# A Totally Asynchronous Block-Based Heavy Ball Algorithm for Convex Optimization

Dawn M. Hustig-Schultz, Katherine Hendrickson, Matthew Hale, and Ricardo G. Sanfelice

***Abstract*— We present a totally asynchronous multiagent algorithm, based on the heavy ball method, that guarantees fast convergence to the minimizer of a twice continuously differentiable, convex objective function over a convex constraint set. The algorithm is parallelized in the sense that the decision variable is partitioned into blocks, each of which is updated by only a single agent. We consider two types of asynchrony: in agents' computations and in communications between agents, both under arbitrarily long delays. We show that, for certain values of the step size and other parameters, the heavy ball algorithm exponentially converges to a minimizer, even under total asynchrony. Numerical results validate these findings and demonstrate significantly faster convergence than a comparable gradient descent algorithm.**

## I. Introduction

Large-scale optimization problems arise in engineering applications including machine learning, signal processing, robotics, and others [1], [2], [3], [4]. As these problems grow in size, parallelized algorithms have been developed to use collections of agents, e.g., networks of computer processors, to solve them faster than they can be solved in a non-parallelized way. Parallelized algorithms decentralize computations among agents, and agents exchange values of their decision variables over time.

It can be difficult to synchronize agents' communications and computations over time, e.g., due to different clock speeds driving computations or congested bandwidth when communicating. This has driven interest in algorithms that are inherently robust to asynchrony. One class of such algorithms is termed *partially asynchronous* [5, Chapter 7], which refers to algorithms that converge if all delays in communications and computations are bounded by some known constant. Algorithms from this class include [6], [7], [8], [9], [10], [11]. It may be difficult, however, for agents to know a bound on all delays in all agents' computations and communications, and, indeed, such a bound may not even exist. In such cases, one can ensure the convergence of an algorithm by designing it for *total asynchrony* [5, Chapter 6], namely, the presence of unbounded delays in communications and computations. The development of such algorithms dates back several decades [12], though, to date, totally asynchronous algorithms are primarily variants of gradient descent. Accelerated algorithms converge faster by making each agent's computations take larger steps towards an optimum. Doing so is of special importance in totally asynchronous settings because agents' computations may be sporadic and infrequent.

Motivated by the lack of accelerated algorithms in totally asynchronous settings, we develop a totally asynchronous version of the heavy ball algorithm. The heavy ball algorithm is appealing because it converges exponentially fast for strongly convex functions [13], [14] and has a rate up to $\mathcal{O}\left(\frac{1}{(k+1)^2}\right)$ for some convex functions [15]. One typically establishes convergence of totally asynchronous algorithms by showing that an update law is a contraction with respect to some block-maximum norm [5, Section 6.3]. The heavy ball update does not necessarily have such a contraction, but under suitable conditions on algorithm parameters, we show that applying the heavy ball update law twice does provide an infinity norm contraction. Due to this property, we propose a totally asynchronous algorithm that, at each step, executes two consecutive updates of a projected heavy ball algorithm for each agent's computation. We show that this algorithm has an exponential convergence rate under the assumption that the objective function is twice continuously differentiable and its Hessian is diagonally dominant[1]. Then we demonstrate in simulation that our algorithm is substantially faster than a comparable totally asynchronous gradient descent algorithm. Related work has developed decentralized synchronous [16], [17] and partially asynchronous versions of accelerated algorithms [18], [19], [20], [21]. The key contribution of our algorithm is total asynchrony, i.e., delays that do not obey a specified upper bound.

The proposed algorithm is block-based, in the sense that each agent only updates a subset of the decision variables in a problem, termed a *block*, and each decision variable is updated by only a single agent. Block-based algorithms date back several decades [12], [22], and have been shown to tolerate total asynchrony for gradient descent applied to some unconstrained problems [12], [23], [24]. For constrained problems, block-based methods have been utilized

D. M. Hustig-Schultz and R. G. Sanfelice are with the Department of Electrical and Computer Engineering, University of California, 1156 High Street, Santa Cruz, CA 95064, USA. Emails: dhustigs@ucsc.edu, ricardo@ucsc.edu. Katherine Hendrickson and Matthew Hale are with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611. Emails: kat.hendrickson@ufl.edu, matthewhale@ufl.edu.
K. Hendrickson and M. Hale were supported by AFOSR under grant FA9550-19-1-0169, ONR under grants N00014-19-1-2543 and N00014-21-1-2495, and by a task order contract from the AFRL Munitions Directorate, Eglin AFB. R. G. Sanfelice and D. M. Hustig-Schultz were partially supported by NSF Grant nos. ECS-1710621, CNS-2039054, and CNS-2111688, by AFOSR Grant nos. FA9550-19-1-0169, FA9550-20-1-0238, and FA9550-23-1-0145, by AFRL Grant nos. FA8651-22-1-0017 and FA8651-23-1-0004, and by ARO Grant no. W911NF-20-1-0253.

[1]It has been observed in [5, Section 6.3.2] that some form of diagonal dominance condition is needed to show convergence of totally asynchronous algorithms, and we therefore include a diagonal dominance condition in this work.

for primal-dual algorithms [25], [26], [27], including totally asynchronous primal-dual algorithms [28]. For the first time, this paper will bring the same tolerance for total asynchrony to set constrained problems solved with an accelerated algorithm. In this preliminary study, we use scalar blocks, i.e., each agent updates a scalar decision variable.

