2024 IEEE 63rd Conference on Decision and Control (CDC)
December 16-19, 2024. MiCo, Milan, Italy

A Totally Asynchronous Nesterov's Accelerated Gradient Method for Convex Optimization

Ellie Pond*,[1], April Sebok*,[2], Zachary Bell[3], and Matthew Hale[1]

Abstract— **We present a totally asynchronous algorithm for convex optimization that is based on a novel generalization of Nesterov's accelerated gradient method. This algorithm is developed for fast convergence under "total asynchrony" which allows arbitrarily long delays between agents' computations and communications. These conditions may arise, for example, due to jamming by adversaries. Our framework is block-based, in the sense that each agent is only responsible for computing and communicating updates to a small subset of the network-level decision variables. In our main result, we present bounds on the algorithm's parameters that guarantee linear convergence to an optimizer. Then, we quantify the relationship between (i) the total number of computations and communications executed by the agents and (ii) the agents' collective distance to an optimum. Numerical simulations show that this algorithm requires 28% fewer iterations than the heavy ball algorithm and 61% fewer iterations than gradient descent under total asynchrony.**

I. INTRODUCTION

Large-scale convex optimization problems are used to model complex problems in several fields, including robotics [1], [2], [3], machine learning [4], [5], and communications [6], [7]. Large systems and/or complex tasks in these applications can lead to large convex programs, and it can be desirable to parallelize computations in order to accelerate the process of finding solutions.

Parallelized algorithms use a collection of agents to solve an optimization problem by partitioning computations among them and having the agents communicate the results of their computations with others in a network. The types of parallelized execution can be succinctly classified as (i) synchronous, (ii) partially asynchronous, or (iii) totally asynchronous. In the synchronous setting, all agents compute and communicate concurrently. However, congested bandwidth or a single slow agent can make synchrony difficult to attain. For partially asynchronous algorithms, the agents must compute and communicate at least once in each time interval of a prescribed length [8]. However, such bounds can be violated due to factors outside agents' control, such as adversaries jamming communications. This challenge can be alleviated by using a totally asynchronous algorithm, which allows for the delays between successive computations and communications to be unbounded for all agents [9], provided that no agent permanently stops computing and communicating.

Existing algorithms that are labeled "totally asynchronous" include variations of gradient descent [10], [11], [12], [13], a Newton-based algorithm [14], and the heavy ball algorithm [15]. In [16] and [17], Nesterov's accelerated gradient (NAG) method is implemented in a partially asynchronous manner, that is, with bounded asynchrony delays. In this paper, we are motivated in part by the totally asynchronous heavy ball algorithm developed in [15], which showed faster convergence in simulation than a comparable gradient descent method. In this work, we seek even faster convergence by using NAG. A feature of the heavy ball algorithm is that it converges monotonically, while NAG can converge faster than heavy ball, but potentially with oscillations [18]. In the totally asynchronous setting that we consider, the faster convergence of NAG is desirable because computations may be infrequent, which makes it critical for each computation to make as much progress as possible towards a minimizer.

Therefore, in this paper, we develop a totally asynchronous NAG algorithm that attains linear convergence to minimizers for a class of optimization problems. We apply the methodology in [9] to prove its convergence under total asynchrony. In particular, [9, pg. 431] Proposition 2.1 shows that if an algorithm is an ∞-norm contraction mapping, then it converges when implemented in a totally asynchronous way. The NAG algorithm itself may not be such a contraction mapping, but we show that the application of two iterations of NAG is an ∞-norm contraction. Then, we establish bounds on the minimum number of computations and communications required by each agent in order for the network's iterates to be within a given distance of a minimum. Finally, we show in simulation that the totally asynchronous NAG algorithm requires up to 61% fewer iterations than gradient descent and 28% fewer than heavy ball.

The rest of this paper is organized as follows. Section II gives problem statements and Section III presents the totally asynchronous NAG algorithm. Section IV proves that this algorithm converges linearly, and Section V gives a convergence rate in terms of each agents' computations and communications. Next, Section VI validates the accelerated convergence of the totally asynchronous NAG algorithm in simulations. Finally, Section VII concludes.

*These authors contributed equally to this work.

[1]EP and MH are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332. Emails: {epond3,matthale}gatech.edu.

[2]AS is with the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, USA 32611. Email: a.sebok@ufl.edu.

[3]ZB is with AFRL/RW at Eglin AFB. Email: zachary.bell.10@us.af.mil.

This work was supported by AFRL under grant FA8651-23-F-A006, AFOSR under grant FA9550-19-1-0169, and ONR under grant N00014-22-1-2435.

II. Preliminaries and Problem Statements

This section lays out notation, reviews the centralized NAG algorithm, and provides problem statements.

