \underline{\mathbf{d}}^k)$, taking values on $\mathcal{N} \times \mathcal{D}$ with some probability $p_{i,\mathbf{d}}^k \triangleq \mathbb{P}((\underline{i}^k, \underline{\mathbf{d}}^k) = (i, \mathbf{d}))$, where $\mathcal{D}$ is the set of all possible delay vectors. Since each delay $d_i \le \tau$ (see Assumption C below), $\mathcal{D}$ is the set of all possible N-length vectors whose components are integers between 0 and τ. More formally, let Ω be the sample space of all the sequences $\{(i^k, \mathbf{d}^k)\}_k$, and let us define the discrete-time, discrete-value stochastic process $\underline{\boldsymbol{\omega}}$, where $\{\underline{\boldsymbol{\omega}}^k(\omega)\}_k$ is a sample path of the process. The k-th entry $\underline{\boldsymbol{\omega}}^k(\omega)$ of $\underline{\boldsymbol{\omega}}(\omega)$—the k-th element of the sequence ω—is a realization of the random vector $\underline{\boldsymbol{\omega}}^k = (\underline{i}^k, \underline{\mathbf{d}}^k) : \Omega \mapsto \mathcal{N} \times \mathcal{D}$. This process fully describes the evolution of Algorithm 2. Indeed, given a starting point $\mathbf{x}^0$, the trajectories of the variables $\mathbf{x}^k$ and $\mathbf{x}^{k-\mathbf{d}}$ are completely determined once a sample path $\{(i^k, \mathbf{d}^k)\}_k$ is drawn by $\underline{\boldsymbol{\omega}}$. The stochastic process $\underline{\boldsymbol{\omega}}$ is fully defined once the joint finite-dimensional probability mass functions $p_{\underline{\boldsymbol{\omega}}^{0:k}}(\omega^{0:k}) \triangleq \mathbb{P}(\underline{\boldsymbol{\omega}}^{0:k} = \omega^{0:k})$ are given, for all admissible tuples $(\omega^0, \dots, \omega^k)$ and k. In fact, this joint distribution induces a valid probability space $(\Omega, \mathcal{F}, P)$ over which $\underline{\boldsymbol{\omega}}$ is well-defined and has $p_{\underline{\boldsymbol{\omega}}^{0:k}}$ as its finite-dimensional distributions. The probabilities $p_{i,\mathbf{d}}^k$ appearing explicitly in Algorithm 2 are just the marginal distributions of $p_{\underline{\boldsymbol{\omega}}^{0:k}}$, and thus given by $p_{i,\mathbf{d}}^k = \sum_{(\omega^0, \dots, \omega^{k-1})} p_{\underline{\boldsymbol{\omega}}^{0:k}}(\omega^0, \dots, \omega^{k-1}, (i, \mathbf{d}))$. We remark that, since Algorithm 2 is just a descriptive model, these probabilities need not be known, they are instead the result of the specific implementation of Algorithm 1. Finally, we need to define the conditional probabilities $p((i, \mathbf{d}) \,|\, \boldsymbol{\omega}^{0:k}) \triangleq \mathbb{P}(\underline{\boldsymbol{\omega}}^{k+1} = (i, \mathbf{d}) | \underline{\boldsymbol{\omega}}^{0:k} = \boldsymbol{\omega}^{0:k})$. We only require some minimal conditions on the model as stated next.

Assumption C (On the global model, Algorithm 2). Suppose that

- **(C1)** There exists a $\tau \in \mathbb{N}_+$, such that $d_i^k \le \tau$, for all i and k;
- **(C2)** For all $i = 1, \dots, N$ and $\boldsymbol{\omega}^{0:k-1}$ such that $p_{\underline{\boldsymbol{\omega}}^{0:k-1}}(\boldsymbol{\omega}^{0:k-1}) > 0$, it holds $\sum_{\mathbf{d} \in \mathcal{D}} p((i, \mathbf{d}) \,|\, \boldsymbol{\omega}^{0:k-1}) \ge p_{\min}$, for some $p_{\min} > 0$;
- **(C3)** $\mathbb{P}\left(\{\omega \in \Omega : \liminf_{k \to \infty} p(\boldsymbol{\omega} \,|\, \boldsymbol{\omega}^{0:k-1}) > 0\}\right) = 1$.