The rest of this paper is organized as follows. Section II gives notation and background. Section III presents the problem statement. Section IV introduces the asynchronous algorithm and its nominal properties. Section V provides simulation results. Due to space constraints, detailed proofs of results will be published elsewhere.

## II. Preliminaries

### A. Notation

We denote the real, positive real, and natural numbers with $\mathbb{R}$, $\mathbb{R}_{>0}$, and $\mathbb{N}$, respectively. For vectors $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^m$, $(v,w) := \left[v^\top, w^\top\right]^\top$, $|v| = \|v\|_2 = \sqrt{v^\top v}$ denotes the Euclidean vector norm of $v$, and $\langle v, w\rangle = v^\top w$ the inner product of $v$ and $w$. The orthogonal projection, with respect to the Euclidean norm, of a vector $v$ onto the convex set $X$ is denoted as $\Pi_X[v] = \arg\min_{w\in X} |w - v|$. We denote the max vector norm as $\|v\|_\infty = \max_i |v_i|$, which is the maximum of the absolute value of its components. The closure of a set $S$ is denoted $\overline{S}$. Given a set-valued mapping $M : \mathbb{R}^m \rightrightarrows \mathbb{R}^p$, the domain of $M$ is the set $\mathrm{dom} M = \{x \in \mathbb{R}^m \ : \ M(x) \neq \emptyset \ \}$.

### B. Properties of Difference Inclusions

In this paper, we consider discrete-time systems with data $(D, G)$ and dynamics defined as

$$z^+ \in G(z) \qquad z \in D \tag{1}$$

where $z \in \mathbb{R}^m$ is the system state, $G : \mathbb{R}^m \rightrightarrows \mathbb{R}^m$ is the right-hand side, and $D \subset \mathbb{R}^m$ is the constraint set. The notation $\rightrightarrows$ indicates that $G$ is a set-valued map. The following definition, from [29], [30], [31], is used in the analysis of the proposed algorithms.

*Definition 2.1 (Basic conditions):* System (1) is said to satisfy the basic conditions if its data $(D, G)$ is such that

(A1) $D$ is a closed subset of $\mathbb{R}^m$;

(A2) $G : \mathbb{R}^m \rightrightarrows \mathbb{R}^m$ is outer semicontinuous and locally bounded relative to $D$, and $D \subset \mathrm{dom}\ G$.

## III. Problem Statement

This paper solves the following problem.

*Problem 1:* Given $f : \mathbb{R}^n \to \mathbb{R}$ and $X \subset \mathbb{R}^n$, design an optimization algorithm using accelerated methods that asynchronously solves $\min_{\xi\in X} f(\xi)$. Such an algorithm must have an exponential convergence rate and show improvement over gradient descent at least in simulation. $\triangle$

To solve Problem 1, we design a totally asynchronous algorithm with exponential convergence rate. The algorithm is designed in three steps. First, we design a synchronous algorithm in which each agent computes a single constrained heavy ball update and communicates with its neighbors once every iteration. Next, we devise a synchronous algorithm in which, during each iteration, each agent computes two constrained heavy ball updates and communicates once with its neighbors. We refer to this algorithm as a double-update algorithm. Then, as a third step, we leverage the framework in [5] to establish that, since the synchronous algorithm has exponential convergence and satisfies the "Synchronous Convergence and Box Conditions" therein, the totally asynchronous algorithm also has an exponential convergence rate. Due to space constraints, the intermediate algorithms will be published elsewhere.

## IV. Asynchronous, Double-Update Heavy Ball

### A. Modeling

In its centralized form, the heavy ball algorithm is

$$\xi(k+1) = \Pi_X\left[\xi(k) - \gamma\nabla f\big(\xi(k)\big) + \lambda\big(\xi(k) - \xi(k-1)\big)\right] \tag{2}$$

for $k \in \mathbb{N}$, where $\xi(k) \in \mathbb{R}^n$ is the state of the algorithm at discrete time $k$, and $\lambda > 0$ and $\gamma > 0$ are tunable parameters representing friction and gravity, respectively [32]. We interpret (2) as a control system consisting of a plant and a control algorithm. Let $z_1 := \xi(k)$, $z_2 := \xi(k-1)$, and $z := (z_1, z_2)$. Then, the plant associated with (2) is given by

$$\begin{bmatrix} z_1^+ \\ z_2^+ \end{bmatrix} = \begin{bmatrix} u \\ z_1 \end{bmatrix} =: G_P(z,u) \quad (z,u) \in X \times X \times \mathbb{R}^n =: D_P \tag{3}$$

with output $y = z$. The control algorithm leading to (2) is $u = \kappa(z) := \Pi_X\left[z_1 - \gamma\nabla f(z_1) + \lambda(z_1 - z_2)\right]$.

To extend (2) to a multiagent setting, allowing arbitrary asynchrony for the agents, we start by extending techniques used in [28] for parallelized multiagent gradient descent. The term "parallelized" means that each decision variable is updated only by a single agent, with each decision variable assigned to each agent referred to as a "block."