A. Notation

We use $\mathbb{R}$, $\mathbb{R}_+$, and $\mathbb{N}$ to denote the real numbers, the strictly positive real numbers, and the natural numbers, respectively. We use $|\cdot|$ to denote the cardinality of a set. We use the column operator for $a, b \in \mathbb{R}^n$ defined as $\text{col}(a,b) = \begin{bmatrix} a^T & b^T \end{bmatrix}^T \in \mathbb{R}^{2n}$. We use $\Pi_{\mathcal{Z}} : \mathbb{R}^n \to \mathbb{R}^n$ to denote the orthogonal projection onto the closed, convex set $\mathcal{X} \subset \mathbb{R}^n$, i.e., $\Pi_{\mathcal{X}}[w] = \arg\min_{x \in \mathcal{X}} ||x - w||_2$. The infinity norm $||\cdot||_\infty$ is defined as $||x||_\infty = \max_{i \in \mathcal{V}} |x_i|$, where $x \in \mathbb{R}^n$ and where we define $\mathcal{V} = \{1, 2, \dots, n\}$. We also use $\nabla_i f = \frac{\partial f}{\partial x_i}$. For an ordered pair $\chi = (v, w) \in \mathbb{R}^n \times \mathbb{R}^n$, we define $\|\chi\|_\infty = \max_{i \in \mathcal{V}} \max\{v_i, w_i\}$. We model the communication topology between agents as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, 2, \dots, n\}$ is the node set and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, such that $(i, j) \in \mathcal{E}$ indicates that agents $i, j \in \mathcal{V}$ communicate with each other.

B. Centralized Nesterov's Accelerated Gradient Method

Consider an objective function $f : \mathbb{R}^n \to \mathbb{R}$ and a constraint set $\mathcal{X} \subset \mathbb{R}^n$. We consider problems of the form: $\min_{x \in \mathcal{X}} f(x)$. The centralized NAG algorithm at iteration $l \in \mathbb{N}$ updates the decision variable $x(l) \in \mathbb{R}^n$ using

$$x(l+1) = \Pi_{\mathcal{X}}\big[x(l) - \gamma \nabla f\big(x(l) + \lambda(x(l) - x(l-1))\big) + \lambda(x(l) - x(l-1))\big], \quad (1)$$

where $\lambda, \gamma \in \mathbb{R}_+$. The γ term is a step size, and here it represents the pull of "gravity," while the λ term helps avoid overshooting the minimizer and represents "friction."

C. Problem Statements

We consider a network of n agents solving $\min_{x \in \mathcal{X}} f(x)$, where $\mathcal{X} \subset \mathbb{R}^n$ is a constraint set. The communication topology of these agents is given by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

For simplicity, in this work we partition x into scalar blocks, i.e., each agent updates a single entry of x, though our results will easily extend to non-scalar blocks. Agent i will compute successive values of x_i using $\frac{\partial f}{\partial x_i}$. We refer to the agents that agent i must communicate with as its "essential neighbors", and we denote this index set as $\mathcal{V}^i \subseteq \mathcal{V}$. We emphasize here that the agents' underlying graphs do not even need to be connected as long as they communicate with their essential neighbors.

Therefore, we consider objective functions f of the form

$$f(x) = \sum_{i=1}^{n} f_i(x_{\mathcal{V}^i}), \quad (2)$$

where $f_i : \mathbb{R}^{|\mathcal{V}^i|+1} \to \mathbb{R}$ and where $x_{\mathcal{V}^i}$ contains agent i's decision variable and all decision variables such that $j \in \mathcal{V}^i$. With this formulation, we state the problems that are the focus of the remainder of the paper.

Problem 1. *Given an objective function $f : \mathbb{R}^n \to \mathbb{R}$ defined in (2) and a constraint set $\mathcal{X} \subset \mathbb{R}^n$, construct a totally asynchronous algorithm based on the NAG method that solves $\min_{x \in \mathcal{X}} f(x)$ over a network of n agents.*

Problem 2. *Show that the totally asynchronous NAG algorithm in Problem 1 converges linearly to a minimizer.*

Problem 3. *Given $\epsilon > 0$, determine lower bounds on the numbers of computations and communications that each agent must execute in order for the iterates of the totally asynchronous NAG algorithm to be within distance ϵ of a minimizer for Problem 1.*

III. Totally Asynchronous Nesterov's Accelerated Gradient Method

In this section we solve Problem 1 by formulating a totally asynchronous NAG algorithm. Specifically, we will first show that two variations of the *synchronous* NAG algorithm satisfy certain technical conditions. Then, using [9], we will show that these properties guarantee convergence in the totally asynchronous setting. Mathematically, we show that the synchronous application of two iterations of the NAG algorithm is an ∞-norm contraction.

To begin, we will make the following three assumptions about the optimization problem in Problem 1.

Assumption 1. *The constraint set $\mathcal{X} \subset \mathbb{R}^n$ is nonempty, convex, and compact. The set can be decomposed as $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$, where $\mathcal{X}_i \subset \mathbb{R}$ for each $i \in \mathcal{V}$.*

Assumption 1 enables the parallelization of a projected update law, which will enable the constant satisfaction of set constraints, even under total asynchrony.

Assumption 2. *The objective function f is twice continuously differentiable.*

Assumption 2 is quite common, and it guarantees the existence and continuity of the gradient and the Hessian of f, both of which are essential in our convergence analyses.