These are quite reasonable assumptions. C1 just limits the age of the old information used in the updates. Condition C2 guarantees that at each iteration every block index i has a positive probability

Algorithm 1 (AsyFLEXA)

Initialization: $k = 0$, $\mathbf{x}^0 \in \mathcal{X}$, $\{\gamma^k\}_k$, $(p_i^{k,c})_{i,c}$.
while a termination criterion is not met, every core c asynchronously and continuously **do**
(S.1): Select a block-index i with probability $(p_c^k)_i$;
(S.2): Compute $\hat{\mathbf{x}}_i(\tilde{\mathbf{x}}_c^k)$ [cf. (2)] using the cached inconsistent iterate $\tilde{\mathbf{x}}_c^k$;
(S.3): Read $\mathbf{x}_i^k$;
(S.4): Update $\mathbf{x}_i$ by setting

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \gamma^k (\hat{\mathbf{x}}_i(\tilde{\mathbf{x}}_c^k) - \mathbf{x}_i^k); \qquad (4)$$

(S.5): Update the iteration counter $k \leftarrow k+1$;
end while

Algorithm 2 (AsyFLEXA: A Global Description)

Initialization: $k = 0$, $\mathbf{x}^0 \in \mathcal{X}$, $\{\gamma^k\}_k$.
while a termination criterion is not met **do**
(S.1): The random variables $(\underline{i}^k, \underline{\mathbf{d}}^k)$ take realization $(i, \mathbf{d})$;
(S.2): Compute $\hat{\mathbf{x}}_i(\tilde{\mathbf{x}}^{k-\mathbf{d}})$ [cf. (2)];
(S.3): Read $\mathbf{x}_i^k$;
(S.4): Update $\mathbf{x}_i$ by setting

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \gamma^k(\hat{\mathbf{x}}_i(\tilde{\mathbf{x}}^{k-\mathbf{d}}) - \mathbf{x}_i^k); \quad (5)$$

(S.5): Update the iteration counter $k \leftarrow k + 1$;
end while

to be updated. Finally, C3 simply says, roughly speaking, that the probability of the event that comprises all $\boldsymbol{\omega}$ such that for an infinite number of iterations the algorithm picks-up a pair index-delay whose probability decreases to zero, must have zero probability.

2.1. Examples and special cases

The proposed model encompasses several instances of practical interest, some of which are discussed next. These examples show that we can easily recover settings analyzed in the literature but also deal with settings that to date could not be reliably analyzed.

Deterministic sequential cyclic Block-Coordinate Descent (BCD) method: Suppose that there is only one core that cyclically updates all block-variables in a given order; for simplicity we assume the natural order, from 1 to N. Since there is only one core, the reading is always consistent and there are no delays: $\mathcal{D} = \{\mathbf{0}\}$. This is a limit case, but it is very interesting that it fits into our framework.

Randomized sequential BCD method: Suppose that there is only one core, but differently from the previous case, at each iteration, the core selects randomly an index i with a positive (bounded away from zero) probability. This is still a special case of our model, with $\tilde{\mathbf{x}}^k = \mathbf{x}^k$ or, equivalently, $\mathcal{D} = \{\mathbf{0}\}$.

Asynchronous BCD with inconsistent read: Consider a generic shared memory system. Since we do not make any particular assumption, the set $\mathcal{D}$ is given by all N-length vectors whose components are any non negative integers between 0 and τ. Suppose that, at every k, all cores select an index uniformly at random, but we leave the possibility that the probabilities with respect to delays are different. In other words, we suppose that for every given $k \geq 0$, $\boldsymbol{\omega}^{0:k}$, and $i \in \mathcal{N}$, we have $\sum_{\mathbf{d}\in\mathcal{D}} p((i,\mathbf{d}) \,|\, \boldsymbol{\omega}^{0:k}) = 1/N$.Note that this setting corresponds to the one analyzed in [13,14]. However, we can easily change the values of the $\sum_{\mathbf{d}\in\mathcal{D}} p((i,\mathbf{d}) \,|\, \boldsymbol{\omega}^{0:k})$ and make the probabilities of selecting indices different one from the other and/or depend on the iteration and/or on the history of the process; these options are not available to [9,13,14]. This possibility has important practical ramifications, since the assumption that the indices are selected with uniform probability is extremely strong. We make here another example hinting at this difficulty. Consider a message passing system; it will be in general true that cores performing operations are different and on top, the number of block-variables dealt by each core may also be different; in this situation uniformity of the probabilities of selecting an index cannot be expected.