We consider $N \in \mathbb{N}_{>0}$ agents indexed by $i \in \mathcal{V} := \{1, 2, \ldots, N\}$. We use $\mathcal{N}_i \subset \mathcal{V}$ to denote the set of essential neighbors of agent $i$, i.e., $\mathcal{N}_i$ is the set of indices of agents $\ell \neq i$ whose decision variables are needed for agent $i$'s computations. More formally, agent $\ell \in \mathcal{N}_i$ is an essential neighbor of agent $i$ if $\nabla_i f$ explicitly depends upon agent $\ell$'s decision variable. Only the essential neighbors $\ell \in \mathcal{N}_i$ need to communicate with agent $i$ to ensure it has the information necessary to compute gradients. Agents exchange information over an undirected graph $\Gamma = (\mathcal{V}, \mathcal{E})$, where edges are pairs in the set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ that directly link essential neighbors. In particular, an edge from agent $\ell$ to agent $i$, denoted $(\ell, i) \in \mathcal{E}$, implies that $\ell$ and $i$ are essential neighbors and agent $\ell$ can send information to agent $i$. Since the graph $\Gamma$ is undirected, agent $i$ is an essential neighbor of agent $\ell$ if and only if agent $\ell$ is an essential neighbor of agent $i$. Conversely, the lack of an edge from agent $\ell$ to agent $i$ (and, hence, $i$ to $\ell$) implies that $\ell$ and $i$ are not essential neighbors and do not communicate.

We use superscripts to denote ownership by an agent, and use subscripts for indexing. For instance, we denote the vector containing agent $i$'s local copy of all decision variables as

$z^i$, and we denote the constraint set corresponding to agent $i$'s decision variable (denoted $\left(z^i_{1,i}, z^i_{2,i}\right)$) as $X_i \times X_i$; see definitions below. With this in mind, we impose the following assumption on the constraint set $X$.

*Assumption 4.1 (Properties of the constraint set):* The constraint set $X \subset \mathbb{R}^N$ is nonempty, compact, and convex. The constraint set can be decomposed as

$$X = X_1 \times X_2 \times \ldots \times X_N \tag{4}$$

where, for each $i \in \mathcal{V}$, $X_i \subset \mathbb{R}$.

In the results to follow, we impose the following assumption on the objective function $f$.

*Assumption 4.2:* The function $f$ is twice continuously differentiable and convex.

Additionally, we impose the following diagonal dominance assumption on the Hessian of the objective function $f$.

*Assumption 4.3 (Diagonal dominance):* The $N \times N$ Hessian matrix $x \mapsto H(x) = \nabla^2 f(x)$ is $\mu$-diagonally dominant on $X \subset \mathbb{R}^N$ for some $\mu > 0$. That is, for each $i \in \mathcal{V}$, $|H_{ii}(x)| - \mu \geq \sum_{j=1,\, j\neq i}^{N} |H_{ij}(x)|$ for all $x \in X$.

For each $i \in \mathcal{V}$, agent $i$ stores its own decision variable and a local copy of the decision variables of all other agents for use in local computations. We denote agent $i$'s value for its own decision variable as

$$\left(z^i_{1,i}, z^i_{2,i}\right) \in X_i \times X_i, \tag{5}$$

where $z^i_{1,i}$ plays the role of $z_1$ in (3) and $z^i_{2,i}$ of $z_2$. We denote agent $i$'s local copy of the decision variable for agent $\ell$ as

$$\left(z^i_{1,\ell}, z^i_{2,\ell}\right) \in X_\ell \times X_\ell. \tag{6}$$

Thus, the full state of agent $i$ is

$$z^i := \left(z^i_1, z^i_2\right) \in X \times X \tag{7}$$

where

$$\begin{aligned} z^i_1 &:= (z^i_{1,1}, z^i_{1,2}, ..., z^i_{1,i}, ..., z^i_{1,N}) \\ z^i_2 &:= (z^i_{2,1}, z^i_{2,2}, ..., z^i_{2,i}, ..., z^i_{2,N}). \end{aligned} \tag{8}$$

As in [28], we want to distribute the discrete-time heavy ball algorithm among multiple agents while allowing agents to compute and share information asynchronously. There are two behaviors that could be asynchronous:

1) *Computation of Updates to Agents' Variables:* Individual agents may update their variables at different times. Namely, the subsets of times at which distinct agents $i$ and $\ell$ compute updates are independent;
2) *Communication of Updated Agents' Variables:* Communication of the agents' variables to their essential neighbors is also totally asynchronous. Namely, the subsets of times at which distinct agents $i$ and $\ell$ communicate with essential neighbors are independent.

To allow total asynchrony, we propose an algorithm that uses the update law

$$\tilde{\kappa}^i(z^i) := \Pi_{X_i}\left[\kappa^i(z^i) - \gamma \nabla_i f(w^i_1(z^i_1)) + \lambda(\kappa^i(z^i) - z^i_{1,i})\right] \tag{9}$$

where $\lambda > 0$ and $\gamma > 0$ come from (2), $X_i$ comes from Assumption 4.1, $z^i$ is defined in (7), $z^i_1$ is defined via (8), $\kappa^i$ is defined as

$$\kappa^i(z^i) := \Pi_{X_i}\left[z^i_{1,i} - \gamma \nabla_i f(z^i_1) + \lambda(z^i_{1,i} - z^i_{2,i})\right] \tag{10}$$

and the function $w^i_1$ is defined as $w^i_1(z^i_1) := \big(z^i_{1,1}, z^i_{1,2}, \ldots, \kappa^i(z^i) \ldots, z^i_{1,N}\big) \in X$, where $\kappa^i(z^i)$ is in the $i$-th entry of $w^i_1(z^i_1)$ and $X$ is defined in (4). The function $w^i_1$ collects the first update to the $z^i_{1,i}$ component of agent $i$'s decision variable and collects each of the $z^i_{1,\ell}$ components of agent $i$'s local copies of the decision variables of all other agents, for use in the computation of $\nabla_i f$ in (9). Each agent $i \in \mathcal{V}$ executes $\tilde{\kappa}^i$ to update its decision variable and communicates to other agents at potentially different times, with arbitrarily long delays allowed.