Assumption 3. *The Hessian matrix, defined as $H(x) = \nabla^2 f(x) \in \mathbb{R}^{n \times n}$, is μ-diagonally dominant on $\mathcal{X} \subset \mathbb{R}^n$ for some $\mu > 0$. That is, for each $i \in \mathcal{V}$, we have the bound $H_{ii}(x) \geq \mu + \sum_{j=1, j \neq i}^{n} |H_{ij}(x)|$ for all $x \in \mathcal{X}$.*

Assumption 3 is standard in the context of totally asynchronous algorithms. Intuitively, this assumption asserts that for agent $i \in \mathcal{V}$, its computations depend more on its own decision variables than on the rest of the agents' decision variables. In [9], it is noted that some form of Hessian diagonal dominance is typically needed for convergence of totally asynchronous algorithms, and we therefore use it here. Assumption 3 implies that f is μ-strongly convex. Therefore it has a unique minimizer over $\mathcal{X}$, which we denote by $x^\star = \text{col}\big(x_1^\star, \dots, x_n^\star\big)$.

In its centralized form in (1), the NAG method depends on the iterate at time l, namely $x(l)$, and the iterate at the previous time step $l-1$, which is $x(l-1)$. We write $y(l) = x(l-1)$ to denote this earlier iterate. For the distributed

solution of Problem 1, each agent will store a local copy of the full decision vector. Onboard agent i at time l, this decision variable is denoted $z^i(l) = \big(x^i(l), y^i(l)\big) \in \mathcal{Z}$.

Over time, agent i computes updates to $z_i^i(l) = \big(x_i^i(l), y_i^i(l)\big) \in \mathcal{Z}_i$, where we define $\mathcal{Z}_i := \mathcal{X}_i \times \mathcal{X}_i$. The subscripts indicate that the terms z_i^i, x_i^i, and y_i^i are agent i's local copies of its own decision variable. Using this notation, at time l agent i's local copy of z is denoted $z^i(l) = (x^i(l), y^i(l)) = \big(\text{col}(x_1^i(l), \ldots, x_i^i(l), \ldots, x_n^i(l)),$ $\text{col}(y_1^i(l), \ldots, y_i^i(l), \ldots, y_n^i(l))\big)$. For any agent $m \notin \mathcal{V}^i$, agent i can set x_m^i and y_m^i to any values over time, since these values do not affect its computations and will not be changed by any communications.

A. The Single-Step Synchronous Method

In this sub-section, we establish the first of two variations of the synchronous NAG algorithm that we use in Section III-C for the analysis of the totally asynchronous NAG algorithm. These algorithms have simultaneous computations and simultaneous communications among essential neighbors.

We will refer to the following update law for $z_i^i(l) = \big(x_i^i(l), y_i^i(l)\big)$ as the "single-step synchronous update law". This update law is defined by $\tilde{u}_x^i$ and $\tilde{u}_y^i$, given by

$$\begin{aligned} x_i^i(l+1) &= \tilde{u}_x^i(x^i(l), y^i(l)) = \Pi_{\mathcal{X}_i}\big[x_i^i(l) - \gamma \nabla_i f\big(x^i(l) \\ &\quad + \lambda(x^i(l) - y^i(l))\big) + \lambda(x_i^i(l) - y_i^i(l))\big] \\ y_i^i(l+1) &= \tilde{u}_y^i(x^i(l), y^i(l)) = x_i^i(l) \end{aligned}$$

for all $l \in \mathbb{N}$ and $i \in \mathcal{V}$. In this update law, we see that at time $l \in \mathbb{N}$, one iteration of NAG is performed and stored in the $x_i^i(l+1)$ variable, while the $y_i^i(l+1)$ variable stores the value of $x_i^i(l)$. For simplicity of notation, let $\tilde{u}^i : \mathcal{Z} \to \mathcal{Z}_i$ denote the single-step synchronous update

$$\begin{aligned} (x_i^i(l+1), y_i^i(l+1)) &= \tilde{u}^i(x^i(l), y^i(l)) \\ &= \big(\tilde{u}_x^i(x^i(l), y^i(l)), \tilde{u}_y^i(x^i(l), y^i(l))\big). \end{aligned} \tag{3}$$

We will also denote (3) by $z_i^i(l+1) = \tilde{u}^i(z^i(l))$ for conciseness. The following lemma establishes that this update law has a single fixed point, which is the solution to Problem 1.

Lemma 1. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, and the Hessian satisfy Assumption 3. Define the points $z^\star = (x^\star, x^\star) \in \mathcal{Z}$ and $z_i^\star = (x_i^\star, x_i^\star) \in \mathcal{Z}_i$. Then, the point $z^\star$ is a fixed point of the single-step synchronous update law in* (3)*, in the sense that $z_i^\star = \tilde{u}^i(z^\star)$ for all $i \in \mathcal{V}$.*

Proof. See Appendix A in technical report [19]. □

In the synchronous algorithm, all agents update and communicate at each time step $l \in \mathbb{N}$. Agent i updates its decision variables according to (3) and communicates this update with agents in its essential neighborhood, indexed by $j \in \mathcal{V}^i$. All agents $j \in \mathcal{V}^i$ incorporate this communication into their own local state vector by setting $(x_i^j(l), y_i^j(l)) \leftarrow (x_i^i(l), y_i^i(l))$. For the agents $m \notin \mathcal{V}^i$, the entries in their local state vector remain the same as at the previous time step, i.e., $(x_i^m(l), y_i^m(l)) \leftarrow (x_i^m(l-1), y_i^m(l-1))$.