Asynchronous BCD with inconsistent read and block-partitioning: This is the setting most often used in numerical experiments, since it has proven to be most effective in practice; it also models a message passing architecture. Suppose we have C cores and that we partition block-variables in C groups $I_1, I_2, \ldots, I_C$. Each block I_c is assigned to core c and only core c can update block-variables in I_c. Our mathematical translation of this setting is exactly the one in the example above with the additional provision of setting for all $\boldsymbol{\omega}^{0:k}$ and i, with $i \in I_c$, $p((i,\mathbf{d}) \,|\, \boldsymbol{\omega}^{0:k}) = 0$, if even just a $\mathbf{d}_j$, $j \in I_c$, is not zero. In fact, since no core $c' \neq c$ can update the variables in I_c, all this variables can always be assumed to have zero delay. Note that in [13] the numerical experiments are carried out for a *without replacement* setting which is just a variant of this setting and for which no theoretical analysis is offered in [13].

2.2. Convergence and iteration complexity

We provide now the main convergence result. In order to do so, we need some conditions on the stepsize sequence $\{\gamma^k\}_k$.

Assumption D (On the stepsize). $\{\gamma^k\}_k$ is chosen so that

(D1) $\gamma^k \downarrow 0$; $\sum_{k=0}^{\infty} \gamma^k = +\infty$; and $\sum_{k=0}^{\infty} (\gamma^k)^2 < +\infty$;

(D2) $\gamma^{k+1}/\gamma^k \geq \eta \in (0,1)$.

Conditions in D1 are quite standard and satisfied by most practical diminishing stepsize rules; see, e.g., [22]. D2 is less standard, but it is however very mild and satisfied by most classical choices of γ^k in diminishing stepsize methods (e.g. monotonically decreasing rules).

We can now state the main convergence result of Algorithm 2, whose proof is omitted because of the space limitation, see [23].

Theorem 1 *Consider Algorithm 2 and the stochastic process $\underline{\boldsymbol{\omega}}$. Given the initial point $\mathbf{x}^0 \in \mathcal{X}$, let $\{\mathbf{x}^k\}_k$ be the sequence generated by the algorithm. Suppose that i) Assumptions A-D hold; and ii) $\{\mathbf{x}^k\}$ is bounded almost surely (a.s.). Then the following hold a.s.: (a) Every limit point of $\{\mathbf{x}^k\}_k$ is stationary for Problem* (1)*; and (b) The sequence of objective function values converges.*

In addition to Assumptions A-D, Theorem 1 requires the sequence $\{\mathbf{x}^k\}_k$ to be bounded a.s.. The following corollary provides some concrete conditions for this boundedness to hold.

Corollary 2 *The same conclusions of Theorem 1 hold if assumption ii) is replaced by the following: F is coercive on $\mathcal{X}$ and $\mathbf{x}_i^k = \tilde{\mathbf{x}}_i^k$, for any $i \in \mathcal{N}$ and $k \geq 0$.*

We note that the equality $\mathbf{x}_i^k = \tilde{\mathbf{x}}_i^k$ is automatically satisfied in a message passing setting or in a shared memory-based architecture if the variables are partitioned and assigned to different cores.

The proposed framework represents a gamut of algorithms, each of them corresponding to a specific choice of the surrogate functions $\tilde{f}_i$ and stepsize γ^k, all converging under the same conditions; see [23] for a discussion of several instances of our framework.

Pseudo-deterministic block selection: The algorithmic framework introduced so far uses random selection rules for the updates of the blocks. However, all convergence results can be extended to the case in which a pseudo-deterministic selection rule of the blocks is performed. In this setting, we do not make any probabilistic assumption, but simply assume that, in Step S.1 of Algorithm 2, i and $\mathbf{d}$ are such that C1 and the following modification of C2 holds:

(C2') Every block is updated every $B \leq \tau$ iterations, i.e., $\forall k$, i, $\exists j \in \{k, k+1, \ldots, k+B\}$ such that $i^j = i$, where i^j denotes the index of the component updated at the jth iteration.