The asynchronous, double-update algorithm for constrained heavy ball is summarized in Algorithm 1. Since computations are totally asynchronous, we denote the set of times at which agent $i$ computes updates as the set $K^i \subset \mathbb{N}$. Note that each $i \in \mathcal{V}$ has its own $K^i$. At each discrete time instant $k \in \mathbb{N}$, if $k \in K^i$ – which defines the time of a *computation event* – the corresponding agent $i$ updates the value of $\left(z^i_{1,i}, z^i_{2,i}\right)$; see lines 4-5 of Algorithm 1. In particular, if $k \in K^i$, then $z^i_{1,i}$ is updated to $\tilde{\kappa}^i(z^i)$ in (9) and $z^i_{2,i}$ is updated to $\kappa^i(z^i)$ in (10). Then, agent $i$ sends $\left(z^i_{1,i}, z^i_{2,i}\right)$ to all agents $\ell \in \mathcal{N}_i$; see line 6 of Algorithm 1. Due to the possibility of communication delays, such information might not be received for some time, and might be received at different times by different agents.

Since communications are totally asynchronous, then we denote the set of times at which agent $i$ receives information from agent $\ell \in \mathcal{N}_i$ as the set $R^{i,\ell} \subset \mathbb{N}$. Note that each $(i, \ell) \in \mathcal{E}$ has its own $R^{i,\ell}$. Then, at each discrete time instant $k \in \mathbb{N}$, if $k \in R^{i,\ell}$ – which defines the time of reception of information, referred to as a *communication event* – agent $i$ updates $\left(z^i_{1,\ell}, z^i_{2,\ell}\right)$ of its state $z^i$ to

$$\left(z^\ell_{1,\ell}(\tau^i_\ell(k)), z^\ell_{2,\ell}(\tau^i_\ell(k))\right) \in X_\ell \times X_\ell, \tag{11}$$

where

$$\tau^i_\ell(k) \in K^\ell \tag{12}$$

denotes the time at which agent $\ell$ originally computed the value of its decision variable onboard agent $i$ at time $k$; see lines 8-12 of Algorithm 1. Note that $\tau^i_i(k) = k$ for all $i \in \mathcal{V}$. For each $\ell$ that is not an essential neighbor of $i$, $(z^i_{1,\ell}, z^i_{2,\ell})$ is left unchanged; see lines 13-15 of Algorithm 1.

To explain how we model the delay between sending and receiving information, and the role of $\tau^i_\ell$ in this delay, we use a simple two-agent example, as follows. Let[2] $i = 1$, $\ell = 2$, $K^1 = \{1\}$, and $R^{2,1} = \{3\}$. At $k = 1$, agent 1 updates $\left(z^1_{1,1}, z^1_{2,1}\right)$ to $\left(\tilde{\kappa}^1(z^1), \kappa^1(z^1)\right)$, and sends these values to agent 2. But these values do not arrive at

[2]Although the sets $K^i$ and $R^{\ell,i}$ are later assumed to be unbounded, we do not consider unboundedness in this example, for simplicity. However, this illustration also applies for unbounded sets $K^1$ and $R^{2,1}$ such that $1 \in K^1$, $0, 2, 3, 4 \notin K^1$, $3 \in R^{2,1}$, and $0, 1, 2, 4 \notin R^{2,1}$.

agent 2 until $k = 3$. When agent 2 receives agent 1's decision variable at $k = 3$, agent 2 updates its value of $\left(z^2_{1,1}, z^2_{2,1}\right)$ to $\left(z^1_{1,1}(\tau^2_1(3)), z^1_{2,1}(\tau^2_1(3))\right)$, where $\tau^2_1(3) = 1 \in K^1$. In this way, $\left(z^2_{1,1}(4), z^2_{2,1}(4)\right) = \left(z^1_{1,1}(2), z^1_{2,1}(2)\right) = \left(\tilde{\kappa}^1(z^1(1)), \kappa^1(z^1(1))\right)$.

**Algorithm 1** Constrained, Asynchronous, Double-Update Heavy Ball

For each $i \in \mathcal{V}$, set the initial state $z^i_\circ$ to an arbitrary value in $X \times X$.
**for** each $k \in \mathbb{N}$ **do**
  **for** each $i \in \mathcal{V}$ **do**
    **if** $k \in K^i$ **then**
      Update $\left(z^i_{1,i}, z^i_{2,i}\right)$ in (5) to $\left(\tilde{\kappa}^i(z^i), \kappa^i(z^i)\right)$, with $\tilde{\kappa}^i$ defined via (9) and $\kappa^i$ defined in (10);
      Agent $i$ sends $\left(z^i_{1,i}, z^i_{2,i}\right)$ to all agents $\ell \in \mathcal{N}_i$. Due to communication delays, $\left(z^i_{1,i}, z^i_{2,i}\right)$ may not be received for some time.
    **end if**
    **for** each $\ell \in \mathcal{N}_i$ **do**
      **if** $k \in R^{i,\ell}$ **then**
        Update $\left(z^i_{1,\ell}, z^i_{2,\ell}\right)$ in (6) to $\left(z^\ell_{1,\ell}(\tau^i_\ell(k)), z^\ell_{2,\ell}(\tau^i_\ell(k))\right)$ in (11).
      **end if**
    **end for**
    **for** each $\ell \notin \mathcal{N}_i$ **do**
      Keep $\left(z^i_{1,\ell}, z^i_{2,\ell}\right)$ constant.
    **end for**
  **end for**
**end for**