We define the true state of the network at time $l \in \mathbb{N}$ to be the vector $z^{\text{true}}(l) = (x^{\text{true}}(l), y^{\text{true}}(l))$, where $x^{\text{true}}(l) = \text{col}(x_1^1(l), x_2^2(l), \ldots, x_n^n(l))$ and $y^{\text{true}}(l) = \text{col}(y_1^1(l), y_2^2(l), \ldots, y_n^n(l))$. These vectors contain each agent's latest value of their own decision variable. To establish convergence for the single-step synchronous NAG algorithm, we will use the map $\hat{u}_{\text{true}}^i(z^{\text{true}}(l)) = x_i^i(l) - \gamma \nabla_i f\big(x^{\text{true}}(l) + \lambda(x^{\text{true}}(l) - y^{\text{true}}(l))\big) + \lambda(x_i^i(l) - y_i^i(l))$, which models the changes in the true state of the network.

Theorem 1. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, and the Hessian satisfy Assumption 3. For each $\gamma \in \big(0, \frac{1}{\max_{i \in \mathcal{V}} \max_{\eta \in \mathcal{X}} |H_{ii}(\eta)|}\big)$ and $\lambda \in \big(0, \frac{\gamma\mu}{2(1-\gamma\mu)}\big)$, the iterates of the synchronous single-step algorithm from the initial state $z(0) \in \mathcal{Z}$ satisfy $||z(l+1) - z^\star||_\infty \leq \alpha ||z(l-1) - z^\star||_\infty$ for all $l \in \mathbb{N}$, where $\alpha = \max\{\alpha_1, \alpha_2\} \in [0, 1)$ and*

$$\begin{aligned} \alpha_1 &= (1 + \lambda - \gamma\mu(1+\lambda))^2 + \lambda(1 - \gamma\mu) \\ &\quad + \lambda(1 - \gamma\mu)(1 + \lambda - \gamma\mu(1+\lambda)) \end{aligned} \tag{4}$$

$$\alpha_2 = 1 - \gamma\mu + 2\lambda(1 - \gamma\mu). \tag{5}$$

Proof. See Appendix B in technical report [19]. □

Theorem 1 proves that the synchronous single-step NAG update law is contractive with respect to the ∞-norm over two time steps, i.e., from time $l-1$ to time $l+1$.

B. The Double-Step Synchronous Method

In this section, we continue the procedure for proving totally asynchronous convergence outlined in [9]. This requires proving that the synchronous variation of the algorithm is contractive with respect to the infinity-norm, which we have shown is the case over two time steps in the previous subsection. We now define the "double-step synchronous update law", which performs two steps of NAG per iteration and hence is contractive at every iteration. We then prove a three-part lemma in regard to the double-step NAG update law that is required for totally asynchronous convergence.

We will now use the variable $k \in \mathbb{N}$ to represent discretized time rather than $l \in \mathbb{N}$. In the k time-scale, the step from $k \to k+1$ is equivalent to $l \to l+2$ in the l time-scale. This is done to make it clear that the NAG algorithm in (1) is applied twice every time an agent performs a computation. For each agent $i \in \mathcal{V}$, the updates for all $k \in \mathbb{N}$ are

$$\begin{aligned} x_i^i(k+1) &= u_x^i(x^i(k), y^i(k+1)) \\ &= \Pi_{\mathcal{X}_i}\big[y_i^i(k+1) + \lambda(y_i^i(k+1) - x_i^i(k)) \\ &\quad - \gamma \nabla_i f\big(y^i(k+1) + \lambda(y^i(k+1) - x^i(k))\big)\big] \end{aligned} \tag{6}$$

$$\begin{aligned} y_i^i(k+1) = u_y^i(x^i(k), y^i(k)) &= \Pi_{\mathcal{X}_i}\big[x_i^i(k) + \lambda(x_i^i(k) \\ &\quad - y_i^i(k)) - \gamma \nabla_i f\big(x^i(k) + \lambda(x^i(k) - y^i(k))\big)\big]. \end{aligned} \tag{7}$$

The local state vector $y^i(k+1)$ that is used in (6) is defined as $y^i(k+1) = \text{col}(y_1^i(k), \ldots, y_i^i(k+1), \ldots, y_n^i(k))$, where the newly updated y_i^i is stored in the i^{th} location and all other entries remain the same as they were at time k, i.e., $y_j^i(k+1) = y_j^i(k)$ for $j \neq i$. Though this may appear non-recursive, the update for x_i^i can be expressed completely

in terms of $(x^i(k), y^i(k))$ using the update law for $y_i^i(k+1)$ in (7). However, we choose to refer to the $(k+1)^{\text{th}}$ iteration of y_i^i for brevity of expression. Nonetheless, both (6) and (7) can be computed simultaneously over one time step $k \in \mathbb{N}$.