Note that the pseudo-deterministic rules are not special cases of the random selection ones, since the latter do not satisfy condition C2'. It can be shown that the conclusions of Theorem 1 still hold, in a deterministic sense, for this variant. Furthermore, one can also estimate an iteration complexity bound, as given next. We first introduce the following stationarity measure $M_F(\mathbf{x}) = \mathbf{x} - \text{argmin}_{\mathbf{y}\in\mathcal{X}}\{\nabla f(\mathbf{x})^{\mathrm{T}}(\mathbf{y} - \mathbf{x}) + g(\mathbf{y}) + (1/2)\cdot\|\mathbf{y}-\mathbf{x}\|_2^2\}$. M_F is a continuous function whose value is zero if and only if $\mathbf{x}$ is a stationary point. We term a point $\mathbf{x}^*$, an ϵ-solution of (1) if $\|M_F(\mathbf{x}^*)\|_2^2 \leq \epsilon$. The following deterministic version of Theorem 1 holds.

Theorem 3 *[24] Consider Algorithm 2 wherein in Step 2 the selection rule for i and $\mathbf{d}$ is such that Assumptions C1 and C2' are satisfied. Assume further that i) Assumptions A and B are satisfied; ii) F*

is coercive on $\mathcal{X}$; and iii) γ^k is a nonincreasing sequence of scalars satisfying $\liminf_k \gamma^k > \eta$, and $\gamma^k < c_{\tilde{f}}/(L_f + (\tau^2 L_f)/2)$, for all k. Given $\mathbf{x}^0 \in \mathcal{X}$, let $\{\mathbf{x}^k\}_k$ be the sequence generated by the algorithm. Then (a) Every limit point of $\{\mathbf{x}^k\}_k$ is stationary for Problem (1)*; and (b) The sequence of objective function values converges; (c) An ϵ-solution of* (1) *is achieved in $O(1/\epsilon)$ iterations.*

It is possible to obtain complexity results also in the case of random selection rules (under additional assumptions); see [24] for details.

3. NUMERICAL RESULTS

In this section we test our algorithms on a convex instance of Problem (1)−the LASSO problem−and a nonconvex one−a quadratic nonconvex problem. All codes were written in C++ using OpenMP. The algorithms were tested on a machine with two 10-Core Intel Xeon-E5 processors (20 cores in total) and 256 GB of RAM. In all the experiments, the starting point was the zero vector.

Example 1: LASSO. Consider the LASSO problem, Problem (1), with $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $G(\mathbf{x}) = \lambda\|\mathbf{x}\|_1$, and $\mathcal{X} = \mathbb{R}^n$, with $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$, and $\lambda > 0$.

We tested AsyFLEXA in the following setting. We set $N = n$ (scalar blocks) and we partitioned the variables among the cores, with only one core per partition. We chose this implementation because it was shown to be particularly effective; see, e.g., [8]. At each iteration, each core selects an index within its partition uniformly at random. The surrogate functions $\tilde{f}_i$ were chosen as $\tilde{f}_i(x_i; \tilde{\mathbf{x}}^k) = f(x_i; \tilde{\mathbf{x}}^k_{-i}) + (\tau^k/2)\cdot(x_i - x_i^k)^2$, where the sequence $\{\tau^k\}_k$, shared among all the cores, was updated every n iterations, according to the heuristic proposed in FLEXA [4]. Note that in this setting, the unique solution $\hat{x}_i$ of each subproblem (2) can be computed in closed form using the soft-thresholding operator [3]. For the stepsize sequence $\{\gamma^k\}_k$ we used the rule $\gamma^{k+1} = \gamma^k(1 - \mu\gamma^k)$ with $\gamma^0 = 1$ and $\mu = 10^{-6}$, which satisfies Assumption D. We compared AsyFLEXA with the state-of-the-art asynchronous schemes AsySPCD [13] and ARock [21]. For these competitors the use of stepsizes that guarantee theoretical convergence leads to very poor practical performance and the algorithms simply do not make any practical progress towards optimality in a reasonable number of iterations. So, we chose values of the free parameters that violate the thresholds set by the analysis in [13,21] for the convergence. For the stepsize of AsySPCD we used $\gamma = 1$ while for the one of ARock we used the same rule employed in our AsyFLEXA, with a safeguard that guarantees that the stepsize never becomes smaller than 0.1. We remark that, in this partitioned setting, our AsyFLEXA is the only algorithm with convergence guarantees (cf. Corollary 2).