Now we model Algorithm 1 mathematically. From Algorithm 1, for each $i \in \mathcal{V}$, agent $i$ has its state $z^i$ in (7) updated via the following difference equation:

$$\left(z^i\right)^+ = G^i_{\text{async}}\left(z^1, z^2, \ldots, z^N\right) \quad z^i \in D^i_{\text{async}} := X \times X, \tag{13}$$

where $G^i_{\text{async}}$ is defined as $G^i_{\text{async}}\left(z^1, z^2, \ldots, z^N\right) := \left(g^i_1\left(z^1, z^2, \ldots, z^N\right), g^i_2\left(z^1, z^2, \ldots, z^N\right)\right)$, where, for each $p \in \{1, 2\}$, $g^i_p$ is defined as $g^i_p\left(z^1, z^2, \ldots, z^N\right) := \left(g^i_{p,1}\left(z^1, z^2, \ldots, z^N\right), g^i_{p,2}\left(z^1, z^2, \ldots, z^N\right), \ldots, g^i_{p,N}\left(z^1, z^2, \ldots, z^N\right)\right)$ and, for each $s \in \mathcal{V}$, $g^i_{p,s}$ is defined as

$$g^i_{1,s}\left(z^1, z^2, \ldots, z^N\right) := \begin{cases} \tilde{\kappa}^i(z^i) & \text{if } s = i, \text{at each computation event} \\ z^s_{1,s} & \text{if } s \in \mathcal{N}_i, \text{at each communication event} \\ z^i_{1,s} & \text{otherwise} \end{cases} \tag{14}$$

if $p = 1$, where $\tilde{\kappa}^i$ is defined in (9), and as

$$g^i_{2,s}\left(z^1, z^2, \ldots, z^N\right) := \begin{cases} \kappa^i(z^i) & \text{if } s = i, \text{at each computation event} \\ z^s_{2,s} & \text{if } s \in \mathcal{N}_i, \text{at each communication event} \\ z^i_{2,s} & \text{otherwise} \end{cases} \tag{15}$$

if $p = 2$, where $\kappa^i$ is defined via (10). In (14)-(15), a computation event by agent $i$ occurs when $k \in K^i$ and a communication event occurs when agent $i$ receives information from agent $s$ at $k \in R^{i,s}$. Due to this, the maps $g^i_1$ and $g^i_2$ depend on the current and past state values. Such a dependency is omitted in (14)-(15), for simplicity of notation.

Then, the full multiagent system corresponding to interconnecting $N$ agents with dynamics as in (13) is

$$\begin{aligned} &\left(z^1, z^2, \ldots, z^N\right)^+ = G_{\text{async}}\left(z^1, z^2, \ldots, z^N\right) \\ &\left(z^1, z^2, \ldots, z^N\right) \in D_{\text{async}} := (X \times X)^N \end{aligned} \tag{16}$$

where $G_{\text{async}}$ is defined as $G_{\text{async}}\left(z^1, z^2, \ldots, z^N\right) := \left(G^1_{\text{async}}\left(z^1, z^2, \ldots, z^N\right), G^2_{\text{async}}\left(z^1, z^2, \ldots, z^N\right), \ldots, G^N_{\text{async}}\left(z^1, z^2, \ldots, z^N\right)\right)$

### *B. Convergence rate of Algorithm 1*

For the forthcoming result for the asynchronous algorithm $(D_{\text{async}}, G_{\text{async}})$ in (16), we impose the following assumption.

*Assumption 4.4:* (Infinitely many events): For each $i \in \mathcal{V}$ and each $\ell \in \mathcal{N}_i$, the sets $K^i \subset \mathbb{N}$ and $R^{i,\ell} \subset \mathbb{N}$ are unbounded. If $\{k_s\}_{s\in\mathbb{N}}$ is an increasing sequence of times in $K^i$, then for each solution to the algorithm in (16), $\lim_{s\to\infty} \tau^\ell_i(k_s) = \infty$ for each $\ell \in \mathcal{V}$ such that $i \in \mathcal{N}_\ell$, where $\tau^\ell_i$ is defined via (12).

In the upcoming result, the convergence rate of (16) is characterized by leveraging results in [26] and [28] in terms of the number of operations – denoted $\text{ops}(k)$ – the agents have completed (counted in the appropriate sequence). Namely, we count operations as follows. Initially, we set $\text{ops}(0) = 0$. Then, after all agents $i \in \mathcal{V}$ have updated $\left(z^i_{1,i}, z^i_{2,i}\right)$ to $\left(\tilde{\kappa}^i(z^i), \kappa^i(z^i)\right)$ and have sent such updates to and had such updates received by all essential neighbors $\ell \in \mathcal{N}_i$ – say, by time $k'$ – we increment ops to $\text{ops}(k') = 1$. Note that it is possible for any agent $i$ to compute and send – and essential neighbors $\ell \in \mathcal{N}_i$ to receive – multiple updates of $\left(z^i_{1,i}, z^i_{2,i}\right)$ between $\text{ops}(0) = 0$ and $\text{ops}(k') = 1$. In other words, different subsequences of $\{k_s\}_{s\in\mathbb{N}}$ in $K^i$ exist for each $i \in \mathcal{V}$ and different subsequences of $\{k_s\}_{s\in\mathbb{N}}$ in $R^{i,\ell}$ exist for each $(i, \ell) \in \mathcal{E}$ between $\text{ops}(0) = 0$ and $\text{ops}(k') = 1$. After $\text{ops}(k') = 1$, then we wait until all agents $i \in \mathcal{V}$ have subsequently computed a new update of $\left(z^i_{1,i}, z^i_{2,i}\right)$ and such updates have been sent to and received by all other essential neighbors. If this occurs at time $k''$, then we set $\text{ops}(k'') = 2$, and this process continues.