Let $u^i : \mathcal{Z} \to \mathcal{Z}_i$ be the double-step synchronous NAG update for agent i, defined as

$$\begin{aligned}(x_i^i(k+1), y_i^i(k+1)) &= u^i(x^i(k), y^i(k)) \\ &= \big(u_x^i(x^i(k), y^i(k+1)), u_y^i(x^i(k), y^i(k))\big). \quad (8)\end{aligned}$$

With an abuse of notation, we will alternately write (8) as $z_i^i(k+1) = u^i(z^i(k))$. As in Section III-A, every agent updates and communicates with their essential neighbors at every time step $k \in \mathbb{N}$. This update law has the same fixed point at the synchronous single-step algorithm.

Lemma 2. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, and the Hessian satisfy Assumption 3. Then, for each $i \in \mathcal{V}$, the minimizer $z^\star \in \mathcal{Z}$ is a fixed point of the double-step synchronous update in* (8)*, in the sense that $z_i^\star = u^i(z^\star)$ for all $i \in \mathcal{V}$.*

Proof. See Lemma 2 in authors' technical report [19]. □

We define $h : \mathcal{Z} \to \mathcal{Z}$ to be the map $h(z) := \text{col}((u_x^1(z), \dots, u_x^n(z))^T, (u_y^1(z), \dots, u_y^n(z))^T)$, which is equivalent to one iterate of the double-step synchronous NAG update law. The point $z^\star \in \mathcal{Z}$ is a fixed point of h due to Lemma 2. The following lemma defines and analyzes a collection of sets $\{Z(k)\}_{k\in\mathbb{N}}$ that essentially serve as Lyapunov sub-level sets, and this collection of sets must satisfy three conditions: (i) the Lyapunov-Like Condition (LLC), (ii) the Synchronous Convergence Condition (SCC), and (iii) the Box Condition (BC), which we define next.

Lemma 3. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, and the Hessian satisfy Assumption 3. Let $z(0) \in \mathcal{Z}$ be given, and define the set $\mathcal{Z}(k)$ as $\mathcal{Z}(k) = \{v \in \mathcal{Z} : ||v - z^\star||_\infty \leq \alpha^k ||z(0) - z^\star||_\infty\}$, where $\alpha = \max\{\alpha_1, \alpha_2\}$ is from Theorem 1. Then, for every $k \in \mathbb{N}$, the set $\mathcal{Z}(k)$ satisfies the following three properties:*

1) *(LLC) The set containment rule $\cdots \subset \mathcal{Z}(k+1) \subset \mathcal{Z}(k) \subset \cdots \subset \mathcal{Z}(0) = Z$ holds for all $k \in \mathbb{N}$.*
2) *(SCC) For the mapping h, given a point $z \in \mathcal{Z}(k)$, we have $h(z) \in \mathcal{Z}(k+1)$ for all $k \in \mathbb{N}$. As well, if $\{z_k\}_{k\in\mathbb{N}}$ is a sequence such that each $z_k \in \mathcal{Z}(k)$ for each $k \in \mathbb{N}$, then $\lim_{k\to\infty} z_k = z^\star$, where $z^\star = (x^\star, x^\star)$ is the fixed point of h.*
3) *(BC) For all $k \in \mathbb{N}$ and $i \in \mathcal{V}$, there are sets $\mathcal{Z}_i(k) \subset \mathcal{Z}_i$ such that $\mathcal{Z}(k) = \mathcal{Z}_1(k) \times \mathcal{Z}_2(k) \times \cdots \times \mathcal{Z}_n(k)$.*

Proof. See Lemma 3 in authors' technical report [19]. □

C. Totally Asynchronous NAG Algorithm

In this section, we build upon the previous subsections and develop the totally asynchronous NAG algorithm. Under total asynchrony, at any time step each agent may or may not perform a computation. Let $K^i \subseteq \mathbb{N}$ be the set of time steps at which agent $i \in \mathcal{V}$ updates its own decision variables.

When performing a computation, agent i uses the same update law in the totally asynchronous setting as in the synchronous double-step setting, namely (8), but it can do so at any time and without coordinating that time with any other agent. To faithfully implement that update law, any communications that agent i receives at time k are not incorporated into the local state vector $y^i(k+1)$ until after agent i has computed an update to both x_i^i and y_i^i. Therefore, at each time $k \in K^i$, agent i updates its decision variables with $(x_i^i(k+1), y_i^i(k+1)) \leftarrow u^i(x^i(k), y^i(k))$.

After an update is computed at a time $k \in K^i$, agent i sends its updated decision variables $(x_i^i(k+1), y_i^i(k+1))$ to each agent $j \in \mathcal{V}^i$. However, in the totally asynchronous setting, agent i may not communicate or there may be communication delays between agent i sending and agent j receiving. To model these communications, let the set of times at which agent i receives a communication from its essential neighbor j be $R_j^i \subseteq \mathbb{N}$. We emphasize that the sets K^i and R_j^i are not known to the agents and are only used to facilitate analysis.