We generated the LASSO problem using the random generator proposed by Nesterov in [1], which permits to control the sparsity of the solution. We considered problems with 40000 variables and matrix $\mathbf{A}$ having 20000 rows, and set $\lambda = 1$; the percentage of nonzero in the solution is 1%. In the implementation of all the algorithms, we computed the matrix $\mathbf{A}^T\mathbf{A}$ and the vector $\mathbf{A}^T\mathbf{b}$ offline. In the left panel of Figure 1 we plot the relative error on the objective function versus the CPU time, using 2 and 20 cores. The curves are averaged over five independent problem realizations. The figure shows that AsyFLEXA significantly outperforms all the other algorithms. Moreover, at difference with the other algorithms, its empirical convergence speed significantly increases with the number of cores. This is mainly due to the facts that i) AsyFLEXA is not a standard proximal-gradient method; ii) AsyFLEXA does not use a stepsize determined by the parameters of f while AsySPCD and ARock use a stepsize determined by the Lipschitz constant of $\nabla_{\mathbf{x}_i} f$ and ∇f, respectively, which drastically slows down the algorithm. In order to quantify the scalability of the algorithms, in the right panel of Figure 1 we plot the speedup achieved by each of the algorithms versus the number of cores for the LASSO problem. We defined the speedup as the ratio between the runtime on a single core and the runtime on multiple cores. The runtimes we used are the CPU times needed to reach a relative error less than 10^{-5} for AsyFLEXA and less than 10^{-1} for AsySPCD and ARock. We used different thresholds because AsySPCD and ARock do not reach a relative error as small as AsyFLEXA; however the comparison in terms of speedup is fair since all the algorithms run for a considerable amount of time. Note that while AsyFLEXA and ARock show a good speedup, near to linear up to 10 cores, AsySPCD lags behind.

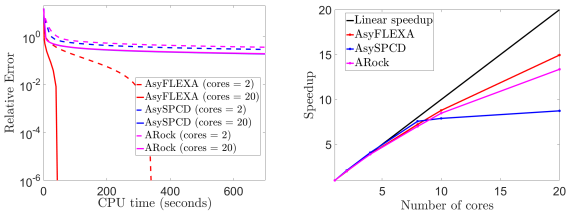

Fig. 1: (Left) LASSO: Relative error versus CPU time. (Right) LASSO: Speedup.

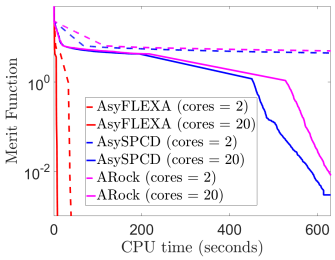

Fig. 2: Quadratic nonconvex problem: Stationarity distance versus CPU time.

Example 2: Nonconvex Quadratic Problem. We consider now the following nonconvex instance of problem (1):

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & F(\mathbf{x}) \triangleq \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 - \frac{\bar{c}}{2}\|\mathbf{x}\|_2^2 + c\|\mathbf{x}\|_1 \\ \text{s.t.} & -b \le x_i \le b \quad i = 1, \ldots, n \end{array} \tag{6}$$

where c is a positive constant chosen so that (6) is nonconvex. In particular, we set $\bar{c} = 1000$, $c = 100$, $b = 1$, $n = 10000$ and the matrix $\mathbf{A}$ having 9000 rows was generated using the Nesterov's method as in the LASSO problem. The tuning of the algorithms is the same as for the LASSO problem, except that for AsyFLEXA we set $\tau^k > \bar{c}$, for all k, so that each surrogate function $\tilde{f}_i$ is strongly convex. Since (6) is nonconvex, we compare the performances of the algorithms using as a merit function the distance from stationarity described in detail in [4, Sec. VI-C]. In Fig. 2 we plot the stationarity measure versus the CPU time, for all the algorithms; the curves are averaged over five independent realizations. All the algorithms were observed to converge to the same stationary solution of (6). Note, however, that for AsySPCD and ARock there is no formal proof of convergence in this nonconvex setting. The figure shows that, even in the nonconvex case, AsyFLEXA outperforms the other algorithms.