Given $X$ in (4), we denote by

$$z := (z_1, z_2) \in X \times X \tag{17}$$

the "true value" of agents' decision variables, in the sense that $z$ in (17) contains the current values of the decision variables of all agents. We define $z_1$ and $z_2$ as $z_1 := \left(z^1_{1,1}, z^2_{1,2}, \ldots, z^N_{1,N}\right)$ and $z_2 := \left(z^1_{2,1}, z^2_{2,2}, \ldots, z^N_{2,N}\right)$. Namely, $z_1$ collects the $z^i_{1,i}$ components and $z_2$ collects the $z^i_{2,i}$ components of the states of all agents $i \in \mathcal{V}$.

For the result to follow, we denote the portion of the algorithm $(D_{\text{async}}, G_{\text{async}})$ in (16) corresponding to the update

of the true value $z$ in (17) as

$$z^+ = g(z) \qquad z \in X \times X \tag{18}$$

where $g$ is defined as $g(z) := \left(\left(\tilde{g}^1_{1,1}(z^1), \tilde{g}^2_{1,2}(z^2), \ldots, \tilde{g}^N_{1,N}(z^N)\right), \left(\tilde{g}^1_{2,1}(z^1), \tilde{g}^2_{2,2}(z^2), \ldots, \tilde{g}^N_{2,N}(z^N)\right)\right)$ where, for each $i \in \mathcal{V}$, $\tilde{g}^i_{1,i}$ and $\tilde{g}^i_{2,i}$ are defined as

$$\tilde{g}^i_{1,i}\left(z^i\right) := \begin{cases} \tilde{\kappa}^i(z^i) & \text{at each computation event by agent } i \\ z^i_{1,i} & \text{otherwise} \end{cases}$$
$$\tilde{g}^i_{2,i}\left(z^i\right) := \begin{cases} \kappa^i(z^i) & \text{at each computation event by agent } i \\ z^i_{2,i} & \text{otherwise} \end{cases} \tag{19}$$

where $\tilde{\kappa}^i$ is defined via (9) and $\kappa^i$ is defined in (10). Recall that a computation event by agent $i$ occurs when $k \in K^i$. Since (18)-(19) represent only the updates of the true values of the decision variables, and not the local copies onboard each agent $i$, communication events are not represented in (19).

*Theorem 4.5: (Exponential convergence rate for (16)) Suppose $X \subset \mathbb{R}^N$ satisfies Assumption 4.1, $f$ satisfies Assumption 4.2, $H$ satisfies Assumption 4.3 with $\mu > 0$, and $z_\circ \in \mathcal{Z}_\circ := X \times X$ denotes the initial state of $z$ in (17). For each $\gamma \in \left(0, \frac{1}{\max_{i\in\mathcal{V}} \max_{\eta\in X} |H_{ii}(\eta)|}\right)$ and $\lambda \in \left(0, \frac{\gamma\mu}{2}\right)$, each maximal solution[3] $k \mapsto \left(z^1(k), z^2(k), \ldots, z^N(k)\right)$ to the asynchronous, double-update heavy ball algorithm $(D_{async}, G_{async})$ in (16) from $\left(z^1_\circ, z^2_\circ, \ldots, z^N_\circ\right) \in \mathcal{Z}^N_\circ$, for which Assumption 4.4 holds, satisfies*

$$\max_{i\in\mathcal{V}} \left\|z^i(k) - z^*\right\|_\infty \leq a^{ops(k)} \max_{i\in\mathcal{V}} \left\|z^i_\circ - z^*\right\|_\infty \tag{20}$$

*for all $k \in \mathbb{N}$, where $a := \max\{a_1, a_2\}$, with $a_1 := (1 - \gamma\mu + \lambda)^2 + \lambda + \lambda(1 - \gamma\mu + \lambda) \in [0, 1)$ and $a_2 := 1 - \gamma\mu + 2\lambda \in [0, 1)$, and $z^* := (x^*, x^*)$, where $x^* \in X$ is the minimizer of $f$ over $X$.*

## V. Numerical Example

To demonstrate the effectiveness of the asynchronous, double-update heavy ball algorithm $(D_{\text{async}}, G_{\text{async}})$ in (16), we compare it in simulation with a multiagent constrained gradient descent algorithm. In particular, we compare $(D_{\text{async}}, G_{\text{async}})$ with a version of the asynchronous primal-dual algorithm for constrained gradient descent, in [28], with the dual variables fixed to zero. This renders that algorithm equivalent to totally asynchronous projected gradient descent. First, we compare the convergence rates of $(D_{\text{async}}, G_{\text{async}})$ and the algorithm [28] analytically. For the asynchronous algorithm in [28], when the dual variables are fixed at zero, the constrained update law for block $i$ simplifies to $\bar{\kappa}^i(z^i_1) := \Pi_{X_i}\left[z^i_{1,i} - \gamma\nabla_i f(z^i_1)\right]$ where

$$\gamma < \frac{1}{\max_{i\in\mathcal{V}} \max_{\eta\in X} \sum_{\ell=1}^N |H_{ij}(\eta)|}. \tag{21}$$

[3]A solution is called maximal if it cannot be extended further, and is called complete if its domain is unbounded. When Assumption 4.4 holds, such solutions are defined for all $k \in \mathbb{N}$.