At any time $k \in R_j^i$, agent i uses the values it receives from agent j to overwrite the previous values that it had received from agent j. Formally, agent i executes the operation $(x_j^i(k), y_j^i(k)) \leftarrow \big(x_j^j(\tau_j^i(k)), y_j^j(\tau_j^i(k))\big) \in \mathcal{Z}_j$. Here, the notation $\tau_j^i(k) \in K^j$ denotes the time at which agent j originally computed the values of x_j^j and y_j^j that agent i has onboard at time k. That is, at all times $k \in \mathbb{N}$, we define τ_j^i to be the earliest time in K^j so that $\big(x_j^i(k), y_j^i(k)\big) = \big(x_j^j(\tau_j^i(k)), y_j^j(\tau_j^i(k))\big)$ holds. For agents $m \notin \mathcal{V}^i$, the entries $(x_m^i(k+1), y_m^i(k+1))$ remain constant.

The full NAG algorithm with totally asynchronous computations and communications is shown in Algorithm 1, and this algorithm solves Problem 1.

Algorithm 1 Totally Asynchronous NAG Algorithm

Input: For $i \in \mathcal{V}$, select an arbitrary initial state $z^i(0) \in \mathcal{Z}$.

```
for k ∈ ℕ do
    for i ∈ 𝒱 do
        if k ∈ K^i then
            (x_i^i(k+1), y_i^i(k+1)) ← u^i(x^i(k), y^i(k))
            if j ∈ 𝒱^i then
                Send (x_i^i(k+1), y_i^i(k+1)) to agent j
        for j ∈ 𝒱^i do
            if k ∈ R_j^i then
                (x_j^i(k), y_j^i(k)) ← (x_j^j(τ_j^i(k)), y_j^j(τ_j^i(k)))
        if m ∉ 𝒱^i then
            (x_m^i(k), y_m^i(k)) ← (x_m^i(k−1), y_m^i(k−1))
```

IV. Convergence Rate

In this section we will show that the totally asynchronous NAG algorithm in Algorithm 1 converges linearly, which will solve Problem 2. To begin, we state an assumption regarding

agents' computations and communications and introduce the notion of an "operation cycle".

Assumption 4 ([9]). *For each agent $i \in \mathcal{V}$ and each essential neighbor $j \in \mathcal{V}^i$, the update set K^i and the communication set R^i_j are infinite. Moreover, if $\{k_s\}_{s\in\mathbb{N}}$ is an increasing sequence of times in K^i, then $\lim_{s\to\infty} \tau^i_j(k_s) = \infty$ for all $j \in \mathcal{V}$.*

This assumption ensures that no agent will ever cease computing or communicating indefinitely, though it allows for the delays between successive computations and communications to be arbitrarily long.

For the totally asynchronous setting, we will define an "operation cycle" to analyze Algorithm 1's convergence. Over time, such cycles are tracked by a counter, and we say that $\text{ops}(k)$ cycles have been completed by time k. Initially, when $k = 0$, we have $\text{ops}(0) = 0$. Say $k' \in \mathbb{N}$ is the first point at which (i) every agent has updated their decision variables, (ii) they have communicated, and (iii) every essential neighbor $j \in \mathcal{V}^i$ has received and incorporated these values. Then, we have $\text{ops}(k') = 1$. After time step k', the next communication cycle is completed after (i)-(iii) have been completed again. Suppose this happens at time $\ell \in \mathbb{N}$. Then we have $\text{ops}(k) = 1$ for all $k' \leq k < \ell$, and $\text{ops}(\ell) = 2$. The value of $\text{ops}(\cdot)$ will remain equal to 2 until the next cycle is completed.

Note that one or more agents may update and communicate more than one time per cycle. However, an operation cycle is not completed until *every* agent has performed a computation, communicated it, and received a communication at least once. At a time where $\text{ops}(k) = 0$, each agent's local state satisfies $(x^i(k), y^i(k)) \in \mathcal{Z}(0) = \mathcal{Z}$. Once the first operation cycle is completed $\text{ops}(k') = 1$, every agent's local state satisfies $(x^i(k'), y^i(k')) \in \mathcal{Z}(1)$.

We now establish an invariance result that is needed to derive the convergence rate of the algorithm.

Lemma 4. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, and the Hessian satisfy Assumption 3. Given an initial state $z^i(0) \in \mathcal{Z}$ for every $i \in \mathcal{V}$, the set $\mathcal{Z}(k)$ is forward invariant for the totally asynchronous NAG algorithm in Algorithm 1. That is, for all $i \in \mathcal{V}$, once $z^i(l) \in \mathcal{Z}(k)$ for some $l \in \mathbb{N}$, it holds that $z^i(p) \in \mathcal{Z}(k)$ for all $p \geq l$.*

Proof. See Appendix A in technical report [19]. □

Lemma 5. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, the Hessian satisfy Assumption 3, and let Assumption 4 hold. Then, for each $i \in \mathcal{V}$, the minimizer $z^\star \in \mathcal{Z}$ is a fixed point of the totally asynchronous NAG update law* (8)*, such that $z^\star_i = u^i(z^\star)$ for all $i \in \mathcal{V}$.*

Proof. See Lemma 5 in authors' technical report [19]. □

Now we present the theorem that establishes that the totally asynchronous NAG algorithm converges linearly in the value of $\text{ops}(k)$, which solves Problem 2.