4. CONCLUSIONS

We proposed a novel SCA-based algorithm for the parallel asynchronous minimization of the sum of a nonconvex smooth function and a convex nonsmooth one. The underlying probabilistic model captures the essential features of modern multi-core architectures by providing a more realistic description of lock-free implementations subject to inconsistent read. We establish almost sure convergence of the proposed scheme. Preliminary numerical results indicate that our algorithm outperforms current asynchronous schemes on both convex and nonconvex problems.

5. REFERENCES

[1] Yu Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, vol. 140, pp. 125–161, August 2013.

[2] Paul Tseng and Sangwoon Yun, "A coordinate gradient descent method for nonsmooth separable minimization," *Mathematical Programming*, vol. 117, no. 1-2, pp. 387–423, March 2009.

[3] Amir Beck and Marc Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, Jan. 2009.

[4] F. Facchinei, G. Scutari, and Simone Sagratella, "Parallel selective algorithms for nonconvex big data optimization," *IEEE Transactions on Signal Proces.*, vol. 63, no. 7, pp. 1874–1889, April 2015.

[5] A. Daneshmand, F. Facchinei, V. Kungurtsev, and G. Scutari, "Hybrid random/deterministic parallel algorithms for convex and nonconvex big data optimization," *IEEE Transactions on Signal Proces.*, vol. 63, no. 13, pp. 3914–3929, August 2015.

[6] Paolo Di Lorenzo and Gesualdo Scutari, "Next: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 120–136, 2016.

[7] A. Nedić, D. P. Bertsekas, and V. S. Borkar, "Distributed asynchronous incremental subgradient methods," *Studies in Computational Mathematics*, vol. 8, pp. 381–407, 2001.

[8] B. Recht, C. Re, S. J. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[9] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2719–2727.

[10] Z. Huo and H. Huang, "Asynchronous stochastic gradient descent with variance reduction for non-convex optimization," *arXiv preprint arXiv:1604.03584*, 2016.

[11] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan, "Perturbed iterate analysis for asynchronous stochastic optimization," *arXiv:1507.06970*, 2016.

[12] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar, "An asynchronous parallel stochastic coordinate descent algorithm," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 285–322, 2015.

[13] Ji Liu and Stephen J Wright, "Asynchronous stochastic coordinate descent: Parallelism and convergence properties," *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 351–376, 2015.

[14] D. Davis, "The asynchronous palm algorithm for nonsmooth nonconvex problems," *arXiv preprint arXiv:1604.00526*, 2016.

[15] M. Hong, "A distributed, asynchronous and incremental algorithm for nonconvex optimization: An admm based approach," *arXiv preprint arXiv:1412.6058*, 2014.

[16] E. Wei and A. Ozdaglar, "On the o (1= k) convergence of asynchronous distributed alternating direction method of multipliers," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 2013, pp. 551–554.

[17] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, "Asynchronous distributed optimization using a randomized alternating direction method of multipliers," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 3671–3676.

[18] Patrick L Combettes and Jonathan Eckstein, "Asynchronous block-iterative primal-dual decomposition methods for monotone inclusions," *arXiv preprint arXiv:1507.03291*, 2015.

[19] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *Journal of the ACM (JACM)*, vol. 25, no. 2, pp. 226–244, 1978.

[20] A. Frommer and D. B. Szyld, "On asynchronous iterations," *Journal of computational and applied mathematics*, vol. 123, no. 1, pp. 201–216, 2000.

[21] Z. Peng, Y. Xu, M. Yan, and W. Yin, "Arock: an algorithmic framework for asynchronous parallel coordinate updates," *arXiv preprint arXiv:1506.02396*, 2015.

[22] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23, Prentice hall Englewood Cliffs, NJ, 1989.

[23] L. Cannelli, G. Scutari, F. Facchinei, and V. Kungurtsev, "Asynchronous parallel algorithms for nonconvex big-data optimization: Model and convergence," *Technical Report, Purdue University*, September 2016. Available online at: `http://web.ics.purdue.edu/~lcannell/AsyFLEXA-TR_I.pdf`.

[24] L. Cannelli, G. Scutari, F. Facchinei, and V. Kungurtsev, "Asynchronous parallel algorithms for nonconvex big-data optimization: Complexity and numerical results," *Technical Report, Purdue University*, September 2016. Available online at: `http://web.ics.purdue.edu/~lcannell/AsyFLEXA-TR_II.pdf`.