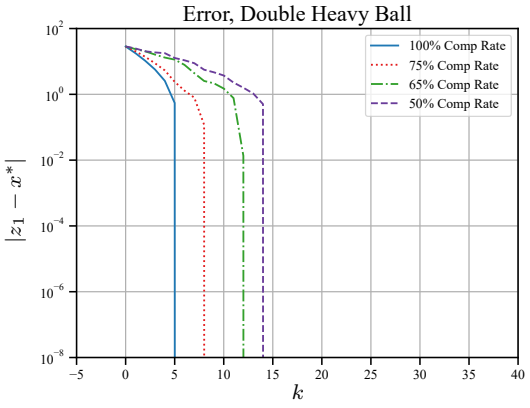


Fig. 1. The evolution over time of $|z_1 - x^*|$ for $N = 10$ agents running the totally asynchronous Algorithm 1.

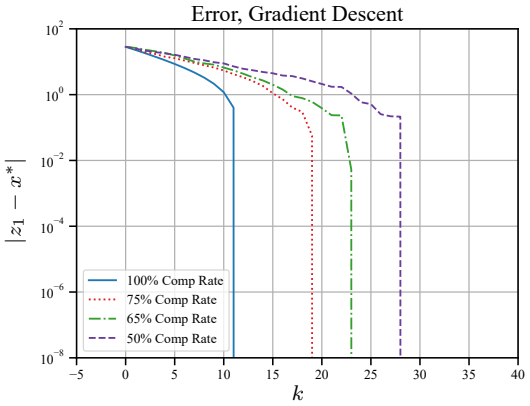


Fig. 2. The evolution over time of $|z_1 - x^*|$ for $N = 10$ agents running totally asynchronous gradient descent.

In [28], the asynchronous algorithm is designed and analyzed for convex functions $f$ which satisfy Assumption 4.2, $H$ satisfies Assumption 4.3, the set $X$ satisfies Assumption 4.1, and the algorithm itself satisfies Assumption 4.4.

It is shown in [28, Theorem 2] that, for the dual variables fixed, each maximal solution $k \mapsto z_1(k)$ to the asynchronous primal-dual algorithm therein satisfies

$$\max_{i\in N} \|z^i_1(k) - x^*\|_\infty \leq q_p^{\text{ops}(k)} \max_{\ell\in N} \|z^\ell_{1_\circ} - x^*\|_\infty \tag{22}$$

where $q_p := (1 - \gamma\mu) \in [0, 1)$ and $x^* \in X$ denotes the fixed point of the constrained gradient descent update law, with the dual variables fixed. Comparing the constant $q_p$ in (22) with the constant $a$, defined below (20), based on the established theoretical bounds, the primal convergence rate of the asynchronous algorithm in [28] is faster than the convergence rate of $(D_{\text{async}}, G_{\text{async}})$. However, as we illustrate below, the convergence rate of $(D_{\text{async}}, G_{\text{async}})$ is much better in practice.

Next, we compare the algorithms in simulation[4] for $N = 10$ agents with the objective function

$$f(z_1) := \frac{3}{10}\sum_{i=1}^N \left(z^i_{1,i}\right)^2 + \frac{1}{200}\sum_{i=1}^N \sum_{\substack{\ell=1 \\ \ell\neq i}}^N \left(z^i_{1,i} - z^i_{1,\ell}\right)^2$$

for which $\mu = \frac{1}{2}$. We also require that $z^i_{1,i} \in X_i = [1, 10]$ and $z^i_{2,i} \in X_i = [1, 10]$ for each $i \in \mathcal{V}$. We use the parameter

[4]Code at `github.com/HybridSystemsLab/MultiagentHBF`.

value $\gamma = 0.3$ for the step size of both algorithms, which satisfies $\gamma \in \left(0, \frac{1}{\max\limits_{i\in\mathcal{V}} \max\limits_{z_1\in X} |H_{ii}(z_1)|}\right)$ for $(D_{\text{async}}, G_{\text{async}})$ in (16) and the definition of $\gamma$ in (21) for the asynchronous algorithm in [28]. Additionally, we use the value $\lambda = 0.075$ for $(D_{\text{async}}, G_{\text{async}})$, which satisfies $\lambda \in (0, \frac{\gamma\mu}{2})$. In this example, both algorithms have a communication rate of 1 (i.e., each agent has a $100\%$ chance of communicating the latest update to another agent at each iteration) and solutions to both algorithms are simulated at computation rates of 1 (i.e., each decision variable has a $100\%$ probability of updating at each iteration), $0.75$, $0.65$, and $0.5$. The initial conditions for $(D_{\text{async}}, G_{\text{async}})$ are $z^i_\circ = (10\mathbf{1}, 10\mathbf{1})$, for all $i \in \mathcal{V}$, where $\mathbf{1}$ is the $10 \times 1$ vector of ones, and for the asynchronous algorithm in [28] are $z^i_{1,\circ} = 10\mathbf{1}$ for all $i \in \mathcal{V}$.