Theorem 2. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, the Hessian satisfy Assumption 3, and let Assumption 4 hold. For each $\gamma \in \left(0, \frac{1}{\max_{i\in\mathcal{V}} \max_{\eta\in\mathcal{X}} |H_{ii}(\eta)|}\right)$ and $\lambda \in \left(0, \frac{\gamma\mu}{2(1-\gamma\mu)}\right)$, the totally asynchronous NAG algorithm in Algorithm 1 satisfies $\max_{i\in\mathcal{V}} \left|\left|z^i(k) - z^\star\right|\right|_\infty \leq \alpha^{\text{ops}(k)} \max_{i\in\mathcal{V}} \left|\left|z^i(0) - z^\star\right|\right|_\infty$ for all $k \in \mathbb{N}$, in which all agents are initialized with the initial condition $z^i(0) \in \mathcal{Z}$, where $\alpha = \max\{\alpha_1, \alpha_2\} \in [0, 1)$ with α_1 and α_2 defined in* (4) *and* (5)*, respectively.*

Proof. See Theorem 2 in authors' technical report [19]. □

V. Operation Complexity

In this section, we will leverage the convergence rate established in Theorem 2 and the network properties of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to quantify the operation complexity of agents' convergence. An operation cycle consists of at least $|\mathcal{V}|$ computation events and at least $2|\mathcal{E}|$ communication events. On the agent level, agent i performs a computation at least one time, sends information to essential neighbors at least $\left|\mathcal{V}^i\right|$ total times, and receives information from essential neighbors at least $\left|\mathcal{V}^i\right|$ total times.

Our goal is to establish bounds on the number of computations and communications that are required in order for the network's solution to be within an ∞-norm ball of radius ϵ that is centered on the minimizer of (2), that is, $\left|\left|z^i(k) - z^\star\right|\right|_\infty \leq \epsilon$ for all i. Of course, the minimzer $z^\star$ is unknown in general, and for this reason we use $D_0 := \max_{v_1, v_2 \in \mathcal{Z}} ||v_1 - v_2||_\infty$ as an upper bound on the initial distance of the agents' iterates to a minimizer.

Theorem 3. *Consider Problem 1, and let $\mathcal{X}$ satisfy Assumption 1, f satisfy Assumption 2, the Hessian satisfy Assumption 3, and let Assumption 4 hold. Given $\gamma \in \left(0, \frac{1}{\max_{i\in\mathcal{V}} \max_{\eta\in\mathcal{X}} |H_{ii}(\eta)|}\right)$ and $\lambda \in \left(0, \frac{\gamma\mu}{2(1-\gamma\mu)}\right)$, in order for agent i to be within ϵ of the minimizer $z^\star$ for all $i \in \mathcal{V}$, i.e., to attain $\max_{i\in\mathcal{V}} \left|\left|z^i(k) - z^\star\right|\right|_\infty \leq \epsilon$, agent i must have performed at least β computations and communicated at least $\beta\left|\mathcal{V}^i\right|$ times, where $\beta = \frac{\log(\epsilon/D_0)}{\log(\alpha)}$.*

Proof. See Theorem 3 in authors' technical report [19]. □

VI. Simulations

To demonstrate the effectiveness of Algorithm 1, we provide a simulation in MATLAB. We consider a network of 10 agents with the objective function

$$f(x) = \frac{3}{10}\sum_{i=1}^{n}(x^i_i)^2 + \frac{1}{200}\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n}(x^i_i - x^i_j)^2,$$

where $\mu = 0.3$. Our constraint set for each agent $i \in \mathcal{V}$ is $\mathcal{Z}_i = [1, 10]$. We define the hyperparameters λ and γ as $\lambda = 0.058$ and $\gamma = 0.345$. The initial conditions for each agent are $z^i_0 = (10\cdot\mathbf{1}, 10\cdot\mathbf{1})$, where $\mathbf{1}$ is the 10-dimensional vector of ones. We use a uniform probability distribution to determine the probability of an agent updating and communicating at each time step, i.e., what time steps are in the sets K^i and R^i_j. We consider a range of probabilities

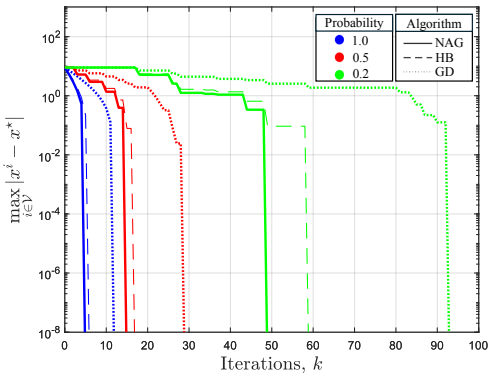

Fig. 1. The evolution of the worst-performing agent in the totally asynchronous NAG algorithm (solid lines), heavy ball algorithm (dashed lines), and gradient descent (dotted lines).

TABLE I

OPERATION COMPLEXITY OF NAG, HB, AND GD.

Probability	Iterations			Percent Reduction	
	NAG	HB	GD	HB	GD
1.0	5	6	12	17%	58%
0.9	7	7	15	0%	53%
0.8	10	11	17	9%	41%
0.7	8	10	20	20%	60%
0.6	10	11	24	9%	58%
0.5	15	17	29	12%	48%
0.4	15	19	34	21%	56%
0.3	24	29	58	17%	59%
0.2	49	59	93	17%	47%
0.1	123	170	314	28%	61%

in the interval $[0.1, 1]$, where a probability 1 of updating and communicating leads to the double-step synchronous algorithm of Section III-B.

Figure 1 shows the distance to the optimum for three probabilities in $[0.1, 1]$, and Table I provides the operation complexity results for a wider range of probabilities. Both demonstrate the faster convergence rate of Algorithm 1 compared to the totally asynchronous heavy ball (HB) algorithm in [15] and the totally asynchronous gradient descent (GD) algorithm. In simulations, it was seen that the HB method converged in, at best, the same time as NAG, though often slower, as Figure 1 and Table I show. The GD method was seen to converge substantially slower than both NAG and HB. In addition, Figure 1 shows that NAG outpaces GD and HB by a wider margin as the probabilities of updating and communicating decrease. We observe a maximum reduction in convergence time of 28% between NAG and HB, and a maximum reduction in convergence time of 61% between NAG and GD. This indicates the improved performance of NAG relative to existing algorithms in the totally asynchronous context.

VII. CONCLUSION

This paper presented what is, to the best of our knowledge, the first totally asynchronous implementation of Nesterov's accelerated gradient algorithm for optimization. We showed that this algorithm converges linearly in the number of agents' computations and communications when counted in a certain sequence, and simulations showed that it converges faster than comparable heavy ball and gradient descent algorithms. Future work will identify additional forms of accelerated algorithms that converge under total asynchrony and explore the implementation of these techniques in teams of mobile robots whose communications are subject to jamming.

REFERENCES

[1] J. C. Derenick and J. R. Spletzer, "Convex optimization strategies for coordinating large-scale robot formations," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1252–1259, 2007.

[2] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015, pp. 109–124.

[3] J. Alonso-Mora, S. Baker, and D. Rus, "Multi-robot navigation in formation via sequential convex programming," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4634–4641.

[4] S. Bubeck *et al.*, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[5] H. Jiang, Y. T. Lee, Z. Song, and S. C.-w. Wong, "An improved cutting plane method for convex optimization, convex-concave games, and its applications," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020, pp. 944–953.

[6] J. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *IEEE Signal processing magazine*, vol. 27, no. 3, pp. 50–61, 2010.

[7] D. P. Palomar and Y. C. Eldar, *Convex optimization in signal processing and communications*. Cambridge university press, 2010.

[8] P. Tseng, D. P. Bertsekas, and J. N. Tsitsiklis, "Partially asynchronous, parallel algorithms for network flow and other problems," *SIAM Journal on Control and Optimization*, vol. 28, no. 3, pp. 678–710, 1990.

[9] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[10] 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.

[11] M. Ubl and M. Hale, "Totally asynchronous large-scale quadratic programming: Regularization, convergence rates, and parameter selection," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 3, pp. 1465–1476, 2021.

[12] K. R. Hendrickson and M. T. Hale, "Totally asynchronous primal-dual convex optimization in blocks," *IEEE Transactions on Control of Network Systems*, vol. 10, no. 1, pp. 454–466, 2023.

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

[14] F. Mansoori and E. Wei, "A fast distributed asynchronous newton-based optimization algorithm," *IEEE Transactions on Automatic Control*, vol. 65, no. 7, pp. 2769–2784, 2019.

[15] D. M. Hustig-Schultz, K. Hendrickson, M. Hale, and R. G. Sanfelice, "A totally asynchronous block-based heavy ball algorithm for convex optimization," in *2023 American Control Conference (ACC)*. IEEE, 2023, pp. 873–878.

[16] 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.

[17] R. Hannah, F. Feng, and W. Yin, "A2bcd: Asynchronous acceleration with optimal complexity," in *International Conference on Learning Representations*, 2019.

[18] D. M. Hustig-Schultz and R. G. Sanfelice, "Uniting nesterov and heavy ball methods for uniform global asymptotic stability of the set of minimizers," *Automatica*, vol. 160, p. 111389, 2024.

[19] E. Pond, A. Sebok, Z. Bell, and M. Hale, "Technical report: A totally asynchronous nesterov's accelerated gradient method for convex optimization," 2024. [Online]. Available: https://arxiv.org/abs/2406.10124