Figures 1 and 2 demonstrate marked performance improvement of $(D_{\text{async}}, G_{\text{async}})$ over the asynchronous gradient descent algorithm, with the error for $(D_{\text{async}}, G_{\text{async}})$ as small as $10^{-8}$ after 5 iterations and the error for asynchronous gradient descent algorithm as small as $10^{-8}$ after 11 iterations, when the computation rate is 1. In other words, $(D_{\text{async}}, G_{\text{async}})$ converges twice as fast as the asynchronous gradient descent algorithm, which is true for all computation rates in Figures 1 and 2. From this example, we see that although the theoretical convergence rate of $(D_{\text{async}}, G_{\text{async}})$ in (20) is slower than the convergence bound of the asynchronous gradient descent algorithm in (22), the bound on $(D_{\text{async}}, G_{\text{async}})$ is conservative compared to its numerical performance, and the asynchronous heavy ball algorithm is much faster in practice.

## VI. Conclusion

We developed a totally asynchronous, block-based optimization algorithm utilizing two constrained heavy ball computations per agent update. The algorithm guarantees fast convergence to the unique minimizer of $f$. Future work includes extending the results to nonscalar blocks and establishing robustness to perturbations in agents' updates.

## References


[1] S. M. Fosson, "Online optimization in dynamic environments: a regret analysis for sparse problems," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 7225–7230.

[2] F. Y. Jakubiec and A. Ribeiro, "D-map: Distributed maximum a posteriori probability estimation of dynamic systems," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 450–466, 2013.

[3] S. M. Fosson, "Centralized and distributed online learning for sparse time-varying optimization," *IEEE Transactions on Automatic Control*, vol. 66, no. 6, pp. 2542–2557, 2021.

[4] O. Arslan and D. E. Koditschek, "Exact robot navigation using power diagrams," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1–8.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice Hall Englewood Cliffs, NJ, 1989.

[6] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[7] A. Nedić and A. Olshevsky, "Distributed optimization over time-varying directed graphs," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 601–615, 2015.

[8] J. Duchi, A. Agarwal, and M. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Trans. on Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[9] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, "Push-sum distributed dual averaging for convex optimization," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 2012, pp. 5453–5458.

[10] M. Zhu and S. Martínez, "On distributed convex optimization under inequality and equality constraints," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 151–164, 2011.

[11] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," in *Advances in neural information processing systems*, 2014, pp. 3068–3076.

[12] D. Bertsekas, "Distributed asynchronous computation of fixed points," *Mathematical Programming*, vol. 27, no. 1, pp. 107–120, 1983.

[13] E. Ghadimi, H. R. Feyzmahdavian, and M. Johansson, "Global convergence of the heavy-ball method for convex optimization," in *14th IEEE European Control Conference*, 2015, pp. 310–315.

[14] P. Ochs, T. Brox, and T. Pock, "ipiasco: inertial proximal algorithm for strongly convex optimization," *Journal of Mathematical Imaging and Vision*, vol. 53, no. 2, pp. 171–181, 2015.

[15] H. Wang and P. C. Miller, "Scaled heavy-ball acceleration of the richardson-lucy algorithm for 3d microscopy image restoration," *IEEE Transactions on Image Processing*, vol. 23, no. 2, pp. 848–854, 2013.

[16] J. Zhang, C. Uribe, A. Mokhtari, and A. Jadbabaie, "Achieving acceleration in distributed optimization via direct discretization of the heavy-ball ode," in *2019 American Control Conference (ACC)*, 2019, pp. 3408–3413.

[17] A. I. Chen and A. Ozdaglar, "A fast distributed proximal-gradient method," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 601–608.

[18] H. Li, H. Cheng, Z. Wang, and G.-C. Wu, "Distributed nesterov gradient and heavy-ball double accelerated asynchronous optimization," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5723–5737, 2020.

[19] D. Ochoa, J. Poveda, C. Uribe, and N. Quijano, "Robust optimization over networks using distributed restarting of accelerated dynamics," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 301–306, 2020.

[20] A. Olshevsky, "Linear time average consensus and distributed optimization on fixed graphs," *SIAM Journal on Control and Optimization*, vol. 55, no. 6, pp. 3990–4014, 2017.

[21] D. Jakovetić, J. Xavier, and J. Moura, "Fast distributed gradient methods," *IEEE Trans. on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.

[22] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[23] S. Hochhaus and M. T. Hale, "Asynchronous distributed optimization with heterogeneous regularizations and normalizations," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 4232–4237.

[24] M. Ubl and M. Hale, "Totally asynchronous distributed quadratic programming with independent stepsizes and regularizations," in *58th IEEE Conf. on Decision and Control (CDC)*, 2019, pp. 7423–7428.

[25] M. Hale and M. Egerstedt, "Cloud-based optimization: A quasi-decentralized approach to multi-agent coordination," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 6635–6640.

[26] M. T. Hale, A. Nedić, and M. Egerstedt, "Asynchronous multiagent primal-dual optimization," *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4421–4435, 2017.

[27] J. Koshal, A. Nedić, and U. V. Shanbhag, "Multiuser optimization: Distributed algorithms and error analysis," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 1046–1081, 2011.

[28] K. R. Hendrickson and M. T. Hale, "Towards totally asynchronous primal-dual convex optimization in blocks," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 3663–3668.

[29] R. G. Sanfelice, *Hybrid Feedback Control*. New Jersey: Princeton University Press, 2021.

[30] R. Goebel, R. G. Sanfelice, and A. R. Teel, *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. New Jersey: Princeton University Press, 2012.

[31] C. M. Kellett, *Advances in converse and control Lyapunov functions*. University of California, Santa Barbara, 2002.

[32] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.